

Agostino Lorenzi
Vittorio Moriggia

C++

TEORIA E AMBIENTE DI PROGRAMMAZIONE

INFORMATICA



Atlas

Agostino Lorenzi

Vittorio Moriggia

INFORMATICA

C++

**Teoria e Ambiente
di programmazione**

ATLAS

ISBN 978-88-268-9095-1

Edizioni:

2	3	4	5	6	7	8	9	10
2012	2013	2014	2015	2016				

Direzione editoriale: Roberto Invernici
Copertina: Vavassori & Vavassori
Videoimpaginazione: Claudio Tognozzi
Disegni: Claudio Tognozzi - Vavassori & Vavassori
Stampa: Vincenzo Bona - Torino

Con la collaborazione della Redazione e dei Consulenti dell'I.I.E.A.



La casa editrice ATLAS opera con il Sistema Qualità conforme alla norma UNI EN ISO 9001: 2008 certificato da CISQ CERTICARGRAF.

L'Editore si impegna a mantenere invariato il contenuto di questo volume, secondo le norme vigenti.

Il presente volume è conforme alle disposizioni ministeriali in merito alle caratteristiche tecniche e tecnologiche dei libri di testo.

L'Editore dichiara la propria disponibilità a regolarizzare errori di attribuzione o eventuali omissioni sui detentori di diritto di copyright non potuti reperire.

Ogni riproduzione del presente volume è vietata.

Le fotocopie per uso personale del lettore possono essere effettuate nei limiti del 15% di ciascun volume/fascicolo di periodico dietro pagamento alla SIAE del compenso previsto dall'art. 68, commi 4 e 5, della legge 22 aprile 1941 n. 633.

Le fotocopie effettuate per finalità di carattere professionale, economico o commerciale o comunque per uso diverso da quello personale possono essere effettuate a seguito di specifica autorizzazione rilasciata da **CLEARedi**, Centro Licenze e Autorizzazioni per le Riproduzioni Editoriali, Corso di Porta Romana 108, 20122 Milano, e-mail autorizzazioni@clearedi.org e sito web www.clearedi.org.

© 2012 by Istituto Italiano Edizioni Atlas
Via Crescenzi, 88 - 24123 Bergamo
Tel. 035/249.711 - Fax 035/216.047 - www.edatlas.it

PRESENTAZIONE

UN'OPERA MISTA, MULTIMEDIALE E DIGITALE

Il testo propone lo studio e l'applicazione delle tecnologie dell'informatica secondo le *Linee guida* della disciplina *Informatica* per il secondo biennio negli Istituti tecnici, settore tecnologico. È una proposta editoriale **mista**, composta di materiali a stampa, materiali *on line*, materiali multimediali e interattivi; inoltre è disponibile in forma digitale (*E-book*) su piattaforma dedicata.

MATERIALI A STAMPA

Il testo propone un'integrazione tra la trattazione teorica dell'informatica e la presentazione delle caratteristiche dell'ambiente di sviluppo nel linguaggio C++.

La parte iniziale, dopo i richiami alla terminologia di base e ai concetti matematici impiegati in informatica, illustra l'architettura e le risorse del sistema di elaborazione, il sistema operativo Windows, la *shell* dei comandi.

Nella seconda parte, dopo aver introdotto le linee metodologiche per la **progettazione degli algoritmi**, vengono presentate le caratteristiche e la sintassi del linguaggio C++ e le note operative per l'uso degli ambienti di sviluppo.

La **programmazione ad oggetti** è introdotta non soltanto a livello teorico, ma mostrando, in modo facile e intuitivo per gli studenti, come essa sia il paradigma che trova un'ampia applicazione pratica nello sviluppo del software moderno, basato su classi e oggetti. Viene presentata anche la **programmazione guidata dagli eventi** e la costruzione di interfacce per l'utente.

L'ultima parte riguarda le reti e Internet, gli elementi fondamentali del codice HTML e dei fogli di stile per creare le pagine Web, oltre al linguaggio di programmazione lato client (*JavaScript*) per la gestione locale di eventi in Internet(*).

Alla fine del testo ci sono le schede **Focus notes** per l'utilizzo del lessico e della terminologia di settore anche in lingua inglese.

Gli aspetti applicativi sono presentati sotto forma di **progetti**, accompagnati da note pratiche e videate di spiegazione, per favorire la didattica laboratoriale a scuola e la rielaborazione personale dello studente a casa. Lo studente viene guidato alla risoluzione di problemi secondo una precisa metodologia di sviluppo del progetto informatico.

All'inizio di ogni capitolo sono enunciati gli obiettivi di apprendimento e i contenuti trattati. Lo studente può svolgere un'efficace attività di **autoverifica** attraverso le domande strutturate (con soluzioni alla fine del testo) e i numerosi problemi proposti come esercitazioni, simili ai progetti svolti all'interno dei capitoli. Ogni capitolo si chiude con la scheda di autovalutazione che contiene l'elenco delle conoscenze e delle abilità previste come esito di apprendimento al termine del capitolo.

(*) Le tecnologie Web lato server (ASP.NET e Php) con accesso ai database in rete sono trattate nel volume: A. Lorenzi, E. Cavalli - Progettazione dei database, Linguaggio SQL, Dati in rete (Access, MySQL, Pagine ASP, Pagine Php) - Atlas, ISBN 978-88-268-1675-3.

MATERIALI ON LINE

I materiali a stampa si integrano e si completano con una grande ricchezza e varietà di materiali on line, in diversi formati, disponibili per Studenti e Docenti sulla **Libreria Web**, accessibile tramite un collegamento diretto alla home page del sito dell'Atlas: www.edatlas.it oppure con il seguente codice QR per dispositivi mobili.



I materiali riguardano non solo ampi approfondimenti e integrazioni dei contenuti trattati nel testo, ma anche i materiali digitali e multimediali utili per lo sviluppo dei progetti proposti, i programmi eseguibili dei progetti del testo con videate delle fasi di esecuzione che lo studente può usare per controllare l'attività di programmazione personale o di gruppo, le note e le lezioni multimediali sull'uso operativo degli ambienti e delle tecnologie software. I riferimenti ai *Materiali on line* sono indicati, in modo puntuale e con numerazione progressiva, al termine dei paragrafi di ogni capitolo, richiamati con l'apposito simbolo.



I riferimenti sono inoltre elencati nell'indice generale del testo.

I materiali sono un *repository* alimentato e aggiornato in modo continuo, perché diventino parte integrante del testo, del quale costituiscono un'espansione didattica a supporto di un migliore approfondimento della disciplina. Inoltre permettono di avere un volume cartaceo più agile.

MATERIALI MULTIMEDIALI E DIDATTICI PER L'INSEGNANTE

A disposizione del Docente ci sono innanzitutto i **Materiali didattici per l'insegnante**, disponibili nell'*area riservata* del sito della Casa Editrice, a cui i Docenti possono accedere con password, e contemplano:

- traccia per la compilazione dei Piani di lavoro per i Consigli di classe
- soluzioni ai problemi del testo
- repertorio di esercizi da assegnare come verifiche in classe.

Inoltre i Docenti possono disporre, a richiesta, di **materiali multimediali e interattivi** su supporto digitale:

- presentazioni in *PowerPoint* e in *pdf* che illustrano i contenuti dei capitoli e che possono essere utilizzati con la LIM per lezioni multimediali in classe;
- codici sorgente completi dei programmi presentati nel volume;
- file completi dei progetti di pagine Web e fogli di stile;
- ulteriore repertorio di esercizi che possono essere assegnati come autoverifiche per gli studenti.

L'Editore

INDICE

PARTE PRIMA - PREMESSE INTRODUTTIVE	11
Capitolo 1 - Nozioni di base e richiami matematici	11
1 L'evoluzione dell'informatica	12
2 Le risorse tecnologiche e le applicazioni	13
3 I termini di uso comune	16
4 I sistemi di numerazione	21
5 Il sistema binario	21
■ Le operazioni nel sistema binario	22
■ Il sistema ottale	24
6 Il sistema esadecimale	25
7 Regole di conversione binario/esadecimale	25
■ Regole di conversione binario/ottale	26
8 La codifica delle informazioni	27
9 Rappresentazione dei numeri interi	28
10 Rappresentazione dei numeri reali	29
11 Rappresentazione delle informazioni alfanumeriche	31
12 Proposizioni e connettivi logici	34
13 Tavole di verità	36
14 Le funzioni	39
AUTOVERIFICA: Domande	40
Problemi	42
SCHEDA DI AUTOVALUTAZIONE	46
PARTE SECONDA - L'AMBIENTE DI LAVORO CON IL COMPUTER	47
Capitolo 2 - Il sistema di elaborazione	47
1 Concetto intuitivo di dispositivo automatico	48
■ Definizione formale di automa	49
2 Struttura generale del sistema di elaborazione	52
3 L'unità centrale di elaborazione	54
4 La memoria centrale	56
5 Le unità di input e di output	60
6 Le memorie di massa	62
7 Il collegamento delle periferiche	65
8 Il software	66
9 Il sistema operativo	67
10 La struttura modulare di un sistema operativo	68
■ La sicurezza	71
11 L'interprete dei comandi	75
12 I linguaggi di programmazione	76
13 Il software applicativo	78
■ Lo sviluppo del software	79
AUTOVERIFICA: Domande	81
Problemi	83
SCHEDA DI AUTOVALUTAZIONE	86
Capitolo 3 - Il sistema operativo Windows	87
1 Caratteristiche generali	88
2 Avvio e arresto del sistema	88
3 Il desktop	90
4 Il Pannello di controllo	92
5 Il mouse	93
6 Installazione e rimozione di nuovi programmi	94
7 Multitasking	94
8 L'interfaccia standard delle applicazioni	96
9 Il pulsante Start	97
■ Il programma per l'editing dei testi	99
10 Cartelle e file	101

11	Le operazioni sui file	104
12	La compressione dei file	108
	■ I virus	110
13	La Guida in linea	111
14	Gestione di una stampante	113
15	Connessione e rimozione di periferiche	114
16	Copie di sicurezza	114
	■ Lavorare in rete	115
	AUTOVERIFICA: Domande	119
	Problemi	120
	SCHEDA DI AUTOVALUTAZIONE	122
Il Prompt dei comandi		123
1	La finestra Prompt dei comandi	123
2	I file e le directory	124
3	I comandi	126
4	I comandi per la gestione di directory e file	127
5	La ridirezione di input e output	131
6	La pipeline	134
	■ Gestione del sistema e delle periferiche	135
7	I file batch	137
	AUTOVERIFICA: Problemi	144
PARTE TERZA - LA PROGRAMMAZIONE		147
Capitolo 4 - Progettazione degli algoritmi		147
1	Modello del problema	148
2	Dati e azioni	149
3	La metodologia di lavoro	152
4	Definizione di algoritmo	153
	■ Caratteristiche dell'algoritmo	154
5	Algoritmo ed esecutore	155
6	Acquisire e comunicare i dati	156
7	Gli operatori	158
8	Strumenti per la stesura di un algoritmo	159
9	L'individuazione dei dati di un problema	163
	■ La macchina di Turing	164
10	Le strutture di controllo	168
11	La struttura di alternativa	173
	■ Osservazioni sulla struttura di alternativa	174
12	La ripetizione	176
	■ Le strutture derivate	178
13	Ripetizione precondizionale	178
	■ Confronto tra le strutture di ripetizione	180
14	La ripetizione con contatore	181
	■ La struttura di scelta multipla	183
15	Organizzazione degli algoritmi	186
16	Procedure con parametri	192
	■ Passaggio di parametri	195
17	Funzioni	196
18	Funzioni ricorsive	197
	AUTOVERIFICA: Domande	199
	Problemi	203
	Problemi di riepilogo	208
	SCHEDA DI AUTOVALUTAZIONE	210
Capitolo 5 - Il linguaggio C++		211
1	Le basi del linguaggio	212
2	Gli statement	215
3	La dichiarazione delle variabili e delle costanti	216
	■ I tipi di dati del linguaggio C++	218

4	Le frasi di commento	219
5	L'assegnazione dei valori alle variabili	219
	■ Operatore ++	222
6	Il casting per la conversione di tipo	223
7	Gli operatori di relazione e logici	225
8	Le istruzioni di ingresso e uscita	227
9	Le fasi della programmazione	230
10	L'importanza della documentazione	233
11	La programmazione strutturata	234
12	La sequenza	235
13	La struttura di alternativa	236
	■ Le strutture annidate di alternativa	239
14	La ripetizione	240
15	La ripetizione precondizionale	242
16	La ripetizione con contatore	245
	■ La struttura di ripetizione <i>for</i> e l'operatore virgola	247
	■ La struttura di scelta multipla	249
17	Lo sviluppo top-down	251
18	Le funzioni	252
19	Funzioni con parametri	255
20	Il passaggio di parametri	257
21	Dichiarazione delle funzioni con i prototipi	262
	■ Regole di visibilità	265
22	Le funzioni predefinite	267
	Namespace e librerie di inclusione	269
23	Function overloading	272
	■ La ricorsione	274
24	Costanti ed enumerazioni	277
25	L'array	279
26	L'array a due dimensioni	283
27	Le strutture	286
28	I puntatori	289
	AUTOVERIFICA: Domande	295
	Problemi	302
	Problemi di riepilogo	307
	SCHEDA DI AUTOVALUTAZIONE	308
	Ambienti di sviluppo in C++	309
	CodeLite	309
	Visual C++	313
	Capitolo 6 - Programmazione ad oggetti	319
1	L'astrazione	320
2	Programmare con gli oggetti	321
3	La classe	322
4	Creazione di una classe	325
5	Costruttore e distruttore	329
	■ Costruttore di copia	333
6	Visibilità dei membri	335
	■ Funzioni friend	338
7	Ereditarietà	342
	■ Gerarchia delle classi ed ereditarietà multipla	346
8	Polimorfismo e overloading	349
	■ Membri <i>protected</i>	350
9	Riscrittura di metodi nelle classi derivate	351
	■ Classi e <i>overloading</i> degli operatori	357
10	Funzioni virtuali	361
	■ Osservazioni sulla programmazione orientata agli oggetti	364
	AUTOVERIFICA: Domande	365
	Problemi	368
	SCHEDA DI AUTOVALUTAZIONE	370

Capitolo 7 - Strutture di dati e algoritmi	371
1 Ricerca di un elemento in un array	372
2 Ordinamento degli elementi di un array	374
3 Array paralleli	376
4 La tabella	378
5 La pila	382
■ Implementazione della pila con un array	382
6 La coda	385
■ Implementazione della coda con un array	385
7 Le liste	387
8 Template di funzioni	393
9 Template delle classi	394
■ Template di classe con due parametri	398
10 La libreria standard STL	401
11 Il contenitore per le stringhe	402
12 Il contenitore <i>vector</i>	404
13 Gestione delle eccezioni	406
14 Iteratori	408
■ Uso di un template di funzione per visualizzare un <i>vector</i>	409
15 Il contenitore <i>list</i>	410
16 Il contenitore <i>map</i>	413
17 Algoritmi generici	418
18 Algoritmi per l'ordinamento	422
■ Altri algoritmi standard	424
AUTOVERIFICA: Domande	425
Problemi	429
SCHEDA DI AUTOVALUTAZIONE	432
 Capitolo 8 - Flussi di dati	 433
1 Le classi di input/output	434
2 Input e output standard	435
■ Overloading degli operatori <<	437
3 Archivi di dati	439
4 File di testo e file binari	440
5 Operazioni e accesso agli archivi	441
6 Lettura e scrittura in un file di testo	442
7 Apertura esplicita dei file	445
8 Errori nelle operazioni sui file	446
■ Gestione strutturata delle eccezioni	447
9 Controllo della fine del file	450
10 Stream su stampante	451
11 Uso degli stream in un'applicazione gestionale	452
12 Accesso sequenziale per gli archivi con record a lunghezza costante	457
13 Iteratori per gli stream	460
AUTOVERIFICA: Domande	466
Problemi	468
SCHEDA DI AUTOVALUTAZIONE	470
 Programmazione guidata dagli eventi e interfaccia utente	 471
1 L'ambiente di programmazione	471
2 Gli oggetti per le interfacce grafiche	474
3 Creazione di applicazioni in Visual C++	475
4 Form e controlli	478
■ Le proprietà degli oggetti	480
5 Form	482
6 Pulsanti di comando, caselle di testo e finestre di messaggio	482
7 Caselle di controllo e pulsanti di opzione	488
8 Casella di riepilogo e casella combinata	491
9 Finestre di dialogo comuni	494
10 I menu	498
11 Metodi e gestione degli eventi	501
AUTOVERIFICA: Problemi	507

PARTE QUARTA - RETI E WEB	509
Capitolo 9 - Reti e Internet	509
1 Le reti	510
2 Classificazione delle reti	510
3 Mezzi trasmissivi e dispositivi di rete	512
4 Le reti pubbliche	515
■ Intranet e Extranet	516
5 I protocolli	517
6 Internet e Web	518
■ Il sistema dei nomi di dominio	519
7 Le applicazioni di Internet e del Web	520
8 La sicurezza delle reti	524
AUTOVERIFICA: Domande	526
Problemi	529
SCHEDA DI AUTOVALUTAZIONE	530
Capitolo 10 - Pagine Web e fogli di stile	531
1 Le pagine per i siti Internet	532
■ I formati degli oggetti multimediali	534
2 Il linguaggio HTML	539
■ Caratteri speciali	543
3 La formattazione	546
■ I nomi dei file	552
■ Unione di celle nelle tabelle	555
4 Elementi multimediali	556
5 I form e l'interazione con l'utente	560
6 Fogli di stile CSS	563
7 Accessibilità e usabilità	586
■ Creazione di siti Web	588
AUTOVERIFICA: Domande	590
Problemi	593
SCHEDA DI AUTOVALUTAZIONE	596
Programmazione Web con JavaScript e HTML5	597
1 Il linguaggio JavaScript	597
2 Le canvas	613
3 Cookie e Webstorage	626
4 Validazione dei dati di un form	631
AUTOVERIFICA: Problemi	633
FOCUS NOTES	635
Operating systems	636
Computer programming	637
Object Oriented Programming	637
Computer networks	638
Internet and Web	639
Publishing Web pages	640
Glossary	641
Acronyms	644
APPENDICE	645
Tabella dei caratteri	646
Pseudocodifica	648
Linguaggio C++	649
Tag HTML	656
Tabella dei colori	659
Soluzioni alle domande di autoverifica	662
Indice analitico	664

Indice dei Materiali on line



Capitolo 1 - Nozioni di base e richiami matematici

1. Sistemi di numerazione con il foglio elettronico Excel
2. Che cos'è Unicode
3. Esempi di Codici Unicode
4. Operatori booleani in Excel
5. Operatore XOR in Excel

Capitolo 2 - Il sistema di elaborazione

1. Esempi di automi
2. Aggiornamenti sui sistemi di elaborazione e i dispositivi hardware
3. Il software open source
4. Traccia riassuntiva sui sistemi di elaborazione (hardware e software)

Capitolo 3 - Il sistema operativo Windows

1. Lezioni multimediali sull'ambiente di Windows
2. Lezioni multimediali sulla gestione dei file
3. Lezioni multimediali sulla gestione delle stampe
4. Eseguire copie di sicurezza (lezione multimediale)
5. Condividere una cartella (lezione multimediale)

Il Prompt dei comandi

1. Editor di testi
2. Attributi sui file
3. Comandi avanzati per i file batch
4. File batch per le operazioni aritmetiche

Capitolo 4 - Progettazione degli algoritmi

1. Diagrammi a blocchi con Word
2. Esempi di Macchina di Turing
3. Teoria della complessità algoritmica
4. Approfondimento sul Teorema di Böhm-Jacopini
5. Ripetizione precondizionale per il controllo di fine input
6. Algoritmi diversi per la soluzione dello stesso problema
7. Esempi di procedure e funzioni ricorsive

Capitolo 5 - Il linguaggio C++

1. I linguaggi C e C++
2. Manipolatori per l'Input/Output formattato
3. Utilizzo degli output formattati
4. Namespace e file di intestazione
5. Programmi eseguibili e videate dei progetti
6. File di intestazione e file di inclusione
7. Scomposizione dei programmi con le funzioni
8. Programmi eseguibili e videate dei progetti
9. Programmi eseguibili e videate dei progetti

Capitolo 6 - Programmazione ad oggetti

1. Strumenti per disegnare i diagrammi delle classi (UML)
2. Programmi eseguibili e videate dei progetti del capitolo
3. Lezioni multimediali sulla programmazione ad oggetti in C++

Capitolo 7 - Strutture di dati e algoritmi

1. Ricerca di un elemento in un array ordinato (ricerca dicotomica)
2. Passaggio di array alle funzioni
3. Gestione di pila e coda con i puntatori
4. La struttura dati ad albero binario
5. La funzione *getline()*
6. Altri contenitori standard
7. Programmi eseguibili e videate dei progetti presentati nel capitolo
8. Lezioni multimediali sulla programmazione delle strutture di dati in C++

Capitolo 8 - Flussi di dati

1. Applicazione dell'overloading dell'operatore <<
2. Manipolatori e formattazione dell'output
3. Programmi eseguibili e videate dei progetti sui file di testo
4. Accesso diretto per gli archivi con record a lunghezza costante
5. La stringa come stream
6. Programmi eseguibili e videate dei progetti sugli stream
7. Lezioni multimediali sulla gestione dei file in C++

Programmazione guidata dagli eventi e interfaccia utente

1. Progetti con interfaccia grafica per l'utente
2. Altri controlli per l'interfaccia utente
3. Barra degli strumenti
4. Menu contestuali
5. Programmi eseguibili dei progetti

Capitolo 9 - Reti e Internet

1. Nomi di dominio
2. Ricerche avanzate in Internet
3. La Netiquette in Internet
4. Telefonare con il VoIP
5. Impostare la privacy di Facebook
6. La licenza GNU GPL

Capitolo 10 - Pagine Web e fogli di stile

1. Verifica della compatibilità del browser con la versione HTML5
2. Programmi per l'editing di pagine Web con guida pratica per il loro utilizzo
3. Mappe sensibili al mouse
4. Validazione del codice HTML
5. Validatori dei fogli di stile CSS
6. Software per creare fogli di stile CSS
7. Linee guida e norme per l'accessibilità
8. Tool per l'accessibilità

Programmazione Web con JavaScript e HTML5

Codice HTML e JavaScript dei progetti

1

parte prima

Premesse introduttive

Nozioni di base e richiami matematici

OBIETTIVI DI APPRENDIMENTO

In questo capitolo imparerai a riconoscere e ad usare alcuni termini di uso comune nel contesto specifico dell'informatica.

Potrai inoltre vedere come i concetti della matematica siano utilizzati nell'informatica, in particolare i metodi di rappresentazione e memorizzazione delle informazioni e i principi di base della logica formale.

L'evoluzione dell'informatica

Le risorse tecnologiche e le applicazioni

I termini di uso comune

I sistemi di numerazione

Il sistema binario

Il sistema ottale

Il sistema esadecimale

Regole di conversione binario/esadecimale

Regole di conversione binario/ottale

La codifica delle informazioni

Rappresentazione dei numeri interi

Rappresentazione dei numeri reali

Rappresentazione delle informazioni alfanumeriche

Proposizioni e connettivi logici

Tavole di verità

Equivalenza logica e proprietà dell'algebra booleana

Le funzioni

1 L'evoluzione dell'informatica

Il rapido sviluppo dell'informatica e delle sue potenzialità ha determinato un'estensione delle risorse informatiche in molti ambiti applicativi, non solo in quelli legati alla gestione aziendale. La tecnologia informatica inoltre si è legata strettamente alle telecomunicazioni, fornendo un sistema integrato di elaborazione e di comunicazione.

Le attività connesse al settore dell'informatica sono indicate con la sigla **ICT** (*Information and Communication Technology*), cioè un insieme vario di tecnologie che serve a conservare, elaborare e trasmettere informazioni.

Le macchine informatiche che si usano nei processi automatizzati non sono dei semplici computer, ma piuttosto sistemi di elaborazione, cioè insiemi organizzati di risorse di tipo diverso finalizzati all'attività di elaborazione, ma anche di comunicazione.

Il computer non è soltanto una macchina e uno strumento per scrivere programmi (**programmazione**), oppure per usare i programmi scritti da altri (**elaborazione**), ma è diventato un potente mezzo per **comunicare** e per **accedere** alla rete, rendendo ininfluenti le distanze geografiche, nei rapporti tra le persone, gli enti, le aziende.



Lo scopo dell'**Informatica**, neologismo che deriva dall'unione di due parole **Informazione** e **Automatica**, consiste nel progettare e usare macchine che siano in grado di trattare informazioni senza l'intervento dell'uomo, in modo appunto *automatico*.

Gli oggetti dell'informatica servono ad acquisire informazioni dal mondo esterno, per memorizzarle su supporti permanenti, e per elaborarle al fine di ottenere nuove informazioni.

I dati arrivano al computer non solo dalla tastiera dell'utente, ma anche da altri strumenti, per esempio lo scanner o la macchina fotografica, che consentono di acquisire documenti e immagini in formato digitale, oppure da apparecchiature in grado di trasmettere al computer dati o comandi tramite messaggi vocali.

L'apertura verso le telecomunicazioni consente di ottenere informazioni, non solo dall'interno dell'azienda, ma anche da banche dati esterne e geograficamente lontane.

La conservazione dei dati viene realizzata attraverso memorie permanenti di capacità sempre più elevate, costituite da supporti magnetici od ottici.

L'elaborazione dei dati avviene a velocità molto alte e i risultati possono essere messi a disposizione in tempi brevi, anche per utenti esterni all'azienda, attraverso le infrastrutture della comunicazione.

Le tecnologie dell'informazione e delle telecomunicazioni diventano centrali in ogni forma di innovazione, nelle aziende, nella Pubblica Amministrazione, nelle famiglie.

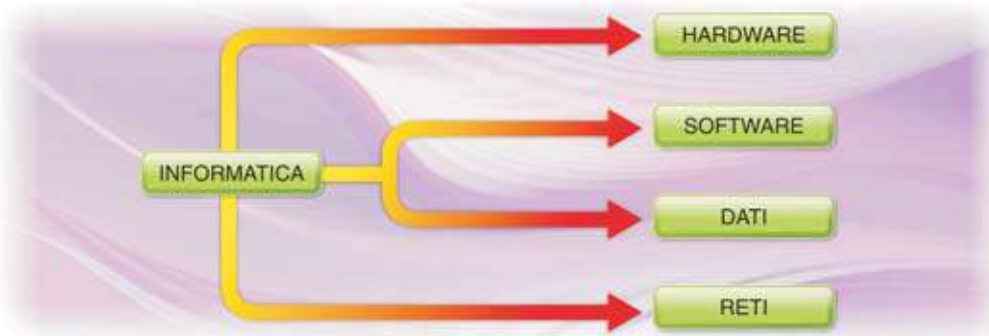
Le nuove tecnologie sono il frutto della convergenza di settori diversi quali l'informatica, l'elettronica e le telecomunicazioni. Questo consente di realizzare applicazioni nelle quali sono presenti componenti informatiche, provenienti dai computer, insieme a componenti multimediali provenienti dai media e dall'elettronica e componenti di comunicazione provenienti dalle reti.



2 Le risorse tecnologiche e le applicazioni

I problemi, gli strumenti, le risorse dell'informatica e le loro applicazioni possono essere classificati secondo quattro grandi categorie:

- **hardware**, cioè la parte fisica dei sistemi di elaborazione;
- **software**, cioè i programmi che fanno funzionare il computer e realizzano le applicazioni informatiche;
- **organizzazione e supporti per i dati**, cioè le modalità e gli oggetti che vengono utilizzati per conservare le informazioni;
- **reti di comunicazione**, cioè i protocolli e i servizi applicativi delle reti aziendali e della rete Internet.



L'evoluzione tecnologica, per quanto riguarda l'hardware, è caratterizzata dalla disponibilità di processori molto veloci e da memorie di grandi capacità. Il grande sviluppo dell'elettronica, o meglio della microelettronica, negli ultimi decenni ha prodotto strumenti di dimensioni sempre più piccole (**miniaturizzazione**), con capacità elaborative o di memorizzazione molto elevate, con costi di produzione bassi, e tempi brevi di assemblaggio.

I sistemi di elaborazione personali (detti comunemente **Personal Computer** o **PC**) hanno dimensioni limitate e capacità elaborative elevate, in termini di velocità di elaborazione, oltre che di memoria, per le applicazioni e i dati dell'utente. Le dimensioni ridotte e i prezzi bassi hanno fatto diventare il personal computer un prodotto di massa, diffuso nelle imprese, negli enti pubblici, negli studi professionali, ma anche nelle famiglie.

Lo sviluppo del software riguarda prima di tutto i **sistemi operativi**, cioè l'insieme dei programmi di base che fanno funzionare il computer: essi si sono estesi nelle funzioni e nel miglioramento delle prestazioni, con l'obiettivo di adeguare le attività degli utenti alle crescenti potenzialità dell'hardware.

Nel corso degli anni sono cambiate anche le metodologie per la costruzione di programmi, attraverso nuovi **linguaggi di programmazione**, meno simbolici e più vicini al linguaggio naturale: essi da una parte rendono più facile il lavoro del programmatore, e dall'altra consentono di creare programmi più complessi e ricchi di funzionalità.

In questo contesto viene assegnata grande importanza alle modalità di interazione tra utente e computer, attraverso la costruzione di **interfacce** software di tipo grafico, organizzate a menu o a finestre con le icone, per rendere più facile l'approccio al computer e alle applicazioni. L'informazione è una risorsa strategica per l'azienda, e perciò i dati devono essere a disposizione in modo aggiornato e integrato: questo porta alla costruzione di **basi di dati**, cioè collezioni organizzate di archivi in linea, che richiedono, per la memorizzazione, dischi e nastri di grandi capacità.

Le nuove **applicazioni multimediali** dell'informatica necessitano poi di grandi quantità di memoria per registrare immagini, filmati, suoni; sono perciò entrati nell'uso comune i **dischi ottici**, e in particolare i CD-ROM e i DVD, che consentono, a costi bassi, l'utilizzo di informazioni organizzate e di programmi di tipo diverso, quali enciclopedie o raccolte di leggi, ma anche giochi e filmati.



Le nuove opportunità offerte dalla tecnologia consentono quindi di rendere più veloci le tradizionali applicazioni gestionali e di trattare in tempi brevi i dati contenuti in archivi di grandi dimensioni.

L'adozione di strumenti e programmi informatici nelle aziende avviene oggi in modo più razionale e organico, in modo da diventare soluzioni integrate che vanno a incidere anche nella struttura, nell'organizzazione e nei costi aziendali.

Gli **aspetti applicativi** più rilevanti riguardano:

- il consolidamento degli strumenti hardware e software per l'automazione dei tradizionali lavori di ufficio (**office automation**) e in particolare il trattamento dei testi e l'archiviazione dei documenti;
- la gestione integrata degli archivi aziendali in basi di dati (**database**), per mettere a disposizione, a diversi utenti e per diverse applicazioni, le informazioni che servono, garantendo l'aggiornamento e la consistenza dei dati;

- l'uso di strumenti informatici e di software non solo ai livelli operativi e amministrativi dell'azienda, ma anche da parte di quadri aziendali e manager, per il controllo e il supporto alle decisioni.

Le soluzioni per l'azienda offerte dall'informatica non solo incidono nel miglioramento e nella modernizzazione degli uffici, ma soprattutto determinano l'aumento della produttività nelle imprese.

Le infrastrutture avanzate della comunicazione (**reti informatiche**) diventano strategicamente importanti per la competitività delle aziende.

I computer sono collegati tra loro a formare connessioni che possono essere di tipo locale o di tipo geograficamente più esteso, ma che hanno in comune l'obiettivo di condividere risorse (hardware, software o dati). Le distanze geografiche non sono importanti, così come le dimensioni e le caratteristiche dei computer che stanno nei nodi della rete. Questo consente di definire soluzioni informatiche basate sul **networking**, cioè il lavoro cooperativo in rete tra utenti diversi, con due aspetti fondamentali: le risorse non sono concentrate, ma distribuite, e le risorse non sono duplicate, ma condivise.



L'idea di poter comunicare tramite i computer, o di poter accedere alle informazioni di generi diversi, è entrata nella cultura di massa attraverso la diffusione di **Internet**, la rete di reti che è diventata l'infrastruttura fondamentale di collegamento, e che rende le distanze insignificanti, prefigurando una nuova società dell'informazione.

L'informatica mobile rende disponibile l'accesso alla rete Internet da qualsiasi luogo, attraverso dispositivi di piccole dimensioni, quali telefoni cellulari e computer portatili.

Lo sviluppo di Internet consente alle aziende l'uso della rete per svolgere attività essenziali per le imprese moderne:

- la **posta elettronica**, cioè lo scambio di messaggi attraverso i computer connessi a Internet, con possibilità di collegamenti con l'intero pianeta;
- la possibilità di accedere a **banche dati** esterne all'azienda per avere informazioni sempre aggiornate, che sono una risorsa strategicamente importante;
- la presentazione e la vendite dei servizi e dei prodotti dell'azienda tramite Internet, con il commercio elettronico a livello internazionale.

3 I termini di uso comune

Prima di entrare nel dettaglio dei fondamenti dell'informatica e del funzionamento del computer, è opportuno richiamare alcuni concetti che verranno usati nel seguito della trattazione: essi riguardano la definizione dei termini secondo la connotazione specifica assunta in informatica e le premesse matematiche indispensabili per comprendere gli argomenti presentati nei capitoli successivi.

Informazione

In generale si può dire che l'**informazione** è tutto ciò che possiede un significato per l'uomo, e che viene conservato o comunicato in vista di un'utilità pratica, immediata o futura.

L'informazione presuppone da una parte un fenomeno del mondo reale oppure una persona che possiede una conoscenza, e dall'altra una persona che non sa, ma vuole ottenere una maggiore conoscenza.

Le informazioni presuppongono l'esistenza di un insieme di regole comuni all'emittente e al ricevente, chiamate **codice**, secondo le quali le informazioni vengono trasformate per essere comprese dal ricevente.

L'esempio più semplice di codice è costituito dalla lingua naturale (per esempio, la lingua italiana).



La persona A (**emittente**), che vuole comunicare un'informazione, opera una trasformazione (**codifica**) prima di trasmettere alla persona B (**ricevente**), la quale a sua volta deve completare la trasformazione inversa (**decodifica**) per comprendere il significato dell'informazione. Il procedimento viene ripetuto a rovescio quando la persona B diventa l'emittente e la persona A diventa il ricevente.

Questo vale anche quando l'entità B, anziché essere una persona, è costituita da un computer che non è in grado di comprendere il significato delle informazioni nelle forme che noi utilizziamo nei rapporti sociali e interpersonali.

Dati ed elaborazione

L'osservazione del mondo reale ci consente di ottenere gli elementi che utilizziamo per esprimere un'opinione, per caratterizzare un fenomeno oppure per risolvere un problema.

A questi elementi diamo il nome di **dati**: il concetto di dato presuppone la volontà di volere conservare qualcosa nella memoria o negli archivi, in vista di un successivo trattamento per produrre le informazioni, che forniscono una maggiore conoscenza della realtà sulla quale si intendono attivare operazioni di controllo, modifica o direzione.

Quindi c'è una distinzione tra il dato e l'informazione nel senso che:

- il *dato* descrive aspetti elementari di entità o fenomeni
- l'*informazione* è un insieme di dati elaborati e presentati sulla base dell'esigenza di utilizzo pratico da parte delle persone interessate.

Il trattamento dei dati per ottenere le informazioni viene indicato con il termine **elaborazione**.

Il caso più semplice di elaborazione è costituito dalla trascrizione (per esempio, l'impiegato dell'ufficio dell'anagrafe copia i dati di un cittadino sulla tessera elettorale).

Un'elaborazione più complessa è il calcolo matematico, che opera su dati numerici e presuppone la conoscenza di regole predefinite.

Altri tipi di elaborazione ancora più complessi sono: la trasformazione di figure piane, la definizione di uno schema, la costruzione di una teoria scientifica.

Ogni tipo di elaborazione necessita di dati in ingresso (**input**) e produce dati in uscita (**output**). Il risultato del lavoro di elaborazione è costituito da altri dati, che possono essere a loro volta utilizzati per altre elaborazioni a un livello superiore.



Trasmissione

Perché sia possibile la comunicazione delle informazioni tra emittente e ricevente è necessario che essi riescano a capirsi.

Si pensi allo scambio di informazioni tra due persone che parlano due lingue diverse.

Perché questa esigenza venga soddisfatta occorre che siano stabiliti e rispettati:

- il **codice** della trasmissione, cioè le regole secondo le quali il messaggio viene trasformato prima di essere inviato;
- la **tecnica** di trasmissione, cioè la modalità e gli strumenti con i quali viene effettuata la trasmissione;
- il **protocollo** della trasmissione, cioè l'insieme delle regole che permettono uno scambio ordinato e corretto delle informazioni.



I caratteri e i simboli che compongono il messaggio vengono trasformati per viaggiare sul mezzo trasmissivo che collega l'emittente e il ricevente e prendono il nome di **segnali**.

Figurativamente si dice che il flusso dei segnali sul mezzo trasmissivo avviene attraverso un **canale** di trasmissione.

Nella pratica quando viene effettuata la trasmissione di informazioni, oltre al messaggio da comunicare, vengono trasmessi altri segnali, per controllare il traffico dei messaggi e le operazioni di avvio, mantenimento e conclusione della trasmissione.

Si pensi in una comune telefonata al segnale di connessione alla rete telefonica oppure al segnale di numero libero o occupato.

Nelle telecomunicazioni i segnali possono essere di tipo **digitale** quando sono associati a cifre, oppure di tipo **analogico** quando vengono rappresentati da grandezze che variano con continuità nel tempo. Per esempio, il computer tratta segnali digitali, mentre sulle linee telefoniche viaggiano segnali analogici.

Comunicazione

La **comunicazione** consiste nel dialogo stabilito tra due persone, tra una persona e una macchina, in generale tra due enti, per realizzare un flusso di informazioni: la comunicazione di solito è di tipo interattivo e bidirezionale, ed è costituita da domande e risposte.

Questo presuppone che esista un insieme di parole chiave conosciute da entrambi gli enti in comunicazione, e un insieme di regole per combinare le parole in frasi: in sostanza un **linguaggio** per la comunicazione.

Nell'ambito dell'informatica l'insieme delle parole con le quali l'utente può chiedere l'attivazione dei lavori a un computer si chiama **linguaggio di comandi**; le parole e le regole per costruire le frasi utilizzate nei programmi per i computer costituiscono il **linguaggio di programmazione**.

Quando due enti di natura diversa vengono messi in comunicazione è necessario utilizzare un meccanismo che possa rendere possibile la comunicazione e facilitare l'interazione: questo meccanismo prende il nome di **interfaccia**.

In informatica il termine *interfaccia* viene usato per indicare:

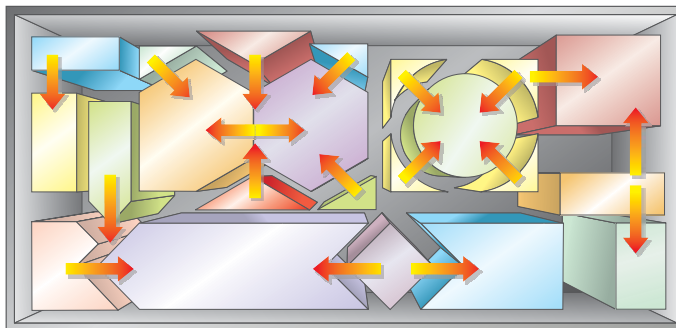
- a. le regole e gli strumenti attraverso i quali l'utente realizza la comunicazione con il computer;
- b. l'apparecchiatura elettronica che consente la comunicazione tra un computer e un dispositivo esterno ad esso collegato, per esempio una strumentazione di laboratorio oppure una stampante.



Sistema e modello

Nell'osservazione di fenomeni del mondo reale, specie se complessi, si è soliti utilizzare il concetto di sistema per descriverne il comportamento, in quanto il funzionamento del tutto può essere spiegato solo attraverso il comportamento delle parti.

Un **sistema** viene definito come un insieme anche complesso di elementi (sottosistemi) di natura differente che interagiscono tra loro in modo dinamico e comunque finalizzato al raggiungimento di certi obiettivi.



Ogni sistema è definito quando sono definite:

- le **parti** che lo compongono,
- le **correlazioni** tra esse,
- le **finalità** del sistema.

Le finalità servono per comprendere le ragioni che muovono il sistema, mentre le parti e le relative correlazioni servono per definirne il funzionamento.

Nella vita comune la visione sistemica di una realtà viene usata in modo molto frequente: sistema nervoso del corpo umano, sistema dei trasporti di una città, sistema politico di una Nazione. Per esempio, con il termine Sistema Sanitario Nazionale si indica l'insieme delle strutture, delle attrezzature, delle persone, degli organi di gestione, che forniscono i servizi sanitari nei diversi comuni e località.

Per definire il sistema dobbiamo descrivere le parti (farmacie, reparti ospedalieri, pazienti, medici di base, medici specialisti, ambulatori specialistici, direzione sanitaria, fornitori di materiali), le correlazioni (per esempio, la richiesta di un medico di base attiva l'accesso alla prestazione specialistica di un ambulatorio), le finalità (gestire gli interventi di diagnosi e di terapia, promuovere la prevenzione delle malattie, controllare la produzione e il consumo di farmaci).

Nel linguaggio corrente dell'informatica si usa spesso il termine **sistema di elaborazione**, anziché il termine computer, perché il lavoro con l'elaboratore si avvale non di un solo oggetto, ma di un insieme organizzato di apparecchiature e di programmi che interagiscono fra loro, finalizzati all'elaborazione automatica delle informazioni.

Così come il termine **sistema operativo** indica in informatica l'insieme organizzato dei programmi di base che fanno funzionare l'hardware di un computer, rendendolo disponibile per le applicazioni dell'utente.

L'osservazione del funzionamento di un sistema, oppure l'analisi di un problema da risolvere, viene favorita dalla descrizione mediante un **modello**, cioè dalla rappresentazione semplificata della realtà osservata, che ne evidenzia gli aspetti fondamentali riducendoli in termini trattabili con il lavoro manuale, mentale o automatizzato.



Il tipo di modello che viene utilizzato è determinato da quello che si vuole fare: si va da un livello di astrazione bassa rispetto alla realtà osservata, come nel caso di modelli **iconici**, fino a un livello di astrazione alta come nel caso dei modelli **matematici**.

Un esempio di modello iconico è rappresentato dalla piantina di una città; il piano dei conti della contabilità di un'azienda è un esempio di modello matematico.

Nel lavoro comune di tutte le discipline e di tutte le attività, i modelli vengono sempre utilizzati, perché consentono di concentrare l'attenzione sugli aspetti fondamentali di un sistema o di un problema, trascurando quelli che hanno un'influenza minore sul funzionamento del sistema o sulla possibilità di risoluzione di un problema.

Processo e processore

Quando si deve eseguire un lavoro occorre conoscere l'elenco delle cose da fare e i dati su cui operare.

Con il termine **processo** si indica l'insieme formato dalle operazioni da svolgere in sequenza e dai dati che vengono elaborati durante queste operazioni per svolgere il compito assegnato.

Anche nel linguaggio comune si usa il termine processo per indicare un'attività, con l'idea di mettere in risalto lo svolgersi, il progredire, il compiersi nel tempo: per esempio processo penale, processo produttivo, processo infiammatorio.

Le caratteristiche fondamentali del concetto di processo sono:

- l'evoluzione del processo nel tempo
- la sua sequenzialità.

Il **processore** è l'esecutore del processo, cioè l'ente che causa l'evoluzione del processo nel tempo.

Lo studente, quando svolge una prova scritta in classe, svolge le funzioni di processore rispetto al processo costituito dalla risoluzione degli esercizi assegnati.

Il treno è il processore nel trasporto delle persone da una stazione all'altra, il trasporto è il processo.

Il robot è il processore nel processo di produzione industriale automatizzato.

Il computer, come vedremo in seguito, è un particolare tipo di processore che esegue una sequenza di operazioni per elaborare in modo automatico i dati ad esso forniti.

AUTOVERIFICA

Domande da 1 a 6 pag. 40

4 I sistemi di numerazione

Le quantità numeriche vengono espresse generalmente utilizzando il **sistema di numerazione** decimale, che si chiama così perché utilizza 10 cifre (0,1,2,3,4,5,6,7,8,9) per rappresentare i numeri.

Si dice anche che i numeri sono rappresentati in base 10.

Si noti l'uso del termine *sistema* come insieme di oggetti e di regole organizzati per svolgere la funzione costituita dalla rappresentazione delle grandezze numeriche: anche in questo caso il sistema risulta definito dalle parti (le cifre), dalle relazioni (le regole di calcolo) e dalle finalità (rappresentare le grandezze).

Le cifre possiedono un valore posizionale, cioè un valore a seconda della posizione occupata nella scrittura del numero.

Per esempio, nel numero 724 la cifra 7 vale 700, 7 centinaia (potenza di 10 con esponente 2), la cifra 2 vale 20, 2 decine (potenza di 10 con esponente 1) e la cifra 4 vale 4 unità (potenza di 10 con esponente 0).

10^2	10^1	10^0
7	2	4

In informatica vengono usati altri sistemi di numerazione: il sistema binario (numeri in base 2), il sistema ottale (numeri in base 8) e il sistema esadecimale (numeri in base 16).

In particolare il sistema binario si lega strettamente alla tecnologia del funzionamento dei computer in quanto, come vedremo più in dettaglio nel seguito, l'elaboratore utilizza dispositivi elementari che possono assumere soltanto due stati, ai quali vengono associate le cifre 0 e 1, che sono appunto le cifre del sistema binario.

Nel linguaggio informatico le cifre binarie vengono indicate con il termine **bit**, dalla contrazione delle due parole inglesi *Binary digit* (cifra binaria).

Poiché i dati all'interno del computer sono rappresentati utilizzando le cifre binarie, il bit diventa l'unità elementare per la misura dell'informazione o la più piccola unità di informazione.

5 Il sistema binario

In modo analogo al sistema decimale, le cifre del **sistema binario** 0 e 1 assumono un valore posizionale nella scrittura del numero binario con riferimento alle potenze di 2, anziché alle potenze di 10.

Le potenze di 2 con esponente da 0 a 10 sono:

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1024	512	256	128	64	32	16	8	4	2	1

Quindi il numero binario 1001 corrisponde al numero decimale 9, perché

2^3		2^2		2^1		2^0	
1		0		0		1	
8	+	0	+	0	+	1	= 9

La trasformazione di un numero binario in un numero decimale avviene secondo la seguente regola: si moltiplica ciascuna cifra binaria per la corrispondente potenza di 2 e si sommano i prodotti ottenuti.

Per esempio, dato il numero binario 1 1 0 1 0:

$$0 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 = 26$$

La trasformazione inversa, da un numero decimale a un numero binario, viene effettuata secondo la seguente regola:

Si divide il numero dato per 2 e si scrive il resto (che può essere 0 o 1); il quoziente ottenuto viene a sua volta diviso per 2 ottenendo un nuovo resto; si prosegue fino a quando si ottiene come quoziente il numero 0.

La sequenza dei resti, letta dall'ultimo al primo, fornisce il numero binario corrispondente al numero decimale dato.

Per esempio, la trasformazione del numero decimale 35 si ottiene in questo modo:

QUOZIENTI	RESTI
35	
17	1
8	1
4	0
2	0
1	0
0	1



$$35_{10} = 1\ 0\ 0\ 0\ 1\ 1_2$$



LE OPERAZIONI NEL SISTEMA BINARIO

L'*addizione* di numeri binari è analoga a quella del sistema decimale: le cifre vengono sommate in colonna a partire da destra, ricordando gli eventuali riporti.

A questo proposito è opportuno notare che nell'addizione binaria vale:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10 \text{ (cioè 0 con riporto di 1).}$$

Esempi:

(sopra i numeri del primo addendo vengono ricordati i riporti)

$$\begin{array}{r}
 1 \\
 101001+ \\
 11011 \\
 \hline
 1000100
 \end{array}$$

$$\begin{array}{r}
 1 \\
 11011+ \\
 111 \\
 \hline
 100010
 \end{array}$$

La *moltiplicazione* in colonna viene effettuata come con i numeri decimali, ricordando le regole dell'addizione binaria quando si devono sommare i risultati parziali.

Esempio:

$$\begin{array}{r}
 1011 \times \\
 11 \\
 \hline
 1011 \\
 1011 \\
 \hline
 100001
 \end{array}$$

La *sottrazione* in colonna viene calcolata ricordando che:

$$\begin{array}{l}
 0 - 0 = 0 \\
 0 - 1 = 1 \quad (\text{con prestito di 1 dalla cifra a sinistra}) \\
 1 - 0 = 1 \\
 1 - 1 = 0
 \end{array}$$

Esempio:

(il prestito corrisponde alla diminuzione di 1 nella cifra più a sinistra del minuendo e viene indicato con la cifra 0 in alto)

$$\begin{array}{r}
 000 \\
 10110 - \\
 1011 \\
 \hline
 01011
 \end{array}$$

Per comprendere più facilmente la *divisione* di due numeri binari, facciamo riferimento a un esempio.

Esempio:

$$1111011 \text{ diviso } 101$$

- dapprima si separano le prime tre cifre del dividendo e si dividono per il divisore ottenendo come quoziente 1

$$1111011 : \begin{array}{r} 101 \\ 1 \end{array}$$

- si moltiplica il quoziente e il divisore scrivendo il risultato incolonnato sotto il dividendo e si effettua la sottrazione (ricordare le regole della sottrazione binaria)

$$\begin{array}{r}
 1111011 : \begin{array}{r} 101 \\ 1 \end{array} \\
 \underline{101} \\
 10
 \end{array}$$

- si abbassa la cifra a destra e si ripete la *divisione*, come prima

$$\begin{array}{r}
 1111011 : \begin{array}{r} 101 \\ 11 \end{array} \\
 \underline{101} \\
 101 \\
 \underline{101} \\
 0
 \end{array}$$

- si abbassa un'altra cifra a destra, e così via fino a quando tutte le cifre sono state abbassate, ottenendo alla fine il quoziente e il resto della divisione

$$\begin{array}{r}
 1111011 : 101 \\
 \underline{101} \\
 101 \\
 \underline{101} \\
 00 \\
 0 \\
 0 \\
 01 \\
 0 \\
 11 \\
 0 \\
 11
 \end{array}$$

Per esercizio, si effettui la divisione di due numeri binari e poi si controlli la correttezza del risultato trasformando dividendo, divisore, quoziente e resto in numeri decimali e confrontando il quoziente e resto, ottenuti con la divisione binaria, con il quoziente e resto ottenuti con la divisione decimale.



IL SISTEMA OTTALE

Il sistema ottale si chiama così perché utilizza 8 cifre nella rappresentazione dei numeri: 0,1,2,3,4,5,6,7.

Il valore posizionale delle cifre è fornito dalle potenze di 8:

8^4	8^3	8^2	8^1	8^0
4096	512	64	8	1

Le regole per la conversione di un numero ottale in decimale e da decimale ad ottale sono analoghe a quelle già descritte per il sistema binario.

Vediamo due esempi:

- trasformazione del numero ottale 325 in numero decimale:

$$\begin{array}{ccc}
 8^2 & 8^1 & 8^0 \\
 3 & 2 & 5 \\
 5 \times 8^0 + 2 \times 8^1 + 3 \times 8^2 = 5 + 16 + 192 = 213
 \end{array}$$

- trasformazione del numero decimale 1602 in numero ottale:

QUOZIENTI	RESTI
1602	
200	2
25	0
3	1
0	3



$$1602_{10} = 3102_8$$

6 Il sistema esadecimale

Il **sistema esadecimale** utilizza 16 cifre:

0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Quindi la lettera A rappresenta il numero decimale 10 in esadecimale, B il numero 11, C il numero 12, D il numero 13, E il numero 14, F il numero 15. Il valore delle cifre dipende dalla posizione nella scrittura del numero secondo le potenze di 16:

16^4	16^3	16^2	16^1	16^0
65536	4096	256	16	1

Mostriamo la trasformazione di un numero da esadecimale a decimale e viceversa con due esempi:

- trasformazione del numero esadecimale 3AF2 in numero decimale:

16^3	16^2	16^1	16^0
3	A	F	2

$$2 \times 16^0 + 15 \times 16^1 + 10 \times 16^2 + 3 \times 16^3 = 15090$$

- trasformazione del numero decimale 16034 in numero esadecimale:

QUOZIENTI	RESTI
16034	
1002	2
62	A
3	E
0	3



$$16034_{10} = 3 \text{ E A } 2_{16}$$

7 Regole di conversione binario/esadecimale

Nelle operazioni di conversione dei numeri dal sistema in base 2 al sistema in base 16, è conveniente ricordare la seguente tabella:

Binario	Esadecimale
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

• **Da binario a esadecimale**

Si raggruppano le cifre del numero binario a gruppi di *quattro* a partire da destra, e si trasformano le cifre di ciascun gruppo nel corrispondente numero esadecimale, secondo la tabella di conversione.

Per esempio, il numero binario

1011110111

si può scrivere come

10	1111	0111
2	F	7

quindi: $1011110111_2 = 2F7_{16}$

• **Da esadecimale a binario**

Si fa corrispondere a ciascuna delle cifre esadecimali che compongono il numero un gruppo di *quattro* bit secondo la tabella di conversione.

Per esempio, dato il numero esadecimale

C3B

si può scrivere

C	3	B
1100	0011	1011

quindi: $C3B_{16} = 110000111011_2$



REGOLE DI CONVERSIONE BINARIO/OTTALE

• **Da binario a ottale**

Si raggruppano le cifre del numero binario a gruppi di *tre* a partire da destra, e si trasformano le cifre di ciascun gruppo nel corrispondente numero ottale.

Per esempio, il numero binario

1011110111

si può scrivere come

1	011	110	111
1	3	6	7

quindi: $1011110111_2 = 1367_8$

• **Da ottale a binario**

Si fa corrispondere a ciascuna delle cifre ottali che compongono il numero un gruppo di *tre* bit.

Per esempio, dato il numero ottale

625

si può scrivere

6	2	5
110	010	101

quindi: $625_8 = 110010101_2$

AUTOVERIFICA

Domande da 7 a 9 pag. 40-41
Problemi da 1 a 22 pag. 42-43-44



MATERIALE ONLINE

1. Sistemi di numerazione con il foglio elettronico Excel

8 La codifica delle informazioni

Le cifre binarie 0 e 1 (bit) sono gli elementi fondamentali della scienza dei calcolatori per la rappresentazione delle informazioni all'interno dell'elaboratore.

Solitamente le cifre binarie all'interno di un calcolatore vengono trattate a gruppi o pacchetti contenenti un numero costante di bit: in particolare, per essere elaborate, le cifre binarie vengono raggruppate in sequenze o stringhe di 8 bit.

Una stringa di 8 bit prende il nome di **byte**.

Per il trattamento dei dati, gli elaboratori operano su sequenze composte da un numero fisso di byte. Tali stringhe di byte prendono il nome di **parole**.

Esistono elaboratori che operano con parole di

1 byte (8 bit)

2 byte (16 bit)

4 byte (32 bit)

8 byte (64 bit).

Il tempo per elaborare una parola va considerato come costante e quindi la lunghezza delle parole costituisce uno degli indici per determinare la potenza e la velocità di calcolo degli elaboratori.

Dati N simboli da disporre in un numero fisso P di posti, tutte le possibili combinazioni sono in numero pari a N^P .

Utilizzando i due simboli delle cifre binarie, le combinazioni che si ottengono con 4 posti sono:

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

16 in totale (con 2 simboli in 4 posti si ottengono 2^4 combinazioni diverse).

Utilizzando i 16 simboli del sistema esadecimale da disporre in 2 posti, si ottengono $16^2 = 256$ combinazioni.

9 Rappresentazione dei numeri interi

Abbiamo visto che tutti i numeri possono essere espressi in forma binaria e che gli elaboratori utilizzano parole di lunghezza fissa.

Supponiamo di utilizzare parole di 2 byte (16 bit) e prendiamo in considerazione soltanto numeri interi con segno. Dei 16 bit a disposizione il primo bit a sinistra viene dedicato alla rappresentazione del segno: 0 per il segno +, 1 per il segno –.

Per i numeri positivi si hanno, quindi, a disposizione 15 bit, cioè 2 simboli (le cifre binarie), da disporre in 15 posti: possiamo quindi rappresentare 2^{15} numeri diversi, vale a dire dal numero 0 al numero $2^{15} - 1 = 32767$.

Per i numeri negativi, invece, si ricorre al criterio del **complemento a 2**: dato un numero espresso in forma binaria, per ottenere il complemento a 2 del numero dato, ad ogni cifra si sostituisce 0 dove c'è 1, 1 dove c'è 0 e alla fine si somma 1.

Esempi:

dato il numero:		0	1	0	1	0	0	1	1
si scambiano le cifre:		1	0	1	0	1	1	0	0
e si somma 1	+								1
complemento a 2:		1	0	1	0	1	1	0	1
dato il numero:		0	1	0	1	1	1	1	0
si scambiano le cifre:		1	0	1	0	0	0	0	1
e si somma 1	+								1
complemento a 2:		1	0	1	0	0	0	1	0
dato il numero:		0	1	0	1	0	0	0	0
si scambiano le cifre:		1	0	1	0	1	1	1	1
e si somma 1	+								1
complemento a 2:		1	0	1	1	0	0	0	0

Sussiste la seguente regola: fissato il numero di bit su cui operare, e convenendo di ignorare i riporti che creino nuove cifre a sinistra eccedenti il numero di bit fissato, il complemento a 2 di un numero dato risulta l'opposto del numero.

Per esempio, fissato a 8 il numero di bit su cui operare:

dato il numero	5	in binario:	0 0 0 0 1 0 1
l'opposto è	-5	il complemento è:	1 1 1 1 1 0 1 1
la somma è	0		1 0 0 0 0 0 0 0
			└──────────┘
			8 bit utili

Il riporto a sinistra degli otto bit utilizzati viene ignorato.

Per calcolare l'opposto di un numero binario si utilizza il complemento a 2 del numero.

Utilizzando quindi parole di 2 byte e volendo rappresentare i numeri interi positivi, poiché il primo bit di sinistra è dedicato all'indicazione del segno, si ottiene che i numeri interi positivi rappresentabili in 2 byte vanno da 0 a $2^{15} - 1$.

Per i numeri negativi, invece, si utilizza il criterio del complemento a 2 e quindi i numeri negativi rappresentabili in 2 byte vanno da -1 a -2^{15} .

Infatti, utilizzando 16 bit:

il numero 1 viene rappresentato da 0000000000000001
 il numero -1 dal complemento di 1, cioè 1111111111111111
 il numero $+2^{15}$ in binario risulta uguale a 1000000000000000

e non può essere rappresentato in una parola di 2 byte perché il primo bit a sinistra indica il segno -.

Invece il complemento a 2 del numero precedente, ovvero l'opposto di 2^{15} , risulta ancora $1000000000000000 = -2^{15}$. Riassumendo, i numeri interi rappresentabili in una parola di due byte sono (in ordine decrescente)

0111111111111111	= $32767 = 2^{15} - 1$
0111111111111110	= 32766
0111111111111101	= 32765
.....	
0000000000000011	= 3
0000000000000010	= 2
0000000000000001	= 1
0000000000000000	= 0
1111111111111111	= -1
1111111111111110	= -2
1111111111111101	= -3
.....	
1000000000000010	= -32766
1000000000000001	= -32767
1000000000000000	= $-32768 = -2^{15}$

Il primo e l'ultimo numero della tabella precedente sono
 $2^{15} - 1$; -2^{15} .

L'esponente di 2 è 15 perché si considerano parole di 16 bit di cui il primo è utilizzato per il segno.

Nel caso si utilizzino parole di 4 byte, i bit a disposizione risultano 32 di cui uno è dedicato al segno e quindi:

$$\text{massimo} = 2^{31} - 1 \text{ e minimo} = -2^{31}$$

10 Rappresentazione dei numeri reali

La rappresentazione dei numeri **reali** (non interi) può essere fatta usando

- la virgola fissa (**fixed point**):

1.5 0.00123 12.564

oppure

- la virgola mobile (**floating point**):

3E-4 12E+18 1.47E-3

(Si noti che per scrivere i numeri decimali viene usato sempre il punto al posto della virgola, secondo la notazione anglosassone).

Il secondo modo di rappresentare i numeri si chiama anche **notazione scientifica o rappresentazione esponenziale**.

La lettera E sta al posto di **10 elevato a** ed è seguita dall'esponente a cui elevare la base 10: la potenza di 10 va moltiplicata per il numero (**mantissa**) che precede la lettera E.

Per esempio:

7.2342 E-5

significa

$$7.2342 \times 10^{-5} = 0.000072342$$

Questo tipo di rappresentazione viene usato solitamente in calcoli scientifici, quando si devono trattare numeri molto grandi o molto piccoli scrivendoli in una forma compatta e facilmente leggibile.

Per esempio, il numero

0.0000000000000003

può essere scritto, molto più semplicemente, nella forma

3 E-15

oppure il numero

1230000000000000

può essere scritto

1.23 E+14

Un numero in virgola mobile può essere scritto rappresentando la mantissa con un numero, in valore assoluto, sempre minore di 1.

Per esempio:

1.23 E+14

si può scrivere

0.123 E+15

o più semplicemente

.123 E+15

Questa rappresentazione, che si chiama notazione esponenziale **normalizzata**, mette in evidenza la mantissa (la parte prima di E) e l'esponente (la parte dopo di E) che prende il nome di **caratteristica**.

Il criterio per rappresentare un numero in forma normalizzata è il seguente: ogni numero può essere rappresentato come prodotto tra un numero decimale e una potenza di 10, in modo tale che la parte intera del numero decimale risulti sempre uguale a 0 e la prima cifra decimale risulti sempre diversa da 0; per ottenere questo risultato l'esponente di 10 è calcolato secondo la regola: se il numero dato è in valore assoluto > 1 , l'esponente è positivo e uguale al numero delle cifre intere; se il numero dato è in valore assoluto < 1 , l'esponente è negativo e uguale al numero degli zeri dopo la virgola.

Per brevità illustriamo ora solo il metodo di rappresentazione dei numeri reali in virgola mobile. Supponendo di avere a disposizione parole di 4 byte la rappresentazione interna di un numero reale utilizza i 32 bit a disposizione in questo modo (contando i bit da sinistra):

- bit 1: 0 se il numero è positivo, 1 altrimenti
- bit da 2 a 8: valore binario della caratteristica aumentato di 64 (tutti i numeri > 64 indicano caratteristica > 0 , i numeri < 64 caratteristica negativa)
- bit da 9 a 32: valore binario della mantissa; per la mantissa vengono quindi utilizzati 24 bit: il massimo numero rappresentabile è 16.777.215; nel caso in cui la mantissa oltrepassi questo valore vengono perse le ultime cifre decimali a destra, cioè le meno significative, e la rappresentazione interna all'elaboratore subisce un'approssimazione.

Quindi la rappresentazione interna dei numeri nella memoria dell'elaboratore subisce una limitazione dovuta alle dimensioni fisiche della cella di memoria, che si traduce in un limite per il numero di cifre significative associate al numero.

Tutto questo viene descritto con il termine **precisione** della rappresentazione interna dei numeri: in genere si chiama **precisione semplice** (o **precisione singola**) quando i numeri reali sono rappresentati con 4 byte, e **precisione doppia** nel caso di rappresentazione con 8 byte. Occorre osservare che quanto esposto per la rappresentazione dei numeri reali ha un valore puramente indicativo: il criterio descritto presenta alcune varianti nei diversi sistemi di elaborazione in commercio.

11 Rappresentazione delle informazioni alfanumeriche

Con 1 bit si hanno due combinazioni possibili: 0, 1. Con 1 byte abbiamo 256 combinazioni possibili: da 00000000 a 11111111.

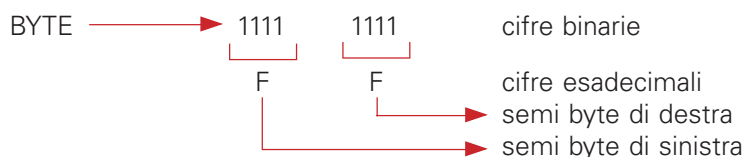
Infatti 11111111 in base 2 = $128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$.

Il massimo valore che può esprimere un byte è 255 decimale.

Il massimo numero di combinazioni che può contenere un byte è 256 (compresa la combinazione 00000000).

Un byte nel quale tutti i bit sono uguali a 1 (11111111) è equivalente a FF esadecimale.

Per rappresentare un simbolo del sistema esadecimale occorrono 4 bit (4 cifre binarie).



La rappresentazione di una cifra esadecimale richiede un semibyte.

Tutte le informazioni non numeriche (alfabetiche o alfanumeriche) sono esprimibili mediante una combinazione di lettere, cifre o caratteri speciali: affinché un elaboratore riesca a riconoscere e a trattare tali informazioni deve essere stabilita una corrispondenza che ad ogni carattere utilizzato per rappresentare le informazioni associ una particolare configurazione degli 8 bit di un byte.

La rappresentazione dei dati all'interno di un elaboratore è quindi realizzata attraverso l'associazione di una combinazione binaria del byte ad un determinato simbolo (lettera, cifra o carattere speciale): questa associazione è chiamata **codifica**.

La codifica di base per i caratteri di un testo si chiama **ASCII** (*American Standard Code for Information Interchange*, in italiano *codice americano standard per lo scambio di informazioni*) che utilizza 7 bit per codificare un singolo carattere; per esempio a 0110000 corrisponde il carattere "0" (cifra zero) o a "1110001" corrisponde la lettera "q" (minuscola).

I 7 bit del codice consentono di rappresentare 128 (2^7) caratteri diversi tra loro e quindi non sono sufficienti per codificare tutte le variazioni e tutti i simboli utilizzati nelle varie lingue. Questo sistema è stato esteso a 8 bit, raddoppiando il numero di caratteri disponibili ($256 = 2^8$), con la definizione del codice chiamato **ASCII esteso**.

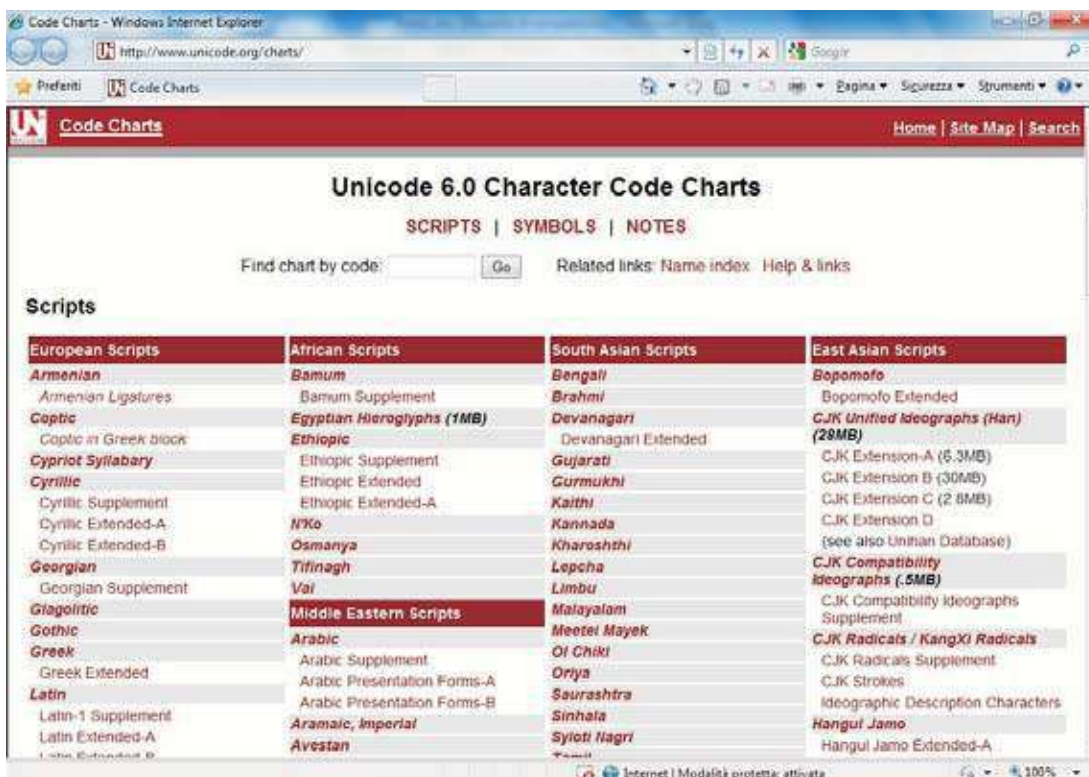
Nel codice ASCII standard ci sono 32 caratteri (i primi 31 e l'ultimo) **non stampabili**, ma che hanno un significato particolare, per esempio *carattere vuoto*, *fine del testo*, *segnale acustico*, *Cancel*, *Esc*, *Delete*.

Successivamente si trovano i caratteri stampabili:

- da **32** (010 0000) a **47**, segni di punteggiatura: punto esclamativo, punto, virgola
- da **48** (011 0000) a **57**, cifre da 0 a 9
- da **58** (010 0000) a **64**, altri segni di punteggiatura: due punti, maggiore, uguale, minore
- da **65** (100 0001) a **90**, lettere maiuscole da "A" a "Z"
- da **91** (101 1011) a **96**, altri segni di punteggiatura: per esempio, apostrofo, parentesi quadre
- da **97** (110 0001) a **122**, lettere minuscole da "a" a "z"
- da **123** (111 1011) a **126**, altri segni di punteggiatura: per esempio, parentesi graffe, tilde.

Nel 1991 è stata sviluppata una nuova codifica, chiamata **Unicode**, che si è diffusa rapidamente ed è in continua evoluzione: è diventata lo standard di fatto per la rappresentazione delle informazioni nei documenti elettronici, in particolare nelle pagine Web, utilizzando i simboli delle numerose lingue esistenti nel mondo.

Le tabelle dei codici *Unicode* sono disponibili sul sito <http://www.unicode.org/charts>.



I primi caratteri di *Unicode* sono esattamente gli stessi della codifica ASCII, in modo da mantenere la compatibilità con il sistema preesistente. All'inizio la codifica utilizzava 2 byte (16 bit, con la possibilità di codificare 65.536 caratteri), ma poi è stata estesa a 32 bit, permettendo la rappresentazione di più di un milione di caratteri differenti. Ovviamente non sono stati codificati subito tutti i caratteri: essi vengono via via assegnati nelle nuove versioni.

L'obiettivo generale di *Unicode* è di creare una codifica che comprenda tutti i caratteri, con tutte le variazioni possibili, di tutte le lingue esistenti, oltre ai simboli utilizzati in matematica e nelle scienze.

Per semplificare le operazioni sono state poi create versioni ridotte del codice che permettono di scrivere i caratteri di uso più frequente in modo più breve: **UTF-8** (a 8 bit), **UTF-16** (a 16 bit) e **UTF-32** (a 32 bit).

Tutte queste codifiche sono compatibili tra loro almeno per quanto riguarda i caratteri standard (lettere maiuscole, minuscole e numeri), mentre può causare problemi con altri simboli (per esempio le lettere accentate o i simboli di frazione).

Riassumendo, ogni carattere viene convertito in bit per essere trasmesso. Più bit vengono utilizzati, più caratteri differenti possono essere utilizzati e più lunga sarà la sequenza di bit da trasmettere.

Per esempio la stringa di testo "Ciao, mondo!" contenente 12 caratteri (occorre tenere conto di tutti i simboli, compresi lo spazio e il punto esclamativo) occuperebbe 84 bit (12 x 7) se codificata in *ASCII standard*, mentre ne occuperebbe 384 (12 x 32) in *UTF-32*.

La figura seguente mostra la prima parte del documento *Unicode* per la codifica dei caratteri delle lingue arabe (codice *Arabic*): i caratteri corrispondono ai numeri compresi nell'intervallo da 0600 a 06FF in esadecimale.

	Arabic															
	0600	0601	0602	0603	0604	0605	0606	0607	0608	0609	060A	060B	060C	060D	060E	060F
0																
1																
2																
3																
4																
5																
6																

Per esempio il carattere evidenziato con un riquadro rosso corrisponde al codice 0683 in esadecimale (colonna 068, riga 3 della tabella).

Volendo inserire per esempio il carattere in un documento Word, si deve prima convertire in decimale (si può usare la *Calcolatrice* di Windows in *Accessori*):

$$0683_{16} = 1667_{10}$$

Poi, in Word, occorre tenere premuto il tasto *Alt* e premere in successione i tasti 1667 sul tastierino numerico (a destra nella tastiera).

In alternativa, si possono scrivere direttamente nel testo le cifre esadecimali del codice Unicode e premere subito dopo la combinazione di tasti **ALT+X**. Per esempio, scrivendo le cifre 20AC seguite da ALT+X si ottiene il simbolo dell'euro €.



AUTOVERIFICA

Domande da 10 a 11 pag. 41
Problemi da 23 a 25 pag. 44



MATERIALI ONLINE

2. Che cos'è Unicode
3. Esempi di Codici Unicode

12 Proposizioni e connettivi logici

Nel lavoro di programmazione capita spesso di dover ricorrere ai principi della logica degli enunciati e occorre conoscere i concetti di base dell'algebra delle proposizioni. L'algebra delle proposizioni è detta anche **algebra booleana** dal nome del matematico inglese *George Boole* (1815-1864).

Si dice **enunciato** una proposizione che può essere soltanto vera o falsa.

La verità o la falsità di un enunciato sono dette **valori di verità**; un enunciato può essere vero o falso, ma non entrambe le cose.

Esempi

- "oggi piove"; "quel fiore è rosso" sono enunciati.
- "speriamo che non piovano"; "dove siete stati?" non sono enunciati in quanto non sono né veri né falsi.

Alcuni enunciati possono essere **composti**, vale a dire sono formati da sottoenunciati collegati tra loro da **connettivi**.

Esempio

- "egli è intelligente oppure studia tutta la notte" è un enunciato composto dai sottoenunciati "egli è intelligente" e "egli studia tutta la notte" collegati tra loro dal connettivo "oppure".

La proprietà fondamentale di un enunciato composto è che il suo valore di verità è interamente definito dai valori di verità dei suoi sottoenunciati e dal connettivo che li unisce.

Scopo di questo paragrafo è presentare i principali connettivi logici e di illustrarne le proprietà. Per indicare gli enunciati si usano di solito le lettere **p, q, r, ...**.

Congiunzione (AND)

Due enunciati possono essere collegati dal connettivo "**e**" (in inglese e in informatica, **and**), in modo da formare un enunciato composto, detto **congiunzione** degli enunciati di partenza. In simboli **p and q** denota la congiunzione degli enunciati e viene letto "p e q".

Il valore di verità di **p and q** è dato dalla seguente tabella che costituisce la definizione di congiunzione:

p	q	p and q
V	V	V
V	F	F
F	V	F
F	F	F

Dove "V" (vero) e "F" (falso) sono valori di verità.

La prima riga indica in modo sintetico che se *p* è vera e *q* è vera, allora **p and q** è vera. Le altre righe hanno significato analogo. Si osservi che **p and q** è vera solo nel caso in cui sono veri entrambi i sottoenunciati.

Esempio:

- Londra è in Inghilterra e $2+2=4$
- Londra è in Inghilterra e $2+2=5$
- Londra è in Spagna e $2+2=4$
- Londra è in Spagna e $2+2=5$

Solo il primo enunciato è vero. Gli altri sono falsi perché almeno uno dei sottoenunciati è falso.

Disgiunzione (OR)

Due enunciati possono essere collegati dal connettivo “o” (in inglese e in informatica, **or**), in modo da formare un enunciato composto, detto **disgiunzione** degli enunciati di partenza. In simboli **p or q** denota la disgiunzione degli enunciati e viene letto “p o q”. Il valore di verità di **p or q** è dato dalla seguente tabella che costituisce la definizione della disgiunzione:

p	q	p or q
V	V	V
V	F	V
F	V	V
F	F	F

Si osservi che **p or q** è falsa solo nel caso in cui sono falsi entrambi i sottoenunciati.

Esempio:

- Londra è in Inghilterra o $2+2=4$
- Londra è in Inghilterra o $2+2=5$
- Londra è in Spagna o $2+2=4$
- Londra è in Spagna o $2+2=5$

Solo l'ultimo enunciato è falso. Gli altri sono veri perché almeno uno dei sottoenunciati è vero.

Disgiunzione esclusiva (XOR)

Due enunciati possono essere collegati dal connettivo “o esclusivo” (in inglese e in informatica, **xor**), in modo da formare un enunciato composto, detto **disgiunzione esclusiva** degli enunciati di partenza. In simboli **p xor q** denota la disgiunzione esclusiva degli enunciati e viene letto “p xor q”.

Il valore di verità di **p xor q** è dato dalla tabella seguente che costituisce la definizione della disgiunzione:

p	q	p xor q
V	V	F
V	F	V
F	V	V
F	F	F

Si osservi che **p xor q** è vera solo nel caso in cui i due enunciati **p, q** hanno valori di verità diversi, mentre **p xor q** risulta falsa se i valori di verità di **p, q** sono uguali.

Esempio:

- Londra è in Inghilterra o $2+2=4$
- Londra è in Inghilterra o $2+2=5$
- Londra è in Spagna o $2+2=4$
- Londra è in Spagna o $2+2=5$

Solo il secondo e terzo enunciato sono veri. Gli altri sono falsi perché i due sottoenunciati hanno valore di verità uguale.

Osservazione

Nella lingua italiana la particella “o” può assumere due significati diversi. In alcuni casi viene utilizzata come “p o q o entrambi” (disgiunzione *or*), in altri casi viene intesa con il significato di “p o q ma non entrambi” (disgiunzione esclusiva *xor*). Per esempio, se si dice: “ha vinto alla lotteria o ha avuto una eredità”, si intende che può essere vero “ha vinto alla lotteria” (p) o può esser vero “ha avuto una eredità” (q) o possono essere veri entrambi. Se, invece, si dice: “va a Roma o va a Milano”, si esclude che possano essere veri entrambi i sottoenunciati.

Negazione (NOT)

Dato un enunciato **p**, è possibile formare un altro enunciato che si indica con **not p** e che è detto **negazione** di **p**. Nel linguaggio corrente la negazione di **p** si ottiene antepoendo a **p** “non è vero che...” oppure inserendo in **p** la parola “non”.

Il valore di verità di **not p** è dato dalla tabella:

p	not p
V	F
F	V

Esempio:

- Londra è in Inghilterra
- Londra non è in Inghilterra
- Non è vero che Londra è in Inghilterra

Il secondo e il terzo enunciato sono entrambi la negazione del primo.

È opportuno osservare che esistono diverse notazioni per indicare i connettivi “e”, “o” e “non”:

“e”	= p et q, p & q, p×q, p∧q, p and q
“o”	= p vel q, p+q, p∨q, p or q
“o esclusivo”	= p aut q, p xor q
“non”	= non p, p', $\bar{p}$, ¬p, not p

I simboli usati più frequentemente in informatica sono **and**, **or**, **xor**, **not**.



MATERIALI ONLINE

4. Operatori booleani in Excel

5. Operatore XOR in Excel

13 Tavole di verità

Combinando in vario modo gli enunciati semplici del tipo **p, q, r, ...** e i connettivi **and**, **or** e **not** si possono ottenere enunciati composti sempre più complessi.

Quando gli enunciati **p, q, r, ...** che formano un enunciato composto **P(p, q, r, ...)** sono variabili, l'enunciato che si ottiene viene chiamato **forma enunciativa**.

Il valore di verità di una forma enunciativa è noto quando lo sono i valori di verità delle sue variabili.

Un modo semplice e schematico per mostrare questo rapporto è quello di costruire una **tavola di verità**.

Si consideri, per esempio, la forma enunciativa **not(p and not q)**. La tavola di verità si costruisce nel seguente modo:

p	q	not q	p and not q	not(p and not q)
V	V	F	F	V
V	F	V	V	F
F	V	F	F	V
F	F	V	F	V

Le prime due colonne indicano tutte le combinazioni dei valori che possono essere assunti dalle variabili **p** e **q**: ci sono tante righe quante sono queste combinazioni (con due variabili sono necessarie quattro righe, con tre variabili otto righe, ... con n variabili 2^n righe). Segue una colonna per ogni operazione indicata nella forma enunciativa. La sequenza con cui costruire le diverse colonne è definita dalla seguente regola: si procede dalle parentesi più interne verso l'esterno ed entro la medesima parentesi l'ordine di esecuzione delle operazioni è: **not, and, or**. Alla fine si ottiene il valore di verità della forma enunciativa.

Esempio

Si trovi la tavola di verità della forma enunciativa **p or (q and r)**

p	q	r	q and r	p or (q and r)
V	V	V	V	V
V	V	F	F	V
V	F	V	F	V
V	F	F	F	V
F	V	V	V	V
F	V	F	F	F
F	F	V	F	F
F	F	F	F	F



EQUIVALENZA LOGICA E PROPRIETÀ DELL'ALGEBRA BOOLEANA

Si dice che due forme enunciative sono equivalenti quando hanno la medesima tavola di verità.

Esempio

Si dimostri che le forme enunciative **(p and q) or not p** e **not p or q** sono equivalenti.

Tavola di verità di **(p and q) or not p**:

p	q	p and q	not p	(p and q) or not p
V	V	V	F	V
V	F	F	F	F
F	V	F	V	V
F	F	F	V	V

p	q	not p	not p or q
V	V	F	V
V	F	F	F
F	V	V	V
F	F	V	V

Attraverso le equivalenze logiche si possono esprimere le **leggi** (o *proprietà*) dell'algebra booleana. Nella tabella seguente vengono riportate alcune tra le più significative:

Leggi di De Morgan

not (p or q) $\equiv$ not p and not q **not (p and q) $\equiv$ not p or not q**

"Se non piove **e** non tira vento, non esco con l'impermeabile".

14 Le funzioni

Dati due insiemi A e B , in matematica si definisce **relazione binaria** una legge che associa alcuni elementi di B ad alcuni elementi di A .

La **funzione** è un particolare tipo di relazione binaria da A in B tale che, per ogni elemento a appartenente all'insieme A , esiste un unico elemento b appartenente all'insieme B associato ad a dalla legge stabilita dalla funzione.

L'insieme A si chiama **dominio** della funzione, l'insieme B si chiama **codominio** della funzione. Nella rappresentazione delle funzioni si usano le seguenti notazioni:

$$f: A \rightarrow B$$

$$A \xrightarrow{f} B$$

$$b = f(a)$$

L'elemento a si chiama anche **variabile indipendente**, e l'elemento b **variabile dipendente**. Quindi ad ogni elemento dell'insieme A la funzione f associa uno e un solo elemento dell'insieme B : questo significa che ad un elemento b possono essere associati anche più di un elemento dell'insieme A , ma non è vero il viceversa.

Esempi di funzione sono:

- la funzione che associa a ogni automobile la Casa automobilistica che la produce;
- la funzione che associa a ogni città la regione a cui appartiene;
- la funzione che associa a ogni automobile il suo proprietario;
- la funzione che associa il codice fiscale al cittadino;
- la funzione che associa a una persona la sua data di nascita.

Spesso si usano funzioni che rappresentano l'evoluzione di un sistema o di un modello nel tempo: in questi casi la variabile indipendente è costituita dal tempo che si indica comunemente con t . Per esempio nella funzione

$$I = C i t$$

I rappresenta l'interesse, C il capitale, i il tasso e t il tempo: t è la variabile indipendente e I la variabile dipendente.

Potremmo anche scrivere in forma sintetica, per evidenziare la dipendenza dal tempo:

$$I = f(t)$$

In modo analogo nel moto rettilineo uniforme lo spazio percorso dipende dal tempo secondo la formula

$$s = vt$$

e, quindi, possiamo dire che lo spazio è legato al tempo da una funzione

$$s = f(t)$$

L'età di una persona è legata all'anno di nascita secondo la formula:

$$età = anno\ attuale - anno\ di\ nascita$$

Si può, quindi, scrivere anche

$$età = f(\text{anno nascita})$$

L'ammontare dell'IVA si ottiene calcolando

$$IVA = \text{imponibile} : 100 \times \text{aliquota}$$

Se si vuole mettere in evidenza la funzione matematica che associa ad ogni imponibile l'imposta relativa, si può usare la notazione

$$IVA = f(\text{imponibile})$$



AUTOVERIFICA

Domande da 12 a 16 pag. 41-42

Problemi da 26 a 35 pag. 44-45

DOMANDE

Termini di uso comune

- 1 La sigla ICT è abbreviazione di:
 - a) Informatica, Computer, Telecomunicazioni
 - b) Internet, Computer, Telefonia
 - c) Information Communication Technology
- 2 Le nuove tecnologie di elaborazione e di comunicazione dei dati sono frutto dello sviluppo convergente di tre settori (scegli quali).
 - a) Telecomunicazioni
 - b) Informatica
 - c) Elettronica
 - d) Cibernetica
 - e) Matematica
 - f) Programmazione
- 3 L'insieme dei programmi di base che permettono il funzionamento del computer formano
 - a) L'hardware
 - b) L'organizzazione dei dati
 - c) Il sistema operativo
 - d) Il mainframe
- 4 Qual è il significato del termine *networking*?
 - a) Costruzione di autostrade informatiche.
 - b) Lavoro cooperativo in rete tra utenti diversi.
 - c) L'utilizzo della posta elettronica.
 - d) Produzione di software.
- 5 Quale delle seguenti definizioni corrisponde al termine *sistema*?
 - a) Una serie di combinazioni dei possibili esiti di alcune partite di calcio che si compila per tentare di vincere con la schedina.
 - b) Un insieme di enti che interagiscono tra loro per raggiungere un obiettivo comune.
 - c) Un modo di risolvere un problema.
 - d) Un insieme di problemi che devono essere risolti da più persone.
- 6 Qual è il significato del termine *modello*?
 - a) L'osservazione del funzionamento della realtà.
 - b) L'analisi di un problema da risolvere.
 - c) La rappresentazione semplificata della realtà osservata.
 - d) Un programma di gestione aziendale.

Sistemi di numerazione

- 7 Per trasformare un numero da binario a esadecimale si devono raggruppare le cifre binarie
 - a) Per 2
 - b) Per 4
 - c) Per 8 (16:2)
 - d) Per 16

- 8** Per trasformare un numero da binario a ottale si devono raggruppare le cifre binarie:
- a) Per 2
 - b) Per 3
 - c) Per 4
 - d) Per 8
- 9** Quale dei seguenti numeri non può essere un numero ottale?
- a) 800
 - b) 700
 - c) 567
 - d) 777

Codifica delle informazioni

- 10** Quali delle seguenti affermazioni sono vere (V) e quali false (F)?

- a) Cifra binaria e bit sono sinonimi
- b) Un bit è composto da otto byte
- c) Un byte è composto da otto bit
- d) Una parola è una stringa di byte



- 11** Indica la configurazione binaria e il corrispondente valore in base 10 del massimo numero positivo e del minimo numero negativo rappresentabili con il criterio del complemento a 2 in una parola di 8 bit.

Configurazione binaria	Base 10
.....	
.....	

Algebra booleana

- 12** Inserisci al posto dei puntini il connettivo logico corrispondente alla tabella di verità indicata.

p	q	p ... q
V	V	F
V	F	V
F	V	V
F	F	F

p	q	p ... q
V	V	V
V	F	F
F	V	F
F	F	F

p	q	p ... q
V	V	V
V	F	V
F	V	V
F	F	F

- 13** Completa in modo corretto le frasi seguenti:
- a) **p and q** è vera solo nel caso in cui
 - b) **p or q** è falsa solo nel caso in cui
 - c) **p xor q** è vera solo nel caso in cui
 - d) **p and q** è falsa nel caso in cui
- 14** Supponendo $p = \text{"Fa freddo"}$ e $q = \text{"Piove"}$, associa ad ogni enunciato della colonna di sinistra la corrispondente frase del linguaggio comune nella colonna di destra.
- | | |
|---------------------------|------------------------------|
| a) not p | 1) Fa freddo e piove |
| b) p and q | 2) Fa freddo o non piove |
| c) p or q | 3) Non è vero che non piove |
| d) p or not q | 4) Non fa freddo (fa caldo) |
| e) not p and not q | 5) Fa freddo o piove |
| f) not not q | 6) Non fa freddo e non piove |
- 15** Quale delle seguenti espressioni si ottiene applicando la proprietà distributiva all'espressione $a \text{ and } (b \text{ or } c)$?
- a) $a \text{ and } b \text{ or } c$
 - b) $a \text{ and } b \text{ and } c$
 - c) $(a \text{ and } b) \text{ or } (a \text{ and } c)$
 - d) $(a \text{ or } b) \text{ and } (a \text{ or } c)$
- 16** Scrivi, usando le leggi di De Morgan, la negazione per ciascuna delle seguenti espressioni:
- a) $(\text{prezzo} = 80) \text{ or } (\text{peso} > 50)$
 - b) $(\text{nascita} = \text{Milano}) \text{ and } (\text{not } (\text{età} > 18))$
 - c) $(\text{peso} = 80) \text{ or } (\text{età} > 50)$
 - d) $(\text{città} = \text{Roma}) \text{ and } (\text{not } (\text{lingua} = \text{italiana}))$
 - e) $(\text{distanza} < 50) \text{ or } (\text{popolazione} > 20000)$
 - f) $(\text{prezzo} < 500) \text{ or } (\text{quantità} < 100)$
 - g) $(\text{distanza} > 5) \text{ and } (\text{estensione} > 190.000)$
 - h) $(\text{superficie} = 300) \text{ or } (\text{perimetro} > 100)$
 - i) $(\text{nascita} = \text{Milano}) \text{ and } (\text{not } (\text{statura} > 1.70))$

PROBLEMI

Sistemi di numerazione

- 1** Trasformare i seguenti numeri binari nel sistema decimale:
- 11001100
 - 1010101
 - 10000111
 - 1001001
- 2** Trasformare i seguenti numeri decimali nel sistema binario:
- 123
 - 87
 - 96
 - 101

- 3** Fare la somma dei numeri binari
 $1011101 + 101101$
e controllare il risultato passando ai numeri decimali.
- 4** Calcolare la somma $11001 + 111$ e verificare il risultato passando ai decimali.
- 5** Fare la somma dei numeri binari $1011 + 101$ e controllare il risultato passando ai numeri decimali.
- 6** Calcolare la differenza $10100 - 1111$ e verificare il risultato passando ai decimali.
- 7** Fare la sottrazione dei numeri binari $101101 - 101$ e controllare il risultato passando ai numeri decimali.
- 8** Trasformare in binario il numero decimale 66 e il numero ottale 66.
- 9** Trasformare i numeri decimali in ottale:
103
200
222
- 10** Rappresentare il numero binario 100011101 in decimale e poi anche in ottale.
- 11** Trasformare i seguenti numeri esadecimali nel sistema decimale:
2CC3
3B4
22FA
12AE
- 12** Trasformare i seguenti numeri decimali in numeri esadecimali:
32068
800
542
902
- 13** Trasformare il numero esadecimale A57 in decimale e in binario.
- 14** Trasformare il numero esadecimale CB2 in decimale e in binario.
- 15** Trasformare il numero decimale 103 in binario e poi in esadecimale.
- 16** Trasformare il numero decimale 238 in esadecimale.
- 17** Calcolare il valore esadecimale del numero decimale 1760.
- 18** Trasformare i seguenti numeri decimali in esadecimale:
124
237
200

- 19** Trasformare i seguenti numeri binari in base 16 dividendo i bit quattro a quattro:
- 101100110001111
001111100111101
111111100100111
010011001110000
- 20** Trasformare i seguenti numeri esadecimali in base 2 utilizzando la regola che fa corrispondere a ogni cifra esadecimale quattro bit:
- 2A3E
78A1
8C1A
E3A4
- 21** Trasformare i seguenti numeri binari in base 8 (dividere i bit tre a tre):
- 110110110
001001001
101010101
010101010
- 22** Trasformare i seguenti numeri ottali in base 2 utilizzando la regola che fa corrispondere a ogni cifra ottale un terzetto di bit:
- 667
443
745
772
- 23** Scrivere le combinazioni da 00 a FF che si ottengono disponendo i simboli del sistema esadecimale in due posti.
- 24** Scrivere i seguenti numeri in forma esponenziale normalizzata:
- 9867.5543
1.3244
87562123
- 25** Scrivere il numero 6778899000000000000000 in forma esponenziale normalizzata.

Algebra booleana

- 26** Costruire la tabella di verità per le seguenti espressioni booleane:
- (a or b) and c
(a and b) or (not c)
(a or b) or (c and a)
- 27** Se $a = 5$, $b = 7$ l'uso del connettivo and: $(a < 3)$ and $(b > 5)$ produce una proposizione vera o falsa?
- 28** Se $a = 1$, $b = -6$
l'uso del connettivo or:
 $(a > 2)$ or $(b > 0)$
produce una proposizione vera o falsa?

- 29** Supponendo che a sia vera, b falsa e c vera, qual è il valore di verità delle seguenti espressioni:
- $(a \text{ xor } b) \text{ and } c$
 - $a \text{ or } (b \text{ xor } c)$
 - $a \text{ and } (b \text{ or not } c)$
- 30** Supponendo che a sia falsa, b vera e c falsa, qual è il valore di verità delle seguenti espressioni:
- $a \text{ xor } (\text{not } b \text{ and } c)$
 - $(\text{not } a \text{ xor } b) \text{ or not } c$
 - $a \text{ xor } (b \text{ and not } c).$
- 31** Controllare con una tabella di verità la proprietà distributiva della disgiunzione rispetto alla congiunzione.
- 32** Risolvere la negazione nella seconda parte dell'espressione $(a \text{ and } b) \text{ and } (\text{not } (b \text{ or } c))$, per determinare l'espressione equivalente, usando le leggi di De Morgan, e controllare lo sviluppo con una tabella di verità.
- 33** Calcolare la negazione di $(a \text{ and } b) \text{ and } (\text{not } (b \text{ or } c))$.
- 34** Riscrivere l'espressione booleana:
- $\text{not } a \text{ and not } b \text{ or } c$
 - usando soltanto gli operatori not e or.
- 35** Scrivere sotto forma di funzione la relazione tra peso e volume.



SCHEDA DI AUTOVALUTAZIONE

CONOSCENZE

- ☐ Terminologia corrente per l'elaborazione e la trasmissione dei dati
- ☐ Aspetti evolutivi dell'informatica
- ☐ Risorse della tecnologia e applicazioni
- ☐ Termini di uso comune utilizzati nel contesto informatico
- ☐ Sistema di numerazione in base 2
- ☐ Operazioni con i numeri binari
- ☐ Sistema di numerazione in base 16
- ☐ Sistema di numerazione in base 8
- ☐ Regole di conversione
- ☐ Bit e byte
- ☐ Rappresentazione dei dati, interi, reali e alfanumerici
- ☐ Algebra delle proposizioni
- ☐ Tavole di verità
- ☐ Equivalenza logica
- ☐ Proprietà dell'algebra booleana
- ☐ Funzioni

ABILITÀ

- ☐ Saper riconoscere gli aspetti evolutivi della tecnologia e delle applicazioni informatiche
- ☐ Saper spiegare il significato dei termini informazione, dato, elaborazione, trasmissione, codice, segnale, digitale, analogico, interfaccia, sistema, modello, processo, processore
- ☐ Convertire un numero decimale in binario e viceversa
- ☐ Calcolare la somma di due numeri binari
- ☐ Calcolare il prodotto, la differenza e il quoziente di due numeri binari
- ☐ Convertire un numero binario in esadecimale e viceversa
- ☐ Convertire un numero decimale in esadecimale e viceversa
- ☐ Convertire un numero binario in ottale e viceversa
- ☐ Rappresentare le tavole di verità degli operatori and, or, xor e not
- ☐ Usare le tavole di verità per calcolare il valore di un'espressione logica
- ☐ Usare le tavole di verità per verificare l'equivalenza di due espressioni logiche
- ☐ Applicare le proprietà dell'algebra booleana
- ☐ Rappresentare una legge o un fenomeno reale con una funzione



SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 662

2

parte seconda

L'ambiente di lavoro
con il computer

Il sistema di elaborazione

OBIETTIVI DI APPRENDIMENTO

In questo capitolo potrai cogliere l'aspetto sistemico delle macchine utilizzate in informatica, in modo da acquisire una visione d'insieme del sistema di elaborazione e della logica di funzionamento.

Conoscerai inoltre le caratteristiche delle risorse hardware e software e comprenderai le funzioni complessive del sistema operativo.

Concetto intuitivo di dispositivo automatico
Definizione formale di automa
Struttura generale del sistema di elaborazione
L'unità centrale di elaborazione
La memoria centrale
Le unità di input e di output
Le memorie di massa
Il collegamento delle periferiche
Il software
Il sistema operativo
La struttura modulare di un sistema operativo
La sicurezza
L'interprete dei comandi
I linguaggi di programmazione
Il software applicativo
Lo sviluppo del software

Prima di descrivere gli aspetti principali di un sistema di elaborazione reale, per comprenderne più facilmente i principi generali di funzionamento, è opportuno introdurre il concetto di **automa**, in modo intuitivo e in modo formalizzato.

1 Concetto intuitivo di dispositivo automatico

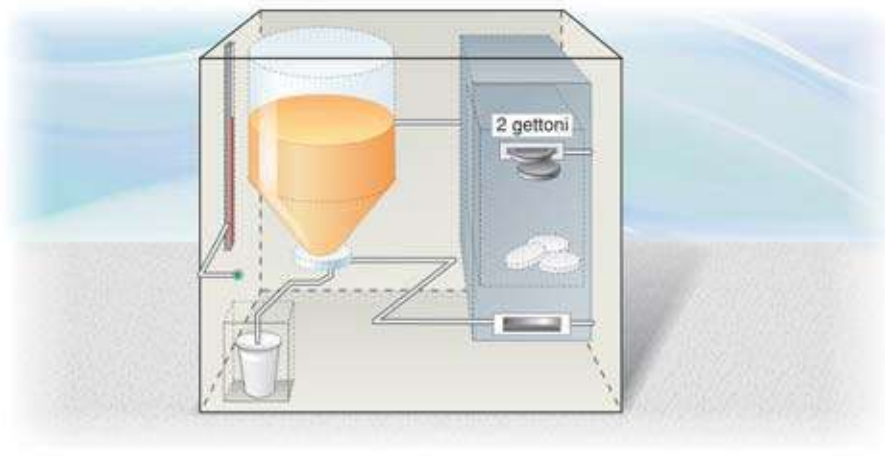
Consideriamo un esempio di macchina che esegue un lavoro predeterminato in modo automatico, senza cioè l'intervento manuale di una persona, secondo una sequenza di operazioni che devono essere eseguite in un ordine stabilito e che devono produrre i risultati che ci interessano. Nel trattare questo esempio non ci interessa come è fatto il dispositivo nella sua realizzazione pratica o come siano i componenti fisici che fanno funzionare la macchina.

Si vuole, invece, mettere in evidenza l'aspetto logico della macchina, cioè le operazioni che compie e quale ordine segue per realizzarle. Inoltre i meccanismi interni che la fanno funzionare vengono schematizzati in modo intuitivo.

Si consideri il funzionamento di una *macchina per la distribuzione di bibite*.

La macchina accetta solo gettoni ed emette una bibita dopo l'introduzione di 2 gettoni.

La macchina inoltre è dotata di un meccanismo che controlla il contenuto del serbatoio della bibita e di un segnalatore luminoso che si accende quando il serbatoio è vuoto. In questo stato non accetta più gettoni.



Il funzionamento del dispositivo è regolato da un procedimento che descrive le operazioni da eseguire per ottenere la bibita in corrispondenza dell'introduzione dei gettoni:

- accettazione di un gettone
- accettazione di un secondo gettone
- riempimento del bicchierino di bibita.

Il procedimento viene ripetuto continuamente per tutto il tempo durante il quale la macchina rimane accesa.

In effetti la macchina deve essere in grado di gestire anche le eccezioni, quali:

- introduzione di gettoni non validi
- controllo sul livello di liquido nel serbatoio.

Per poter funzionare il dispositivo deve essere dotato di un meccanismo di introduzione (per l'ingresso dei gettoni), di un meccanismo interno per il controllo delle varie fasi del lavoro, e di un'apparecchiatura per la distribuzione della bibita.

In modo molto schematico la macchina può essere descritta così:



Possiamo anche immaginare che la macchina durante il suo funzionamento si trovi in situazioni diverse o **stati** del dispositivo, e che possa cambiare la sua situazione interna passando da uno stato all'altro, in corrispondenza di qualche evento interno o esterno.

Più precisamente si può supporre che:

- se non sono stati ancora inseriti gettoni, la macchina si trovi in stato di accettazione e non emetta bibita;
- se il serbatoio si svuota, la macchina si metta in stato di serbatoio vuoto e meccanismo non funzionante, non accettando gettoni e non emettendo bibite;
- se i gettoni sono stati inseriti, la macchina passi allo stato di distribuzione bibite e non accetti più gettoni prima di completare questa fase del lavoro.

A una macchina di questo tipo possiamo dare il nome di **automa**, intendendo indicare con questo termine un dispositivo che sia in grado di eseguire *da solo*, cioè in modo automatico, senza l'intervento di una persona, una sequenza di azioni stabilite in precedenza, dotato di meccanismi per acquisire elementi dall'esterno e produrre elementi verso l'esterno: durante il suo funzionamento inoltre, all'interno, può assumere stati diversi fra loro.

Gli aspetti che caratterizzano il funzionamento di un automa di questo tipo sono:

- i simboli forniti dall'esterno, che l'automa sa riconoscere nel loro significato;
- gli elementi prodotti all'esterno come risultato del lavoro svolto;
- l'insieme di tutti i possibili stati che l'automa può assumere;
- l'insieme di tutte le possibili transizioni da uno stato all'altro.



DEFINIZIONE FORMALE DI AUTOMA

Dal punto di vista formale l'automa viene definito come un insieme di 5 elementi:

- 1) **l'alfabeto dei simboli di input**, cioè l'insieme dei simboli che l'automa è in grado di ricevere dall'esterno, riconoscendoli:

$$I = (i_1, i_2, \dots, i_n)$$

- 2) **l'alfabeto dei simboli di output**, cioè l'insieme dei simboli che l'automa comunica verso l'esterno:

$$U = (u_1, u_2, \dots, u_m)$$

- 3) **l'insieme dei possibili stati** che l'automa può assumere, cioè le situazioni più importanti dell'automa durante il suo funzionamento:

$$S = (s_1, s_2, \dots, s_p)$$

4) **la funzione degli stati successivi**, cioè la relazione che indica lo stato assunto dall'automa (nell'istante t successivo a quello considerato) quando, trovandosi in un determinato stato, accetta dall'esterno un determinato simbolo di input:

$$(i_t, s_{t-1}) \xrightarrow{F} s_t$$

5) **la funzione delle uscite**, cioè la relazione che indica il simbolo che viene emesso verso l'esterno, in corrispondenza di un determinato stato e di un determinato simbolo di input:

$$(i_t, s_{t-1}) \xrightarrow{G} u_t$$

Definiamo quindi l'automa come l'insieme (la quintupla) degli elementi descritti sopra:

$$A = (I, U, S, F, G)$$

Le relazioni stabilite dalla funzione degli stati successivi e dalla funzione delle uscite possono essere rappresentate con un'unica relazione, a cui si dà il nome di **funzione di transizione**, che in corrispondenza di una determinata coppia formata dal simbolo di input e dallo stato in cui si trovava l'automa, indica lo stato successivo e il simbolo di output:

$$(i_t, s_{t-1}) \rightarrow (s_t, u_t)$$

Il funzionamento dell'automa consiste quindi nell'accettare un simbolo dall'esterno ed emettere un simbolo in uscita, producendo un cambiamento di stato.

Possiamo supporre che la macchina possieda le seguenti caratteristiche:

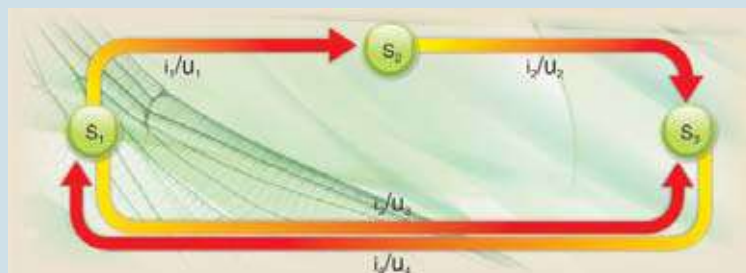
- l'insieme dei simboli di input, l'insieme dei simboli di output e l'insieme degli stati possibili sono in numero finito; per questo motivo si chiama anche **automa a stati finiti**;
- lo stato successivo dell'automa è univocamente determinato dallo stato in cui si trovava nell'istante precedente e dal simbolo di input accettato (**automa deterministico**); questo significa che nella funzione di transizione non ci possono essere due coppie uguali (*simbolo di input, stato*) a cui corrispondono due coppie diverse (*stato successivo, simbolo di output*);
- l'automa funziona secondo una successione discreta di eventi, muovendosi a scatti da uno stato all'altro (**automa discreto**); questo comporta anche che le variabili, che caratterizzano l'automa, assumano valori che non variano con continuità nel tempo.

Un automa con queste caratteristiche si chiama anche **macchina sequenziale**, perché il modo di funzionare della macchina in un determinato istante dipende dal modo di funzionare nell'istante precedente.

Per descrivere il funzionamento di un automa in modo più formalizzato si usano di solito i grafi e le tabelle.

Il **grafo** è uno schema nel quale i cerchi indicano gli stati dell'automa e gli archi orientati che li congiungono indicano le transizioni da uno stato all'altro.

Sugli archi vengono indicati il simbolo di input, che determina il passaggio di stato, e il simbolo di output prodotto all'esterno. I due simboli sono separati dal segno /.



Con riferimento all'esempio della figura si può quindi dire che l'automa, trovandosi nello stato s_1 e ricevendo in input il simbolo i_1 , passa allo stato s_2 , producendo verso l'esterno il simbolo u_1 .

Quando si trova in s_1 e riceve in input il simbolo i_3 , passa allo stato s_3 e produce verso l'esterno il simbolo u_3 .

Le **tabelle** servono a rappresentare schematicamente la funzione degli stati successivi e la funzione delle uscite, descritte precedentemente.

Nella prima tabella, detta **tabella degli stati successivi**, sono indicati sulle righe i simboli di input e sulle colonne gli stati.

All'incrocio tra una riga e una colonna viene indicato lo stato in cui l'automa viene a trovarsi in corrispondenza del simbolo di input e dello stato precedente.

Tabella degli stati successivi

input \ stati	s_1	s_2	s_3	s_4
i_1	s_{11}	s_{12}	s_{13}	s_{14}
i_2	s_{21}	s_{22}	s_{23}	s_{24}
i_3	s_{31}	s_{32}	s_{33}	s_{34}

Nella seconda tabella, detta **tabella delle uscite**, sono indicati, come nella precedente, sulle righe i simboli di input e sulle colonne gli stati; all'incrocio tra una riga e una colonna viene indicato il simbolo di output comunicato all'esterno in corrispondenza del simbolo di input e dello stato precedente.

Tabella delle uscite

input \ stati	s_1	s_2	s_3	s_4
i_1	u_{11}	u_{12}	u_{13}	u_{14}
i_2	u_{21}	u_{22}	u_{23}	u_{24}
i_3	u_{31}	u_{32}	u_{33}	u_{34}

Le due tabelle possono essere raggruppate in un'unica tabella, che schematizza la funzione di transizione e che prende il nome di tabella di transizione o **matrice di transizione**. Sulle righe sono indicati i simboli di input e sulle colonne gli stati: all'incrocio tra una riga e una colonna vengono indicati lo stato successivo e il simbolo di output separati da una virgola.

Matrice di transizione

input \ stati	s_1	s_2	s_3	s_4
i_1	s_{11}, u_{11}	s_{12}, u_{12}	s_{13}, u_{13}	s_{14}, u_{14}
i_2	s_{21}, u_{21}	s_{22}, u_{22}	s_{23}, u_{23}	s_{24}, u_{24}
i_3	s_{31}, u_{31}	s_{32}, u_{32}	s_{33}, u_{33}	s_{34}, u_{34}

ESEMPIO

Si consideri il funzionamento di un distributore automatico di carburante. Dall'esterno accetta la banconota che indicheremo con Ba e la pressione del pulsante del carburante che indicheremo con Pu. Verso l'esterno emette un segnale luminoso (Lu), per indicare che la colonnina è pronta, e il carburante (Ca).

Durante il funzionamento, questo automa può trovarsi in due stati: stato di accettazione delle banconote (**Ac**) e stato di distribuzione di carburante (**Di**).

Il distributore formalizzato come un automa possiede come simboli di input l'insieme

$$I = (Ba, Pu)$$

come insieme dei simboli di output

$$U = (Lu, Ca)$$

e come insieme degli stati

$$S = (Ac, Di).$$

Quando riceve dall'esterno una banconota, trovandosi in stato di accettazione, emette un segnale luminoso e passa allo stato di distribuzione del carburante.

Quando riceve la pressione del pulsante, trovandosi in stato di distribuzione, emette il carburante e passa allo stato di accettazione delle banconote.

Il grafo che descrive il funzionamento dell'automa ha la seguente forma:



Le due tabelle sono rappresentate secondo questi schemi:

Tabella degli stati successivi

stati \ input	Ac	Di
Ba	Di	Di
Pu	Ac	Ac

Tabella delle uscite

stati \ input	Ac	Di
Ba	Lu	-
Pu	-	Ca

AUTOVERIFICA

Domande da 1 a 4 pag. 81

Problemi da 1 a 9 pag. 83-84

**MATERIALI ONLINE****1. Esempi di automi****2 Struttura generale del sistema di elaborazione**

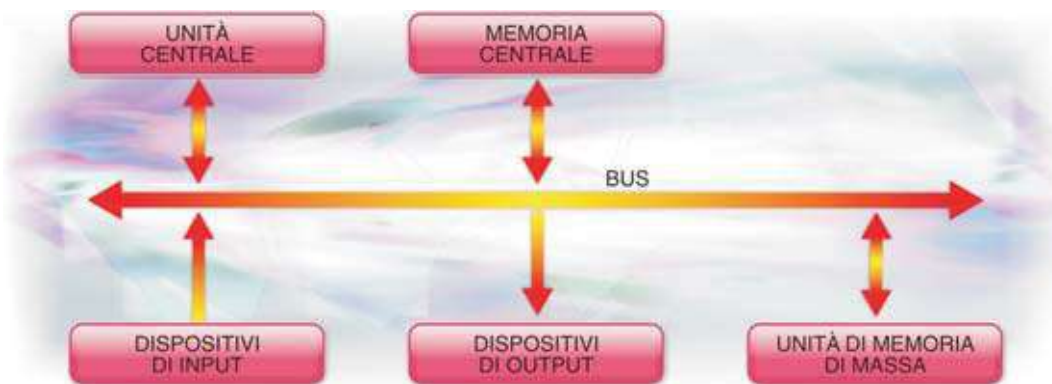
Verranno esaminate ora le caratteristiche fisiche di una macchina reale per l'elaborazione delle informazioni e le funzioni che vengono attivate per consentire all'utente di utilizzare le risorse a disposizione. Essa deve avere alcune caratteristiche fondamentali:

- può essere utilizzata per problemi di tipo diverso, e non solo per calcoli;
- può eseguire operazioni logiche (per esempio il confronto tra due grandezze);
- può trattare non solo numeri, ma anche caratteri qualsiasi;
- è in grado di comprendere ed eseguire una serie di ordini impartiti dall'esterno, organizzati in un procedimento risolutivo e codificati utilizzando appositi linguaggi di programmazione;
- è in grado di elaborare grandi quantità di dati, con una velocità elevata.

Un'apparecchiatura con queste caratteristiche viene chiamata **computer** o **elaboratore** ed è costituita da un insieme di dispositivi di diversa natura (elettrici, meccanici, ottici), in grado di acquisire dall'esterno dati e algoritmi, e produrre in uscita i risultati dell'elaborazione.

Spesso si usa il termine **sistema di elaborazione**, per evidenziare il fatto che l'elaborazione avviene attraverso un insieme organizzato di risorse diverse. Queste risorse possono essere classificate in due grandi categorie: le risorse **hardware**, che sono la parte fisica del sistema, cioè i dispositivi e le apparecchiature meccaniche, elettriche, elettroniche e ottiche, che si possono riconoscere fisicamente, e le risorse **software**, che sono la parte logica del sistema, cioè i programmi e le procedure che fanno funzionare l'hardware.

Tutte le risorse interagiscono tra loro con la finalità di rendere disponibili le funzioni di elaborazione dei dati all'utente: questo giustifica l'uso del termine *sistema*, secondo la definizione data nel capitolo iniziale.



Dal punto di vista dell'hardware, la struttura essenziale di un elaboratore comprende:

- l'unità centrale di elaborazione (**CPU**, acronimo di *Central Processing Unit*, cioè unità centrale di elaborazione),
- la memoria centrale,
- i dispositivi di input,
- i dispositivi di output,
- le unità di memoria di massa.

Le unità comunicano tra loro seguendo regole codificate, chiamate **protocolli di comunicazione**, usando per i trasferimenti di informazioni uno o più percorsi elettrici chiamati **bus** di sistema.

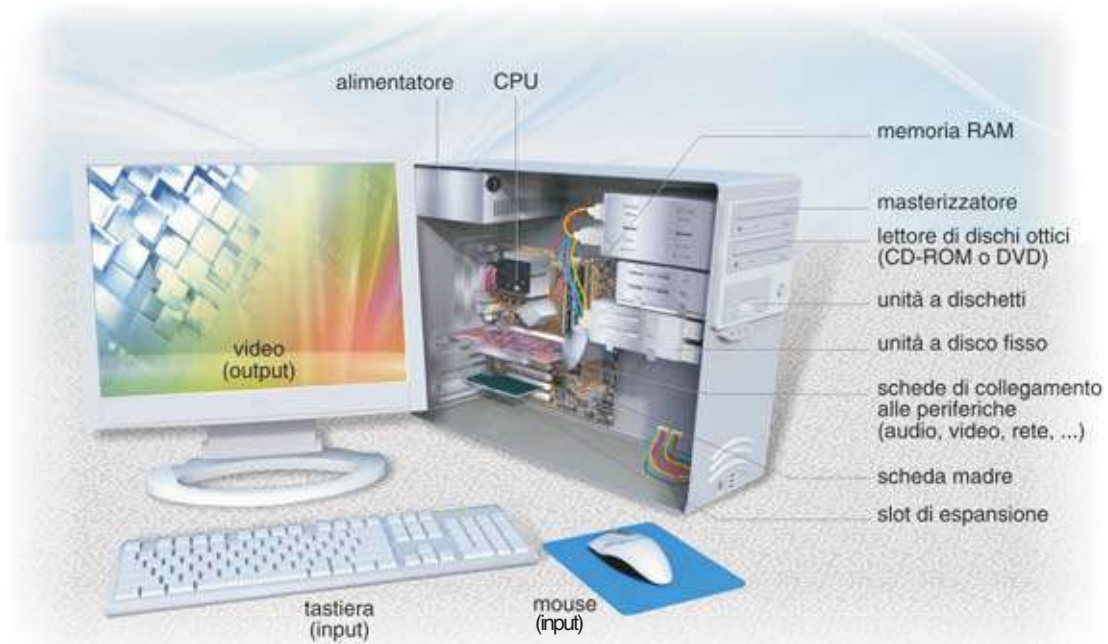
Questo modello di elaboratore o, come si dice in informatica, questa **architettura**, corrisponde alla **macchina di Von Neumann**, dal nome dello scienziato che nel 1952 elaborò negli Stati Uniti un progetto di computer che costituisce tuttora la base concettuale della maggior parte dei sistemi di elaborazione.

Ogni singolo elemento del sistema è caratterizzato da:

- la capacità di interagire con gli altri elementi e con l'esterno;
- la situazione in cui si trova in ogni istante, cioè il suo *stato*;
- l'evoluzione che tale stato compie a seconda dello stimolo ricevuto.

Queste proprietà indicano che i dispositivi sono *automi*, nel senso visto nel paragrafo precedente, e perciò l'insieme formato dal loro assemblaggio è ancora un automa.

Le componenti hardware presenti in un personal computer vengono illustrate con la seguente figura; esse verranno in seguito esaminate nel dettaglio.



I sistemi di elaborazione vengono tradizionalmente classificati secondo diversi livelli, corrispondenti a complessità e potenza di elaborazione crescenti:

1. **calcolatori** tascabili;
2. **piccoli sistemi di elaborazione**: personal computer da scrivania (computer *desktop*), computer portatile (detto anche *laptop* o *notebook*), tablet PC, netcomputer o *netbook*, PDA (*Personal Digital Assistant*) o *palmare*, telefono cellulare con funzionalità avanzate (*smartphone*);
3. **medi sistemi di elaborazione**: minicomputer con terminali (tastiera e video) per gli utenti oppure computer collegati in una rete locale;
4. **grandi sistemi di elaborazione**, detti comunemente *mainframe*.

3 L'unità centrale di elaborazione

L'unità centrale di elaborazione o **CPU** (*Central Processing Unit*) è il dispositivo che esegue materialmente le operazioni logiche, aritmetiche e di trasferimento sui dati secondo il procedimento richiesto; è a tutti gli effetti il cuore del sistema di elaborazione, ovvero quella componente che effettua l'elaborazione delle informazioni.

Si può considerare come costituita da:

- una **unità di elaborazione**, detta **unità aritmetico-logica** o **ALU** (*Arithmetic Logic Unit*), in grado di eseguire le istruzioni richieste
- una **unità di governo**, detta **unità di controllo** o **CU** (*Control Unit*), che stabilisce quali operazioni debbano essere eseguite.



La CPU svolge all'interno del sistema di elaborazione le funzioni di *processore*, secondo la definizione data nel capitolo introduttivo. I processi sono le diverse attività di elaborazione della CPU. Il processore viene realizzato concretamente su un chip di silicio (e prende il nome di **microprocessore**), utilizzando la tecnologia di integrazione su larga scala (**VLSI**, *Very Large Scale Integration*), cioè la possibilità di mettere tanti dispositivi logici su una superficie molto ristretta.

L'unità di elaborazione contiene elementi di memoria detti **registri**, usati per memorizzare dati e risultati delle operazioni, nonché i codici operativi del linguaggio macchina.

Tra i registri utilizzati dalla CPU ci sono:

- *Program Counter*, contenente l'indirizzo della memoria dove si trova la prossima istruzione da eseguire
- *Registro istruzione*, contenente l'istruzione attualmente in esecuzione
- *Registro di stato*, per il controllo delle condizioni o delle anomalie che si verificano durante le operazioni
- *Accumulatore*, utilizzato per effettuare somme e memorizzare risultati parziali delle operazioni
- *Registro dati di memoria*, contenente il dato prelevato da una cella di memoria o pronto per essere trasferito in una cella di memoria
- *Registro indirizzi di memoria*, contenente l'indirizzo della cella di memoria dove si deve prelevare o depositare un dato o un'istruzione.

L'unità di governo fornisce all'unità di elaborazione i segnali elettrici che attivano i diversi dispositivi di memoria o di operazione. Questi segnali vengono forniti in sincrono con un orologio interno della macchina, chiamato con il termine inglese **clock**: ad ogni scatto del clock viene inviato un segnale.

La frequenza con cui il clock scatta fornisce perciò un'importante indicazione sulla velocità a cui opera l'unità centrale: la frequenza si misura in Megahertz (**Mhz**) o Gigahertz (**Ghz**), cioè in milioni (Mega) o in miliardi (Giga) di "colpi", o *cicli*, al secondo. Nei personal computer più recenti questa frequenza può raggiungere il valore dell'ordine dei Gigahertz, cioè miliardi di cicli al secondo.

Alcune operazioni richiedono un solo ciclo di clock per essere eseguite, altre più di uno: in media ci saranno più cicli di clock che istruzioni eseguite e perciò la velocità dell'unità centrale, espressa in istruzioni al secondo sarà inferiore a quella del clock.

L'unità di misura della velocità di un'unità centrale è il **MIPS**, acronimo di *Millions Instructions Per Second* (milioni di operazioni al secondo).

Le istruzioni elementari eseguite dalla CPU sono notevolmente più semplici delle operazioni descritte con un linguaggio di programmazione evoluto usato per costruire i programmi software.

Esempi di istruzioni eseguite dalla CPU sono:

- il trasferimento di un dato dalla memoria in un registro della CPU (*Load*)
- il trasferimento di un dato da un registro della CPU alla cella di memoria (*Store*)
- il confronto tra il contenuto di due registri (*Compare*);
- la somma di dati utilizzando il registro Accumulatore (*Add*).

Si consideri per esempio il problema del calcolo del prodotto di due numeri: sono necessari tre registri, R1, R2, R3 per contenere rispettivamente i due fattori e il prodotto.

Il procedimento di calcolo viene realizzato sommando al contenuto del registro R3, azzerato all'inizio, il valore di R1 per un numero di volte pari al valore di R2.

Esso risulta composto dalle seguenti operazioni elementari:

	inizio
	azzerà il contenuto di R3
	copia dalla memoria in R1 il primo fattore
	copia dalla memoria in R2 il secondo fattore
etichetta 1:	se R2 contiene il valore 0 allora salta all'etichetta 2
	altrimenti aggiungi il contenuto di R1 a R3
	decrementa R2
	torna all'etichetta 1
etichetta 2:	copia in memoria il contenuto di R3
	fine

La moltiplicazione si ottiene, cioè, con somme successive e diventa un'operazione complessa, che deve essere codificata. Operazioni ancora più complesse devono essere realizzate attraverso procedimenti più complessi; per esempio la differenza viene realizzata con la logica del complemento a 2, l'estrazione di radice attraverso tabelle e algoritmi di approssimazione.

4 La memoria centrale

La **memoria centrale** è un dispositivo elettronico in grado di memorizzare istruzioni e dati codificati in forma binaria. Materialmente essa è costituita da uno o più **chip** (che in inglese significa *scheggia*), cioè da piastrine di silicio incapsulate in materiale plastico e fornite di contatti metallici, chiamati *piolini* (in inglese **pin**), che ne consentono l'inserimento su circuiterie più estese chiamate **schede**, e quindi il collegamento elettrico con il resto del sistema. Le dimensioni di un chip sono dell'ordine dei centimetri, e quindi lo spazio occupato dalle unità di memoria è estremamente ridotto.



Il funzionamento dei sistemi digitali, quali i calcolatori elettronici, si basa sul sistema binario per trattare le informazioni. Come già si è visto in precedenza a proposito del sistema binario, l'unità di informazione è il **bit** (dall'inglese *binary digit*, ovvero una cifra che può assumere il valore 0 o il valore 1), che però non è sufficiente per codificare una benché minima quantità di informazioni. Si usa quindi, come base per trattare le informazioni, un suo multiplo, il **byte** corrispondente ad 8 bit, con il quale si possono rappresentare $2^8 = 256$ diverse informazioni, quante sono le combinazioni di 2 elementi (la cifra 0 e la cifra 1) in 8 posti. Con queste combinazioni di bit si rappresentano i caratteri della codifica ASCII numerati da 0 a 255, oppure le istruzioni per il calcolatore espresse in linguaggio macchina.

La memoria centrale è pensabile, dal punto di vista logico, come una sequenza finita di locazioni della dimensione di uno o più byte; ogni locazione prende il nome di **parola** (in inglese **word**).

I parametri che caratterizzano la memoria sono:

- la dimensione della parola,
- la dimensione complessiva della memoria (*capacità*),
- le modalità di accesso (solo lettura o anche scrittura),
- la natura di permanenza o volatilità,
- la velocità con cui risponde alle richieste.

Ogni parola di memoria può essere chiamata anche **cella** di memoria, in quanto è una locazione fisica nella quale è possibile memorizzare uno o più byte, ovvero numeri da 0 a 255, e che può rappresentare diversi oggetti: un numero naturale, un carattere (codifica ASCII) o un codice operativo proprio del linguaggio macchina, ovvero del linguaggio interno di programmazione di un sistema.

Ogni cella di memoria deve essere in qualche modo identificata univocamente, per potervi accedere senza essere confusa con le altre.

Per identificare le celle di memoria viene utilizzato un sistema molto semplice: le celle vengono numerate da 0 in poi, ogni cella viene così ad avere un proprio numero identificativo, che viene detto **indirizzo** (in inglese *address*).

Indirizzo in binario	Indirizzo in esadecimale	
0000 0000 0000 0000	0000	cella 0
0000 0000 0000 0001	0001	cella 1
0000 0000 0000 0010	0002	cella 2
0000 0000 0000 0011	0003	cella 3
.....		
1111 1111 1111 1110	FFFE	cella 65534
1111 1111 1111 1111	FFFF	cella 65535

Gli indirizzi di memoria sono gli oggetti sui quali un calcolatore deve lavorare: se per esempio un programma richiede di recuperare un dato memorizzato, deve specificare l'indirizzo in cui è contenuto, e per far questo deve ricordare anche l'indirizzo che gli serve registrandolo in una locazione di memoria particolare.

Quindi è comodo che un indirizzo sia un numero binario, e che sia memorizzabile in uno spazio delle dimensioni di uno o più byte. Il numero di bit utilizzati dalla CPU per identificare gli indirizzi definisce lo **spazio di indirizzamento** (*addressing*).

Nello schema presentato sopra, la memoria è vista come una sequenza di 65536 celle; per indirizzarle ci serve un numero da 0 a 65535, e per scrivere tutti gli indirizzi ci servono 16 bit, ovvero 2 byte; con 2 byte si possono infatti indirizzare $2^{16} = 65536$ celle. Così come con n bit di indirizzamento si possono gestire memorie di 2^n celle.

Si osservi che è comodo usare come spazi di indirizzamento le potenze del numero 2, vista l'intrinseca natura binaria del sistema elettronico.

I multipli del byte vengono usati per caratterizzare le dimensioni complessive dei dispositivi di memoria:

1024 byte (2^{10} byte) formano un **Kilobyte**, abbreviato con KB o semplicemente K o Kbyte;

1024 Kbyte formano un **Megabyte**, detto anche MB o Mega o Mbyte;

1024 Mega formano un **Gigabyte** o GB o Giga o Gbyte.

1024 Giga formano un **Terabyte** o TB o Tera o Tbyte

La memoria descritta con lo schema precedente, per esempio, può indirizzare fino a 64Kbyte ($64 \times 1024 = 65536$) posizioni di memoria.

Quindi, riassumendo, un Mega è costituito da circa un milione di byte, cioè 8 milioni di bit, mentre un Giga comprende circa un miliardo di byte, ovvero 8 miliardi di bit e un Tera circa mille miliardi di byte, cioè 8 mila miliardi di bit. Queste unità di misura vengono usate per determinare la **capacità di una memoria**, cioè la quantità di informazioni che essa può contenere. Un personal computer ha normalmente una memoria con dimensione di alcuni Gigabyte, mentre sistemi più grandi hanno ovviamente memorie di dimensioni maggiori.

La **velocità** del dispositivo è legata al tempo che intercorre tra la richiesta di accedere ad una certa parola e l'istante in cui l'operazione è eseguita. Tale tempo è dell'ordine dei *nanosecondi*, cioè dei milionesimi di secondo; se si vuole fare un paragone di velocità, per accedere ad un disco fisso si hanno tempi dell'ordine dei millisecondi, ovvero la memoria centrale è circa mille volte più veloce di un disco fisso.

Tra le memorie di tipo elettronico usate nei calcolatori è importante distinguere tra **ROM** o *Read Only Memory* (memoria di sola lettura) e **RAM** o *Random Access Memory* (memoria ad accesso casuale, nel senso di accesso diretto), sulle quali è possibile anche scrivere.

La memoria centrale deve contenere necessariamente dispositivi del secondo tipo, in quanto su di essa occorre poter sia leggere le informazioni che scriverne di nuove, e solo le memorie RAM offrono questa possibilità.

Le memorie ROM vengono usate diffusamente in tutti quei casi in cui non serva modificare il contenuto della memoria: tipicamente contengono le istruzioni per la fase detta di **bootstrap**, ovvero di accensione e avvio del sistema.

Esse, inoltre, contengono programmi in linguaggio macchina eseguiti spesso per la gestione standard dei dispositivi quali video, tastiera, porte di input/output, oppure tabelle matematiche utilizzate dall'unità aritmetico-logica della CPU per svolgere operazioni aritmetiche di base e quindi di frequente consultazione.

La RAM è composta di un grande numero di oggetti elettronici in grado di assumere uno tra due stati stabili, non soggetti cioè a variazioni in assenza di stimoli esterni, come un interruttore a due posizioni. Ad uno dei due stati viene convenzionalmente attribuito il significato '0', all'altro il significato '1'.

Il funzionamento di questi dispositivi è subordinato alla presenza della tensione di alimentazione; in altre parole, se si spegne la macchina tutto il contenuto della RAM viene perduto: si dice che la RAM costituisce una **memoria volatile**.

La struttura delle ROM è invece tale che l'informazione viene ritenuta anche se manca alimentazione (memorie non volatili) e questa caratteristica le rende adatte, per esempio, a contenere i programmi di inizializzazione del sistema, cui abbiamo già accennato.

La funzione della memoria centrale all'interno del sistema di elaborazione è di provvedere in modo rapido a fornire alla CPU le operazioni da eseguire e i dati su cui eseguirle, in una forma codificata che usa solo i simboli dell'alfabeto binario cioè 0 e 1.

Per far questo occorre che si sia stabilita una volta per tutte quale debba essere la codifica usata, e che questa non possa dare adito ad equivoci.

Ogni parola di memoria può contenere un'istruzione oppure uno o più dati o parte di essi. Qualora contenga un'istruzione, parte della parola (per esempio 6 bit) è destinata ad ospitare il **codice operativo**, cioè una sequenza di bit che identifica in modo univoco un'operazione da eseguire, mentre il resto della parola è occupato da informazioni riguardanti gli **operandi** (principalmente il loro indirizzo).

CODICE OPERATIVO

OPERANDI oppure INDIRIZZO DEGLI OPERANDI

I caratteri utilizzati dal computer secondo la codifica ASCII occupano 8 bit e perciò possono essere memorizzati uno per byte: per questa ragione spesso il termine **carattere** viene usato come sinonimo di byte.

I numeri interi vengono rappresentati utilizzando da 16 a 32 bit e perciò occupano dai 2 ai 4 byte, mentre i numeri reali richiedono dai 32 ai 64 bit, cioè 4 o 8 byte, e devono perciò essere memorizzati su più parole.

Dire che i caratteri occupano 8 bit significa che a ciascuno viene convenzionalmente assegnato un codice composto di 8 cifre binarie: questo può essere visto come un numero in base 2 compreso tra 0000 0000 (0 decimale) e 1111 1111 (255 decimale).

Perciò si hanno 256 codici disponibili per altrettanti caratteri: lettere alfabetiche maiuscole e minuscole, cifre decimali, segni di punteggiatura, caratteri di controllo.

I trasferimenti di dati e istruzioni tra memoria centrale e CPU avvengono in **parallelo**, cioè trasmettendo contemporaneamente tutti i bit di una parola.

Tanto maggiore è l'ampiezza della parola, tanto minore è il numero di accessi alla memoria che devono essere eseguiti dalla CPU per accedere ai dati da elaborare, e quindi aumenta la velocità di esecuzione di un programma.

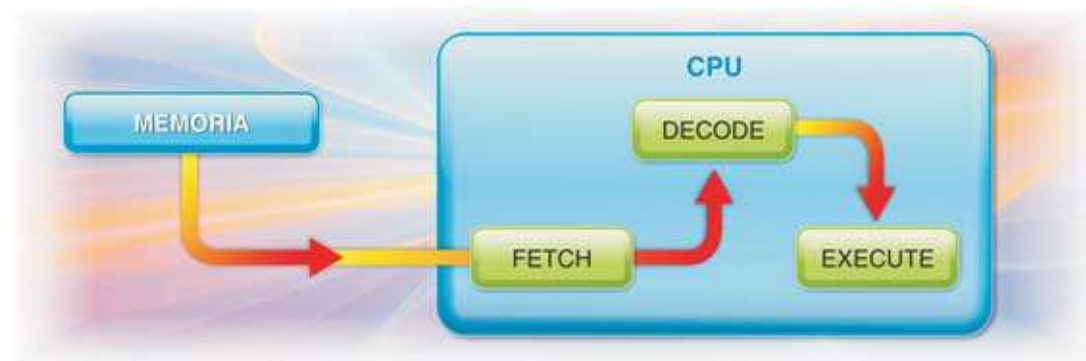
Il numero dei bit che vengono trasferiti in parallelo costituisce quindi uno dei parametri che determinano la potenza di elaborazione di un computer e viene detto **grado di parallelismo**.

I grandi sistemi usano processori che gestiscono parole di 64 bit mentre su elaboratori più piccoli vengono adottate CPU a 16 o 32 bit.

L'attività dell'unità centrale è strettamente legata alla memoria centrale.

La memoria centrale contiene i programmi in esecuzione e fornisce alla CPU, una per volta, le istruzioni da eseguire.

La CPU passa da uno stato detto di **fetching** (prelevamento), nel quale essa legge dalla memoria centrale una sequenza di bit corrispondente ad un'istruzione, ad uno stato detto di **decoding** (decodifica), nel quale l'unità di governo trasforma l'istruzione in segnali inviati all'unità di elaborazione, quest'ultima si viene a trovare nello stato di **executing** (esecuzione), per eseguire materialmente le istruzioni.

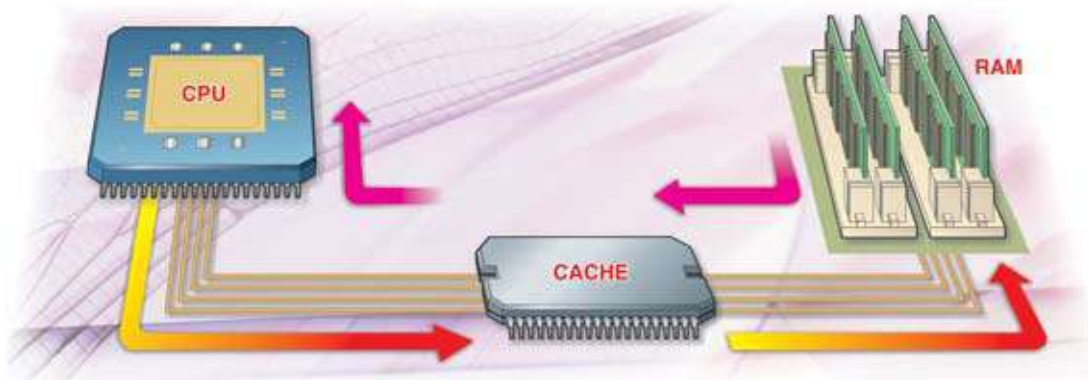


All'aumentare della velocità del processore, può succedere che la memoria centrale non riesca a servire con la massima efficacia l'unità centrale, perché i due dispositivi sono costruiti con tecnologie diverse e operano a velocità diverse. Per risolvere questo genere di problemi sono state introdotte le memorie *cache*.

La **memoria cache** è una memoria temporanea, utilizzata per trasferire dati tra dispositivi operanti a velocità di lavoro diverse (uno veloce e l'altro lento): tipici esempi sono il trasferimento da memoria di massa (dischi) a memoria RAM e tra memoria RAM e CPU.

Il funzionamento di una memoria *cache* si basa sulla memorizzazione di istruzioni e di dati usati più frequentemente dalla CPU, producendo così accessi più veloci.

L'uso della memoria *cache* è dovuto alla differenza tra le velocità del disco e della RAM (1000 volte) e a quella che intercorre tra la velocità della RAM normale e della *cache* per la CPU (alcune volte).



5 Le unità di input e di output

Le unità di ingresso/uscita dei dati (*Input/Output*), indicate comunemente come **unità di I/O**, consentono l'acquisizione dall'esterno dei dati che devono essere elaborati dal sistema e la comunicazione verso l'esterno dei risultati dell'elaborazione.

L'unità di input standard è la **tastiera** (in inglese *keyboard*), abbinata a un video che riproduce i caratteri battuti sui tasti (come un eco alla battitura).

La tastiera più il video formano anche il posto di lavoro, detto **terminale**, che collega l'utente a un sistema di elaborazione di medie o grandi dimensioni.

L'uso esteso delle interfacce grafiche (con finestre e icone), per il colloquio tra utente e computer nei moderni sistemi di elaborazione, ha di fatto reso indispensabile la presenza del mouse, come strumento di input che sostituisce la tastiera per gran parte del lavoro svolto dall'utente.

Il **mouse** è un piccolo oggetto che con il movimento sul piano del tavolo fa muovere un puntatore sul video del computer, consentendo la scelta tra diversi lavori possibili o diversi comandi presentati all'utente sul video del computer; le scelte sono rappresentate attraverso piccoli disegni (**icone**) che ne ricordano la funzione svolta.

In alternativa al mouse tradizionale che viene mosso sul piano del tavolo, si può usare il **trackball**, un mouse formato da due tasti e da una sfera: facendo ruotare la sfera, l'utente può muovere il puntatore sul video.

Il **touchpad** è un dispositivo di puntamento che svolge le stesse funzioni del mouse, costituito da una piccola membrana sensibile al movimento del dito e alla pressione sopra di essa. Viene usato di solito con i computer portatili (*laptop*).

Il **joystick** è utilizzato prevalentemente nei giochi elettronici con il computer, per guidare con più facilità i movimenti del giocatore.



Altre unità di input sono:

- i sistemi ottici per leggere i **codici a barre** (*bar code*), cioè barre verticali di vario spessore, disposte per ottenere una codifica di prodotti commerciali e apposte sulle etichette delle confezioni;
- lo **stilo** utilizzato nei palmari o nei Tablet PC in alternativa alla tastiera o al mouse;
- lo **scanner**, che consente di acquisire immagini o di leggere parti testuali di documenti trasformandoli in segnali digitali che vengono immessi nel computer (funzioni di OCR, *Optical Character Reader*, lettore ottico di caratteri);
- la **tavoletta grafica** (*digitizer*) per rilevare le coordinate dei punti di un disegno;
- la **macchina fotografica digitale** (*digital camera*), in grado di registrare e conservare immagini in formato digitale, che possono poi essere trasferite nel computer;
- la **telecamera digitale** (*Webcam*), in grado di acquisire le immagini in movimento di una persona vicina al computer e di trasmetterle sulla rete Internet;
- il **microfono** per acquisire dall'esterno suoni, discorsi o commenti.

Il **video** (o display o monitor) è l'unità di output standard, perché oltre a fare l'eco alla tastiera, riporta i messaggi di risposta e di errore del computer all'utente, e viene usato per visualizzare messaggi e dati di output.

I monitor moderni si basano sulla tecnologia **LCD** (*Liquid Crystal Display*), cioè a cristalli liquidi.

L'unità di output più usata, oltre al video, è la **stampante** (*printer*), un'apparecchiatura per riprodurre i caratteri di risultati e messaggi su un foglio di carta.

Le principali tecniche di stampa usate nelle stampanti sono:

- **getto di inchiostro** (*inkjet*): la scrittura avviene attraverso un getto di minuscole gocce di inchiostro caricate elettrostaticamente e proiettate contro la carta;
- **laser**: la pagina da stampare, memorizzata in una memoria di tipo RAM della stampante, attraverso un raggio laser viene costruita in modo latente con cariche elettrostatiche sulla superficie di un tamburo rivestito di materiale fotosensibile; l'immagine viene poi realizzata materialmente con inchiostro in polvere (*toner*), caricato elettrostaticamente, che viene fatto aderire al tamburo.

L'inchiostro infine viene trasferito su un foglio di carta e fissato in modo indelebile tramite riscaldamento. Il procedimento è lo stesso utilizzato per le macchine fotocopiatrici, e quindi le stampanti laser sono soggette allo stesso tipo di manutenzione delle fotocopiatrici.



Un'altra unità di output è il **plotter** o tracciatore di disegni, una tavoletta o un rullo sul quale è appoggiato un foglio dove la penna collegata a un braccio meccanico disegna il grafico.

Tra le unità di output possiamo anche classificare le **casse musicali** (*speakers*) o le **cuffie** di cui un personal computer è normalmente dotato: esse servono a riprodurre il suono proveniente da un programma di giochi, da un CD musicale o da un file audio scaricato da Internet.

In un sistema di elaborazione ci sono anche alcune apparecchiature periferiche che sono unità sia di input che di output:

- il **modem** (*modulatore/demodulatore*) è l'apparecchiatura che collega un computer alle linee telefoniche per realizzare le connessioni alle reti ed a Internet, di cui si parlerà più diffusamente in seguito;
- il **video sensibile al tatto** (*touchscreen*, schermo tattile) è uno schermo sul quale l'utente può effettuare una scelta oppure selezionare un'icona attraverso il dito o uno stilo.



6 Le memorie di massa

Le **memorie di massa** o memorie esterne (per distinguerle dalla memoria centrale) sono i supporti che servono per registrare archivi di dati, testi, programmi dell'utente o programmi forniti dalla casa costruttrice del computer. Hanno la caratteristica peculiare di essere *memorie permanenti* ovvero le informazioni in esse contenute vengono conservate indifferente dal fatto che l'unità di elaborazione sia accesa o spenta.

Attualmente le memorie di massa più utilizzate sono costituite da supporti magnetizzabili, oppure da supporti ottici.

Sulle memorie di massa, che si chiamano così perché possono contenere quantità rilevanti di dati rispetto alla memoria centrale, vengono fatte *operazioni di input*, quando si leggono i dati registrati sul supporto di memoria e *operazioni di output*, quando si registrano nuovi dati. Quindi possono essere considerate **unità di I/O**.

Le memorie di massa sono caratterizzate da alcuni parametri fondamentali che ne descrivono le caratteristiche:

- il **tempo di accesso** (*access time*), espresso con i sottomultipli del secondo, indica il tempo richiesto affinché il computer possa ritrovare i dati registrati per poterli elaborare poi in memoria centrale;
- la **capacità** indica la quantità di informazioni (espressa in numero di byte, in sostanza di caratteri, e misurata con i suoi multipli, KByte, MByte, GByte, TByte) che il supporto può contenere;
- la **velocità di trasferimento dei dati** (*transfer rate*) indica la rapidità con la quale i dati vengono trasferiti dal supporto alla memoria centrale e si misura in KByte per secondo (KBps) o MByte per secondo (MBps).

Tutto quello che può essere registrato su un supporto di memoria di massa si indica con il termine generico di **file** (*archivio*): un file può essere un testo, un programma, o un archivio di dati.

Per una migliore organizzazione delle informazioni registrate su una memoria di massa, i file sono raggruppati in **cartelle** (in inglese *directory*), secondo criteri di omogeneità: i file di uno stesso utente oppure i file di una determinata applicazione o progetto.

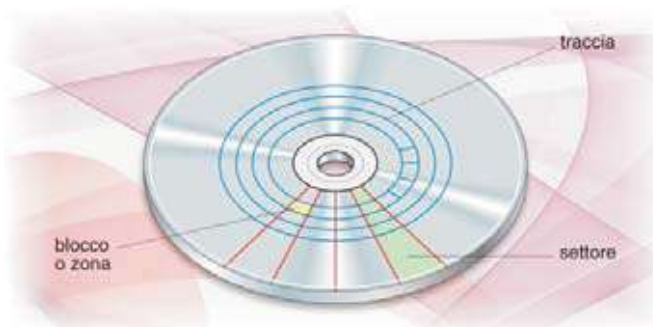
Nella misura della capacità di una memoria di massa, possiamo usare come riferimento la corrispondenza tra un carattere e un byte (secondo il codice ASCII dei caratteri usati dal computer): un file può essere formato anche da alcuni milioni di caratteri e quindi può avere dimensioni di alcuni MByte; una cartella che contiene molti file può anche raggiungere la dimensione di decine o centinaia di MByte.

I supporti di tipo magnetico più diffusi sono i dischi e i nastri.

I **dischi** possono essere di tipo diverso: dischi rigidi o *hard disk*, che possono contenere grandi quantità di dati e vengono usati per i sistemi medi e grandi, e i dischi esterni rimovibili di grandi capacità (da 100, 250 GByte, fino a un TByte).

Quasi sempre i dischi contenuti nelle unità di memoria per sistemi medi e grandi sono più di uno, e sono disposti uno sopra l'altro ruotanti tutti attorno allo stesso asse verticale.

Dal punto di vista fisico il disco è organizzato per **tracce** e **settori**. Le tracce sono tanti cerchi concentrici; i settori sono simili a "fette". All'incrocio tra la traccia e il settore si forma il **blocco fisico** dei dati registrati su disco.



Per funzionare il disco ha bisogno di un'apparecchiatura per leggere e scrivere i dati sulla sua superficie, dotata di una testina che si posiziona sulla traccia desiderata muovendosi dalla periferia al centro del disco e viceversa, mentre il disco ruota velocemente attorno a un asse verticale passante per il centro.

Il meccanismo che muove il disco si chiama **drive**, e spesso questo termine viene usato come sinonimo di unità a dischi.

I **nastri** (in inglese *tape*) hanno funzioni e capacità analoghe a quelle dei dischi, anche se l'organizzazione e l'accesso ai dati registrati seguono una tecnica diversa.



Mentre l'accesso ai dati di un disco avviene in modo diretto, sul nastro i dati sono registrati in modo sequenziale (uno di seguito all'altro) e l'accesso è di tipo sequenziale, che si ottiene con l'avvolgimento veloce in un verso o nell'altro del nastro. Il nastro può avere la forma di una bobina oppure di una cartuccia (*cartridge*).

I **CD-ROM** (*Compact Disk - Read Only Memory*) sono dischi di grandissima capacità basati su un sistema di lettura di tipo ottico.

I dati vengono registrati sul disco ottico, attraverso apparecchiature dette **masterizzatori** per CD-ROM, e il disco diventa di sola lettura per l'utente finale che utilizza per la riproduzione appositi **lettori di CD-ROM** collegati al computer, del tutto simili ai lettori di CD musicali.

Un lettore di CD-ROM viene caratterizzato dalla velocità di lettura (8X, 16X, 32X,...); più alta è la velocità, più efficiente è la presentazione di filmati e immagini in movimento.

I primi lettori di CD-ROM operavano con una velocità di trasferimento dal disco alla memoria del computer pari a 150 KByte per secondo (KBps), indicata con 1X e detta velocità singola (*single speed*). Le versioni successive delle periferiche per la lettura dei CD-ROM hanno raggiunto velocità molto più elevate: le sigle 2X, 4X, ..., 48X e 52X indicano l'incremento della velocità rispetto alla velocità dei primi lettori a singola velocità.

Per esempio un lettore 48X permette una velocità di trasferimento di 7200KBps (KByte per secondo).

Un CD-ROM può contenere fino a 700 MB di dati: risulta quindi particolarmente adatto per conservare documenti multimediali, dizionari, codici, norme fiscali, documentazioni tecniche di prodotti, consultabili facilmente e in tempi brevi: ci si può rendere conto dell'elevata capacità di questi dischi se si pensa che un libro medio occupa circa 2 MB.



I dischi ottici **CD-R** (*Recordable*) sono CD registrabili con il masterizzatore connesso al personal computer, per la creazione di CD che possono essere poi letti con un drive standard per CD; il contenuto del CD però non può più essere modificato con registrazioni successive. Per fare questo occorre usare i dischi **CD-RW** (*Rewritable*), che consentono aggiunte, modifiche e cancellazioni in fasi successive.



Negli anni più recenti la tecnologia ha creato i lettori e i masterizzatori per i nuovi supporti basati sulla tecnologia **DVD** (*Digital Versatile Disk*).

I lettori di dischi DVD possono leggere anche i normali CD-ROM, ma sono stati progettati per trattare le informazioni multimediali in formato digitale con alta qualità di riproduzione nel video e nel suono.

Il vantaggio dei dischi DVD è costituito dall'alta capacità di memorizzazione, che può essere di 4.7 Gigabyte su uno dei suoi due lati, corrispondenti a circa 133 minuti di filmato. Alcuni dischi DVD poi sono non solo a doppia faccia, ma anche a doppia densità di registrazione, quindi possono potenzialmente contenere circa 17 GB di video, audio o altri dati.

Le **chiavi USB** (dette anche *penne*) sono piccoli dispositivi con *memoria flash*, possono contenere grandi quantità di dati (dell'ordine dei Gigabyte) e si connettono al computer tramite la porta standard USB.

Altri supporti, utilizzati per memorizzare fotografie, sono le **memorie SD** (*Secure Digital*) per fotocamere e telefoni cellulari.

I dischi possono essere **fissi** o **rimovibili**, connessi direttamente alla *motherboard* del computer oppure esterni e collegati tramite una porta USB.

Le **unità di rete** sono dischi che stanno fisicamente su computer *server*, ai quali l'utente di rete può connettersi per prelevare o depositare dati, come se usasse un disco del proprio computer.



In ogni sistema di elaborazione, sia esso piccolo o grande, ha una grande importanza, per la sicurezza dei dati, il salvataggio di archivi e programmi su altri supporti di copia (**backup**). Per fare la copia del contenuto di un disco si usano nastri, dischi rimovibili o DVD.

7 Il collegamento delle periferiche

Le componenti di un sistema di elaborazione non facenti parte dell'unità centrale, e quindi le unità di I/O, le memorie di massa, dispositivi come la stampante, lo scanner, la videocamera ed in generale tutte quelle apparecchiature che possono essere messe in comunicazione con un computer, si indicano con il nome di **periferiche**.

Il collegamento tra il computer e le sue periferiche avviene mediante le apparecchiature che si chiamano **interfacce**. Un'interfaccia è in sostanza una scheda di circuiti elettrici che, inserita nella struttura di un computer, permette di eseguire il trasferimento di informazioni dal computer alle sue periferiche e viceversa: la scheda di interfaccia è connessa al bus mediante opportuni connettori sulla piastra madre (*motherboard*) del computer; il collegamento con la periferica è realizzato attraverso le **porte di I/O** presenti sulla scheda e accessibili dall'esterno.

La porta può essere **seriale**, quando i caratteri vengono trasmessi un bit per volta, o **parallela**, quando i bit del carattere vengono trasmessi contemporaneamente.

Per poter connettere apparecchiature periferiche di produttori diversi, il collegamento deve rispettare gli standard fissati a livello internazionale: seguendo questa esigenza, negli anni recenti si è affermato un nuovo standard di interfaccia seriale denominato **USB** (*Universal Serial Bus*).

Questa tecnologia di connessione presenta alcuni vantaggi: possibilità di connettere fino a 127 dispositivi in sequenza sulla stessa porta USB; alta velocità di trasferimento; connessione e configurazione rapida di nuove periferiche anche con computer funzionante (*PnP, plug and play*).

Un'altra porta tipica del computer consente la connessione alla **rete** tramite cavo. La porta **FireWire** è uno standard per la connessione di periferiche multimediali al computer, in particolare per l'archiviazione e il trasferimento veloce di file video.



AUTOVERIFICA

Domande da 5 a 11 pag. 81-82

Problema 10 pag. 84



MATERIALI ONLINE

2. Aggiornamenti sui sistemi di elaborazione e i dispositivi hardware

8 Il software

L'insieme delle apparecchiature, dei circuiti, delle memorie, dei cavi, degli interruttori, che formano un sistema di elaborazione si chiama **hardware**, cioè la parte fisica del computer e delle sue periferiche. Ma per poter funzionare un sistema di elaborazione ha bisogno anche del **software**.

Il **software** è l'insieme dei programmi e delle procedure che servono a finalizzare gli strumenti fisici alla risoluzione del problema presentato dall'utente del sistema.

Con il termine **programma** si indica in generale una sequenza di operazioni che devono essere eseguite dal computer, secondo l'ordine prestabilito. Si usa anche il termine inglese **routine**.

Il termine software comprende:

- il sistema operativo,
- i programmi di utilità e i programmi per la gestione delle periferiche (**driver**),
- i linguaggi di programmazione e i programmi di supporto per il programmatore,
- i programmi applicativi e i programmi scritti dall'utente,
- gli strumenti, indicati con il termine inglese *tools*.

I primi due vengono forniti dalla casa costruttrice del computer e costituiscono parte integrante del sistema di elaborazione. Senza il software infatti un sistema si riduce ad essere solo un insieme di apparecchiature che non producono nessun lavoro, hardware appunto (in inglese *hardware* significa ferramenta). L'insieme dei primi due prodotti software si chiama comunemente anche **software di base** o *software di sistema*. Gli ultimi due tipi di software si indicano con il termine generico di **software applicativo**.

Nel corso del tempo un sistema operativo o un applicativo software necessitano di revisioni per correggere errori, migliorare le prestazioni, aggiungere nuove funzionalità, adattarsi alle nuove caratteristiche fisiche delle piattaforme hardware, fornire modalità di interazione più facili per l'utente.

Questa attività di revisione produce **versioni** successive di un prodotto software indicate con il termine inglese **release**. Per esempio, nel corso degli ultimi anni il sistema operativo Windows è stato rilasciato nelle versioni Windows 95, Windows 98, Windows ME, Windows 2000, Windows XP, Windows Vista e Windows 7.

Un programma applicativo può avere una versione iniziale 1.0 e le successive 1.2, 1.3 o 2.0, 2.1 ecc.: di solito la modifica della parte decimale del numero di release corrisponde a modifiche non sostanziali rispetto alla precedente, mentre la modifica della parte intera del numero di release corrisponde all'aggiunta di nuove funzionalità importanti e migliorative rispetto alla versione precedente.

9 Il sistema operativo

Il **sistema operativo** è un insieme di programmi che consente all'utente, o alle applicazioni informatiche, di accedere alle operazioni di base per utilizzare le risorse del sistema di elaborazione, risorse sia hardware che software.

I compiti generali di questi programmi sono:

- offrire un'interfaccia tra gli utenti e la macchina, mettendo a disposizione strumenti di lavoro, facilitazioni, piccole procedure pronte da utilizzare;
- gestire le risorse (hardware e software) al fine di ottimizzarne l'uso da parte degli utenti.

Le principali **funzioni del sistema operativo** possono essere schematizzate secondo questo elenco:

- gestione dell'unità centrale e del processo di elaborazione;
- inizializzazione e terminazione del lavoro della macchina;
- gestione della memoria centrale, nel senso dell'accesso alla memoria e di distribuzione della capacità di memoria in presenza di più utenti e di più lavori;
- gestione dei processi e dell'ordine con il quale vengono eseguiti;
- gestione dell'I/O (*input/output*), cioè l'uso ottimizzato delle periferiche collegate all'unità centrale, anche in presenza di più utenti che usano la stessa risorsa (per esempio una stampante molto costosa condivisa tra più utenti);
- gestione delle informazioni riguardanti i file registrati sulle memorie di massa e dell'accesso ai dati in essi contenuti;
- gestione delle protezioni degli archivi di dati e dei programmi da accessi non autorizzati;
- supporto all'utente-programmatore nella preparazione e nella messa a punto dei programmi.

Gli utenti finali utilizzano le prestazioni del sistema operativo attraverso i programmi applicativi. Gli utenti-programmatori realizzano i programmi applicativi attraverso i linguaggi di programmazione.

Comunemente il sistema operativo viene definito come l'interfaccia tra l'utente e l'elaboratore, nel senso che è costituito da un insieme di comandi e di programmi che consentono di rendere compatibile il software e i lavori dell'utente con l'hardware e le prestazioni della macchina, e nel contempo consente all'utente di attivare le funzioni e i programmi applicativi attraverso un'**interfaccia utente** che può essere di tipo testuale (per esempio nel *Prompt dei comandi*) o di tipo grafico (per esempio in Windows).

Prima dell'avvento dei personal computer ogni casa costruttrice installava sulle macchine sistemi operativi proprietari.

Lo sviluppo recente dei sistemi ha portato a sistemi operativi basati sull'interfaccia grafica, come il sistema **Windows** della Microsoft.

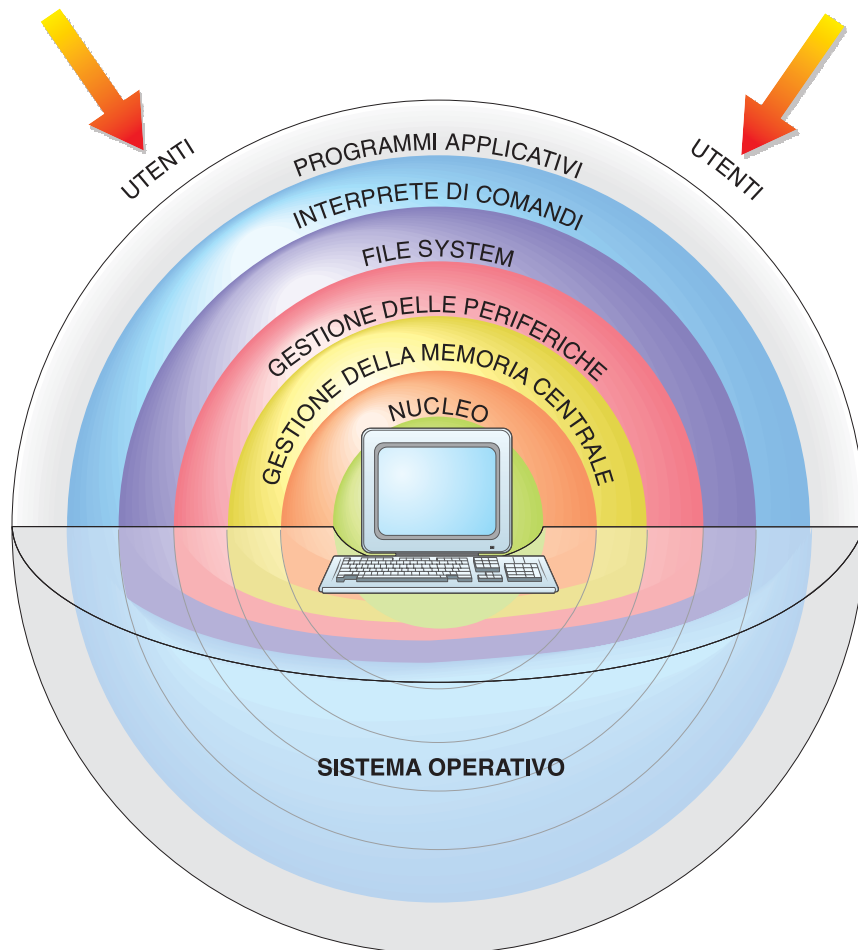
Il sistema operativo **Linux** viene utilizzato su personal computer, ma anche per gestire **server Internet**, ovvero per rendere disponibile l'accesso alle informazioni della rete Internet per gli utenti che lo richiedono. Funzionalità simili per la gestione dei server e delle reti sono disponibili anche nelle versioni *server* del sistema operativo **Windows**.

Altri sistemi operativi sono **Mac OS** per computer *Apple*, **iOS**, **Windows Phone** e **Android** per i netbook e i telefoni cellulari che includono funzionalità di *smartphone*.

10 La struttura modulare di un sistema operativo

In generale, un sistema operativo è organizzato a diversi livelli, ognuno con compiti specifici. Ogni livello vede quelli sottostanti come una macchina che svolge determinate funzioni.

Il centro è rappresentato dall'hardware, su cui agisce solamente il nucleo. Tale modello prende il nome di *struttura a buccia di cipolla*, che indica chiaramente che ogni livello sfrutta solo le funzioni di quello immediatamente sottostante; se un livello richiama funzioni di livelli più bassi di quello sottostante, si hanno complicazioni progettuali nel momento in cui si devono rivedere alcuni moduli.



In un'azienda le funzioni di un ruolo vengono controllate dalla persona di livello immediatamente superiore; il dirigente andrà a verificare eventuali problemi con i suoi collaboratori e non direttamente con i dipendenti, se non in casi eccezionali: così anche nei sistemi operativi, gli accessi ai livelli più bassi sono rari e dettati da ragioni di efficienza.

Il **nucleo**, chiamato anche **kernel** (nocciolo), è la parte del sistema operativo più vicina alla macchina, ed è strettamente dipendente dall'hardware. I microprogrammi che lo compongono sono chiamati **primitive** del nucleo, e sono scritte in linguaggio macchina, specifico di un determinato microprocessore.

Le funzioni del nucleo sono:

- creazione e terminazione dei processi;
- assegnazione della CPU ai diversi processi;
- sincronizzazione tra i processi;
- sincronizzazione dei processi con l'ambiente esterno.

Quando si richiede l'esecuzione di un programma al calcolatore, è necessario che questo provveda a mettere a disposizione del programma le risorse di cui ha bisogno.

Le risorse principali delle quali il programma necessita sono:

- il tempo di utilizzo del processore,
- la memoria.

Per quanto riguarda il primo, occorre dire che il computer deve essere in grado di garantire che ogni programma del quale si chiede l'esecuzione possa terminare e quindi, deve dedicare del tempo alla sua esecuzione, eventualmente dividendo il tempo complessivamente disponibile tra più programmi in esecuzione. Questo obiettivo può essere ottenuto in modi diversi, in particolare:

- assegnando tutto il tempo macchina disponibile ad un singolo programma finché questo termina e passando, poi, al successivo;
- assegnando un quanto di tempo a tutti i programmi in successione, durante la loro esecuzione.

Il primo criterio di assegnazione è tipico dei sistemi detti **uniprogrammati**, mentre il secondo caratterizza il funzionamento dei sistemi **multiprogrammati**.

Per quanto concerne la **gestione della memoria**, in quasi tutti i sistemi essa viene divisa in blocchi logici che vengono chiamati *pagine* o *segmenti* a seconda del modo in cui vengono costruiti ed usati.

Una **pagina** di memoria è un blocco di dimensione fissa, che viene assegnato tutto insieme ad un programma che ne faccia richiesta: in altre parole, quando un programma necessita di nuova memoria per la sua prosecuzione, ad esso viene assegnata una pagina intera indipendentemente da quanta ne servirebbe in effetti. Se un programma ha bisogno di spazio dove mettere pochi o molti dati non fa differenza: comunque gli verrà assegnata una nuova pagina intera.

Un **segmento**, viceversa, non ha una dimensione predeterminata, ma variabile a seconda delle richieste del programma: il sistema risponde alle richieste di memoria dei programmi fornendo a ciascuno solo quella necessaria.

Il vantaggio della gestione a pagine è soprattutto la semplicità: per tenere nota della situazione di ciascun programma attivo è sufficiente una tabella che metta in corrispondenza ciascuna pagina con il programma cui è assegnata.

Con la gestione a segmenti, la tabella si complica, perché i segmenti non sono messi in posizioni fisse della memoria, e quindi per ciascun segmento occorre tenere nota anche di dove si trova e di quanto spazio occupa, oltre che del programma cui è assegnato.

I vantaggi della gestione a segmenti rispetto a quella a pagine sono almeno due:

- non si spreca memoria perché a ciascun programma si assegna solo quella necessaria;
- si possono scrivere programmi già divisi in segmenti, in modo che non sia necessario ricorrere frequentemente alla lettura di segmenti nuovi.

Per capire questo secondo punto ricorriamo ad un esempio: una équipe di tecnici lavora ad un certo progetto, e ad un certo punto il capo progetto vuole sapere qual è la situazione. Se ogni tecnico cura un singolo aspetto del problema, ciascuno sarà in grado di esporre in una sola volta lo stato del proprio lavoro, viceversa, se ognuno sa qualcosa di ogni punto in argomento occorre che ciascuno prenda la parola molte volte per precisare e puntualizzare ciò che dice qualcun altro.

I segmenti vanno visti come tecnici specializzati: ogni segmento va costruito in modo che si possa occupare in modo il più possibile completo di un problema, con un minimo bisogno di interventi da parte di altri segmenti. In tal modo, quando il programma ne richiede l'intervento si può prevedere che esso non avrà bisogno di altri segmenti per un certo periodo di tempo. Viceversa, con la gestione a pagine questo non si verifica, perché il programma viene diviso in modo arbitrario e può succedere che le istruzioni contenute in una pagina facciano riferimento a dati contenuti in un'altra, e così diventa necessario continuare a cambiare le pagine assegnate.

In sintesi, la gestione a segmenti ha caratteristiche di maggiore efficienza dal punto di vista della frequenza con cui occorre aggiornare la tabella. Infatti ogni volta che il sistema deve intervenire per rispondere ad una richiesta sottrae tempo ai programmi, e questo può rallentare sensibilmente la loro esecuzione.

Il modulo di **gestione delle periferiche** consente l'utilizzo delle periferiche da parte degli utenti a livello superiore, lasciando al sistema operativo tutti gli aspetti di gestione riguardanti le loro caratteristiche fisiche.

Mette a disposizione le **periferiche virtuali**, gestendo in modo non visibile agli utenti i problemi di utilizzo concorrente, rendendo più efficace e ottimizzato l'uso delle periferiche reali con tecniche quali lo **SPOOL**: l'utente ha quindi l'impressione di avere a disposizione comunque la risorsa richiesta, anche se in realtà questo avviene solo a livello virtuale, in quanto i moduli del sistema operativo assegnano la stessa risorsa a più utenti in tempi diversi.

Nel sistema operativo monoutente (nei personal computer) lo schema è più semplice, perché essendoci un solo processo principale le risorse (per esempio, la memoria centrale) sono a disposizione di quell'unico processo. Funzioni di *spooling* vengono rese disponibili in vista dell'inserimento del sistema monoutente in ambienti di rete dove è necessario condividere alcune risorse.

Il **File System** è il modulo del sistema operativo che gestisce i dati relativamente alla loro organizzazione nelle memorie di massa.

Le sue funzioni principali sono:

- rendere trasparente all'utente l'utilizzo degli archivi su disco, nascondendo i problemi relativi alla memorizzazione fisica e creando una struttura logica;
- consentire l'accesso ai file in lettura e scrittura, risolvendo i problemi di concorrenza che potrebbero verificarsi in regime di multiprogrammazione;
- predisporre funzioni di utilità quali la lista dei file, la loro cancellazione, duplicazione, disposizione nella struttura logica;
- proteggere le informazioni sia per quanto riguarda la loro integrità che la loro riservatezza.

I file sono identificati in modo logico tramite un **nome** che ne ricorda il contenuto e un'**estensione** che ne ricorda il tipo (dati, testo, eseguibile, immagine, ecc.)

Il File System organizza i file nelle memorie di massa con le **directory**.

Quando un disco nuovo viene preparato per essere utilizzato dal computer, attraverso l'operazione detta di **formattazione** del disco, il sistema operativo crea sul disco stesso una directory vuota, detta **root** (*directory radice*).

All'interno della root l'utente poi può creare altre directory, che a loro volta possono contenere altre **sottodirectory** (o *subdirectory*), secondo una struttura organizzata in modo gerarchico detta ad *albero*, nella quale la radice è costituita dalla directory root, i nodi sono le sottodirectory e le foglie sono i file.

Con questa struttura gerarchica è molto comodo accedere a un qualsiasi file contenuto in una sottodirectory diversa dalla directory corrente, indicando il cammino all'interno dell'albero: si costruisce in questo modo un elenco di nomi delle sottodirectory (nel sistema operativo Windows, separati dal segno \) che devono essere attraversate per ritrovare il file richiesto.



Per esempio, per accedere al file MOVIM.DAT contenuto nella sottodirectory CONTAB, che a sua volta è contenuta nella directory UTENTI, che è stata creata all'interno della root, si deve indicare come nome del file la seguente scrittura:

\UTENTI\CONTAB\MOVIM.DAT

Questa scrittura prende il nome di **pathname** (*nome con indicazione del cammino*). Il primo segno di \ indica la root.



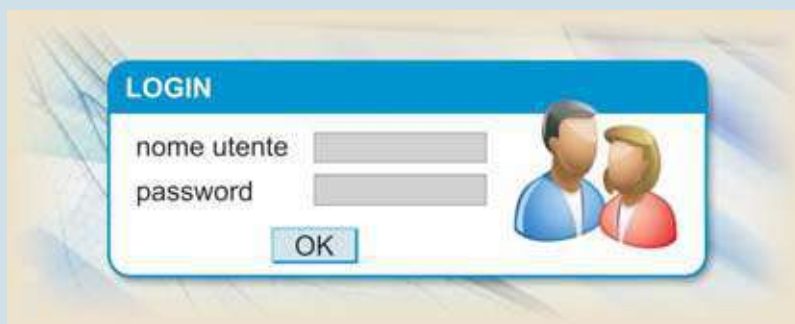
LA SICUREZZA

La gestione di un sistema di elaborazione ha un aspetto cruciale rappresentato dalla necessità di garantire la **consistenza** dei dati in esso memorizzati, sia nel caso di un computer personale, sia nel caso di sistemi informatici aziendali: i dati devono essere significativi ed essere effettivamente utilizzabili nelle applicazioni. I dati devono, quindi, essere protetti per impedire perdite accidentali dovute a improvvise interruzioni dell'attività del sistema, guasti hardware o interventi dannosi da parte di utenti o programmi: la protezione deve riguardare anche gli interventi dolosi sui dati dovuti ad accessi non autorizzati con operazioni di modifica o di cancellazione.

In sostanza **sicurezza** significa impedire che il sistema e i suoi dati vengano danneggiati da interventi accidentali o non autorizzati; **integrità** significa garantire che le operazioni effettuate sul sistema e sui dati da parte di utenti autorizzati non provochino una perdita di consistenza ai dati.

Gli archivi di dati possono essere usati da più applicazioni: perciò il progettista di un'applicazione deve preoccuparsi di definire chi e come può accedere ai dati, ma in un ambiente che comprende tutti i dati aziendali occorre avere una responsabilità centralizzata in grado di distribuire le autorizzazioni e le modalità di accesso.

In generale ad ogni utente viene associato un **profilo**, il cui accesso è controllato mediante l'**identificativo dell'utente** (*user ID* o *username*) e la **parola d'ordine** (*password*): il primo rappresenta il nome, che può anche essere pubblico, con il quale l'utente accede alla procedura di riconoscimento (*login*), mentre la seconda rappresenta la parola chiave, di solito scelta dall'utente stesso, che è privata e riservata. Per questo motivo la password dovrebbe essere non banale (nomi di persona, date di nascita, ecc.) e dovrebbe essere frequentemente cambiata.



Al profilo sono poi associati i **diritti di accesso**, cioè le azioni, i permessi e i divieti relativi alle attività svolte dall'utente: quali programmi può utilizzare, quali archivi può consultare, quali permessi su un singolo archivio (solo lettura oppure lettura e scrittura), la possibilità di installare nuovi programmi.

Questi controlli vengono realizzati a diversi livelli: gestione degli archivi di dati, sistemi operativi, reti di utenti. Essi sono gestiti al livello di **amministratore del sistema** (*system administrator*), che nel personal computer coincide con l'utente stesso, mentre per sistemi più complessi o per le reti è una specifica figura professionale del settore informatico. Gli archivi elettronici hanno più o meno gli stessi problemi degli archivi cartacei, per esempio un'unità a dischi si può rompere provocando la perdita di tutti i dati in essa contenuti, oppure un errore degli operatori può provocare la cancellazione non voluta di uno o più archivi: pertanto si rende necessaria la copia periodica degli archivi e la conservazione delle copie in posti sicuri.

L'attività di copia di archivi grandi ha durate significative e di solito viene eseguita giornalmente: questa attività si chiama **backup** (salvataggio). L'attività opposta si chiama **restore** (ripristino) e serve per ricaricare su disco tutti o parte degli archivi salvati precedentemente.



Nelle situazioni dove non è consentita l'interruzione dell'attività di un sistema di elaborazione (*downtime*), come per esempio in una banca o in un supermercato, la sicurezza viene garantita anche nella parte hardware attraverso la duplicazione di parti o dell'intero sistema: queste tecniche vengono indicate con il termine **fault tolerance** (tolleranza del guasto) e riguardano principalmente le memorie di massa nei server delle reti e nei sistemi di medie e grandi dimensioni.

Di seguito vengono esaminate le **cause** che possono incidere negativamente, e talvolta in modo irreparabile, sulla sicurezza del sistema informatico di un'azienda nel suo complesso.

Errori del personale

L'enorme sviluppo dell'elettronica moderna consente di accumulare grandi quantità di dati in spazi molto piccoli e secondo modalità che non sono quasi mai simili alle operazioni svolte secondo la gestione manuale, per cui anche errori banali possono produrre disastri. Le norme per prevenire tali eventualità di solito consistono in:

- una netta separazione tra l'ambiente di esercizio e l'ambiente di sviluppo del software, per evitare che il programmatore durante le sue prove inquina o peggio distrugga i dati che servono per le procedure aziendali;
- copia periodica di tutti i dati aziendali su supporti di backup da conservare in luogo sicuro per un successivo ripristino se richiesto: normalmente vengono conservati più salvataggi di giornate differenti al fine di una maggiore protezione;
- intervento di livelli professionali differenti in caso di ripetizione dell'errore in modo da evitare qualsiasi rischio;
- definizione di regole organizzative e figure professionali preposte a verificare tutte le anomalie nei dati di input o di output.

Pirata informatico

Il termine *pirata informatico* sta ad indicare una persona che si collega, con sistemi di elaborazione, sfruttando i meccanismi di rete senza avere l'autorizzazione all'accesso. Questi inserimenti indesiderati con messaggi o programmi possono anche portare alla paralisi del sistema.

Anche in questo caso la difesa è costituita dal controllo degli accessi, tenendo conto del fatto però che l'elemento più fragile è rappresentato dall'uso di password banali e facilmente prevedibili.

Per questo è opportuno integrare i sistemi di controllo degli accessi con l'uso di tessere magnetiche di riconoscimento.

Virus

I *virus* sono programmi chiamati così perché sono stati costruiti con lo scopo di:

- "infettare" un programma senza alterarne il funzionamento
- rendersi visibili in qualche modo
- autoriprodursi nell'ambiente in cui sono inseriti creando un'"epidemia" che può essere dannosa (distruzione di dati e programmi) o solo contagiosa estendendosi a tutti i programmi e a tutti i computer.

Possono essere prodotti per gioco oppure talvolta con finalità ricattatorie.

Non esiste un modo attraverso il quale distinguere un programma da un virus e quindi un virus diventa tale solo dopo la sua scoperta e la sua denominazione.

La protezione contro i virus può essere realizzata attraverso il controllo sistematico di tutti i supporti che vengono usati e attraverso il controllo degli accessi della rete.

Esistono poi organizzazioni che si sono specializzate nelle identificazioni dei virus e degli effetti che producono, e nella costruzione di programmi antivirus (*vaccino*) per eliminarli. In alcuni casi la rimozione del virus può richiedere la cancellazione dei programmi danneggiati: pertanto diventa fondamentale il lavoro di filtro sugli accessi, sui dati e sul software, con scopi di prevenzione.

I principali tipi di virus per computer sono:

- Virus che provocano *infezione sui file*, in particolare sui file eseguibili .COM o .EXE e che vengono caricati nella memoria del computer ogni volta che il programma viene eseguito. Sono tipi di virus che si solito arrivano sul computer attaccati ad altri programmi o come allegati a messaggi di posta elettronica.
- Virus che attaccano il sistema e in particolare il *boot-record* del disco, cioè il blocco del disco che contiene le informazioni sull'avvio del computer.
- Virus per le *Macro*, che sono comandi simbolici costruiti dall'utente all'interno dei programmi Office (Word, Excel, Access) e che consentono di eseguire una sequenza di operazioni elementari frequentemente utilizzate. L'infezione di questo tipo di virus provoca malfunzionamenti nei programmi Office.
- Cavalli di Troia (*Trojan*): sono programmi all'apparenza normali che possono però distruggere i file sul disco oppure l'intero disco. A differenza dei virus precedenti, non replicano se stessi su altri computer.
- I virus *Worm* sono programmi autoreplicanti che hanno come caratteristica principale la capacità di diffondersi rapidamente sulla rete di computer e in Internet.

Regole pratiche per prevenire l'infezione dei virus per computer

1. Installare e usare un buon programma antivirus.
2. Aggiornare frequentemente le definizioni dei virus, scaricando gli aggiornamenti dal sito Internet del fornitore del software antivirus.
3. Non aprire messaggi di posta elettronica di dubbia provenienza.
4. Non aprire allegati contenuti all'interno di messaggi di posta elettronica con mittente sconosciuto o non sicuro.
5. Fare la scansione dei supporti di memoria provenienti da fonte insicura prima di usare i file e i programmi in essi contenuti.
6. Non scaricare musiche, file multimediali o filmati da un sito Internet che non offra garanzie di sicurezza.
7. Dopo aver scaricato un programma da Internet, effettuare il controllo antivirus prima di installarlo ed eseguirlo.
8. Operare con cautela la condivisione di file e musiche con altri utenti della rete; durante la connessione con condivisione di cartelle sul disco, il computer è aperto all'ingresso non autorizzato e all'attacco da parte di utenti esterni.
9. Effettuare frequentemente copie di sicurezza (su altri dischi o chiavi USB) dei documenti e file che riteniamo importanti per il nostro lavoro.

Furti di apparecchiature mobili

Un aspetto particolare della sicurezza riguarda il rischio per l'azienda derivante dal furto di computer portatili o telefoni cellulari. Queste apparecchiature possono essere facilmente asportate in modo furtivo e possono contenere dati, archivi, contatti, password e numeri telefonici riservati o confidenziali.

L'uso fraudolento di queste informazioni può provocare gravi danni a una singola persona o a un'azienda, ben più gravi del valore commerciale dell'apparecchiatura rubata.

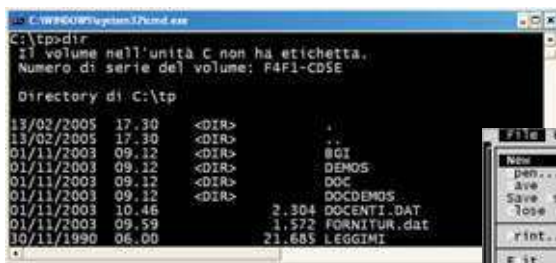
11 L'interprete dei comandi

Le prestazioni del sistema operativo vengono attivate dall'utente o dalle applicazioni con modalità differenziate a seconda del grado di sviluppo del sistema operativo stesso, in particolare del livello di **interprete dei comandi**, ossia quel livello direttamente a contatto con l'utente finale. Una possibilità è quella di fornire da tastiera il comando, con i parametri necessari, corrispondente alla funzione richiesta. Il comando viene letto dall'interprete di comandi del sistema operativo, che controlla la correttezza sintattica, e in caso positivo ne attiva la funzione. Se il sistema operativo ha un'interfaccia grafica, le varie funzioni e i vari comandi possono essere attivati attraverso la selezione di menu a tendina o attraverso l'uso di apposite icone.

L'interprete di comandi viene spesso indicato con il termine inglese **shell**.

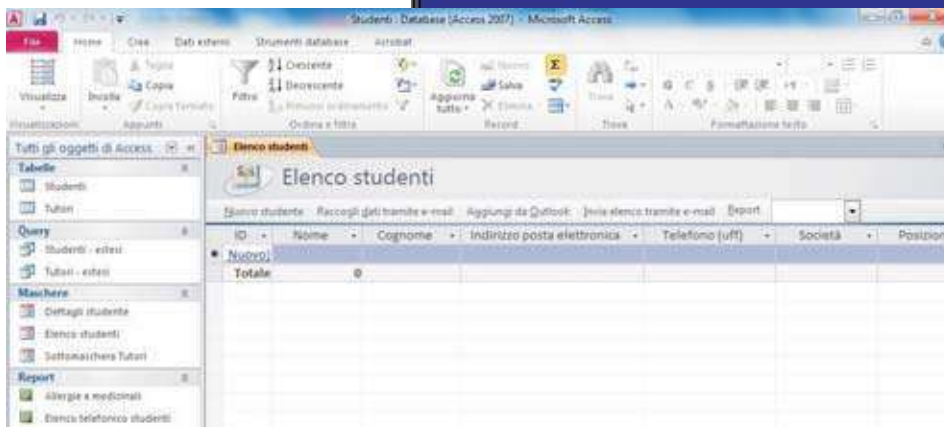
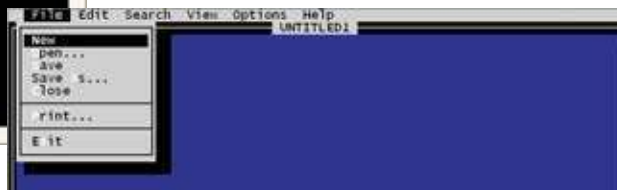
L'interprete di comandi consente all'utente di specificare le funzioni richieste attraverso un'**interfaccia utente**, che ha subito negli anni una continua evoluzione per rendere sempre più amichevole l'utilizzo delle risorse di un sistema di elaborazione soprattutto da parte di utenti non specialisti:

- **a linea di comando**: l'utente scrive sulla tastiera il comando e il video svolge la funzione di eco alla battitura; è di questo tipo l'interfaccia del *Prompt dei comandi* di Windows.
- **a menu**: è presente in una parte dello schermo un elenco di comandi, azionabili con la pressione di un tasto, per mezzo dei quali si può accedere a funzioni o a sottomenu.
- **grafica** (*finestre, icone, mouse*): le funzioni di sistema operativo, il software di base, i file, i programmi applicativi vengono rappresentati graficamente sullo schermo nella forma di icone autoesplicative, che vengono selezionate da un puntatore comandato da un mouse. Gli oggetti citati (finestre, icone, mouse) costituiscono gli strumenti di base della cosiddetta **interfaccia grafica** o **GUI** (*Graphical User Interface*), che viene utilizzata soprattutto nei sistemi operativi per personal computer.



Interfaccia a linea di comando

Interfaccia a menu



Interfaccia grafica

I moderni sistemi operativi offrono nuove possibilità per migliorare l'accessibilità del computer. In generale, l'**accessibilità** è la disponibilità di dispositivi e programmi che consentono l'uso del computer e delle sue risorse, oltre che la connessione alle reti e la fruizione dei servizi nel Web, anche per persone con disabilità fisiche (permanenti o temporanee).

Il software per il **riconoscimento vocale** consente agli utenti di inserire informazioni e controllare il computer tramite la voce, quando sono presenti difficoltà nella scrittura o nell'utilizzo delle mani. In questi casi può essere vantaggioso anche l'uso della **tastiera riprodotta sullo schermo**.

I programmi di **screen reader** (lettura dello schermo) producono la lettura vocale del contenuto del video (testo o pagine Web) per persone non vedenti o ipovedenti.

Per le difficoltà visive sono disponibili anche le funzionalità per ingrandire il formato di immagini e caratteri (**zoom**).

12 I linguaggi di programmazione

Il linguaggio di programmazione è il linguaggio che viene utilizzato per tradurre il procedimento risolutivo di un problema (**algoritmo**) in una forma comprensibile all'elaboratore. Il risultato di questa traduzione si chiama **programma**.

I linguaggi di programmazione in genere sono orientati a specifiche applicazioni e a classi di problemi (scientifici o commerciali); ci sono poi i linguaggi adatti per tutti i problemi (*general purpose*).

I moderni linguaggi di programmazione sono di tipo **evoluto** (o di alto livello), nel senso che utilizzano termini del linguaggio naturale, sono più vicini al problema e facilitano il lavoro del programmatore: sono i linguaggi che più si adattano a tradurre in modo facile un procedimento ben organizzato e dettagliato.

I nomi dei linguaggi meno recenti sono:

FORTRAN (1956): *FORmula TRANslation*, nato soprattutto per applicazioni tecnico-scientifiche;

COBOL (1960): *COmmon Business Oriented Language*, per applicazioni di tipo commerciale e gestionale;

BASIC (1964): *Beginners All-purposes Symbolic Instruction Code*, linguaggio facile, molto interattivo, per applicazioni generali; si è sviluppato insieme al diffondersi dei piccoli sistemi di elaborazione;

RPG (1966): *Report Program Generator*, per applicazioni di tipo commerciale, e in particolare per la preparazione di prospetti.

I linguaggi di programmazione più recenti sono:

PASCAL (1971): orientato ai principi della programmazione strutturata;

C e **C++** (1974): linguaggio utilizzato nell'ambito dello sviluppo dei sistemi operativi e del software di base;

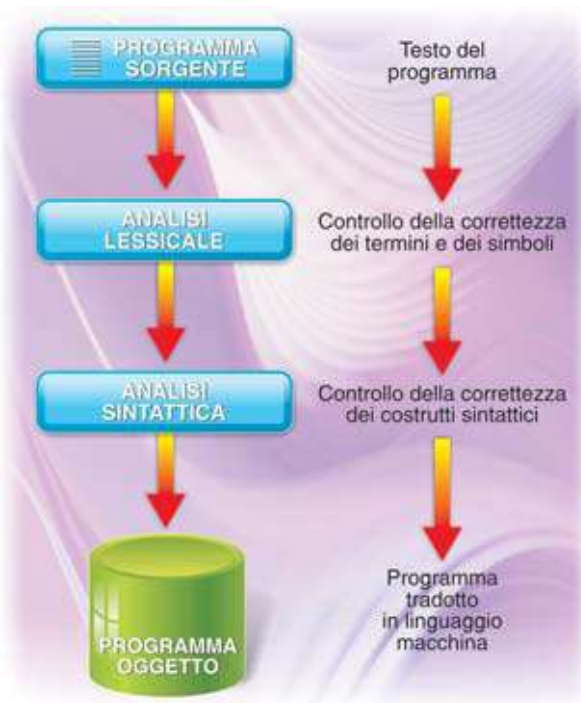
JAVA (1995): linguaggio utilizzato per realizzare applicazioni che possono funzionare in modo interattivo sulla rete Internet;

VISUAL BASIC (1994): linguaggio particolarmente adatto per creare in modo semplice applicazioni con interfaccia grafica per l'ambiente Windows.

Il testo di un programma scritto usando un linguaggio di programmazione, detto programma **sorgente** (in inglese *source*), deve essere tradotto in **linguaggio macchina**, praticamente nel linguaggio formato dalle cifre binarie 0 e 1, per poter essere eseguito dal computer.

Questa traduzione viene effettuata da un programma traduttore che produce una sequenza di istruzioni eseguibili dall'elaboratore. Due sono le modalità di base per eseguire questa traduzione:

- **interpretazione**, il programma che prende il nome di *interprete* considera il testo sorgente istruzione per istruzione e lo traduce mentre lo esegue; su questo principio lavorano linguaggi quali il Basic o alcuni linguaggi per la gestione di basi di dati;
- **compilazione**, il programma traduttore si chiama *compilatore* e trasforma l'intero programma sorgente in linguaggio macchina, memorizzando in un file il risultato del proprio lavoro. Così un programma compilato una sola volta può essere eseguito, senza bisogno di altri interventi, quante volte si vuole. Il risultato della compilazione si chiama programma **oggetto** (in inglese *object*).



La compilazione consente di avere programmi detti **eseguibili**, che possono essere utilizzati senza possedere il compilatore sul sistema dove il programma deve funzionare, mentre con l'interpretazione serve l'insieme di routine che consentono di tradurre le righe del programma sorgente (di norma fornite in uno o più file). L'interpretazione non richiede tempi di compilazione, ma, quando il programma è eseguito, ha tempi di traduzione che lo rendono più lento di un programma eseguibile. Per questi motivi l'interpretazione è comoda quando il programma è in fase di progettazione, mentre la compilazione è assolutamente preferibile per un programma applicativo destinato all'utente finale.

La compilazione deve essere seguita da un'ulteriore operazione detta **linking** (collegamento), che viene svolta da un programma apposito chiamato **linker** (collegatore). Tale operazione consiste nell'aggiungere al programma compilato i moduli del compilatore che realizzano le funzioni richieste dai vari comandi, e contemporaneamente di risolvere i riferimenti a celle di memoria o a variabili.

Alla fine di questo lavoro si ottiene il programma **eseguibile**.

Il compilatore genera il proprio output a condizione che l'input, cioè il programma sorgente, sia *formalmente* corretto, nel senso che deve rispettare le regole del linguaggio scelto. Ogni volta che questo non si verifica, il compilatore emette un messaggio che segnala l'errore.

Gli errori possono riguardare l'uso di termini non appartenenti al linguaggio (**errori di tipo lessicale**) oppure la costruzione di frasi non corrette dal punto di vista delle regole grammaticali del linguaggio (**errori di tipo sintattico**).

Naturalmente il compilatore non è in grado di rilevare **errori logici**, cioè riguardanti la correttezza dell'algoritmo: se un programma traduce un algoritmo sbagliato ma è scritto nel rispetto delle regole del linguaggio, il compilatore non può accorgersi dell'errore.

Così come con la compilazione non si possono rilevare situazioni di errore che possono verificarsi durante l'esecuzione del programma (**runtime errors**), sulla base di particolari valori assunti dai dati durante l'elaborazione: si pensi per esempio alla divisione per un numero che assume durante l'esecuzione il valore 0.

Gli interpreti trasformano il programma sorgente in linguaggio macchina un'istruzione per volta facendola eseguire subito dopo.

Eventuali errori formali (lessicali o sintattici) vengono rilevati e segnalati solo quando l'istruzione errata viene tradotta e causano l'interruzione dell'esecuzione.

Un particolare programma di utilità durante il lavoro di programmazione è il **debugger** (letteralmente significa *spulciatore*), che serve per individuare eventuali errori logici nel programma. Esso può per esempio consentire di:

- eseguire il programma da correggere un'istruzione per volta, così da verificarne la corretta evoluzione;
- controllare i valori assunti dalle variabili in certi punti dell'esecuzione.

I moderni linguaggi di programmazione sono **linguaggi ad oggetti** che consentono al programmatore di definire degli oggetti software con le loro proprietà e funzionalità, ottenendo due vantaggi fondamentali:

- il programma è più leggibile e comprensibile rendendo più facile la manutenzione del software nel tempo;
- il programmatore può riutilizzare oggetti propri o creati da altri all'interno dei programmi software.

13 Il software applicativo

I programmi applicativi sono quelli realizzati dal *programmatore* del sistema di elaborazione utilizzando le prestazioni del sistema operativo, i programmi di utilità e i linguaggi di programmazione.

Tra essi si distinguono:

- i programmi di applicazione del computer a problemi di tipo **gestionale** (contabilità aziendale, fatturazione, magazzino, stipendi); questi programmi vengono di solito forniti all'utente finale da aziende specializzate nella produzione di programmi, dette **software house**;
- gli strumenti software (*tools*), detti anche pacchetti o prodotti **office**, perché vengono forniti già pronti per l'uso all'utente finale e sono orientati alla produttività nella gestione dei più comuni lavori di ufficio:
- **Elaborazione di testi** (*word processing*) per scrivere lettere e rapporti usando il computer.
- **Editoria elettronica** (DTP, *DeskTop Publishing*) per la produzione all'interno dell'azienda di manuali tecnici, notiziari, listini.
- **Gestori di archivi elettronici** (*database*) per creare e gestire con il computer gli schedari presenti nell'ufficio, quali rubriche personali o di gruppo, i listini prezzi, gli indirizzari.
- **Fogli elettronici** (*spreadsheet*) per usare tabelle di calcolo nelle quali inserire dati e formule di ricalcolo, con eventuali rappresentazioni attraverso grafici statistici.
- **L'agenda personale** per pianificare il tempo e gli impegni; le diverse agende personali possono essere integrate nell'**agenda di gruppo**, per trovare le date possibili per una riunione oppure per fissare la prenotazione di una risorsa comune.
- La **gestione del progetto** (*project management*) per definire il piano per raggiungere l'obiettivo finale del progetto e la definizione di tutte le risorse necessarie per completare le singole attività.
- Il programma per la **posta elettronica** (*e-mail*) con l'invio di messaggi attraverso la rete aziendale o la rete Internet.
- Il programma **browser** per consultare le pagine Web dei siti Internet.

Sotto la denominazione di **software applicativo** rientrano poi tutti i programmi che vengono costruiti per utilizzare il computer nelle diverse attività delle aziende, della Pubblica Amministrazione, degli studi professionali e degli uffici, nella ricerca, nel controllo della produzione industriale, nella progettazione.

Sono programmi applicativi anche:

- i software per la grafica e l'**elaborazione delle immagini** che permettono di modificare le dimensioni, il formato, la risoluzione delle immagini e delle fotografie, oltre a fare operazioni di ritaglio e fotoritocco.
- i **videogiochi**, software per l'intrattenimento che riproducono sul computer giochi di ruolo, di avventura o di azione.



LO SVILUPPO DEL SOFTWARE

La costruzione del software si ottiene tramite l'esecuzione di diverse attività; ognuna raggiunge obiettivi ben precisi. La metodologia di **sviluppo del software** identifica queste attività e le organizza per ottimizzare il lavoro. Possiamo suddividere la metodologia in sezioni che chiamiamo **fasi** o passi. Ogni fase è costituita dalle attività da svolgere, dai risultati da raggiungere e dai controlli da eseguire.

Di seguito vengono descritte le diverse fasi della metodologia per lo sviluppo di progetti informatici.

a) La conoscenza degli obiettivi

Questa fase è anche chiamata **analisi preliminare**. Per ottenere la conoscenza necessaria è indispensabile poter disporre di esperti della materia ai quali fare l'**intervista conoscitiva**.

b) L'analisi

La fase di **analisi** ha l'obiettivo di determinare e di descrivere dettagliatamente tutte le componenti del progetto. È la fase di maggiore importanza: il risultato di questo lavoro diventa la base e la guida di tutto il progetto.

In un progetto informatico una componente fondamentale è costituita dai **dati**. L'attività di definizione dei dati consiste nel decidere quali siano le informazioni che necessitano al progetto affinché esso funzioni.

L'altra componente fondamentale di un progetto informatico è costituita dalle **funzioni**. L'attività di definizione delle funzioni consiste nel decidere quali siano le cose che il sistema dovrà fare. I **dati** del progetto saranno memorizzati in archivi elettronici, le **funzioni** del progetto diventeranno i programmi.

La terza componente della fase di analisi è costituita dal **flusso dei dati tra le funzioni**. Determinare il flusso dei dati tra le funzioni significa abbinare ad ogni funzione quali dati utilizza in input e quali dati produce in output.

c) La progettazione e la transizione

L'obiettivo della fase di **progettazione** è definire e descrivere dettagliatamente le caratteristiche del sistema informatico del progetto.

L'obiettivo della fase di **transizione** è quello di definire le modalità di passaggio dal sistema esistente al sistema del nuovo progetto.

d) La realizzazione

La fase della **realizzazione** genera il prodotto del progetto. Dopo lo studio, l'ideazione e la concretizzazione delle scelte di progetto, si arriva alla sua realizzazione. Trattandosi di un progetto informatico il prodotto è costituito da moduli software (chiamati comunemente *programmi*) e dagli archivi elettronici utilizzati dai programmi stessi.

e) La documentazione, le prove e la formazione

L'obiettivo della fase di **documentazione** è fornire a tutti gli utenti del nuovo sistema i documenti da consultare nel corso della loro attività lavorativa (manuale per l'utente).

La finalità specifica della fase di prove o **testing** è quella di verificare il grado di aderenza del progetto alle necessità indicate dall'utente-committente e di rimuovere eventuali errori presenti nel codice software.

Il risultato finale delle prove consiste nella **certificazione del progetto**: significa che l'utente ha trovato soddisfazione nel sistema di prove e questa situazione permette il passaggio alla fase successiva di produzione.

Obiettivo della fase di **formazione** è quello di mettere in condizione tutti gli utenti del nuovo sistema di operare conoscendo le caratteristiche del progetto, attraverso l'erogazione di corsi di formazione rivolti agli utenti del sistema.

f) La produzione

L'ultima fase prevista dalla metodologia è la **produzione**, che si può anche chiamare **rilascio del progetto**. L'obiettivo di questa fase è attivare il progetto presso gli utenti, controllare che i processi elaborativi producano i risultati attesi e verificare globalmente il sistema.

Per quanto riguarda le caratteristiche dei programmi moderni, si deve osservare che il software, soprattutto quello gestionale, è diventato più complesso, perché si sono estese le applicazioni alle diverse funzioni aziendali, e perché deve tener conto di tutti i collegamenti con altri programmi, nella logica di una gestione integrata dell'azienda.

Inoltre le risorse informatiche vengono oggi utilizzate da tutto il personale dell'azienda, e non solo da specialisti dell'informatica: il lavoro deve essere reso semplice per l'utente finale, con molta **interattività** tra utente e computer, con il controllo sulle attività dell'utente, la disponibilità di messaggi e pagine di aiuto (**help**), la facilità nell'utilizzo.

Quindi la produzione di programmi applicativi contiene al suo interno una grande parte dedicata alla realizzazione di **interfacce** che consentono all'utente di usare con facilità il computer, ponendosi sempre più lontano dalle caratteristiche fisiche della macchina.

Le nuove procedure applicative devono avere alcune caratteristiche, che sono importanti nelle imprese moderne:

- la possibilità di scambiare dati con le procedure già esistenti;
- la necessità di stabilire collegamenti tra computer di tipo diverso, posti anche a distanza;
- l'organizzazione dei dati in modo integrato per offrire alle funzioni aziendali una base di dati sempre aggiornata e consistente.

I programmi software, inoltre, non riguardano più soltanto le tradizionali procedure amministrative e gestionali di un'azienda (contabilità, magazzino, personale, ecc.), ma si orientano verso altri settori, quali la multimedialità, i giochi, il software per la didattica, l'interazione tramite Internet, il commercio elettronico sulla rete.

AUTOVERIFICA

Domande da 12 a 16 pag. 82-83

Problemi da 11 a 12 pag. 85

**MATERIALI ONLINE****3. Il software open source****4. Traccia riassuntiva sui sistemi di elaborazione (hardware e software)**

DOMANDE

Automi

- 1 Quali sono gli elementi che definiscono un automa in modo formale?
 - a) algoritmo
 - b) insieme degli stati
 - c) simboli di input
 - d) simboli di output
 - e) simboli di stato
 - f) funzione degli ingressi
 - g) funzione delle uscite
 - h) funzione degli stati successivi
- 2 Che cosa indica la funzione degli stati successivi?
 - a) Lo stato assunto da un automa.
 - b) Il simbolo emesso verso l'esterno.
 - c) Il numero degli stati possibili.
 - d) L'insieme degli stati possibili.
- 3 Che cosa indica funzione delle uscite?
 - a) Lo stato assunto da un automa.
 - b) Il simbolo emesso verso l'esterno.
 - c) Il numero dei simboli di output.
 - d) L'insieme dei simboli di output.
- 4 Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
 - a) La matrice di transizione corrisponde alla tabella degli stati successivi V F
 - b) Un automa si dice discreto perché funziona secondo una successione discreta di eventi V F
 - c) Un automa si dice a stati finiti perché il suo funzionamento termina dopo un numero finito di passi V F
 - d) Un automa produce un cambiamento di stato dopo aver accettato un simbolo di input e dopo aver emesso un simbolo di output V F

Hardware

- 5 Quale delle seguenti definizioni corrisponde al termine *architettura* riferito ad un computer?
 - a) La disposizione delle unità che compongono un sistema di elaborazione.
 - b) Il modello utilizzato per la progettazione di un sistema di elaborazione.
 - c) Le parti hardware di un sistema di elaborazione.
 - d) Le parti software di un sistema di elaborazione.
- 6 Quale termine indica l'insieme dei dispositivi fisici che permettono il funzionamento del computer?
 - a) Hardware
 - b) Memoria ROM
 - c) Sistema operativo
 - d) Mainframe
- 7 Qual è il significato dell'acronimo CPU?
 - a) Control Programming Unit
 - b) Computer Program Unit
 - c) Calculator Process Unit
 - d) Central Processing Unit

- 8 Quale delle seguenti unità di misura è utilizzata per la velocità del processore?
 - a) Mbyte
 - b) Mbps
 - c) Kbyte
 - d) GHz
- 9 Quale delle seguenti frasi riferite alla memoria RAM (*Random Access Memory*) è corretta?
 - a) È la principale area di memoria utilizzata da un programma in esecuzione.
 - b) Era la memoria principale dei vecchi computer, sostituita ora dalla ROM.
 - c) Mantiene il proprio contenuto anche dopo lo spegnimento del computer.
 - d) È una memoria di massa che può essere inserita ed estratta dal computer.
- 10 Dove si trovano i programmi in esecuzione?
 - a) Nella memoria RAM.
 - b) Nella memoria ROM.
 - c) Nella CPU.
 - d) Nella memoria cache.
- 11 Quale delle seguenti definizioni corrisponde al termine *memoria di massa*?
 - a) La memoria di lavoro della CPU.
 - b) La riproduzione su carta dei dati di output.
 - c) Un supporto di dati e di programmi dell'utente di un computer.
 - d) L'insieme della memoria RAM e della memoria ROM.

Software

- 12 Quale termine indica l'insieme dei programmi di base che permettono il funzionamento del computer?
 - a) Hardware
 - b) Memoria ROM
 - c) Sistema operativo
 - d) Mainframe
- 13 Tenendo presente il tipo di organizzazione di un *sistema operativo*, assegna a ciascun modulo della colonna di sinistra una o più funzioni tra quelle elencate a destra.

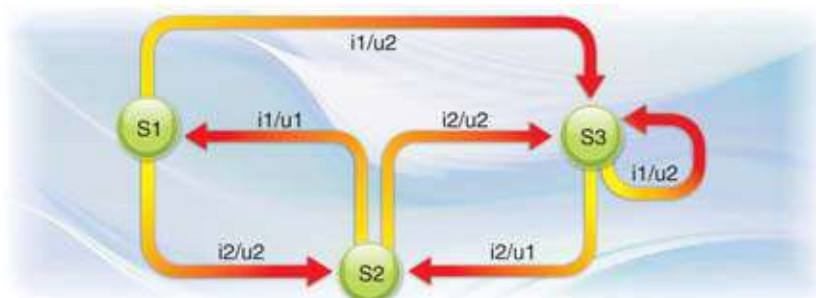
a) Nucleo	1) ottimizzazione dei tempi di accesso a un disco
	2) gestione dell'organizzazione di un disco
b) File System	3) gestione delle pagine o dei segmenti di memoria
c) Modulo di gestione delle periferiche	4) terminazione dei processi
	5) assegnazione della CPU ai diversi processi
d) Interprete dei comandi	6) istruzioni per la formattazione di un disco
	7) assegnazione dei diritti di accesso ad un file
e) Modulo di gestione della memoria	8) controllo della correttezza sintattica di una linea di comando
	9) sincronizzazione dei processi
- 14 Quali tra i seguenti software sono sistemi operativi?
 - a) Windows
 - b) Excel
 - c) Mac OS
 - d) Office

- 15 Qual è il principale vantaggio offerto dalla GUI (*Graphical User Interface*) agli utenti del computer?
- Consentire l'uso di stampanti grafiche a colori.
 - Rendere più veloce l'esecuzione dei programmi.
 - Rendere più facile l'uso dei file su disco.
 - Facilitare l'utilizzo delle applicazioni con icone e mouse.
- 16 Quale tra i seguenti programmi applicativi risulta più adatto per gestire una tabella di calcolo?
- Database
 - Elaborazione testi
 - Foglio elettronico
 - Programma di presentazione

PROBLEMI

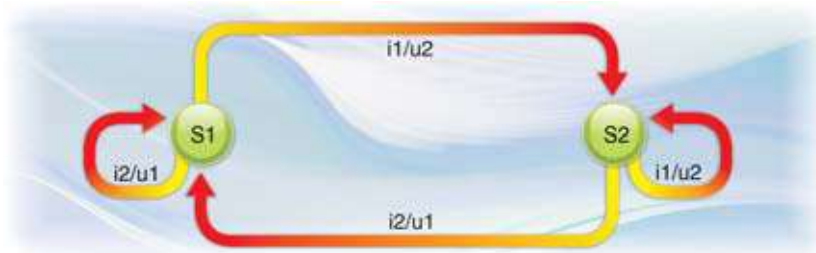
Automi

- 1 Dato il seguente grafo che descrive il funzionamento di un automa



costruire la tabella degli stati successivi e la tabella delle uscite.

- 2 Dato il seguente grafo che descrive il funzionamento di un automa



costruire la corrispondente matrice di transizione.

- 3 Rappresentare con un grafo il funzionamento di un automa, del quale vengono fornite la tabella degli stati successivi e la tabella delle uscite:

stati input	s_1	s_2	s_3
i_1	s_3	s_3	s_3
i_2	s_2	s_1	s_2

stati input	s_1	s_2	s_3
i_1	u_2	u_2	u_1
i_2	u_3	u_1	u_2

- 4 Rappresentare con un grafo il funzionamento di un automa, del quale viene fornita la matrice di transizione:

input \ stati	s ₁	s ₂
i ₁	u ₁ , s ₂	u ₁ , s ₁
i ₂	u ₂ , s ₁	u ₂ , s ₂
i ₃	u ₂ , s ₂	u ₂ , s ₁

Definire in modo formale i seguenti automi, descrivendone il funzionamento con un grafo e con le tabelle degli stati successivi e delle uscite.

- 5 Un distributore automatico di biglietti al casello d'ingresso dell'autostrada è normalmente predisposto per l'emissione, con il semaforo rosso acceso. Quando viene premuto il pulsante, la macchina emette un biglietto e accende il semaforo verde. Dopo il transito del veicolo, viene acceso il semaforo rosso e la macchina è pronta ad emettere un nuovo biglietto.
- 6 L'ingresso dei dipendenti in un'azienda è controllato attraverso una macchina che è in grado di leggere una tessera magnetica in possesso di ogni dipendente: dopo aver letto la tessera, la macchina apre la sbarra di ingresso.
- 7 Un telecomando controlla la visualizzazione di pagine di testo sul televisore. Il meccanismo consente di passare alla pagina precedente o successiva pigiando il tasto P o il tasto S. Per semplicità si supponga che le pagine disponibili siano in numero limitato: i simboli di input sono i tasti pigiati, l'output è costituito dalle pagine visualizzate; la macchina possiede tanti stati quante sono le pagine a disposizione.
- 8 L'ascensore di un edificio, per semplicità di 5 piani, può essere pensato come un automa avente tanti stati quanti sono i piani, oltre al piano terra e al sotterraneo: i simboli di input sono costituiti dalle cifre dei piani oltre a T per il piano terra e S per il sotterraneo (corrispondenti al tasto pigiato), l'output è costituito dai diversi tipi di movimento dell'ascensore (su, giù, fermo).
- 9 Lo sportello Bancomat per la distribuzione automatica di denaro è una macchina automatica che legge la tessera magnetica inserita dall'utente, accetta la cifra richiesta attraverso la tastiera, ed emette le banconote.

Hardware

- 10 Classificare gli oggetti elencati mettendo una crocetta nella colonna corrispondente.

	input	output	input/output
disco			
tastiera			
lettore di CD-ROM			
stampante laser			
mouse			
video			
nastro			
lettore di barcode			
plotter			
chiavetta USB			
scanner			

Software

11 Classificare gli oggetti elencati mettendo una crocetta nella colonna corrispondente.

	hardware	software
chip		
compilatore		
CPU		
debugger		
disco magnetico		
file system		
interfaccia		
kernel		
linker		
masterizzatore		
memoria cache		
microprocessore		
periferica		
plotter		
programma sorgente		
RAM		
scanner		
shell		
sistema operativo		
SPOOL		
tools applicativi		
video		
CD-R		

12 Descrivere le caratteristiche tecniche del proprio computer domestico oppure del computer utilizzato nel Laboratorio di Informatica della scuola, seguendo la traccia della tabella seguente:

Produttore	
Modello	
Processore	
Memoria cache	
Memoria RAM	
Slot di espansione	
Collegamenti I/O (porte seriali, parallele, USB)	
Schema video	
Memorie di massa	
Multimedia (scheda audio, CD, casse)	
Tastiera	
Mouse	
Modem	
Scheda di rete	
Sistema operativo	
Software pre-installato	



SCHEDA DI AUTOVALUTAZIONE

CONOSCENZE

- ☐ Principi generali di funzionamento di una macchina automatica
- ☐ Caratteristiche fondamentali delle macchine in grado di eseguire procedimenti in modo automatico
- ☐ Definizione formale di automa
- ☐ Caratteristiche e funzioni delle componenti fondamentali di un sistema di elaborazione
 - processore
 - memoria centrale
 - unità di input/output
 - memorie di massa
- ☐ Moduli del sistema operativo
- ☐ Caratteristiche dei software applicativi
- ☐ Sicurezza dei sistemi
- ☐ Fasi di sviluppo del software

ABILITÀ

- ☐ Definire semplici automi individuando
 - simboli di input
 - simboli di output
 - insieme degli stati
 - funzione degli stati successivi
 - funzione delle uscite
- ☐ Rappresentare il funzionamento di un automa con i grafi e le tabelle
- ☐ Saper individuare le unità che compongono un sistema di elaborazione
- ☐ Utilizzare parametri e unità di misura per definire le caratteristiche di apparecchiature hardware
- ☐ Saper riconoscere i diversi supporti di memorizzazione delle informazioni
- ☐ Riconoscere le funzioni fondamentali di un sistema operativo
- ☐ Saper adottare semplici misure per garantire la sicurezza del sistema di elaborazione



SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 662

3

parte seconda

L'ambiente di lavoro
con il computer

Il sistema operativo Windows

OBIETTIVI DI APPRENDIMENTO

In questo capitolo imparerai i moduli di uno specifico sistema operativo multitasking e utilizzerai le funzioni del sistema operativo attraverso l'interfaccia grafica.

Sarai in grado di operare nell'ambiente di lavoro con icone e finestre e di organizzare i file e le cartelle sul disco.

Conoscerai anche le modalità per installare nuovo hardware e nuovo software e per condividere le risorse in una rete.

Caratteristiche generali

Avvio e arresto del sistema

Il desktop

Il Pannello di controllo

Il mouse

Installazione e rimozione di nuovi programmi

Multitasking

L'interfaccia standard delle applicazioni

Il pulsante Start

Il programma per l'editing dei testi

Cartelle e file

Le operazioni sui file

La compressione dei file

I virus

La Guida in linea

Gestione di una stampante

Connessione e rimozione di periferiche

Copie di sicurezza

Lavorare in rete

1 Caratteristiche generali

Il sistema operativo **Microsoft Windows** si occupa della gestione ottimizzata delle risorse hardware e software di un sistema di elaborazione, consentendo all'utente di riferirsi ad esse con comandi simbolici e rappresentazioni grafiche: il lavoro diventa quindi più semplice e gradevole attraverso un'interfaccia utilizzabile da chiunque intenda operare elaborazioni di dati senza possedere specifiche competenze.

Windows è classificabile come un sistema operativo con un'**interfaccia utente** di tipo grafico (**GUI**, *Graphical User Interface*), a differenza di sistemi meno recenti e più piccoli, quale il DOS, che avevano un'interfaccia utente di tipo testuale. I moduli di Windows includono tutte le funzioni fondamentali di un sistema operativo; vengono inoltre inserite, come programmi applicativi, funzioni aggiuntive destinate a rendere l'utente subito operativo con l'installazione del sistema Windows, e orientate alla gestione delle nuove tecnologie: ci sono moduli per gestire reti locali, Internet, strumenti multimediali. Il tutto è facilitato dall'interfaccia utente, di tipo grafico e di semplice uso, che consente di rendere autonoma sia la gestione delle componenti hardware del sistema (*driver di periferica*), sia la gestione del software e degli archivi di dati presenti sui diversi supporti, dai dischi magnetici ai supporti ottici o altro (*operazioni per la gestione dei dischi*).

Nelle versioni più recenti, poi, il sistema operativo Windows si è integrato in modo completo con l'accesso alla **rete Internet**, fornendo un'unica visualizzazione per le risorse locali e per quelle di rete: le pagine Internet e le risorse presenti sulla rete diventano oggetti trattabili in modo uguale alle risorse del computer dell'utente.

2 Avvio e arresto del sistema

All'avvio del computer, quando l'utente preme l'interruttore di accensione, vengono attivati i processi di inizializzazione con i quali si effettua il controllo della configurazione del sistema e delle componenti hardware disponibili e il caricamento dei moduli fondamentali del sistema operativo dal disco alla memoria centrale. Alcuni di questi moduli rimangono poi residenti in memoria, altri vengono caricati in seguito quando servono.

Questa fase, detta **bootstrap** del sistema, si conclude quando il sistema operativo attiva il processo di **identificazione** dell'utente (in inglese, **login**), cioè presenta sul video una finestra con la quale vengono richiesti il **nome dell'utente** e la **password** per accedere al sistema.

Dopo il riconoscimento, il sistema operativo predispone l'ambiente di lavoro e le impostazioni predefinite per l'utente.



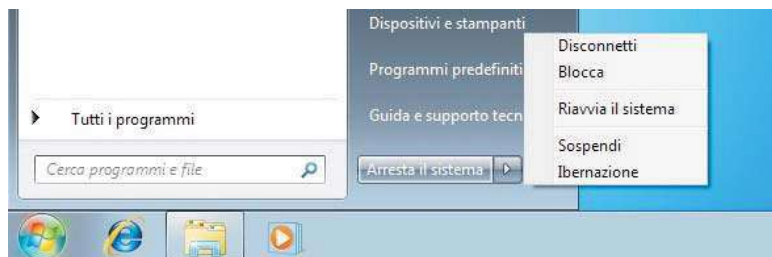
Alla fine del lavoro, prima di spegnere fisicamente il computer, il sistema operativo deve eseguire le seguenti **operazioni di chiusura**:

- eliminare tutti i file temporanei;
- chiudere le applicazioni attive, avvertendo l'utente qualora qualche documento non sia stato salvato;
- chiudere tutti i **file di log**, cioè file di informazioni sulle sessioni di lavoro utili alla ricerca di problemi quando si è in presenza di guasti o malfunzionamenti del sistema.

La sessione di lavoro deve essere quindi conclusa correttamente prima di spegnere il computer.

Per arrestare il sistema

Fare clic sul pulsante **Start**, in basso a sinistra nel desktop, e scegliere **Arresta il sistema**.



I modelli recenti di personal computer possiedono la funzione di spegnimento automatico, cioè viene tolta l'alimentazione dopo aver completato le operazioni di arresto del sistema.

Facendo clic sulla piccola freccia accanto a **Arresta il sistema**, si possono utilizzare anche le opzioni:

- **Riavvia il sistema**: per riavviare il sistema dall'inizio senza spegnere fisicamente il computer.
- **Sospendi**: per gestire il risparmio energetico e ridurre il consumo di energia delle periferiche o dell'intero sistema.

La sospensione del computer provoca la disattivazione del monitor e del disco rigido, il salvataggio su disco di tutti i dati presenti in memoria e lo spegnimento del computer. Quando si avvia nuovamente il sistema, il desktop di Windows verrà riportato alle condizioni precedenti lo spegnimento.

Per disconnettersi come utente del sistema

Fare clic sul pulsante **Start**, poi un clic sulla piccola freccia e scegliere **Disconnetti**.

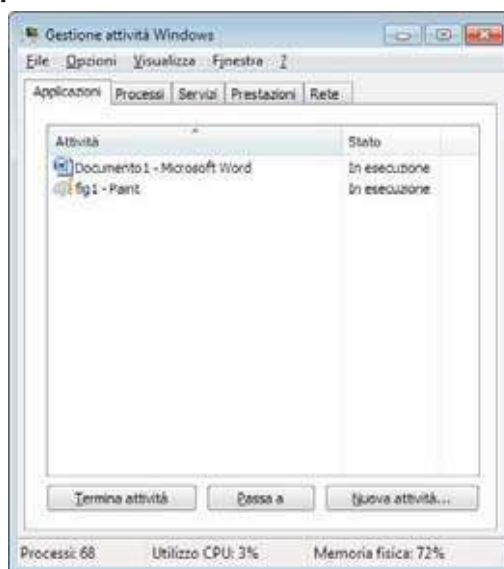
Durante il lavoro, può accadere che un programma si blocchi e non ci sia la possibilità di riprendere il lavoro.

Per chiudere un programma che non risponde

Aprire **Gestione attività** con una tra le seguenti tre modalità:

1. Fare clic con il tasto destro del mouse in un'area vuota sulla *Barra delle applicazioni* e scegliere **Avvia Gestione attività** oppure
2. Premere contemporaneamente i tasti **Crtl+Maiusc+Esc** oppure ancora
3. Premere contemporaneamente i tasti **Crtl+Alt+Canc** e poi fare clic su **Avvia Gestione attività**.

In tutti i casi selezionare poi la scheda **Applicazioni**, fare clic sull'applicazione che non risponde e quindi su **Termina attività**.



3 Il desktop

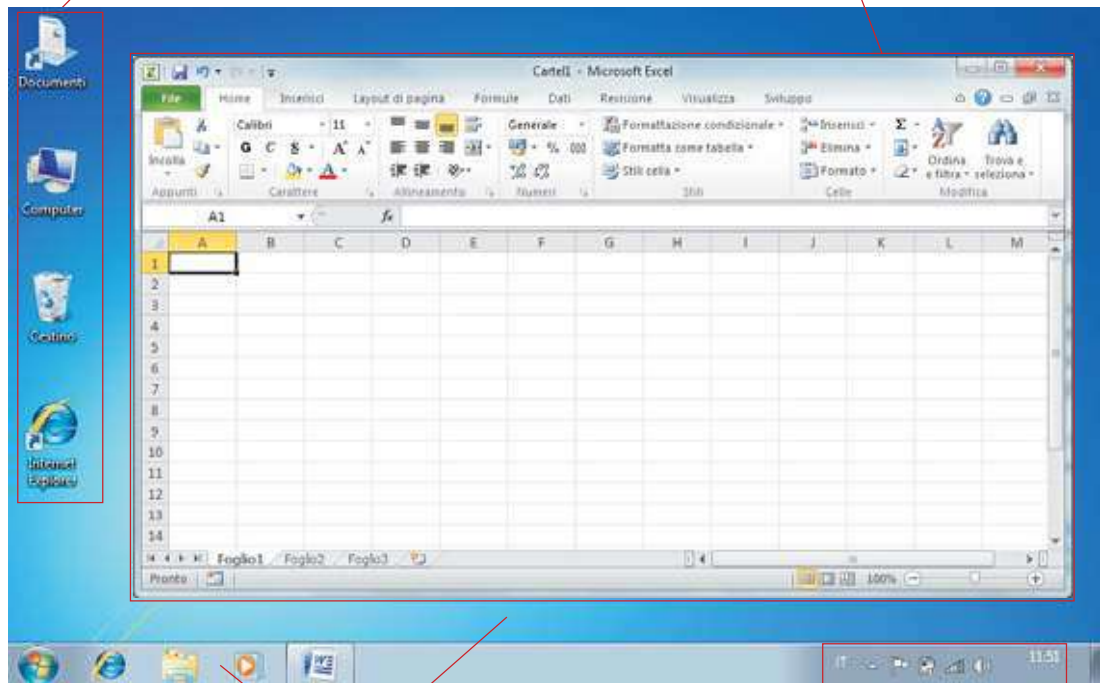
L'ambiente di lavoro di Windows realizza la metafora della scrivania (**desktop**) attraverso l'interfaccia grafica. Il **video** rappresenta il piano di lavoro della scrivania, le **icone** gli oggetti che vengono usati durante il lavoro, le **finestre** i fogli di carta che possono essere messi o tolti dalla scrivania. Ci sono poi altri strumenti che ricordano gli abituali oggetti presenti in un ufficio: per esempio il cestino serve ad eliminare i documenti che non servono più.

In pratica le icone rappresentano le applicazioni che possono essere avviate dall'utente; le finestre sono i programmi attivi; in basso la *Barra delle applicazioni* indica i lavori attualmente aperti sulla scrivania.

Sul desktop di Windows, presentato sul video, si possono posare tutti gli oggetti disponibili: i programmi, le finestre per interagire con il sistema, i documenti, gli archivi di ogni genere, le risorse del sistema.

Le icone sono piccole immagini che possono rappresentare un programma oppure dei dati. Le principali icone visualizzate sul desktop sono: l'icona della cartella dei **Documenti**, l'icona per gestire il proprio computer (**Computer**), l'icona del **Cestino** (contiene i dati cancellati) e l'icona del programma **Internet Explorer**.

Le **finestre dei programmi** sono aree rettangolari all'interno delle quali vengono eseguiti i programmi.



La **Barra delle applicazioni** contiene i pulsanti dei programmi aperti

Il **desktop**, o scrivania, è la schermata che compare sul video dopo aver acceso il computer.

La **tray bar** (o area di notifica) contiene i programmi di servizio del sistema.

L'aspetto della scrivania è personalizzabile a discrezione dell'utente, il quale può disporre in modo diverso le icone, oppure modificare la risoluzione video, lo sfondo, l'aspetto dei colori e lo **screen saver**, cioè un programma che viene attivato in automatico dopo alcuni minuti di inattività del sistema, per risparmiare l'utilizzo dello schermo e l'energia elettrica che mantiene acceso quest'ultimo.

Tutte le impostazioni di personalizzazione e di configurazione del sistema vengono attivate dal pulsante **Start**, facendo clic su **Pannello di controllo**.

Per modificare la risoluzione video

Una **risoluzione** dello schermo più elevata riduce la dimensione degli oggetti e consente di visualizzare più informazioni. Le risoluzioni standard per i monitor variano, a seconda della scheda video del computer, per esempio 800x600 pixel oppure 2048x1536 pixel. I pixel sono i punti luminosi che si possono accendere sul monitor.

1. Fare clic su **Start**, scegliere **Pannello di controllo**.
2. Nella categoria **Aspetto**, fare clic su **Modifica risoluzione schermo** oppure fare clic con il tasto destro del mouse in un punto qualsiasi del desktop e selezionare **Risoluzione dello schermo**.
3. Nella casella **Risoluzione** scegliere la nuova risoluzione.
4. Fare clic sul pulsante **Applica**. Lo schermo diventerà nero per qualche istante: l'utente ha a disposizione 15 secondi per confermare le modifiche apportate.
5. Fare clic su **Mantieni le modifiche** per confermare le modifiche, altrimenti fare clic su **Ripristina**.



Per modificare lo sfondo del desktop

1. Fare clic su **Start**, scegliere **Pannello di controllo**.
2. Selezionare **Aspetto e personalizzazione**, quindi **Cambia lo sfondo del desktop**.
3. Selezionare un'immagine tra quelle predefinite di Windows. Si può usare anche un'immagine registrata sul disco, selezionandone il nome dal pulsante **Sfoglia**.
4. Nella casella **Posizione immagine**, specificare la modalità di posizionamento dell'immagine, scegliendo tra: **Estendi**, **Affianca** (per coprire l'intero schermo con diverse copie di un'immagine di sfondo di piccole dimensioni), **Centra**, **Riempi** o **Adatta**.
5. Fare clic sul pulsante **Salva modifiche** e chiudere la finestra di dialogo. Si possono anche selezionare più immagini per creare una presentazione.



Per impostare uno screen saver

1. Fare clic su **Start**, scegliere **Pannello di controllo**.
2. Selezionare **Aspetto**, quindi **Cambia screen saver** nel gruppo **Schermo**.
3. Nella scheda **Screen saver** selezionarne uno tra quelli predefiniti di Windows.
4. Per personalizzare lo *screen saver*, fare clic sul pulsante **Impostazioni**.
5. Per vederne un'anteprima, fare clic sul pulsante **Prova**.
6. Fare clic su **Applica**.

Lo *screen saver* viene avviato quando il computer rimane inattivo per il numero di minuti specificato nella casella **Attesa**. Lo *screen saver* viene disattivato quando si muove il mouse oppure si preme un tasto qualsiasi.



Per documentare il proprio lavoro oppure per tenere traccia di una situazione di errore o malfunzionamento, oppure, ancora, per spiegare l'uso delle funzionalità del sistema operativo o di un software, può essere utile catturare l'intero schermo o una singola finestra conservandola in un file di immagine.

Per utilizzare la funzione stampa schermo

1. Premere il tasto **Stamp** per catturare lo schermo intero.
2. Per catturare solo la finestra attiva, premere contemporaneamente i tasti **Alt+Stamp**.



+



L'immagine catturata può poi essere incollata in un'applicazione, per esempio nel programma di grafica *Paint* che si trova negli *Accessori* di Windows.

4 Il Pannello di controllo

Dal pulsante **Start**, con la scelta **Pannello di controllo**, vengono effettuate molte altre impostazioni, per esempio:

- **Data e ora**

selezionare **Orologio e opzioni internazionali** e poi fare clic su **Imposta la data e ora**. Occorre fare attenzione alla modifica della data e dell'ora, perché è un'informazione che descrive le caratteristiche dei file registrati sul disco e che viene usata per sincronizzare i computer di una rete.

- **Volume audio**

selezionare **Hardware e suoni**, quindi fare clic su **Regola volume di sistema** nel gruppo **Audio**. Il volume audio può essere regolato anche facendo clic sull'icona **Speakers** nell'*area di notifica (tray bar)* in basso a destra sul desktop.



- **Lingua della tastiera**

selezionare **Orologio e opzioni internazionali** e poi **Paese e lingua**: nella scheda **Tastiere e lingue**, fare clic sul pulsante **Cambia tastiere**. Nella nuova finestra che si apre, si possono selezionare, aggiungere o rimuovere le tastiere.



Per verificare le informazioni di base del computer

Le informazioni riguardano la versione del sistema operativo, il processore, la quantità di memoria RAM installata.

Fare clic su **Start**, scegliere **Pannello di controllo**, quindi **Sistema e sicurezza** e poi **Sistema**.

5 Il mouse

La semplicità di utilizzo del sistema operativo con interfaccia grafica si basa sul mouse: il movimento della mano su un piano provoca lo spostamento di una freccia sul video (*puntatore*), che sostituisce il vecchio cursore delle interfacce testuali.

Di norma con il **tasto sinistro** (il più usato) del mouse si spostano o si attivano i menu e le icone.

Il tasto sinistro serve per:

- fare una scelta (clic)
- selezionare o trascinare (tasto premuto)
- attivare un programma (doppio clic).

Con il **tasto destro** del mouse si accede ai menu di scelta rapida e al menu generale per la gestione dell'oggetto puntato, oppure alle caratteristiche dell'icona o ai programmi di configurazione annessi all'oggetto stesso.



6 Installazione e rimozione di nuovi programmi

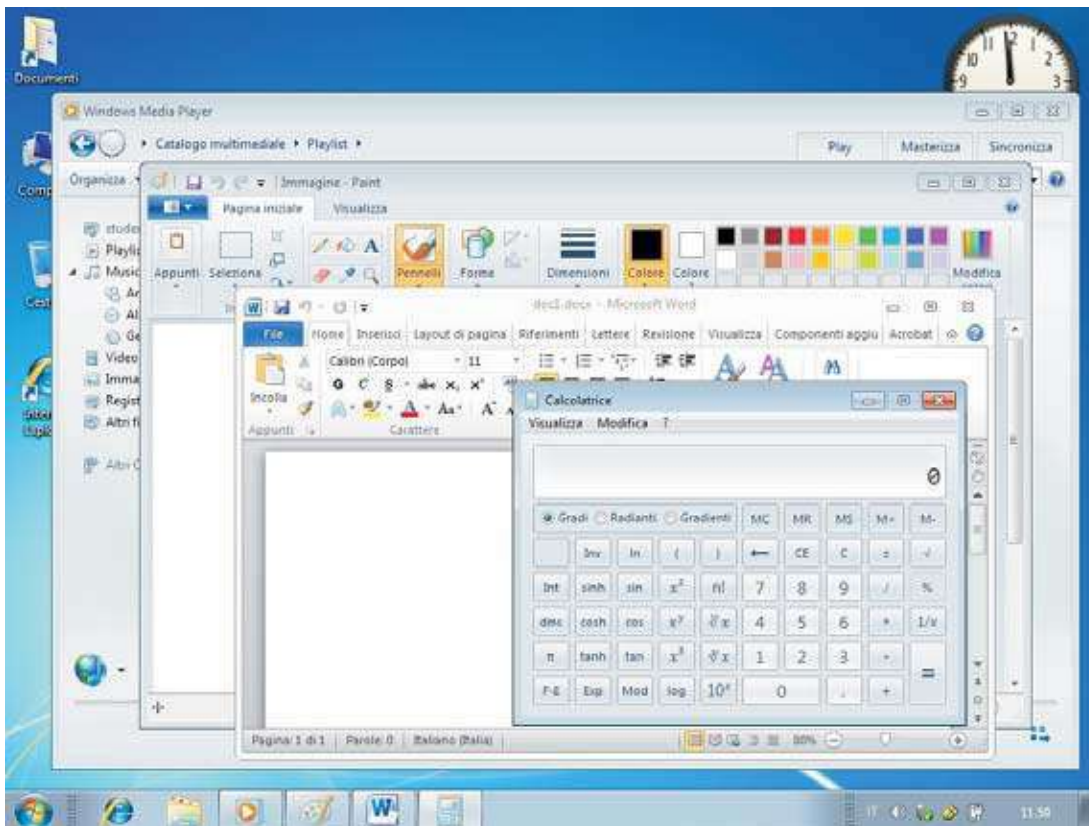
In ambiente Windows è inserito un programma per semplificare la procedura di installazione di nuovi programmi ottenuti tramite supporto esterno (CD o DVD) oppure scaricati da Internet. Le moderne versioni del software sono distribuite tramite CD o DVD che si avviano automaticamente con la procedura di installazione quando vengono inseriti nel lettore di CD o DVD. Negli altri casi si ha a disposizione un file eseguibile (per esempio **setup.exe**) per l'installazione. Facendo doppio clic sulla sua icona, viene avviata la procedura guidata per l'installazione del nuovo software.

Per rimuovere un programma dal computer

1. Fare clic su **Start**, scegliere **Pannello di controllo**.
2. Selezionare **Disinstalla un programma** nella categoria **Programmi**.
3. Selezionare con un clic il software da disinstallare.
4. Fare clic sul pulsante **Disinstalla** per avviare la procedura di rimozione.

7 Multitasking

Normalmente nella parte inferiore del desktop è collocata la **Barra delle applicazioni**. A sinistra della barra si trova il pulsante **Start**, mentre a destra è collocata l'**Area di notifica** (*tray bar*) con l'orologio, la regolazione del volume se è presente una scheda audio, l'icona della stampante quando è attivo il gestore delle stampe (*Print Manager*) e il livello della batteria nei sistemi portatili.



La *Barra delle applicazioni* consente all'utente di ricordare i "fogli" che ha posato sulla scrivania, corrispondenti alle finestre, o che ha messo temporaneamente da parte o nascosto sotto altre finestre: in ogni istante è possibile con un colpo d'occhio visualizzare l'elenco delle applicazioni attive e attivare le finestre in quel momento aperte sul desktop.

Per passare da una finestra all'altra

1. Fare clic sul pulsante corrispondente al programma sulla *Barra delle applicazioni*.
2. Se la *Barra delle applicazioni* è nascosta, spostare il puntatore del mouse sull'area dello schermo in cui è posizionata la barra (normalmente nella parte inferiore dello schermo): essa apparirà automaticamente.
oppure
tenere premuto il tasto **Alt** e premere ripetutamente il tasto **Tab** (il tasto con le doppie frecce vicino al tasto Q).



Si introduce con questo strumento il **multitasking**, una caratteristica molto importante dei moderni sistemi operativi.

In un sistema monoutente è molto utile far funzionare più applicazioni in contemporanea: per esempio, mentre si scrive un testo, si vuole inserire in esso un grafico creato con un altro programma senza dover terminare il programma di elaborazione dei testi (*word processing*), ma sospendendo temporaneamente l'applicazione, oppure si vuole ascoltare un motivo di sottofondo riproducendo le canzoni contenute in un CD musicale.

Un processo attivo prende il nome di **task** (*attività*); un programma può attivare uno o più processi per una o più funzioni; la possibilità di far funzionare questi processi in contemporanea corrisponde al multitasking.

Occorre far notare che per realizzare il multitasking è necessario disporre anche di risorse hardware molto efficienti e in passato costose (in particolare di una quantità rilevante di memoria RAM): ecco perché i primi personal computer non potevano appesantire il processore (CPU) con una gestione multitasking.

Per ogni programma attivo viene creato sulla *Barra delle applicazioni*, nella parte inferiore dello schermo, un pulsante che ne riporta l'icona, il nome ed eventuali documenti aperti in quel programma. Per passare da un programma all'altro, basta spostarsi con il mouse sul pulsante e fare clic con il tasto sinistro. La *Barra delle applicazioni* diviene così lo strumento principale tramite il quale si attiva il multitasking, mantenendo nel contempo in evidenza le applicazioni aperte sul desktop.

Per ridurre a icona tutte le finestre aperte

Fare clic sul pulsante **Mostra desktop** in fondo a destra nella *Barra delle applicazioni* (vicino alla data e ora).

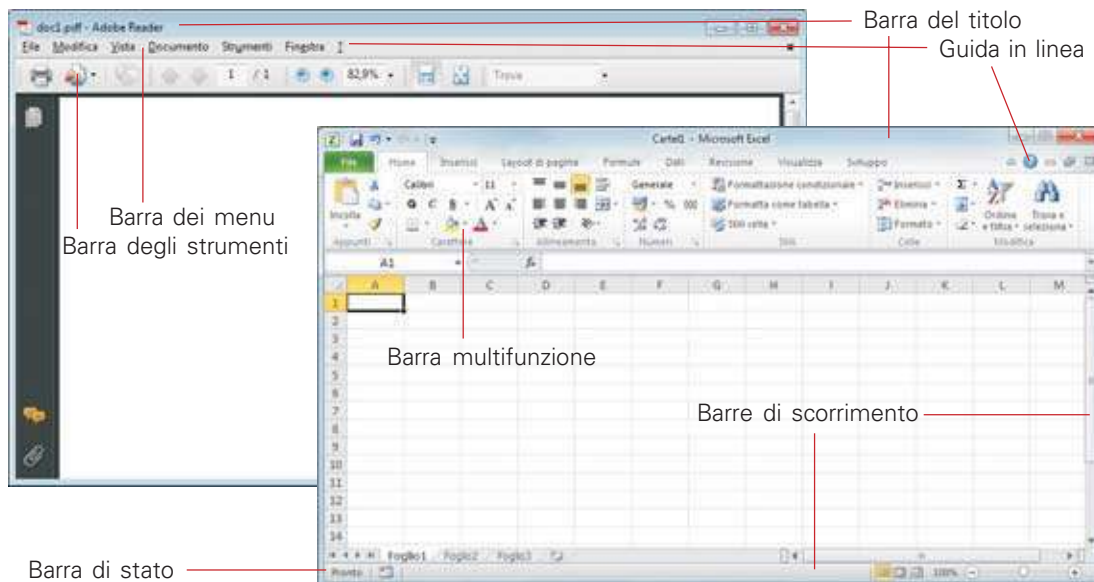
Tutte le finestre aperte diventano pulsanti sulla *Barra delle applicazioni*.

Per ripristinare lo stato precedente delle finestre, fare nuovamente clic sul pulsante *Mostra desktop*.

In alternativa, per ridurre a icona tutte le finestre aperte, si può usare la scorciatoia da tastiera: tenere premuto il tasto **logo di Windows**  (il tasto vicino al tasto *Alt*) mentre si preme il tasto **M**. Per visualizzare velocemente il desktop si può usare anche la combinazione di tasti **logo di Windows** + tasto **D**.

8 L'interfaccia standard delle applicazioni

I programmi e le attività dell'utente vengono rappresentate con riquadri sullo schermo che prendono il nome di **finestre**: esse hanno caratteristiche standard che si presentano simili per i diversi programmi applicativi nell'ambiente Windows.



Gli elementi fondamentali sono:

- La **Barra del titolo** in alto.
- La **Barra dei menu**, che vengono aperti dall'alto verso il basso (menu a tendina) facendo clic su una delle voci della barra.
- La **Barra degli strumenti** oppure la **Barra multifunzione**: è l'insieme delle icone che rappresentano i pulsanti dei comandi posti sotto la barra del titolo. Nella *Barra multifunzione* i pulsanti sono organizzati in *schede*; ogni scheda comprende diversi *gruppi*, ciascuno dei quali include comandi correlati tra loro; un *comando* è un pulsante, una casella per l'immissione di informazioni o un menu.
- Il punto interrogativo o il pulsante per attivare la **Guida in linea** (*help*), che fornisce un aiuto all'utente durante l'esecuzione del programma.
- Le **Barre di scorrimento**, attivate automaticamente in senso verticale e orizzontale quando le dimensioni del documento eccedono le dimensioni della finestra o dello schermo.
- La **Barra di stato** posta nella parte bassa della finestra, riporta informazioni o messaggi relativi al contenuto o alle operazioni della finestra.

Ci sono poi in alto a destra i tre pulsanti standard di ogni finestra:



- il primo riduce l'applicazione a un'icona sulla *Barra delle applicazioni* (**Riduci a icona**);
- il secondo ingrandisce la finestra a tutto schermo (**Ingrandisci**), oppure la riduce a dimensioni più piccole se la finestra è già aperta a tutto schermo (**Ripristina giù**);
- il terzo chiude l'applicazione arrestandone l'esecuzione (**Chiudi**).

Per *ingrandire* la finestra o *ripristinarne* le dimensioni originali, è inoltre possibile fare doppio clic sulla relativa barra del titolo.

Per muovere una finestra

1. Ridurre la finestra a dimensioni più piccole se la finestra è aperta a tutto schermo (con il pulsante **Ripristina giù**).
2. Tenere il tasto sinistro del mouse premuto sulla barra del titolo della finestra e trascinarla sul desktop fino alla posizione desiderata.

Per ridimensionare una finestra

1. Per **modificare la larghezza** della finestra, posizionare il puntatore del mouse sul bordo sinistro o destro della finestra. Quando il puntatore assume la forma di una freccia orizzontale a due punte, trascinare il bordo verso destra o verso sinistra.
2. Per **modificare l'altezza** della finestra, posizionare il puntatore del mouse sul bordo superiore o inferiore della finestra. Quando il puntatore assume la forma di una freccia verticale a due punte, trascinare il bordo verso l'alto o verso il basso.
3. Per **modificare contemporaneamente l'altezza e la larghezza** della finestra, posizionare il puntatore del mouse su un angolo della finestra. Quando il puntatore assume la forma di una freccia diagonale a due punte, trascinare l'angolo nella direzione desiderata.

Per chiudere una finestra

fare clic sul terzo pulsante  nell'angolo in alto a destra della finestra.

Per chiudere una finestra dalla *Barra delle applicazioni*, fare clic con il tasto destro del mouse sul pulsante nella *Barra delle applicazioni* e quindi scegliere **Chiudi finestra**.

Per la chiusura di una finestra si può anche usare la scorciatoia con la combinazione dei

tasti **Alt+F4**.  + 

9 Il pulsante Start

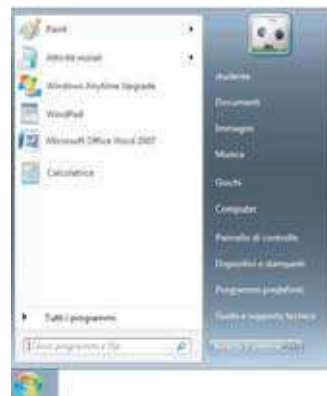
Il pulsante di **Start** (o *Avvio* in altre versioni di Windows) è lo strumento da cui si attivano tutte le applicazioni che si vogliono utilizzare; la sua selezione con il mouse consente di attivare in modo facile ogni funzionalità del sistema operativo: l'utente non deve ricordare particolari comandi o la sintassi di codici operativi, tutte le funzioni appaiono sotto forma di menu e di icone. Per la loro attivazione basta spostare sopra di esse il mouse e premere il tasto sinistro.

Per aprire il menu *Start* si può anche premere il tasto con il *logo di Windows*  , in basso a sinistra nella tastiera (vicino al tasto *Alt*).

Per avviare un programma

1. Fare clic sul pulsante **Start**, quindi scegliere **Tutti i programmi**.
2. Per avviare un programma si punta con il mouse sulla voce dei sottomenu (senza fare clic) e un clic finale sul nome del programma scelto.

Dopo avere avviato il programma, verrà visualizzato un pulsante ad esso corrispondente sulla *Barra delle applicazioni*. Per passare da un programma in esecuzione all'altro, fare clic sul pulsante corrispondente della *Barra delle applicazioni*.



Il menu di *Start*, nella sua forma standard, contiene le seguenti opzioni principali:

Opzione	Azione
<i>Tutti i programmi</i>	Visualizza l'elenco dei programmi installati che è possibile avviare
<i>Documenti</i>	Visualizza il contenuto della cartella Documenti, che contiene i dati archiviati dall'utente
<i>Immagini</i>	Visualizza il contenuto della cartella Immagini
<i>Musica</i>	Visualizza il contenuto della cartella Musica
<i>Giochi</i>	Accede a una raccolta di giochi installati nel computer
<i>Computer</i>	Visualizza le unità presenti nel sistema, dischi e periferiche
<i>Pannello di controllo</i>	Consente di accedere ai programmi di visualizzazione e modifica delle impostazioni del sistema
<i>Dispositivi e stampanti</i>	Visualizza le caratteristiche delle principali unità periferiche
<i>Programmi predefiniti</i>	Consente di selezionare i programmi predefiniti per il Web e altre attività
<i>Guida e supporto tecnico</i>	Attiva la guida in linea con la quale è possibile avere accesso a tutte le informazioni inerenti un particolare aspetto dell'ambiente di lavoro di Windows
<i>Cerca programmi e file</i>	Fornisce le opzioni per la ricerca di un file, di una cartella o di un testo all'interno delle risorse a disposizione dell'utente
<i>Disconnetti</i>	Consente di disconnettersi rapidamente e connettersi al sistema di nuovo utilizzando un nome utente differente
<i>Arresta il sistema</i>	Prima di spegnere il sistema è importante attivare questo comando per la corretta chiusura del sistema: devono essere infatti eliminati tutti i file temporanei e chiusi regolarmente i processi in corso di esecuzione. È disponibile anche la possibilità di riavviare il sistema senza doverlo fisicamente spegnere e accendere.

Tutti gli oggetti utilizzabili nel sistema operativo sono rappresentati con le **icone**:

- Le icone del disco fisso C: (*hard disk*) e delle altre unità di memoria di massa (CD, DVD, chiave USB) sono visibili nella finestra che si apre facendo clic su *Start* e poi su *Computer*. 
- Le cartelle (*directory*) sono rappresentate da icone gialle e servono per organizzare i documenti e gli oggetti registrati sui dischi. Le cartelle possono contenere file, programmi o altre cartelle. 
- Le icone con forma diversa da quella della cartella sono icone di file che possono essere di dati o di programma.  
- Il cestino sul desktop è rappresentato con un'icona che lo richiama e, comunque, ne ha il nome. 
- Le applicazioni software sono identificate attraverso l'icona che rappresenta il logo dell'applicazione stessa, assegnata al momento dell'installazione. 
- Un'icona a forma di stampante rappresenta i dispositivi di stampa installati. 

Per selezionare un'icona è sufficiente fare clic una volta su di essa: il nome che la caratterizza viene evidenziato. Le icone del desktop possono essere spostate dopo averle selezionate: basta trascinare l'icona nel punto desiderato tenendo premuto il tasto sinistro del mouse.

Per aprire un file, una cartella o un'applicazione software

Fare doppio clic con il tasto sinistro del mouse sull'icona del file, della cartella o del programma.

Per un efficiente utilizzo del sistema operativo, l'ambiente di lavoro può essere personalizzato anche modificando il menu del pulsante *Start*. È infatti possibile inserire nuove voci, cancellare quelle non utilizzate, o riorganizzare la disposizione dei menu a cascata che si attivano con questo tasto. I programmi più usati possono essere associati ad un'icona sul desktop (con un collegamento) o inseriti nel menu di *Start*.

Per creare un collegamento sul desktop

1. Fare clic su **Start** e poi su **Computer**.
2. Fare clic con il tasto destro del mouse sul file, programma o cartella, per cui si desidera creare un collegamento e scegliere **Invia a** e poi **Desktop (crea collegamento)**.

Per **rimuovere un collegamento**, selezionarlo facendo clic su di esso e premere il tasto **Canc**,



oppure selezionarlo con il tasto destro del mouse e scegliere **Elimina** dal menu di scelta rapida. L'oggetto originale rimarrà nella posizione in cui è memorizzato sul disco.

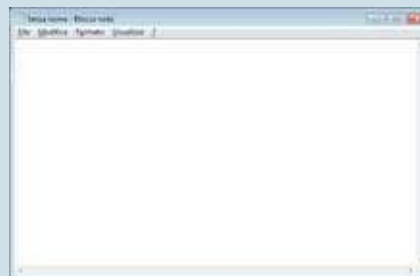
Per aggiungere un nuovo sottomenu al menu Tutti i programmi

1. Dal pulsante **Start**, fare clic con il tasto destro del mouse su **Tutti i programmi**, quindi scegliere **Apri**.
2. Fare doppio clic sulla cartella **Programmi** per selezionarla.
3. Fare clic sul pulsante **Nuova cartella** nella barra in alto nella finestra (si può fare anche con la scelta **Nuovo** e poi **Cartella** nel menu di scelta rapida che si apre con un clic del tasto destro del mouse).
4. Digitare il nome che si desidera utilizzare per il sottomenu.
5. Premere il tasto **Invio**, quindi aprire la cartella appena creata con un doppio clic del mouse.
6. Fare clic con il tasto destro del mouse in un punto vuoto della finestra della cartella e scegliere **Nuovo** e poi **Collegamento**.
7. Utilizzare la procedura guidata **Crea collegamento** per aggiungere voci al sottomenu.

**IL PROGRAMMA PER L'EDITING DEI TESTI**

Il programma **Blocco Note** (o *Notepad*) si attiva dal pulsante *Start*, *Tutti i programmi*, *Accessori*. Esso presenta un'interfaccia semplice, con alcune scelte essenziali organizzate in cinque menu a tendina:

- **File** per la gestione delle operazioni di apertura, salvataggio e stampa;
- **Modifica** per alcune scelte di uso comune sui testi;
- **Formato** per definire le caratteristiche dei caratteri e dei paragrafi;
- **Visualizza** per la visualizzazione della barra di stato;
- **?** per la guida in linea.

**Per creare un nuovo documento**

Scegliere **Nuovo** dal menu **File**.

Il programma assegna al nuovo testo il nome provvisorio "*Senza nome*".

Dopo aver scritto il testo occorre registrarlo su disco assegnando un nome.

Per salvare un nuovo testo senza nome

1. Scegliere **Salva** oppure **Salva con nome** dal menu **File**.
2. Digitare il nome nella casella **Nome file**: il programma aggiunge automaticamente l'estensione *txt*, a meno che non si specifichi un'estensione diversa. In questo secondo caso occorre scegliere *Tutti i file (*.*)* nella casella **Salva come**.
3. Se si vuole salvare il testo in un'unità o in una cartella diversa da quella visualizzata, occorre cambiare la cartella dove salvare il file nella casella dei percorsi, in alto nella finestra (*Barra degli indirizzi*).
4. Fare clic sul pulsante **Salva**.

In alternativa si può operare su un testo salvato in precedenza.

Per aprire un testo esistente

1. Scegliere **Apri** dal menu **File**.
Come scelta predefinita, vengono cercati i file di testo con estensione *.txt*.
2. Nella casella **Nome file** digitare il nome del testo che si desidera aprire, oppure cambiare la cartella dove cercare il file nella casella dei percorsi, in alto nella finestra (*Barra degli indirizzi*).
3. Fare doppio clic sul nome del file desiderato
oppure un solo clic per selezionarlo e poi un secondo clic sul pulsante **Apri**.

Dopo aver operato le modifiche o le aggiunte al testo, occorre registrare la nuova versione sopra quella precedente.

Per salvare il testo

Scegliere **Salva** dal menu **File**.

Per chiudere il programma

Scegliere **Esci** dal menu **File**

oppure

fare clic sul pulsante in alto a destra (**Chiudi**) della finestra di *Blocco Note*.

Per stampare da una stampante predefinita

1. Scegliere **Stampa** dal menu **File**.
2. Selezionare la stampante tra quelle disponibili.
3. Fare clic sul pulsante **Stampa**.

È possibile cambiare le opzioni di stampa e il tipo di stampante con la scelta **Imposta pagina** dal menu *File* prima di lanciare la stampa.

Durante la stampa del testo viene visualizzata un'icona a forma di stampante nell'*Area di notifica* in basso a destra nel desktop. 

Al termine della stampa l'icona scompare. Facendo un doppio clic sull'icona si apre la finestra che visualizza i documenti in coda di stampa.



MATERIALI ONLINE

1. Lezioni multimediali sull'ambiente di Windows

10 Cartelle e file

Windows, essendo un sistema operativo, deve fornire anche le funzioni tipiche del livello **file system**, riguardanti la gestione dei file e delle unità fisiche destinate a contenerli.

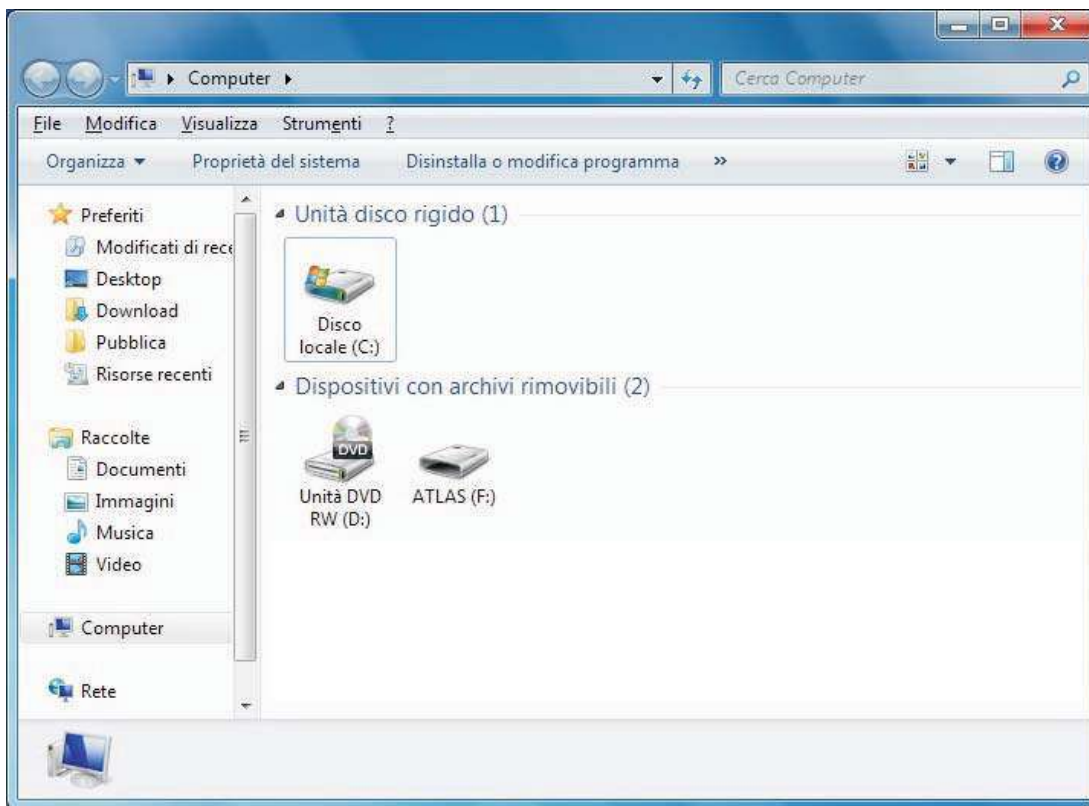
Il sistema operativo memorizza i file e le cartelle sulle **unità periferiche**: sui dischi fissi, sui dischi esterni rimovibili, sulle chiavi USB (dette anche *penne* o *memorie stick*), sui CD-RW (CD riscrivibili), sui DVD-RW (DVD riscrivibili) e sui dischi di rete. Queste unità si chiamano comunemente **memorie di massa** perché possono contenere una grande quantità di dati.

Le unità sono identificate con le lettere, normalmente **C:** è il disco fisso, **D:** è il lettore di CD o DVD oppure il masterizzatore di CD e DVD. Le lettere successive (**E:**, **F:**, ecc.) sono usate per identificare chiavi USB o altre unità a disco, **Z:** per i dischi condivisi da un server o da altri computer della rete.

I file registrati su supporti di memoria collegati al computer, quindi **online** (in linea) consentono l'accesso immediato ai dati in essi contenuti durante l'attività di elaborazione di un programma. Inoltre i file *online* possono essere messi in condivisione con altri utenti collegati in rete.

Il sistema operativo Windows offre le funzioni di gestione di qualsiasi file attraverso un'interfaccia grafica, che rende semplice l'utilizzo del sistema.

Per le operazioni riguardanti il livello *file system* di Windows basta aprire la finestra **Computer** dal pulsante **Start**, con la quale vengono visualizzate le unità disco e le altre unità collegate al computer.



In alternativa è possibile vedere il contenuto di dischi e cartelle con il programma **Esplora risorse**.

Per aprire Esplora risorse

1. Fare clic sul pulsante **Start**, scegliere **Tutti i programmi**.

2. In **Accessori** fare clic su **Esplora Risorse**.

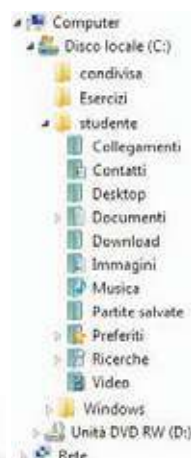
La parte sinistra della finestra si chiama **Riquadro di spostamento** che può essere aperto o chiuso con l'opzione **Layout** nel menu **Organizza** (in alto a sinistra).



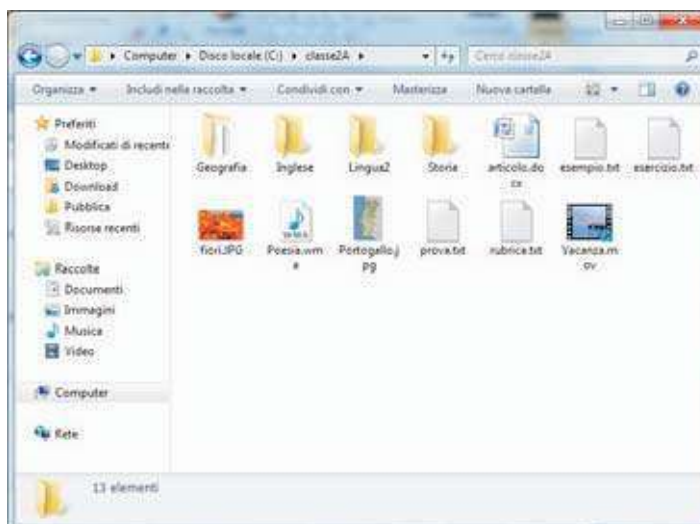
In Windows i file registrati sui dischi sono organizzati con una **struttura gerarchica** (ad albero) nella quale si possono inserire dei nodi e dei sottoalberi; i nodi sono le **cartelle** (*directory*), all'interno delle quali ci sono sottocartelle o singoli documenti (**file**).

Il triangolino bianco accanto a una cartella, significa che quella cartella contiene sottocartelle non visualizzate. Il triangolino nero indica che la cartella è espansa e che il suo contenuto è visualizzato subito sotto, rientrato per mostrare che le sottocartelle sono subordinate. Quindi un clic sul triangolino bianco espande la vista delle unità e delle cartelle (il triangolino diventa nero), mentre un clic sul triangolino nero minimizza la vista (il triangolino diventa bianco).

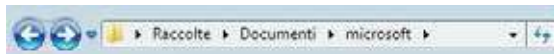
La cartella viene rappresentata con un'icona di colore giallo, che ricorda un raccoglitore di fogli.



Per aprire una cartella e vedere che cosa contiene, basta fare doppio clic sulla cartella selezionata (**finestra della cartella**).



Ci si può muovere all'interno della struttura gerarchica con la casella dell'**Indirizzo** e con i pulsanti **Indietro** e **Avanti** nella parte alta della finestra.



Ogni file è rappresentato con una propria icona, un nome e un'estensione (la parte dopo il punto nel nome del file).

Il **nome** deve essere significativo perché deve ricordarne il contenuto anche a distanza di tempo e perché facilita l'organizzazione delle informazioni sul disco.

La forma dell'**icona** viene assegnata automaticamente dal sistema operativo a seconda del programma applicativo con il quale il file è stato creato.

Per esempio:

Documento di testo



Documento di Microsoft Word



Documento PDF



Pagina HTML di Internet



Documenti multimediali



Nella visualizzazione dei file con icone medie e grandi (pulsante **Modifica visualizzazione** in alto a sinistra) per le immagini, i filmati e i suoni, al posto dell'icona, viene visualizzata un'anteprima.

Anche l'**estensione** del file dipende dal programma con cui è stato creato il file:

.doc e .docx	per i testi creati con Word,
.xls e .xlsx	per i fogli elettronici di Excel,
.mdb e .accdb	per i database di Access,
.ppt e .pptx	per le presentazioni di PowerPoint,
.gif e .jpg	per le immagini,
.rtf	per i file di tipo <i>Rich Text Format</i> creati con Word,
.pdf	per i documenti nel formato standard di Adobe,
.wav o .mp3	per i file audio e musicali,
.avi o .mov	per i filmati,
.zip o .rar	per i file compressi,
.tmp	per i file temporanei,
.exe	per i file eseguibili.

Le **dimensioni** dei file e delle cartelle sono rappresentate con i multipli del **byte** ($1 \text{ byte} = 8 \text{ bit}$): **KB** equivale a 1024 byte, **MB** equivale a 1024 KB (circa un milione di byte), **GB** equivale a 1024 MB (circa un miliardo di byte), **TB** equivale a 1024 GB (circa mille miliardi di byte).

La dimensione di un file o di una cartella compare nella colonna **Dimensione** della finestra del disco o della cartella (fare clic sul pulsante **Modifica visualizzazione** e scegliere la visualizzazione **Dettagli**), oppure nella parte inferiore della finestra quando si fa clic sull'icona del file.

Si può fare anche clic con il tasto destro sull'icona del file (o del disco o della cartella) e scegliere **Proprietà** dal menu di scelta rapida.

Le informazioni sui file e le cartelle riguardano *Nome, Ultima modifica, Tipo, Dimensione*.

Per ordinare i file

Fare clic sul titolo delle colonne *Nome*, *Ultima modifica*, *Tipo*, *Dimensione* a seconda del criterio di ordinamento. Un secondo clic sulla stessa colonna provoca l'ordinamento in senso opposto (crescente o decrescente).

La stessa operazione può essere attivata anche facendo clic con il tasto destro in un punto qualsiasi nella finestra della cartella e scegliendo **Ordina per** dal menu di scelta rapida.

AUTOVERIFICA

Domande da 1 a 4 pag. 119

Problemi da 1 a 4 pag. 120

11 Le operazioni sui file

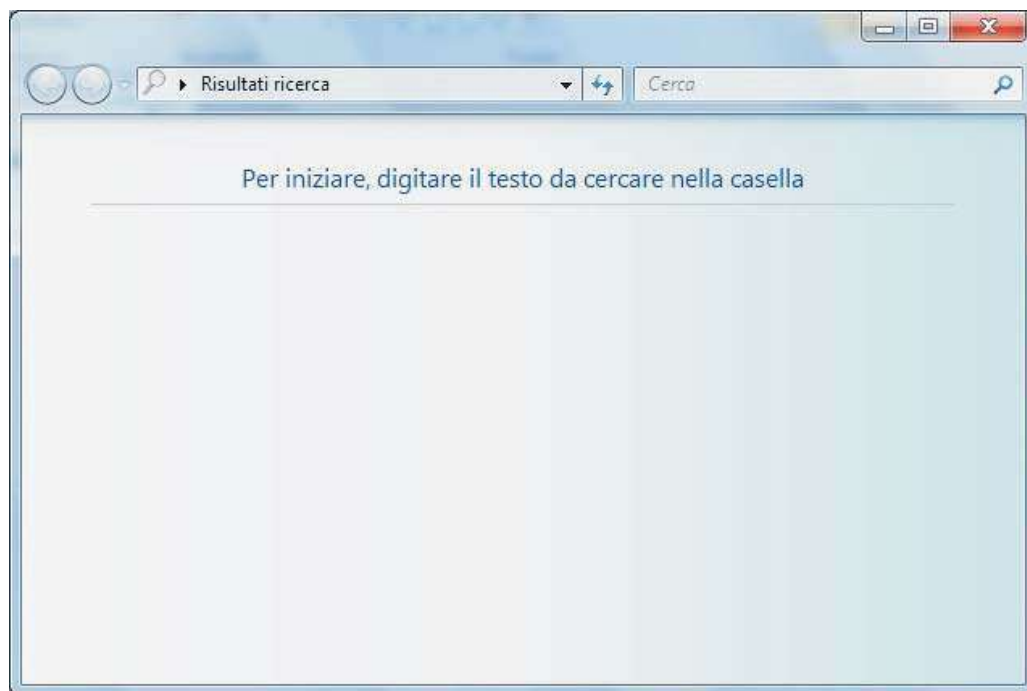
L'operazione di ricerca di un file è importante perché un'unità disco, fisso o rimovibile, può contenere una grande quantità di file.

Per trovare un file o una cartella

La ricerca può essere fatta direttamente scrivendo il nome del file, o parte di esso, nella casella **Cerca programmi e file** sopra il pulsante **Start**.



Per la ricerca digitare una parola o parte di una parola nella casella. Durante la digitazione del testo, vengono visualizzati tutti file presenti nei vari percorsi del computer che corrispondono al testo digitato.



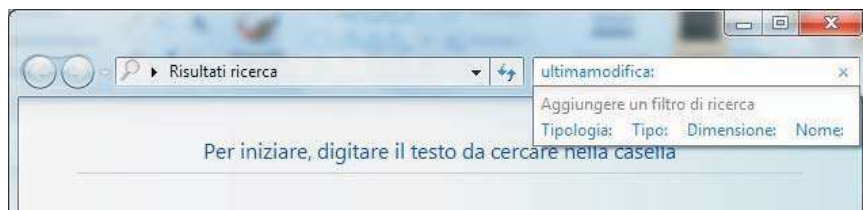
La finestra **Risultati ricerca** può essere aperta con la scorciatoia da tastiera: tasto **logo di Windows** (vicino al tasto Alt) + il tasto **F**.

La casella **Cerca** si trova anche nelle finestre dei dischi e delle cartelle in alto a destra.



Con le opzioni avanzate di ricerca, si possono usare come criteri la data di modifica, la data di creazione o la dimensione del file espressa in byte, il nome del file oppure gli autori.

Facendo clic sulla casella **Cerca** in alto a destra, si può aggiungere un filtro di ricerca: per **Tipologia, Ultima modifica, Dimensione, Nome**.



Se si conosce solo una parte del nome è possibile utilizzare i **caratteri jolly** per cercare tutti i file contenenti quella parte di nome. Per esempio:

specificando **a*.*** verranno trovati tutti i file che iniziano con la lettera **a**;

specificando ***.mp3**, verranno trovati tutti i file con estensione **mp3** e nome qualsiasi;

specificando **a???.txt** verranno trovati tutti i file che iniziano con la lettera **a**, aventi il nome lungo tre caratteri ed estensione **txt**.

In generale il carattere jolly ***** (asterisco) indica una sequenza qualsiasi di caratteri, il carattere jolly **?** (punto interrogativo) indica un solo carattere qualsiasi.

Spesso la ricerca si riferisce a file o documenti di uso recente.

Per visualizzare l'elenco di file usati di recente

1. Nella *Barra delle applicazioni*, fare clic con il tasto destro del mouse sul pulsante dell'applicazione: vengono localizzati gli ultimi 10 elementi (**Jump List**).
2. Fare clic sul nome di uno di essi per aprirlo.

Le **cartelle** (*directory*) sono insiemi organizzati di file.

Per creare una nuova cartella

1. Fare clic sul pulsante **Start**, quindi clic su **Computer** e un doppio clic sull'icona che rappresenta il disco fisso C:
2. Fare clic sul pulsante **Nuova cartella** nella barra in alto. La nuova cartella viene visualizzata con un nome temporaneo.
3. Digitare il nome della nuova cartella e premere il tasto **Invio**.
4. Fare clic all'esterno della cartella per deseleggerla.

All'interno di una cartella si possono poi creare **sottocartelle**, dopo aver fatto doppio clic sull'icona della cartella per aprire la finestra della cartella.

Per creare una nuova cartella (o sottocartella) è anche possibile fare clic con il tasto destro del mouse su un'area vuota nella finestra dell'unità o della cartella e, nel menu di scelta rapida, selezionare **Nuovo** e quindi **Cartella**.

Per rinominare un file o una cartella

1. Selezionare il file o la cartella desiderata facendo clic sull'icona.
2. Scegliere **Rinomina** dal menu **Organizza**.
3. Digitare il nuovo nome e premere **Invio**.

Per modificare il nome di un file o di una cartella è inoltre possibile fare clic con il tasto destro del mouse sull'icona del file o della cartella e quindi scegliere **Rinomina** dal menu di scelta rapida. È importante mantenere la corretta estensione quando si rinomina un file, perché il sistema operativo identifica il file con un'icona e un'estensione che ricorda l'applicazione con la quale il file è stato creato. Inoltre il file viene automaticamente aperto nell'applicazione quando si fa doppio clic sull'icona del file stesso. Modificando l'estensione, il file potrebbe diventare inutilizzabile, perché la nuova applicazione, che viene automaticamente associata al file, potrebbe non essere in grado di aprire il file nel nuovo formato.

Per inviare un file o una cartella a un disco esterno

1. Se si desidera copiare un file o una cartella su un disco rimovibile oppure su una chiave USB, collegare l'unità esterna al computer.
2. Nella finestra dell'unità disco o nella finestra di una cartella, fare clic con il tasto destro del mouse sopra l'icona del file o della cartella da copiare.
3. Scegliere **Invia a** dal menu di scelta rapida e fare clic sul nome dell'unità disco esterna.

Il file o la cartella di cui è stato eseguito l'invio ad altra unità mantengono la posizione originale.

Per copiare un file o una cartella

1. Nella finestra di un disco o di una cartella fare clic sul file che si desidera copiare.
2. Scegliere **Copia** dal menu **Organizza**.
3. Aprire la cartella in cui si desidera copiare l'oggetto.
4. Scegliere **Incolla** dal menu **Organizza**.

In alternativa, per copiare il file, tenere premuto **Ctrl** mentre si trascina il file dalla posizione di origine a quella di destinazione. Se si trascina un file su un'altra unità disco, il file verrà copiato e non spostato.

Le operazioni di **Copia/Incolla** possono essere attivate con le voci del menu di scelta rapida del tasto destro del mouse, oppure con la combinazione dei tasti della tastiera **Ctrl+C** per copiare, **Ctrl+V** per incollare.

Si possono copiare anche più file con una sola operazione.

Per **selezionare più file o cartelle** prima di copiarli, tenere premuto il tasto **Maiuscolo** (*Shift*) se i file sono contigui (fare clic sul primo e sull'ultimo), oppure il tasto **Ctrl** se i file non sono vicini, mentre si selezionano gli oggetti desiderati con il clic del mouse.

Il file o la cartella di cui è stata eseguita la copia mantengono la posizione originale.

Per spostare un file o una cartella

1. Nella finestra di un disco o di una cartella fare clic sul file che si desidera spostare.
2. Scegliere **Taglia** dal menu **Organizza**, oppure dal menu di scelta rapida del tasto destro del mouse.
3. Aprire la cartella in cui si desidera spostare l'oggetto.
4. Scegliere **Incolla** dal menu **Organizza**, oppure dal menu di scelta rapida del tasto destro del mouse.

In alternativa, per spostare il file, trascinare il file dalla posizione di origine a quella di destinazione. Se si trascina un file su un'altra unità disco, il file verrà copiato e non spostato.

Per spostare l'oggetto, tenere premuto il tasto **Maiuscolo** (*Shift*) mentre si trascina.

Le operazioni di **Taglia/Incolla** possono essere attivate con le voci del menu di scelta rapida del tasto destro del mouse, oppure con la combinazione dei tasti della tastiera **Ctrl+X** per tagliare, **Ctrl+V** per incollare.

Si possono spostare anche più file con una sola operazione.

Per **selezionare più file o cartelle** prima di spostarli, tenere premuto il tasto **Maiuscolo** (*Shift*) se i file sono contigui (fare clic sul primo e sull'ultimo), oppure il tasto **Ctrl** se i file non sono vicini mentre si selezionano gli oggetti desiderati con il clic del mouse.

Il file o la cartella di cui è stato eseguito lo spostamento vengono eliminati dalla posizione originale.

Per eliminare un file o una cartella

1. Nella finestra di un disco o di una cartella fare clic sul file che si desidera eliminare.
2. Scegliere **Elimina** dal menu **Organizza**
oppure fare clic con il tasto destro del mouse sopra l'icona del file o della cartella e poi scegliere **Elimina** dal menu di scelta rapida
oppure
premere il tasto **Canc** sulla tastiera.



Per l'eliminazione è anche possibile trascinare l'icona del file o della cartella sopra l'icona del **Cestino** sul desktop, tenendo premuto il tasto sinistro del mouse per trascinare.

Gli oggetti eliminati rimangono nel *Cestino* fino a quando non viene svuotato. Se tuttavia si tiene premuto il tasto **Maiuscolo** (*Shift*) mentre un oggetto viene trascinato sul *Cestino*, tale oggetto verrà immediatamente eliminato dal disco senza essere memorizzato nel *Cestino*.

Con l'operazione di cancellazione il sistema operativo mette a disposizione la possibilità di utilizzare lo spazio, così liberato, per altre applicazioni e nello stesso tempo consente l'eventuale recupero di alcuni file inavvertitamente cancellati.

Per recuperare un file cestinato per errore

1. Fare doppio clic sull'icona del **Cestino** sul desktop.
2. Nell'elenco dei file o delle cartelle cestinate scegliere gli oggetti da recuperare.
3. Per avviare il recupero fare clic sul pulsante **Ripristina elemento** nella barra in alto
oppure
scegliere **Ripristina** dal menu di scelta rapida attivato con il tasto destro sull'icona del file.

Per svuotare il cestino

Fare clic sull'icona del **Cestino** con il tasto destro del mouse e scegliere **Svuota cestino** nel menu di scelta rapida

oppure

dopo aver aperto con un doppio clic la finestra del **Cestino** fare clic sul pulsante **Svuota cestino** nella barra in alto.

È ovvio che lo svuotamento del cestino comporta l'annullamento di eventuali successivi recuperi; è quindi importante essere sicuri che ciò che si è cancellato non serva più, esattamente come succede con i cestini reali. Si tenga presente che è bene svuotare periodicamente il cestino, anche perché gli oggetti presenti occupano una porzione, anche se minima, di spazio sul disco.

Normalmente i file hanno uno stato di **lettura e scrittura**, cioè possono essere aperti, visualizzati, modificati e cancellati. È possibile tuttavia modificare lo stato di un file in modo che risulti bloccato nella modalità di **sola lettura**.

Inoltre un file può essere **nascosto** o reso visibile.

Per modificare lo stato di un file

1. Fare clic sull'icona del file con il tasto destro del mouse e scegliere **Proprietà** dal menu di scelta rapida.
2. Per l'attributo **Sola lettura**, mettere o togliere la selezione nella casella accanto all'attributo **Sola lettura**.
3. Per nascondere o rendere visibile il file, mettere o togliere la selezione nella casella accanto all'attributo **Nascosto**.

AUTOVERIFICA

Domande da 5 a 6 pag. 119

Problemi da 5 a 18 pag. 121

**MATERIALE ONLINE****2. Lezioni multimediali sulla gestione dei file**

12 La compressione dei file

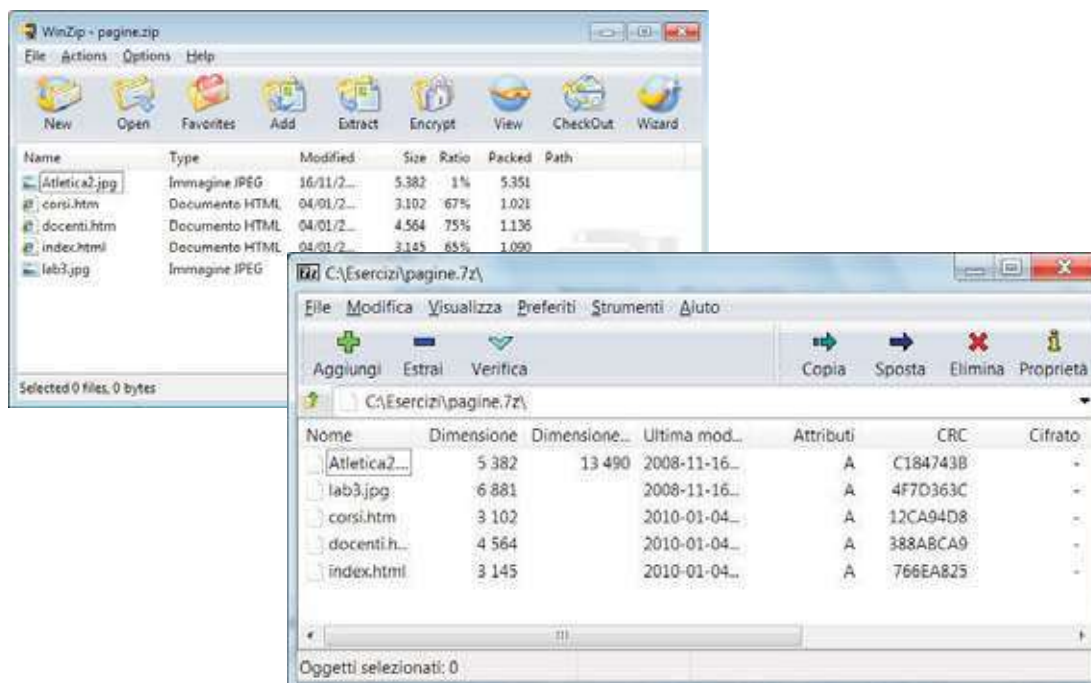
Comprimere un file significa ridurne le dimensioni, conservando il contenuto e le caratteristiche originali, in modo da occupare meno spazio sui supporti di memorizzazione, oppure per diminuire i tempi di trasmissione dei file sulle linee di rete o di Internet o degli allegati ai messaggi di posta elettronica.

Per comprimere i file di una cartella o di un insieme di cartelle si devono utilizzare programmi di compressione, quali *WinZip*, *WinRAR* e *7Zip*.

Tra questi il più usato è il programma *WinZip*. Per questo motivo i file compressi si chiamano spesso anche file *zippati* e le operazioni di compressione e decompressione si indicano spesso anche con **zip** e **unzip**.

I file creati con il programma *WinZip* hanno estensione **.zip**.

Altri formati di compressione sono **.arj**, **.rar** e **.7z**.



Per estrarre i file da un file compresso

1. Scegliere il file *zip* da decomprimere.
2. Fare doppio clic su di esso: automaticamente si apre il programma **WinZip**.
3. Nella finestra compare l'elenco dei file contenuti nel file zippato con le informazioni per ciascuno di essi (nome, data, dimensione originale e fattore di compressione).
4. Facendo clic sull'icona **Extract** della barra degli strumenti, si apre la finestra di dialogo con la quale si fornisce al programma il nome della cartella nella quale si vuole depositare i file dopo l'estrazione. Se la cartella non esiste, viene automaticamente creata dal programma.
5. Fare poi clic sul pulsante **Extract** nella finestra di dialogo.

È possibile anche fare doppio clic su un singolo file all'interno dell'elenco dei file compressi: in questo caso viene aperto automaticamente all'interno dell'applicazione con la quale il file è stato creato. L'operazione contraria consiste nel creare un file compresso (*zippato*) a partire da uno o più file originali.

Per comprimere i file

1. Fare clic su **New** nella barra degli strumenti del programma **WinZip** e fornire il nome e la cartella con cui salvare il nuovo file *zip*.
2. Nella finestra successiva (**Add**) scegliere i file da comprimere e poi premere il pulsante **Add**.

In alternativa è possibile, nella finestra di una cartella, selezionare uno o più file da comprimere (usare il tasto **Ctrl** per selezionare più file), poi premere il tasto destro del mouse sulla selezione effettuata e scegliere **Add to zip** dal menu di scelta rapida che si apre.

Se nel computer è installato un programma di compressione, si può utilizzare il programma di utilità **Cartella compressa** di Windows.

Per comprimere i file

1. Selezionare uno o più file o cartelle che si desidera comprimere.
2. Fare clic con il tasto destro del mouse su di essi, scegliere **Invia a** nel menu di scelta rapida e quindi fare clic su **Cartella compressa**.
Verrà creata una nuova cartella compressa.
Per rinominarla, fare clic con il tasto destro del mouse sulla cartella, scegliere **Rinomina** e quindi digitare il nuovo nome.

Se è già stata creata una cartella compressa e si deve aggiungere un nuovo file o cartella, basta trascinare i file da aggiungere nella cartella compressa.

Per estrarre i file compressi

1. Fare doppio clic sulla cartella compressa per aprirla.
2. Trascinare quindi il file o la cartella dalla cartella compressa in una nuova cartella.
3. Per estrarre l'intero contenuto della cartella compressa, fare clic con il tasto destro del mouse sulla cartella e scegliere **Apri con** e poi **Esplora risorse** dal menu di scelta rapida (oppure selezionare il nome del programma di compressione).

Si osservi che alcuni tipi di file vengono memorizzati sul disco dal programma applicativo in formato compresso, per esempio le immagini in formato JPEG. Questo significa che la compressione di un insieme di file immagine di tipo JPEG non produce una sostanziale riduzione nelle dimensioni.

**I VIRUS**

I virus sono programmi che possono danneggiare il contenuto di file, cartelle o di interi dischi: spesso essi sono in grado di autoriprodursi diffondendosi anche ad altri computer attraverso il trasferimento di file con unità esterne o con messaggi di posta elettronica.

I modi più frequenti con i quali i virus si trasmettono nei computer sono:

- l'uso di dischi esterni o chiavi USB di dubbia provenienza
- l'apertura di messaggi o allegati di posta elettronica
- la consultazione di pagine Web che attivano programmi eseguibili sul computer dell'utente collegato
- lo scambio di file o musica nella rete Internet.

La **protezione** contro i virus può essere realizzata attraverso il controllo dei supporti che vengono usati per memorizzare i dati.

In alcuni casi la rimozione del virus può richiedere la cancellazione dei programmi danneggiati: pertanto diventa fondamentale il lavoro di filtro sugli accessi, sui dati e sul software, con scopi di prevenzione. È inoltre di fondamentale importanza operare un sistematico salvataggio su altri supporti delle copie dei documenti e degli archivi più importanti.

Un **programma antivirus** è uno strumento software che viene caricato nella memoria del computer al momento dell'accensione in modo da controllare il contenuto dei file e delle cartelle sul disco durante il lavoro dell'utente (in *background*, cioè in sottofondo), impedendo l'ingresso e l'attivazione dei virus.

Inoltre il programma consente di fare la scansione di un disco o di un'unità esterna per controllare se contiene virus. I più diffusi programmi antivirus sono *McAfee VirusScan* e *Symantec Antivirus*.

Poiché ogni giorno vengono prodotti nuovi virus di diverso tipo, è necessario mantenere aggiornata la tabella delle **definizioni dei virus**, scaricando dal sito del produttore del programma antivirus i file di aggiornamento e installando poi le nuove definizioni sul proprio computer.

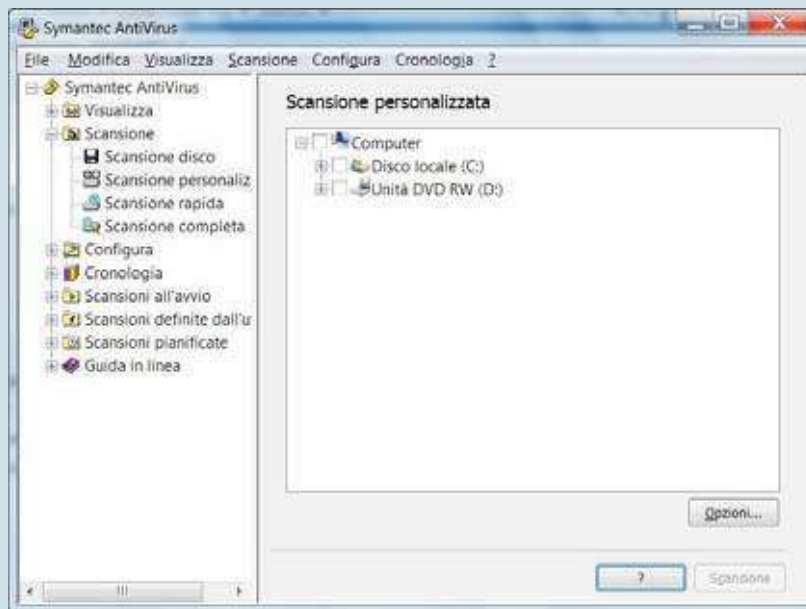
Per controllare l'aggiornamento del software antivirus

Normalmente Windows è in grado di rilevare e controllare lo stato del software antivirus installato nel computer.

1. Fare clic su **Start** e poi su **Sistema e sicurezza**.
2. Fare clic su **Centro Operativo**.
3. Fare clic su **Sicurezza**.

Attivando il **programma di scansione** è possibile selezionare l'unità da controllare e i tipi di file, compresi quelli contenuti nei file compressi. Se durante la scansione il programma trova un file con un virus, esso tenta di ripulire il file infettato oppure, in caso contrario, chiede all'utente la possibilità di cancellare il file.

Al termine della scansione il programma produce un rapporto con il numero di file controllati e i tipi di virus trovati.



Un ruolo importante svolto dal programma antivirus consiste nel controllo dei messaggi di **posta elettronica** in arrivo e dei loro allegati e nella protezione del sistema durante l'attività di consultazione di **pagine Web** tramite il browser, avvisando l'utente sulla possibilità di rimuovere il virus riscontrato.

È comunque buona norma evitare di aprire i messaggi di posta e gli allegati che appaiono come possibili portatori di virus.

13 La Guida in linea

La **Guida e supporto tecnico** (*guida in linea* o **help**) è il programma che mette a disposizione dell'utente la possibilità di consultare il manuale del sistema operativo.

Con lo stesso programma di guida e con le stesse modalità operative vengono anche visualizzati i manuali delle varie applicazioni eventualmente installate nel sistema.

Le Case produttrici di software, sempre più frequentemente, tendono a distribuire la manualistica su supporto informatico anziché cartaceo; i vantaggi sono notevoli:

- maggiore efficienza nella ricerca di informazioni, perché il programma consente di attivare ricerche tra indici di argomenti o per parole chiave, in tempi ridottissimi, fornendo all'utente risposte istantanee;
- l'utente può avere l'aiuto nel momento stesso in cui si trova in difficoltà, nel senso che può richiedere informazioni relative al contesto nel quale sta operando, e risolvere il problema utilizzando le schede guida, che vengono visualizzate sulla scrivania insieme alle altre finestre;
- maggiore economicità, perché il costo della produzione di materiale cartaceo incide notevolmente sul prezzo del pacchetto software finale; è molto più conveniente produrre qualche CD in più o occupare un po' più di spazio su un DVD piuttosto che produrre grandi quantità di carta stampata;
- minore spazio richiesto negli uffici per la conservazione della documentazione sul software. Per tutti questi motivi la documentazione cartacea è limitata al minimo indispensabile (di solito si allega un libretto con le istruzioni di installazione e di un primo approccio al software), ed è eventualmente fornita in modo completo solo su richiesta dell'utente.

Saper utilizzare il programma Guida e supporto tecnico diviene così una competenza fondamentale per l'utente di Windows.

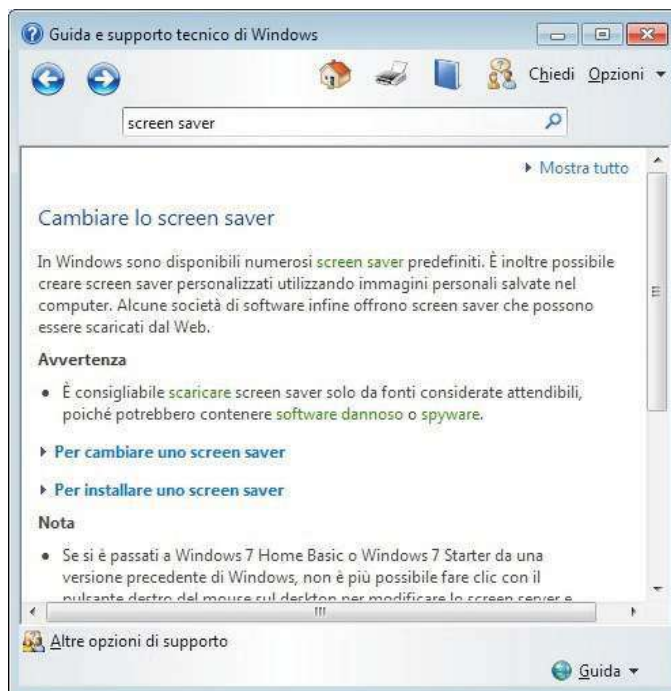
Per attivare la Guida di Windows:

1. Fare clic sul pulsante **Start**.
2. Selezionare **Guida e supporto tecnico**.

La scorciatoia da tastiera per attivare la Guida è la combinazione di tasti: **logo di Windows** (il tasto vicino al tasto Alt) + tasto **F1**.



Per avviare la ricerca su una funzionalità del sistema operativo o su un problema tecnico, occorre scrivere una parola chiave o una frase all'interno della casella di ricerca in alto, per esempio "screen saver" per accedere alle pagine che spiegano come personalizzare il salvaschermo del desktop.



Vengono visualizzati tutti gli argomenti che si riferiscono alla richiesta. In un argomento ci sono riferimenti ad altri argomenti correlati, per accedere ai quali è sufficiente premere il tasto del mouse sopra di essi. Essi sono rappresentati in azzurro e sottolineati, per indicare la presenza di un *link* (collegamento) ad un'altra pagina della Guida. Questo realizza una modalità di accesso alle informazioni di tipo *ipertestuale*, cioè viene offerta la possibilità di una consultazione non necessariamente sequenziale.

Facendo clic sull'icona con una piccola casa nella barra degli strumenti in alto, viene visualizzata la **Pagina iniziale** della Guida. Inoltre ci si può muovere all'interno delle pagine visitate con i pulsanti **Indietro** e **Avanti** in alto a sinistra.

Una pagina può essere stampata con un clic sul pulsante **Stampa**.

Per svolgere una determinata attività l'utente può seguire passo passo le istruzioni che vengono fornite in modo dettagliato dalla Guida e ricevere anche suggerimenti nel caso di problemi o malfunzionamenti.

14 Gestione di una stampante

La stampante, per poter essere utilizzata dai programmi applicativi in ambiente Windows, deve prima essere installata.

Per installare una stampante

1. Fare clic su **Start**, scegliere **Dispositivi e stampanti**.
2. Fare clic sul pulsante **Aggiungi stampante** nella barra in alto, oppure fare clic con il tasto destro del mouse nella finestra e scegliere **Aggiungi stampante**.
3. Seguire le istruzioni contenute nelle finestre di dialogo per l'installazione guidata della stampante.

Se il modello di stampante non è compreso nell'elenco già presente in Windows, all'utente verrà richiesto l'inserimento del disco contenente il programma di gestione (*driver*) relativo a quella stampante.

È opportuno controllare la correttezza dell'installazione attraverso la stampa di una pagina di prova, assicurandosi innanzitutto che la stampante sia accesa e pronta per la stampa.

Terminata l'installazione della stampante, in *Dispositivi e stampanti* del menu *Start* appare l'icona corrispondente.

Se si attiva con un doppio clic questa icona si possono vedere i file che in quel momento sono in fase di stampa: infatti Windows utilizza la tecnica dello **spooling**, cioè invia prima su disco il file da stampare e poi il programma di *spooling* lo trasferisce da disco alla stampante. Questo passaggio consente sia di gestire in modo efficace periferiche lente, sia di rendere disponibile la stampante ad un gruppo di lavoro, ovvero, in una rete locale di computer, a più utenti.

Tra le stampanti disponibili, una può essere impostata come **stampante predefinita**, cioè come la stampante che viene usata normalmente dai programmi applicativi per la stampa di documenti e prospetti su carta, in mancanza di una diversa scelta (in inglese, stampante di *default*).

Per impostare una stampante predefinita

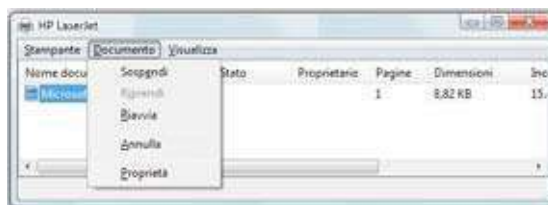
1. Fare clic su **Start**, scegliere **Dispositivi e stampanti**.
2. Fare clic con il tasto destro del mouse sull'icona della stampante e scegliere **Imposta come stampante predefinita** nel menu di scelta rapida

Dopo aver inviato il comando di stampa da un programma applicativo, viene avviato il processo di stampa che può essere controllato dal **gestore delle stampe**.

Durante la stampa del testo viene visualizzata un'icona a forma di stampante  nell'*Area di notifica* in basso a destra accanto alla *Barra delle applicazioni*. Al termine della stampa l'icona scompare.

Facendo un doppio clic sull'icona si apre la finestra che visualizza i documenti in coda di stampa. Nella finestra che si apre, dal menu **Stampante** si possono operare le scelte **Sospendi stampa** ed **Elimina tutti i documenti** dalla coda di stampa.

Facendo clic su un singolo documento all'interno dell'elenco dei documenti in coda di stampa, dal menu **Documento** si possono fare le scelte **Sospendi**, **Riprendi**, **Riavvia**, oppure **Annulla** per annullare il processo di stampa.



MATERIALI ONLINE

3. Lezioni multimediali sulla gestione delle stampe

15 Connessione e rimozione di periferiche

Quando si connette una nuova periferica al computer tramite la porta standard **USB**, per esempio una macchina fotografica digitale oppure una chiave o memoria stick (*penna*), il sistema operativo Windows riconosce la nuova apparecchiatura e assegna ad essa una lettera identificativa, per esempio **E:**, oppure **F:**, oppure **G:**, a seconda di quante periferiche sono già collegate al computer.

La nuova periferica viene rappresentata con un'icona nella finestra *Computer*.



Al termine delle operazioni sulla periferica, si deve eseguire con cura la rimozione del dispositivo, per evitare la perdita di dati.

Per rimuovere la periferica

1. Fare clic sull'icona **Rimozione sicura dell'hardware** che si trova nel gruppo di icone che si apre facendo clic sulla piccola freccia nell'*Area di notifica*, in basso a destra  oppure fare clic con il tasto destro sull'icona della periferica e selezionare **Espellì**.
2. Attendere che il sistema operativo visualizzi il messaggio "È possibile rimuovere l'hardware".



16 Copie di sicurezza

Il termine **backup** indica le copie di sicurezza dei file. Il *backup* è importante perché si può verificare la perdita di file a causa della relativa eliminazione o sostituzione accidentale, dell'attacco di virus, di un errore hardware o software oppure di un errore dell'intero disco rigido. L'operazione di backup deve essere un'attività svolta con regolarità, anche ogni giorno, soprattutto quando riguarda lavori e dati importanti o che comunque richiederebbero tempi lunghi per il recupero.

L'operazione a rovescio, cioè il recupero di dati o documenti dai supporti contenenti le copie di sicurezza, si chiama **ripristino**, in inglese *restore*.

Le copie di *backup* vanno effettuate su supporti diversi da quelli contenenti i file originali, perciò si usano unità di memoria di massa rimovibili, quali chiavi USB, CD, DVD o nastri magnetici.

Si possono copiare singoli file o cartelle importanti per il proprio lavoro oppure si può avviare la procedura di backup: dal pulsante **Start**, scegliere **Pannello di controllo, Sistema e sicurezza** e poi **Backup o ripristino**.

È buona norma conservare poi i supporti di backup in luoghi separati da quello di lavoro, per garantire una conservazione più sicura e una protezione da furti o danneggiamenti accidentali.



MATERIALI ONLINE

4. Eseguire copie di sicurezza (lezione multimediale)

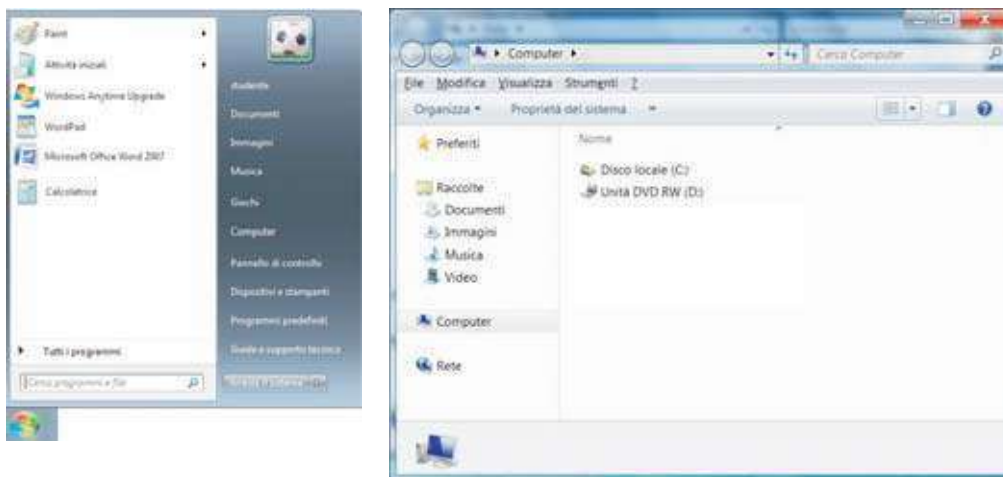
LAVORARE IN RETE

Se il computer appartiene ad una rete locale, il sistema operativo Windows, alla partenza del sistema, attiva la procedura di identificazione dell'utente di rete, con la richiesta del nome di utente e dell'eventuale password. In ogni istante l'utente, connesso alla rete, riesce a vedere quali siano le risorse condivise nella rete.

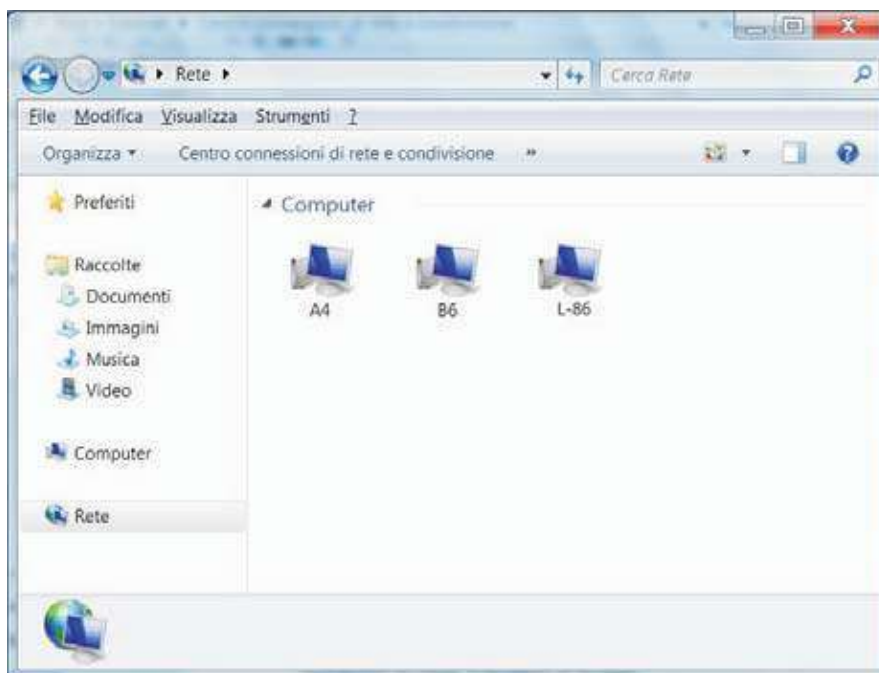
Per visualizzare i computer e i dispositivi di rete

Fare clic sul pulsante **Start** e poi su **Computer**.

Nella parte sinistra della finestra, nel riquadro di spostamento, scegliere **Rete**.



Si possono ottenere informazioni sulla rete anche dal **Pannello di controllo**, scegliendo **Reti e Internet**.



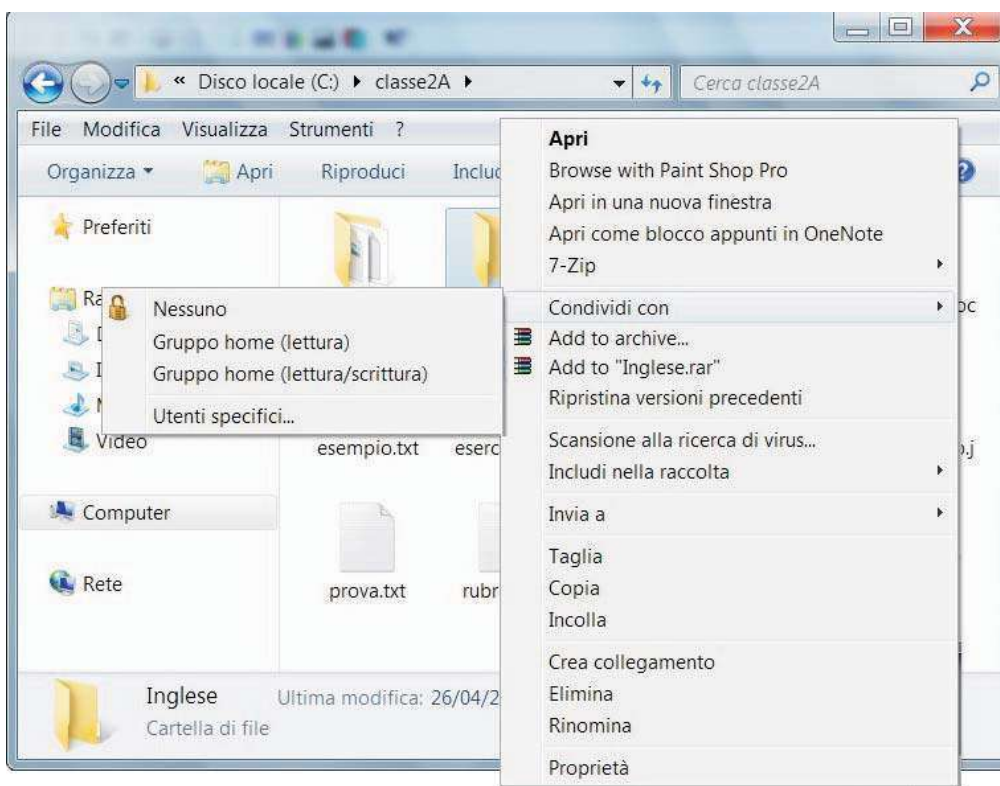
Ogni computer collegato alla rete possiede un nome che lo identifica; facendo doppio clic sull'icona di uno tra i computer elencati, è possibile visualizzare le risorse che il computer ha messo in condivisione (cartella o file).

In generale, la risorsa di rete individuata nell'elenco degli oggetti condivisi può essere aperta e utilizzata come se fosse residente sul proprio computer locale.

Il lavoro in rete si basa sulla collaborazione tra gli utenti: quindi, oltre ad utilizzare le risorse di rete, l'utente stesso può mettere in condivisione proprie risorse per gli altri utenti.

Per mettere in condivisione una cartella

Selezionare la cartella e scegliere **Condividi con** nel menu **File**, oppure fare clic con il tasto destro del mouse sull'icona della cartella che si vuole mettere in condivisione; scegliere **Condividi con** dal menu che si apre.



Le informazioni della risorsa (nella parte inferiore della finestra della cartella) contengono l'impostazione *Condiviso* e un'icona con la figura di due persone.

La condivisione può essere impostata per il gruppo di appartenenza del computer (con autorizzazioni alla sola lettura o alla lettura e alla scrittura), oppure per uno o più utenti specifici. Tra gli utenti specifici, è compreso anche l'utente **Everyone** che rappresenta tutti gli utenti che si connettono alla rete.

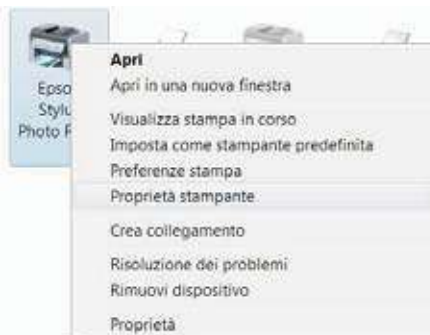
Quando il computer è collegato ad altri computer in una rete locale, l'utente che non ha a disposizione una propria stampante può utilizzare quella di rete che viene condivisa con altri utenti; in altri casi può egli stesso mettere a disposizione una stampante, connessa al computer in uso, per tutti gli altri utenti della rete.

Per mettere in condivisione una stampante

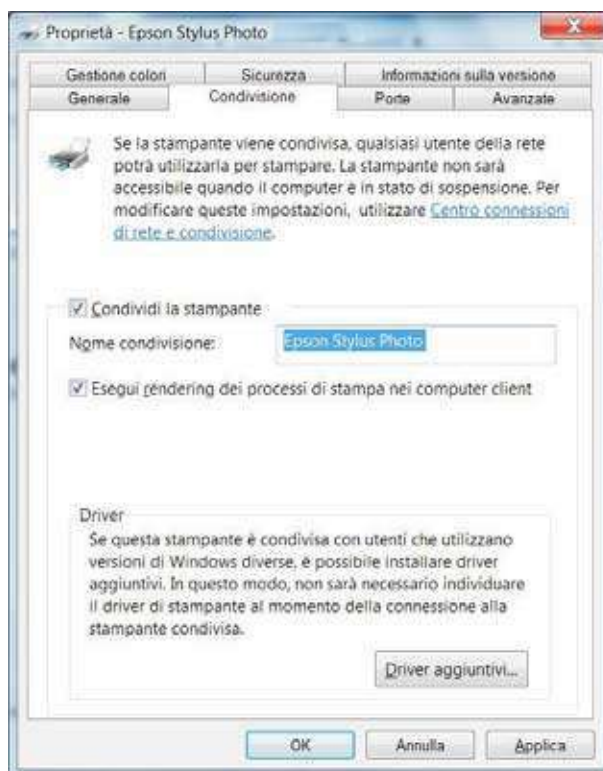
Fare clic su **Start** e poi su **Dispositivi e stampanti**.

Fare clic con il tasto destro del mouse sull'icona della stampante da condividere.

Scegliere **Proprietà stampante** dal menu di scelta rapida.



Nella finestra che si apre fare clic sulla scheda **Condivisione**. Fare clic sul pulsante **Modifica opzioni condivisione** e scegliere l'opzione **Condividi la stampante**, assegnando anche un nome logico con il quale la stampante può essere identificata come risorsa di rete. Confermare con **OK**.



L'icona della risorsa viene modificata aggiungendo la figura di due persone.



Per utilizzare una stampante di rete condivisa

1. Fare clic sul pulsante **Start**, scegliere **Dispositivi e stampanti** e fare doppio clic sul pulsante **Aggiungi stampante** nella barra in alto, oppure fare clic con il tasto destro del mouse all'interno della finestra e scegliere **Aggiungi stampante**.
2. Seguire le istruzioni visualizzate per la procedura di *Installazione guidata stampante*. All'interno di questa procedura si deve scegliere prima di tutto **Aggiungi stampante di rete** e quindi, a seconda dei casi, si può selezionare il nome della stampante dall'elenco delle stampanti disponibili in rete, oppure si può indicare direttamente il nome o l'indirizzo di rete della stampante.

Dopo aver completato l'installazione, sarà possibile utilizzare la stampante come se fosse fisicamente collegata al computer in uso.

AUTOVERIFICA

Domande da 7 a 10 pag. 120

Problemi da 19 a 24 pag. 121

**MATERIALI ONLINE****5. Condividere una cartella (lezione multimediale)**

DOMANDE

Interfaccia per l'utente

- 1 Quali di queste affermazioni, riferite all'uso del *mouse* in Windows, sono vere (V) e quali false (F)?

a) Il tasto destro premuto serve a trascinare gli oggetti	V	F
b) Il doppio clic con il tasto destro attiva un programma	V	F
c) Un clic con il tasto destro attiva un menu di scelta rapida	V	F
d) Il tasto sinistro premuto seleziona una parte di testo	V	F
e) Un clic con il tasto destro effettua la scelta all'interno di un menu	V	F
- 2 Quale tra queste definizioni corrisponde al termine *multitasking*?
 - a) Lo spegnimento automatico del monitor dopo un certo tempo di inattività.
 - b) L'apertura di un programma di trattamento del testo.
 - c) La possibilità di eseguire più applicazioni contemporaneamente.
 - d) La possibilità di usare filmati e suoni in un computer.
- 3 Quando sul desktop sono aperte più applicazioni, quale azione devi compiere per passare da un'applicazione all'altra?
 - a) Un clic sull'icona di *Computer*.
 - b) Un clic sul pulsante corrispondente nella *Barra delle applicazioni*.
 - c) Un clic sul pulsante *Start*.
 - d) Un clic su *Programmi* nel *Pannello di controllo*.
- 4 Quale menu del programma di editing dei testi contiene le scelte per copiare e incollare parti di testo?
 - a) File
 - b) Modifica
 - c) Formato
 - d) Visualizza

Cartelle e file

- 5 Quale tra queste funzioni deve essere attivata in Windows per gestire le operazioni tipiche del *file system* di un sistema operativo?
 - a) Computer
 - b) Pannello di Controllo
 - c) Impostazioni
 - d) Dati recenti
- 6 Associa ad ogni azione a sinistra la voce corrispondente a destra:

a) Cancellare un file	1) Rinomina
b) Cambiare il nome a un file	2) Nuovo
c) Cercare un file	3) Elimina
d) Creare una cartella	4) Invia a
e) Recuperare un file	5) Copia
f) Copiare un file su chiavetta USB	6) Taglia
g) Spostare un file	7) Ripristina
h) Copiare un file	8) Trova

Gestione del sistema e delle risorse

- 7 Completa le frasi seguenti utilizzando una tra le parole elencate alla fine della domanda.
- Per vedere le applicazioni attive si usa la nella parte bassa della scrivania.
 - Con il pulsante in alto a destra di una finestra si effettua la di un'applicazione.
 - Per raggruppare i file su disco si usa la
 - Per conoscere l'uso dei comandi e delle funzioni di Windows si usa la
 - Un'applicazione aperta sul desktop viene rappresentata con una
- cartella, periferica, Barra delle applicazioni, connessione, icona, guida in linea, risorsa, finestra, chiusura.**

- 8 Quali di queste affermazioni sono vere (V) e quali false (F)?
- Per riavviare il sistema dall'inizio senza spegnere fisicamente il computer si deve scegliere *Sospendi* V F
 - Per arrestare il sistema basta spegnere fisicamente il computer V F
 - Il *Pannello di controllo* si trova nel menu *Programmi* V F
 - Per creare una nuova cartella si può usare anche il menu di scelta rapida aperto con il tasto destro del mouse V F
 - I nomi dei file in Windows possono essere lunghi al massimo 8 caratteri V F
 - I file messi nel *Cestino* sono recuperabili anche dopo averlo svuotato V F
- 9 Quali tra le seguenti sono le motivazioni più importanti per l'uso di file compressi?
- Eliminare le immagini da un documento.
 - Occupare meno spazio sui supporti di memorizzazione.
 - Usare dimensioni più piccole per i caratteri di un testo.
 - Rendere più veloce la trasmissione dei file all'interno di una rete.
- 10 Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
- L'utente non può mettere in condivisione proprie risorse per gli altri utenti, perché è un compito dell'Amministrazione del sistema V F
 - La scelta *Everyone* consente di condividere una cartella con tutti gli utenti V F
 - Per utilizzare una stampante di rete condivisa, occorre scegliere *Aggiungi stampante* nel Pannello di controllo V F
 - Ogni computer collegato alla rete possiede un nome che lo identifica V F

PROBLEMI

Modalità operative

- Cambiare lo sfondo per il desktop scegliendolo tra quelli predefiniti di Windows.
- Controllare ed eventualmente regolare la data e l'ora di sistema utilizzando il *Pannello di controllo*.
- Avviare l'editor di testi, scrivere tre righe di testo e salvare il documento con il nome *Es1*.
- Attivare il programma di editing dei testi per scrivere l'elenco dei nomi dei compagni di classe.

Cartelle e file

- 5 Si deve spiegare con una nota scritta come si fa a copiare il file *Prova.doc* dalla cartella *Documenti* del disco fisso sulla chiavetta USB. Scrivere la nota usando l'editor di testi e salvarla in un file di nome *PerCopia.Txt*.
- 6 Visualizzare l'elenco dei file contenuti nella cartella *Documenti* prima in ordine alfabetico e poi in ordine di data.
- 7 Controllare quanto spazio è disponibile su un disco esterno, oppure su una chiavetta USB, che contiene già alcuni file.
- 8 Copiare il file *Leggimi* contenuto nella cartella *Windows* del disco C: sulla chiavetta USB.
- 9 Creare una cartella di lavoro all'interno della cartella *Documenti* assegnandole il nome *Esercizi*.
- 10 Cambiare il nome al file *Es1* con il nuovo nome *Exerc1* usando il menu di scelta rapida (pulsante destro del mouse).
- 11 Trovare la posizione di tutti i file con nome *Leggimi* presenti sul disco fisso.
- 12 Attivare la procedura per svuotare il cestino.
- 13 Determinare la sequenza delle operazioni necessarie per trovare, leggere e stampare un file di nome *Rete.txt* che si trova sul disco fisso.
- 14 Copiare, dopo averne identificato la posizione, il file *Eserc1.Txt* su una chiavetta USB. Quindi cambiare il nome di tale file in *EsOld.Txt*.
- 15 Creare un file con il nome *Prova1.doc*, contenente alcune righe. Successivamente cancellare il file e poi, utilizzando il *Cestino*, ripristinarlo come era prima dell'eliminazione.
- 16 Spostare dalla chiavetta USB il file *Prova2.doc* nella cartella *Documenti* del disco fisso.
- 17 Elencare tutti i file creati o modificati nell'ultimo mese utilizzando l'opzione *Ultima modifica* del programma di ricerca.
- 18 Usando il menu di scelta rapida (con il pulsante destro del mouse), creare una nuova cartella sulla chiavetta USB con il nome *Prima*; all'interno di essa creare una cartella di nome *Cartel2*.

Gestione del sistema e delle risorse

- 19 Utilizzando la *Guida in linea*, trovare le modalità per connettersi ad Internet.
- 20 Utilizzando la *Guida in linea*, trovare le procedure per creare sul desktop l'icona per il collegamento al programma *Excel*.
- 21 Avviare la procedura per installare una nuova stampante.
- 22 Scegliere due file dal disco e creare un unico file compresso per contenerli entrambi.
- 23 Controllare quali risorse sono disponibili nella rete a cui è collegato il computer.
- 24 Mettere in condivisione una cartella del disco del computer.



SCHEDA DI AUTOVALUTAZIONE

CONOSCENZE

- ☐ Apertura e chiusura delle sessioni di lavoro
- ☐ Significato di multitasking
- ☐ Caratteristiche generali dell'interfaccia delle applicazioni Windows
- ☐ Editing di testi
- ☐ Gestione delle risorse e dei file
- ☐ Compressione dei file
- ☐ Virus informatici
- ☐ Guida in linea
- ☐ Stampanti
- ☐ Copie di backup
- ☐ Condivisione di risorse in rete

ABILITÀ

- ☐ Saper avviare e spegnere il computer
- ☐ Verificare le informazioni sulle caratteristiche del computer
- ☐ Saper operare con l'interfaccia grafica (mouse, finestre, icone)
- ☐ Saper usare l'editor di testi
- ☐ Saper operare con cartelle e file
- ☐ Saper creare un file compresso e saper decomprimere un file
- ☐ Utilizzare un programma antivirus
- ☐ Saper usare la guida in linea
- ☐ Effettuare il backup
- ☐ Gestire una stampante
- ☐ Mettere in condivisione una risorsa di rete
- ☐ Utilizzare una risorsa condivisa nella rete



SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 622

Il Prompt dei comandi

1 La finestra Prompt dei comandi

Il **Prompt dei comandi** è una *shell*, cioè un interprete di comandi, disponibile nel sistema operativo Windows che consente di eseguire i comandi che vengono scritti su una linea del video (*linea di comando*). La shell esegue un comando attraverso un nuovo processo e fornisce i risultati in modalità testuale.

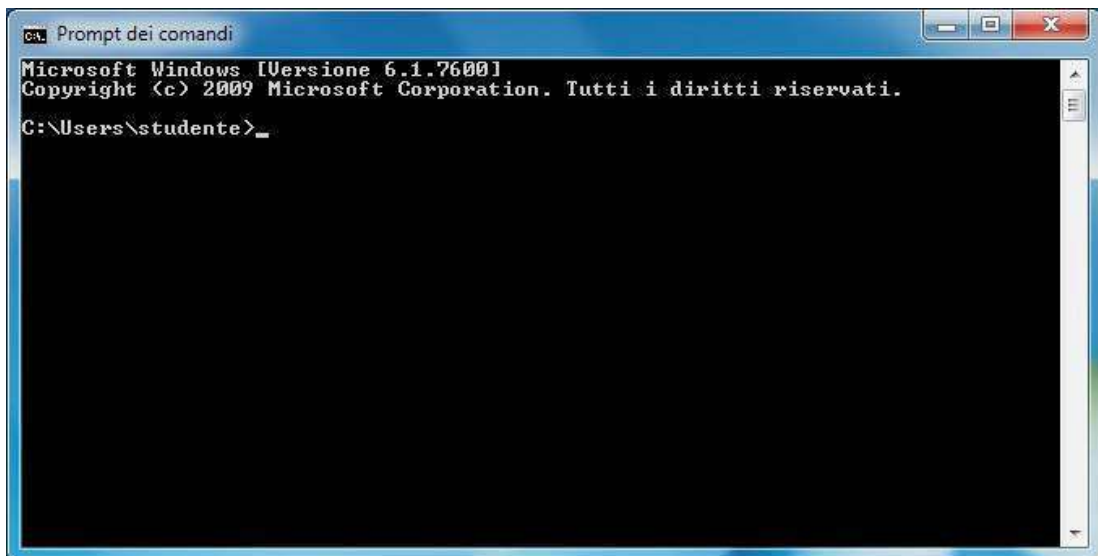
È la funzionalità che permette l'utilizzo dei comandi del sistema operativo MS-DOS, che era il sistema standard dei primi personal computer. Inoltre si possono attivare i comandi, in ambiente Windows, senza utilizzare l'interfaccia grafica con finestre, icone e mouse. Il prompt dei comandi è contenuto nel file **cmd.exe**.

Rispetto all'interfaccia grafica, l'uso del prompt di comandi rende più veloce l'attivazione delle funzionalità del sistema operativo, ma richiede la conoscenza delle parole chiave e della sintassi nella scrittura corretta delle linee di comando.

I comandi disponibili sono in numero limitato, ma l'utente può programmare script di comandi da eseguire in modalità *batch* (**file batch**, che hanno generalmente l'estensione **.bat**), come vedremo nei paragrafi successivi.

Il termine **prompt** (letteralmente *sollecito*) in informatica indica precisamente la sequenza di caratteri che compare sullo schermo quando la shell è pronta ad accettare un nuovo comando. Normalmente la sequenza contiene il nome dell'unità disco, il nome della home directory dell'utente e il segno di maggiore >, per esempio:

```
C:\Users\studente>
```



La finestra con la riga di comandi viene spesso indicata nei sistemi operativi anche come finestra di **console**.

Per aprire la finestra Prompt dei comandi

1. Fare clic su **Start** e poi clic su **Tutti i programmi**.
2. Nella cartella **Accessori**, scegliere **Prompt dei comandi**.

Per aprire la finestra Prompt dei comandi dalla casella di ricerca

1. Fare clic su **Start**.
2. Nella casella **Cerca programmi e file**, scrivere **cmd** e premere il tasto **Invio**.
In alternativa:
 1. Premere il tasto con il **logo di Windows** (in basso a sinistra sulla tastiera) insieme al tasto **R**.
 2. Nella finestra di dialogo che si apre, scrivere **cmd** nella casella **Apri** e poi fare clic su **OK**.

Queste modalità servono anche a lanciare l'esecuzione di un programma o di un applicativo da linea comandi. Per esempio, scrivendo nella casella **Apri**:

```
excel.exe
```

si apre il programma Excel.

Per chiudere la finestra Prompt dei comandi

Scrivere **exit** al prompt della linea di comando
oppure
fare clic sul pulsante di chiusura della finestra in alto a destra.

2 I file e le directory

I file sono identificati attraverso un **nome** simbolico, che può permettere di ricordarne il contenuto, e un'**estensione** che, di solito, ricorda il tipo di file oppure il programma applicativo con il quale è stato creato. Il nome viene separato dall'estensione tramite il carattere . (punto).

Nel sistema operativo DOS il nome del file doveva avere una lunghezza massima di 8 caratteri e l'estensione di 3 caratteri. Nei sistemi Windows i nomi e le estensioni dei file non sono soggetti a queste restrizioni e possono essere molto più lunghi.

Ci sono alcune estensioni di uso comune:

<i>nome</i> .EXE	programma eseguibile
<i>nome</i> .COM	file di comandi
<i>nome</i> .SYS	file di sistema
<i>nome</i> .BAT	file batch, ossia una sequenza di comandi o di programmi da eseguire
<i>nome</i> .BAK	copia di sicurezza di un file
<i>nome</i> .TXT	testo semplice leggibile con un qualsiasi editor di testi.

Se il file che interessa non è registrato sul disco sul quale si sta lavorando, occorre far precedere al nome del file la lettera indicante l'**unità** sulla quale si trova il file (A, B, C, D, F, Z), seguita dal segno : (due punti).

DRIVE

:

NOME DEL FILE

.

ESTENSIONE

Di norma le unità con i floppy disk sono identificate con le lettere A: e B:, l'unità con l'hard disk con la lettera C:, le unità CD-ROM e DVD con le lettere D: ed E:, le memorie USB e i dischi esterni con F:, G:. Le unità e i dischi di altri computer connessi alla rete locale, di solito, sono indicate con le ultime lettere dell'alfabeto, per esempio W: oppure Z:.

Per esempio D:LETTERA.DOC indica un file di nome LETTERA ed estensione DOC che si trova sul disco ottico inserito nell'unità D:.

Per indicare i file in un comando è spesso utile far ricorso ai **caratteri jolly**, che sono il punto interrogativo e l'asterisco.

Il punto interrogativo **?** posto all'interno del nome del file, o dell'estensione, indica che un qualsiasi carattere può occupare quella posizione nel nome o nell'estensione del file.

Per esempio: *Prove?.dat* indica tutti i file che iniziano con *Prove* seguito da un qualsiasi carattere prima del punto, e di estensione *stu* (*Prove1.dat*, oppure *Prove5.dat*, oppure *Provex.dat*.)

L'asterisco ***** inserito all'interno del nome del file o dell'estensione sta ad indicare che qualsiasi sequenza di caratteri può trovarsi in quella posizione all'interno del nome o dell'estensione del file.

Alcuni esempi:

MA*.* indica tutti i file che iniziano per MA, aventi un'estensione qualsiasi;

M*.C* indica tutti i file con nome che inizia per M ed estensione per C;

*M.*C indica tutti i file con nome che termina con M ed estensione che termina con C;

*.TXT indica tutti i file di qualsiasi nome, aventi l'estensione TXT;

. indica tutti i file di qualsiasi nome con qualsiasi estensione.

Dal punto di vista logico, i file sono organizzati su disco attraverso le directory e le sottodirectory. La **directory** è un file particolare che contiene le informazioni di un insieme di file raggruppati di norma secondo un criterio di omogeneità (i file di uno stesso utente oppure i file di una determinata applicazione). Di ogni file vengono ricordate le seguenti informazioni: il nome, le dimensioni, la posizione fisica sul disco, la data e l'ora di creazione o dell'ultimo aggiornamento.

La directory di lavoro nella quale si trovano i file utilizzati in una determinata fase di elaborazione si chiama **directory corrente**.

La directory principale è detta **root** (*directory radice*) perché è la radice di una struttura ad **albero** costituita dalle sottodirectory e dai file.

Per accedere a un qualsiasi file contenuto in una sottodirectory diversa dalla directory corrente, si indica il cammino all'interno dell'albero, cioè l'elenco di nomi delle sottodirectory, separati dal segno **** (*backslash*) che devono essere attraversate per ritrovare il file richiesto.

Per esempio, per accedere al file *Esercizio.txt* contenuto nella sottodirectory *Lavori*, che a sua volta è contenuta nella directory *Utenti*, che è stata creata all'interno della root, si deve indicare come nome del file la seguente scrittura:

```
\Utenti\Lavori\Esercizio.txt
```

Questa scrittura prende il nome di **pathname** (*nome con indicazione del cammino*). Il primo segno di **** indica la root.

Se il pathname inizia con il segno **** la ricerca inizia dalla root (**pathname assoluto**), altrimenti inizia dalla directory corrente (**pathname relativo**) e prosegue verso il basso nella struttura gerarchica delle directory.

Naturalmente dalla directory corrente si può tornare anche verso l'alto nella struttura gerarchica o direttamente alla directory root.

3 I comandi

I comandi sono costituiti da una **parola chiave** che ne ricorda la funzione, da uno o più **argomenti**, dal nome del disco o, in generale, dell'unità contenente i file, dal nome dei file, e da una o più **opzioni** precedute dal simbolo / (barra o *slash*), che servono a specificare un utilizzo più specifico del comando.

La sintassi generale dei comandi è la seguente:

parola chiave

argomenti

unità:

nomefile

/opzioni

La lettera dell'unità periferica è seguita dal segno : (due punti), le opzioni sono precedute dal segno /.

Eventuali opzioni multiple applicate allo stesso comando sono indicate con segni / separati, per esempio:

```
DIR /P /O
```

Alcuni comandi non riguardano i file, pertanto l'indicazione dell'*unità* e del *nome file* può non essere presente. Se non si specifica l'*unità*, si intende quella corrente, normalmente il disco C:. Inoltre nell'esecuzione dei comandi, il sistema operativo considera la tastiera come **standard input**, il video come **standard output** e come **standard error**, cioè come supporto per i messaggi di errore.

I comandi sono scritti su una riga del video (**linea comandi**) dopo la sequenza di caratteri detta **prompt**

```
C:\>
```

e sono eseguiti dopo aver premuto il tasto **Invio**.

C: significa che si sta lavorando sul disco fisso, \ rappresenta la directory *root*; il segno di >, seguito da un trattino di sottolineatura lampeggiante (detto **cursore**), sta ad indicare il punto del video dove l'utente deve digitare il comando.

Secondo le impostazioni predefinite del prompt, quando si cambia la directory corrente, il prompt cambia, specificando il pathname della directory nella quale si sta lavorando. Questa impostazione può essere modificata con il comando **PROMPT**.

L'utente può ottenere informazioni sui comandi utilizzabili tramite l'*help in linea*, attivato con il comando **HELP**. Il comando senza argomenti visualizza un elenco rapido dei comandi disponibili con una breve spiegazione della loro funzione:

```
HELP
```

La descrizione più dettagliata della sintassi di uno dei comandi elencati si ottiene scrivendo la parola **HELP** seguita dal nome del comando:

```
HELP nomecomando
```

In alternativa si può scrivere il nome del comando con l'opzione /?:

```
nomecomando /?
```

La finestra *Prompt dei comandi* non distingue la scrittura dei comandi con lettere maiuscole o minuscole, cioè non è *case sensitive*.

Durante la scrittura o la modifica dei comandi si possono usare i **tasti freccia** della tastiera: i tasti verso l'alto e il basso permettono di scorrere in avanti e indietro la sequenza dei comandi forniti in precedenza, mentre le frecce verso sinistra e destra consentono di spostare il cursore lungo la linea dei comandi, effettuando eventuali modifiche o aggiunte a righe di comando precedenti. Il tasto **Home** consente di posizionarsi all'inizio della linea comandi, il tasto **Fine** alla fine della linea stessa.

**MATERIALE ONLINE**

1. Editor di testi

4 I comandi per la gestione di directory e file

I comandi che operano su directory o file fanno riferimento ad essi secondo la notazione generale:

[unità:][percorso][nomefile]

Le parentesi quadre indicano l'opzionalità degli argomenti: *unità* o *percorso* possono essere omessi se il file si trova nell'unità corrente oppure nel percorso corrente; il *nomefile* è omesso quando i comandi operano su directory o sottodirectory.

Di seguito vengono elencate le funzioni svolte e la sintassi dei comandi di uso più comune.

• MKDIR

(*make directory*)

crea una nuova directory.

Esempio:

```
MKDIR \Contab
```

crea sul disco C: una nuova sottodirectory di nome *Contab* all'interno della directory root. Il comando può essere usato anche nella forma abbreviata **MD**.

```
MD \Contab
```

• RMDIR

(*remove directory*)

cancella una directory (la directory non deve contenere file).

Esempio:

```
RMDIR C:\Utenti\Contab
```

la sottodirectory *Contab* della directory *Utenti* viene eliminata dalla struttura delle directory (l'utente deve prima cancellare tutti i file in essa presenti). Il comando può essere usato anche nella forma abbreviata **RD**. Il comando con l'opzione **/S** rimuove l'intero albero di una directory.

• CHDIR

(*change directory*)

seleziona una nuova sottodirectory come directory corrente.

Esempi:

```
CHDIR \Utenti\Contab
```

consente di posizionarsi in una nuova directory di lavoro *Contab* che è una sottodirectory della directory *Utenti*.

```
CHDIR\
```

consente di tornare alla directory root.

```
CHDIR ..
```

consente di risalire alla directory immediatamente sopra la directory corrente nella struttura gerarchica delle directory.

```
CHDIR
```

senza parametri consente di visualizzare il nome della directory corrente.
Il comando può essere usato anche nella forma abbreviata **CD**.

• DIR

(*directory*)

elenca sul video i file contenuti nella directory corrente o nella directory specificata.

Esempi:

```
DIR D:
```

fornisce il catalogo dei file che si trovano nell'unità D:, per ciascuno viene elencato il nome, l'estensione, l'occupazione di spazio su disco (espressa in byte), la data di creazione e l'ora di creazione del file.

```
DIR /W
```

elenca solo i nomi dei file con l'estensione.

```
DIR /P
```

visualizza l'elenco completo, fermandosi dopo aver visualizzato una pagina di video (24 righe), per consentire una comoda consultazione quando l'elenco è lungo.

```
DIR /P /O
```

visualizza l'elenco completo in ordine alfabetico di nome con 24 righe per pagina di video.

```
DIR D:Q*.*
```

fornisce l'elenco dei file il cui nome inizia con la lettera Q e che stanno sull'unità D:

```
DIR C:Prova.*
```

elenca i file che stanno sul disco fisso C: e hanno il nome *Prova*, con qualsiasi estensione.

```
DIR C:*.*doc
```

elenca i file che stanno sul disco fisso che hanno l'estensione uguale a *doc* e con qualsiasi nome. Le diverse opzioni del comando DIR possono essere visualizzate con il comando che attiva l'help in linea:

```
DIR /?
```


• TREE

visualizza con uno schema grafico la struttura dell'albero delle directory di un pathname o di un disco.

Esempio:

```
TREE \Temp
```

Visualizza l'albero delle sottodirectory e dei file nella directory *Temp* che è una sottodirectory della root.

• TYPE

visualizza il contenuto di un file, di cui viene specificato il nome.

Esempio:

```
TYPE Rubrica.txt
```

consente di ottenere sul video il testo delle righe contenute nel file *Rubrica.txt*.

• COPY

copia un file in un altro file.

Il primo argomento del comando indica l'origine, il secondo la destinazione della copia.

Esempio:

```
COPY D:Clienti C:Anag
```

copia il file di nome *Clienti* che si trova sul disco in D: sul disco C: assegnandogli il nome *Anag*.

• DEL

(*delete*)

cancella dal disco il file di cui viene specificato il nome. Il comando può eliminare anche un insieme di file specificati tramite i caratteri jolly (? e *).

Esempio:

```
DEL Prova.*
```

cancella tutti i file di nome *Prova* e di qualsiasi estensione.

Per eliminare un file si può usare anche il comando **ERASE**.

• RENAME

consente di cambiare il nome a un file registrato su disco.

Il primo argomento del comando indica il vecchio nome, il secondo il nuovo nome.

Esempio:

```
RENAME Clienti Cliold
```

il file di nome *Clienti* assume il nuovo nome *Cliold*.

Il comando può essere usato anche nella forma abbreviata **REN**.

• MOVE

Sposta un file o una directory all'interno dell'albero delle directory. Questa operazione corrisponde al cambiamento del *pathname* di un file o di una directory, quindi il comando **MOVE** può essere usato anche per rinominare un file o una directory.

Il primo argomento del comando indica l'origine, il secondo la destinazione dello spostamento.

Esempi:

```
MOVE C:\Esercizi\Ese1 C:\Corretti\Problema1
```

sposta il file *Ese1* dalla directory *Esercizi* nella directory *Corretti* cambiando il nome in *Problema1*.

```
MOVE C:\Esercizi\Ese1 C:\Corretti
```

sposta il file *Ese1* dalla directory *Esercizi* nella directory *Corretti* mantenendone il nome.

```
MOVE C:\Esercizi C:\Esercizisvolti
```

cambia il nome della directory *Esercizi* con il nuovo nome *Esercizisvolti*.

• REPLACE

Sostituisce un file con un altro file. Il primo argomento del comando indica il percorso del file origine, il secondo la directory di destinazione della sostituzione.

Esempio:

```
REPLACE C:\Esercizi\Problema1 C:\Corretti
```

Il file *Problema1* che si trova nella directory *Esercizi* sostituisce il file con lo stesso nome che si trova nella directory *Corretti*.

• MORE

visualizza il contenuto di un file 24 righe per volta.

Viene usato per visualizzare i file che richiedono più videate. La sintassi generale del comando è la seguente:

```
MORE nomefile
```

Il comando manda al video le righe del file, facendo una pausa quando il video è stato riempito, e indicando nella parte bassa del video la parola *More* (in italiano *ancora*); per continuare con la videata successiva si deve premere la **barra spaziatrice**, per continuare riga per riga si usa invece il tasto **Invio**; per uscire senza vedere il resto si preme il tasto **Q**.

• SORT

ordina i dati contenuti in un file.

Normalmente il comando ordina alfabeticamente il contenuto di un file a partire dal primo carattere a sinistra di ogni riga. Il seguente comando

```
SORT Nomi
```

mette in ordine alfabetico le righe contenute nel file *Nomi* e visualizza il risultato.

• Ricerca di stringhe nei file

Ci sono due comandi per ricercare una o più stringhe in uno o più file, restituendo le linee nelle quali è stata trovata.

La differenza tra i due comandi consiste nella disponibilità di opzioni per raffinare la ricerca.

Il comando più semplice è **FIND** che ha la seguente sintassi generale:

```
FIND opzioni "stringa" nomefile
```

dove *stringa* indica la sequenza di caratteri da ricercare e *nomefile* può essere semplicemente il nome di un file nella directory corrente oppure il percorso di un file che si trova su un'altra unità o in un'altra directory.

Esempio:

```
FIND "Rossi" Rubrica.txt
```

visualizza le righe del file *Rubrica.txt* che contengono il cognome *Rossi*.

La stringa da cercare deve essere racchiusa tra virgolette.

Il comando **FINDSTR** svolge la medesima funzione, ma possiede molte più opzioni che possono essere visualizzate in modo completo con l'help in linea:

```
FINDSTR /?
```

Inoltre questo secondo comando può effettuare ricerche multiple con più stringhe.

La sintassi generale è analoga a quella del comando **FIND**:

```
FINDSTR opzioni stringhe nomefile
```

Per specificare più file su cui effettuare la ricerca si possono usare i caratteri jolly *?* e ***, oppure si può indicare il nome della directory che li contiene.

Per esempio il comando

```
FINDSTR "importo" *.dat
```

trova le righe che contengono la stringa *importo* nei file con estensione *dat*.

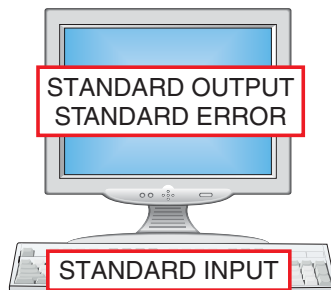


■ MATERIAL ONLINE

2. Attributi sui file

5 La ridirezione di input e output

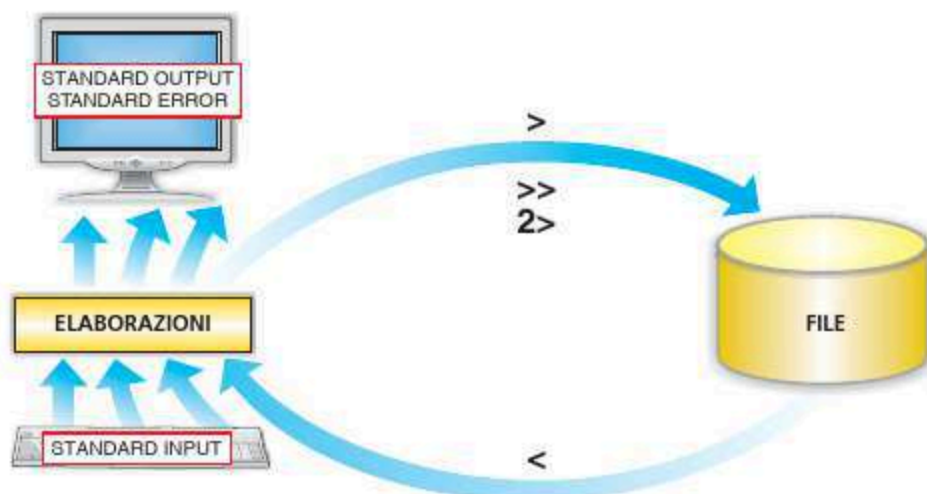
Un comando della *shell* opera normalmente leggendo i dati dalla tastiera, denominata come **standard input**, cioè l'unità standard dalla quale vengono acquisiti i dati, e scrivendo i risultati sul video, indicato come **standard output**, cioè l'unità del sistema alla quale vengono inviati i risultati dell'elaborazione. Il video serve anche per produrre verso l'esterno i messaggi di diagnostica del sistema in caso di errore e per questo motivo viene indicato come l'unità che funziona da **standard error**.



Se si vogliono modificare queste assegnazioni standard per l'input, l'output e l'error occorre indicare il nome del file alternativo, poiché dal punto di vista del sistema i tre standard sono considerati file. Questa operazione si chiama **ridirezione**.

Essa viene rappresentata con i seguenti caratteri speciali (metacaratteri):

- < considera il file indicato come lo standard input
- > invia lo standard output al file indicato
- >> appende l'output in coda al file indicato
- 2> ridirige lo standard error nel file indicato (il numero 2 è il tipo di descrittore esplicito per lo standard error, mentre 0 indica lo standard input e 1 lo standard output).



Esempi:

```
DIR *.txt >Testi
```

registra l'elenco dei file con estensione *txt* nel file *Testi*, anziché visualizzarlo sullo schermo.

```
DIR *.doc >>Testi
```

aggiunge l'elenco dei file con estensione *doc* in coda al file *Testi* senza sovrascrivere le righe già registrate.

```
SORT Nomi >Nomord
```

ordina alfabeticamente il file *Nomi* e registra l'elenco ordinato nel file *Nomord*.

Se il file *Prova.dat* non esiste nella directory corrente, il seguente comando:

```
DIR Prova.dat
```

produce sul video il nome della directory e poi il seguente messaggio di errore:

```
File non trovato
```

Con il comando:

```
DIR Prova.dat 2>errori
```

il messaggio di errore non compare sul video perché viene registrato nel file *errori*, dove potrà essere letto in un momento successivo (si può verificare il contenuto del file *errori* con il comando *TYPE errori*).

Volendo eliminare anche la visualizzazione del nome della directory si può usare il comando con la ridirezione dello standard output e dello standard error:

```
DIR Prova.dat >messaggi 2>errori
```

In questo modo sul video non compare alcun output del comando.

Output ed errori possono essere ridiretti anche su un file fittizio, avente il nome predefinito **NUL**, quando si vogliono fare ridirezioni senza registrare file nel disco; per esempio:

```
DIR Prova.dat 2>NUL
```

elimina la visualizzazione dei messaggi di errore e non li registra su un file fisico.

La ridirezione dell'input consente di fornire a un comando i dati di input acquisendoli da un file invece che dalla tastiera come avviene normalmente.

Si consideri il comando SORT senza argomenti:

```
SORT
```

non avendo specificato alcun file, la shell attende che l'utente inserisca i dati dallo *standard input* (tastiera); immettendo riga per riga, con il tasto *Invio* alla fine di ciascuna, i dati vengono acquisiti dal comando SORT; per interrompere l'inserimento si deve premere la combinazione di tasti **Ctrl + Z** e poi *Invio*; a questo punto il risultato dell'ordinamento viene mandato allo *standard output* (video).

Quindi il comando

```
SORT <Nomi
```

è una ridirezione dello standard input sul file *Nomi*, cioè i dati da ordinare vengono acquisiti dal file *Nomi*, anziché dalla tastiera.

Questa operazione è equivalente al comando visto nel paragrafo precedente per l'ordinamento di file

```
SORT Nomi
```

La seconda forma inoltre rende più rapida l'elaborazione.

In modo analogo il comando

```
SORT <Nomi >Nomord
```

ridirige sia lo standard input che lo standard output.

Un secondo esempio di ridirezione dell'input consiste nell'uso di un programma eseguibile *Prog1.exe* (realizzato compilando un sorgente scritto con un linguaggio di programmazione) che normalmente acquisisce i dati da tastiera: in modo diverso si può creare con l'editor un file di testo di nome *Dati.dat* contenente per ogni riga i dati da elaborare. In questo caso l'esecuzione si ottiene con la linea di comando:

```
Prog1.exe < Dati.txt
```

Questo modalità non richiede l'interazione con l'utente per fornire i dati al momento dell'esecuzione.

6 La pipeline

La **pipeline** è il modo più conveniente per utilizzare l'output di un comando come input di un altro comando senza creare file intermedi. Si chiama così perché letteralmente in inglese il termine *pipeline* significa tubatura oppure oleodotto: con questa metafora si vuole rappresentare una linea di comandi separati dal segno **|** (barra verticale, detta **pipe**, che si trova sulla tastiera a sinistra in alto sotto il tasto *Esc*) tale che l'output di un comando diventa l'input del comando successivo.

comando | comando | comando



Per esempio, se si vuole ottenere la lista dei file presenti nella propria directory e creati il 14 dicembre, si possono usare i due comandi:

```
DIR >Lista  
FINDSTR "14/12/" Lista
```

Usando una pipeline si ha il vantaggio di ottenere lo stesso risultato con una sola riga di comandi e senza la necessità di creare il file temporaneo *Lista*:

```
DIR | FINDSTR "14/12/"
```

L'output del comando *DIR* diventa l'input per il comando *FINDSTR*.

L'effetto della precedente riga di comando è il seguente: *DIR* genera l'elenco dei nomi dei file, uno per riga, ciascuno descritto anche da dimensione, data e ora. Anziché visualizzare la lista sullo schermo, la shell la manda in ingresso al comando successivo, cioè *FINDSTR*, che provvede ad estrarre le linee contenenti la sequenza di caratteri specificata ed a visualizzarle. Se la lista è molto lunga, si può usare un'altra pipeline per visualizzare una pagina per volta, attraverso il comando *MORE*, nel modo seguente:

```
DIR | FINDSTR "14/12/" | MORE
```

L'output di *FINDSTR* viene così ulteriormente elaborato nel modo desiderato.

Per mezzo delle pipeline è possibile quindi effettuare operazioni molto complesse con una singola riga di comandi formata anche da più di due comandi successivi separati dai segni di *pipe*.

Esempi

```
DIR | SORT /+13
```

lista i file della directory corrente in ordine crescente di ora di creazione.

```
DIR C:\Esercizi /S /B | FINDSTR "arch" | MORE
```

produce l'elenco senza intestazione (*/B*) dei nomi di file che si trovano nella directory *Esercizi* o nelle sue sottodirectory (*/S*) e che contengono nel nome la stringa *"arch"*; la visualizzazione avviene per pagine (24 righe per volta).

```
TREE C:\ | MORE
```

visualizza la struttura dell'albero delle directory del disco C: per pagine di video.

```
FINDSTR "Rossi" Fatture.txt | SORT
```

produce l'elenco ordinato delle righe del file *Fatture.txt* che si riferiscono al cliente *Rossi*.

```
FINDSTR /? |MORE
```

visualizza l'help in linea del comando FINDSTR per pagine di video.

```
HELP | MORE
```

visualizza l'elenco dei comandi, con una breve descrizione, per pagine di video.

Una pipeline può anche essere utilizzata con una ridirezione dell'output per costruire comandi complessi; si osservi che in questo caso l'operazione di ridirezione deve essere in coda alla riga di comando. Per esempio:

```
DIR /B | FINDSTR "arch" > ListaArch.txt
```



GESTIONE DEL SISTEMA E DELLE PERIFERICHE

In questo paragrafo vengono illustrati alcuni tra i comandi del sistema operativo Windows che riguardano la gestione del sistema di elaborazione, delle sue periferiche e il controllo dello stato dei processi. Anche se richiamano operazioni che sono più propriamente di competenza dell'*Amministratore del sistema*, essi risultano spesso utili anche per l'utente generico. Si deve osservare, comunque, che alcuni di questi comandi richiedono esplicitamente i privilegi di *Amministratore*.

- **DATE**

consente di visualizzare o modificare la data del computer (giorno, mese, anno).

- **TIME**

consente di visualizzare o modificare l'ora del computer (ore, minuti, secondi e centesimi di secondo)

- **FORMAT**

serve a preparare un supporto di memoria di massa per essere usato sul computer. Esempio:

```
FORMAT F:
```

viene formattato il disco che si trova nell'unità F:

- **VOL**

visualizza l'etichetta di volume e il numero seriale di un disco (fisso o esterno). Esempio:

```
VOL C:
```

visualizza le informazioni del disco C:

- **DISKCOPY**

copia il contenuto di un disco su un altro disco.

Esempio:

```
DISKCOPY A: B:
```

copia il disco che si trova nell'unità A: (originale) sul disco che si trova nell'unità B: (disco destinazione).

- **CHKDSK**

controlla un disco e visualizza il relativo rapporto sullo stato dei file e delle cartelle, oltre ad informazioni su capacità del disco, spazio occupato, numero di file e cartelle.

Esempio:

```
CHKDSK E:
```

effettua un controllo della memoria esterna USB identificata con la lettera E:

Le informazioni sulla **configurazione del sistema** di elaborazione possono essere visualizzate con i seguenti comandi:

- **SYSTEMINFO**

visualizza la configurazione e le proprietà delle componenti del sistema di elaborazione in uso.

- **VER**

visualizza la versione del sistema operativo in uso.

- **HOSTNAME**

visualizza il nome del computer in uso. È un comando senza argomenti.

- **IPCONFIG**

visualizza l'indirizzo IP del computer in uso.

Utilizzato senza parametri

```
IPCONFIG
```

visualizza, per ogni scheda presente nel computer e associata al protocollo TCP/IP, l'*indirizzo IP*, la *subnet mask* e il *gateway* predefinito.

Il seguente comando

```
IPCONFIG /ALL
```

visualizza le informazioni in modo completo, in particolare per ogni scheda di rete è possibile ottenere l'indirizzo fisico (*MAC address*).

- **PING**

verifica la connessione ad un altro computer della rete di cui si specifica l'indirizzo; per esempio:

```
PING 192.168.1.2
```

Con questo comando viene inviato un pacchetto di dati di prova al computer avente l'indirizzo IP indicato come parametro del comando: se non si ottiene una risposta entro pochi secondi, significa che l'altro computer non è connesso alla rete. Ovviamente l'esecuzione senza successo di questo comando potrebbe anche dipendere dal fatto che il proprio computer non è correttamente connesso alla rete.

Il comando PING seguito da un indirizzo Internet consente di ottenere l'indirizzo IP del server Internet associato nel DNS (*Domain Name System*); per esempio:

```
PING www.google.it
```

verifica la possibilità di connessione al server *www.google.it* e visualizza il suo indirizzo IP.

7 I file batch

La shell è in grado di mandare in esecuzione comandi contenuti in file caratterizzati dalle estensioni COM, EXE e BAT. I file dei tipi COM ed EXE contengono codice in linguaggio macchina e, una volta caricati in memoria, possono essere immediatamente eseguiti.

I file di tipo **BAT**, invece, sono file di testo che contengono sequenze di comandi di shell, perciò l'interprete dei comandi li tratta come se rispondesse a input da tastiera.

I file di tipo BAT, spesso chiamati **file batch** (o anche **script**), costituiscono la maniera più pratica di automatizzare sequenze di comandi che vengono adoperate spesso oppure che sono lunghe da eseguire.

Nel primo caso si ottiene un risparmio di battute, e quindi di errori di battitura, mentre nel secondo l'uso di uno script permette di non attendere il termine di ciascun comando per impartire il successivo, ottimizzando il tempo totale di esecuzione della sequenza di comandi. Inoltre il lavoro può essere eseguito in modo non interattivo (modalità *batch*) senza la necessità della presenza dell'utente.

Si voglia per esempio costruire un batch file di nome *Copia.bat* che svolga le seguenti funzioni: copiare alcuni file dal disco fisso all'unità E: (per esempio una memoria USB) e cancellare in seguito i file copiati per motivi di segretezza.

```
@ECHO OFF
CLS
ECHO Introdurre il disco nell'unità E:
PAUSE
COPY C:\Docum\Testo.doc E:\File1
COPY C:\Tabelle\Vendite.tab E:\File2
COPY C:\Lettere\Lettera.txt E:\File3
COPY C:\Fatture\Luglio E:\File4
DEL C:\Docum\Testo.doc
DEL C:\Tabelle\Vendite.tab
DEL C:\Lettere\Lettera.txt
DEL C:\Fatture\Luglio
```

Il comando può essere eseguito dalla linea comandi scrivendo:

```
Copia.bat
```

o, più semplicemente:

```
Copia
```

Con questo batch file, è possibile chiedere al sistema l'esecuzione dei comandi in esso contenuti nello stesso ordine con il quale sono stati scritti nel testo del file.

Si noti l'istruzione **@ECHO OFF** che serve a disattivare la ripetizione sullo schermo del comando in esecuzione; il simbolo **@** disattiva anche la visualizzazione del comando *echo off* stesso.

Il comando **CLS** cancella lo schermo e il comando **ECHO** seguente visualizza un messaggio informativo.

Il comando **PAUSE** sospende l'esecuzione che può essere riattivata premendo un tasto qualsiasi.

I comandi successivi eseguono la copia dei file dal disco fisso all'unità esterna e la successiva cancellazione dei file copiati.

Un secondo esempio mostra il testo di un batch file che serve a ripristinare i file su disco fisso:

```
COPY E:\File1 C:\Docum\Testo.doc
COPY E:\File2 C:\Tabelle\Vendite.tab
COPY E:\File3 C:\Lettere\Lettera.txt
COPY E:\File4 C:\Fatture\Luglio
```

Il file batch *Copia.bat* può essere migliorato introducendo le seguenti righe dopo la sequenza dei comandi COPY.

```
DIR E:\
PAUSE
```

Questo fa sì che il contenuto dell'unità E: venga visualizzato e l'esecuzione sospesa prima di cancellare i file sul disco fisso. Se si constata che uno o più file non sono stati copiati si può interrompere l'esecuzione e controllare che cosa sia andato male.

All'interno di uno script è possibile in qualche misura il controllo del flusso di esecuzione, vale a dire che si può alterare la sequenza con cui i comandi vengono eseguiti, a seguito del verificarsi o meno di certe condizioni. Per esempio si può controllare che un comando sia stato eseguito correttamente prima di passare al successivo, oppure si può verificare che un file esista prima di eseguire operazioni che lo riguardano.

Vediamo qualche esempio:

```
IF NOT EXIST Lettera.doc GOTO ERRORE
COPY Lettera.doc C:\Archivio
DEL Lettera.doc
GOTO FINE
:ERRORE
ECHO Errore: il file non esiste
:FINE
```

Quando la shell esegue questo file inizia con la verifica, richiesta con la prima riga, dell'esistenza di un file e prende una decisione di conseguenza: se il file non esiste (**IF NOT EXIST**) esegue il comando **GOTO ERRORE**, in modo che il prossimo comando da interpretare sia quello che segue l'etichetta *ERRORE*, indicata entro il file con *:ERRORE* (con i due punti iniziali). Questo comando è **ECHO frase**, che causa la visualizzazione di *frase* sul video.

Se invece il file esiste, l'interpretazione prosegue con la riga successiva all'IF, che ricopia il file *Lettera.doc* nella directory *C:\Archivio*, quindi con **DEL Lettera.doc**, che cancella la versione originale del file; infine salta all'etichetta *FINE*, anch'essa indicata entro il file con il simbolo *:*

(due punti) seguito dal nome dell'etichetta. Oltre a IF NOT EXIST esiste il controllo complementare, cioè **IF EXIST**.

Durante l'esecuzione dello script, tutti i comandi in esso contenuti vengono visualizzati: se si desidera evitare questo, si può introdurre nella prima riga il comando **@ECHO OFF** che appunto sopprime l'eco di ogni comando a video.

Desiderando ripristinarlo, si usa il comando **ECHO ON**.

Per alcuni comandi, è possibile controllare se la loro esecuzione è terminata in modo normale, oppure con errori, attraverso la struttura **IF ERRORLEVEL**. Questo è reso possibile dall'uso del codice di uscita del programma avente la seguente sintassi generale:

```
IF ERRORLEVEL n comando
```

dove *n* è un numero intero maggiore o uguale a zero.

Il nome **ERRORLEVEL** definisce una variabile predefinita della programmazione batch (**variabile di ambiente**).

Questa forma di controllo verifica se il codice di uscita è stato posto ad un numero pari a *n* o superiore, e in caso positivo la shell manda in esecuzione il comando specificato. Un valore 0 (zero) indica che il comando è terminato con successo, perciò in generale:

```
IF ERRORLEVEL 1 GOTO ERRORE
```

causa il salto all'etichetta *ERRORE* se il comando è terminato in modo anomalo; infatti qualsiasi codice di uscita diverso da zero viene rilevato da questo controllo.

Per determinare quale errore sia effettivamente accaduto, bisogna controllare tutti i codici in ordine decrescente, in modo che si abbia riscontro positivo solo al codice effettivamente trasmesso.

Se è noto che un certo comando può terminare con un codice compreso tra 0 e 5, occorre controllare i codici nell'ordine da 5 a 1: se il codice di uscita è 3, i controlli su 5 e 4 sono negativi, mentre è positivo quello su 3.

```
.....
IF ERRORLEVEL 5 GOTO ERRORE5
IF ERRORLEVEL 4 GOTO ERRORE4
IF ERRORLEVEL 3 GOTO ERRORE3
IF ERRORLEVEL 2 GOTO ERRORE2
IF ERRORLEVEL 1 GOTO ERRORE1
:TUTTOOK
...
GOTO FINE
:ERRORE1
...
GOTO FINE
:ERRORE2
...
GOTO FINE
...
:ERRORE5
...
GOTO FINE
:FINE
```

Non tutti i comandi usano il codice di uscita, per esempio il comando DEL non lo adopera e se si cerca di cancellare un file inesistente, che costituisce evidentemente una situazione anomala, non si ha modo di verificarlo per mezzo di ERRORLEVEL.

Altri comandi usano più codici di uscita, per esempio FORMAT è in grado di comunicare diversi eventi anomali che possono verificarsi durante la sua esecuzione per mezzo di diversi codici di uscita: il loro uso consente di scrivere file batch più raffinati, in grado di prendere le decisioni giuste in ogni circostanza.

Gli script sono in grado di trattare **parametri di chiamata**, ovvero stringhe fornite in ingresso insieme al comando.

Per esempio, lo script seguente accetta in ingresso i nomi di due file e ne visualizza il contenuto, uno dopo l'altro e una pagina per volta.

```
@ECHO OFF
TYPE %1 | MORE
PAUSE
TYPE %2 | MORE
```

Dopo la visualizzazione del primo file, lo script si ferma e attende che l'utente prema un tasto per continuare (comando **PAUSE**).

Scrivendo un file *Scrivi.bat* contenente i comandi specificati, se si digita

```
Scrivi Primo.txt Secondo.txt
```

si ottiene l'output dei due file sul video.

Infatti %1 assume il valore corrispondente alla prima stringa che segue il nome del comando e %2 quello della seconda, perciò i due comandi entro lo script vengono espansi in:

```
TYPE Primo.txt | MORE
PAUSE
TYPE Secondo.txt | MORE
```

Il comando **SET** assegna un valore a una variabile (*SET nomevariabile = valore*). Per indicare il valore di una variabile si usa la notazione *%variabile%*.

Per controllare l'uguaglianza tra variabili, o tra variabili e stringhe, si usa l'operatore con il doppio segno == per il confronto:

```
IF %variab%==%variab2% comando
IF NOT %variab%==%variab2% comando
IF %variab%==stringa comando
IF NOT %variab%==stringa comando
```

Variabili delle quali si deve controllare il contenuto sono per esempio quelle passate allo script quando lo si esegue.

```
IF %1==C% comando
IF NOT %2==PIPP0 comando
```

Si osservi che viene usato solo un simbolo di percentuale per espandere il contenuto delle variabili passate allo script.

Se nel secondo caso lo script contenente il controllo viene lanciato senza almeno due stringhe, il controllo dà luogo ad un errore, perché %2 viene espanso in una stringa nulla, contenente zero caratteri, e quindi la riga assume l'aspetto seguente

```
IF NOT ==PIPP0 comando
```

che contiene un errore di sintassi, perché non si può mettere il doppio segno == senza un argomento che lo preceda. È perciò opportuno usare la forma seguente:

```
IF NOT "%2"=="PIPPO" comando
```

In tal modo, se %2 è la stringa nulla, l'espansione genera la riga seguente:

```
IF NOT ""=="PIPPO" comando
```

che sintatticamente è corretto.

Usando questo tipo di controlli si possono creare script in grado di eseguire compiti diversi, in dipendenza da diversi valori di una stringa in ingresso.

Lo script che segue, che chiamiamo *Uti.bat*, richiede che vengano forniti due o tre parametri in ingresso, il primo dei quali determina il tipo di operazione da eseguire, i successivi sono nomi di file sui quali operare.

All'interno del testo sono state introdotte alcune righe di commento per spiegare le funzionalità del file batch. Le righe che iniziano con la parola chiave **REM** contengono i commenti.

```
@ECHO OFF
REM Gestione di file

REM Variabile per il controllo del tipo di errore
SET TIPO_ERR=0

IF "%1"=="c" GOTO COPIA
IF "%1"=="m" GOTO MUOVI
IF "%1"=="d" GOTO CANCELLA
IF "%1"==" " GOTO ISTRUZIONI

REM Istruzioni per copiare un file in un altro
:COPIA
IF "%3"==" " GOTO ISTRUZIONI
REM Se uso c devo dare anche 2 nomi di file

IF EXIST %3 GOTO ERRORE
SET TIPO_ERR=1
IF NOT EXIST %2 GOTO ERRORE
COPY %2 %3
GOTO FINE

REM Istruzioni per spostare un file in un altro
:MUOVI
IF "%3"==" " GOTO ISTRUZIONI
REM Se uso m devo dare anche 2 nomi di file

IF EXIST %3 GOTO ERRORE
SET TIPO_ERR=1
IF NOT EXIST %2 GOTO ERRORE
MOVE %2 %3
GOTO FINE
```

```
REM Istruzioni per cancellare un file
IF "%2"==" " GOTO ISTRUZIONI
REM Occorre un nome di file
:CANCELLA
SET TIPO_ERR=1
IF NOT EXIST %2 GOTO ERRORE
DEL %2
GOTO FINE

REM Se il comando viene eseguito con i parametri
REM sbagliati, vengono visualizzate le istruzioni per l'uso
:ISTRUZIONI
ECHO Istruzioni per l'uso del comando Uti:
ECHO Uti c File1 File2  copia File1 in File2 se File2 non esiste
ECHO Uti m File1 File2  muove File1 in File2 se File2 non esiste
ECHO Uti d File1  cancella File1
GOTO FINE
:ERRORE
IF %TIPO_ERR%==0 ECHO Attenzione, il File %3 esiste gia'
IF %TIPO_ERR%==1 ECHO Attenzione, il File %2 non esiste

:FINE
```

Come si può vedere dalle istruzioni per l'uso, lo script permette di copiare un file in un altro, cancellando o no l'originale, oppure di cancellare un file.

Vengono eseguiti alcuni controlli, di correttezza formale e di possibilità di esecuzione. Se la forma non è rispettata, cioè se i parametri non sono corretti, vengono visualizzate le istruzioni per l'uso, mentre se un'operazione non può essere eseguita compare un messaggio di errore.

Lo script riconosce due tipi di errore: l'esistenza del file di destinazione e la non esistenza del file da copiare o cancellare. La gestione è affidata alla variabile *TIPO_ERR*, che vale 0 all'inizio e durante il controllo su %3, e viene posta ad 1 subito prima del controllo su %2. In questo modo, se si ha effettivamente un errore si può controllare il valore di *TIPO_ERR* per visualizzare il messaggio corretto.

In sintesi, gli script permettono di eseguire sequenze di comandi con un certo grado di sicurezza in modo automatico, a condizione che si prevedano le circostanze anomale che possono verificarsi, soprattutto se lo script deve essere eseguito in modo *unattended*, cioè senza la sorveglianza continua di un operatore. La corretta gestione dei codici di uscita e delle condizioni di esistenza dei file può impedire di causare danni al sistema, o comunque di eseguire operazioni sbagliate, mentre l'uso delle variabili permette di scrivere script più versatili e potenti.

Il seguente esempio presenta un programma batch che può essere utilizzato per installare su disco fisso i programmi per la gestione della contabilità, prelevandoli dal CD che si trova nell'unità D:.

Il programma sarà registrato sullo stesso CD contenente il software con il nome *Installa.bat* e la sua esecuzione potrà essere attivata scrivendo da linea comando:

```
D:Installa
```


Il testo del file *Installa.bat* è il seguente:

```
@ECHO OFF
CLS
ECHO  INSTALLAZIONE SOFTWARE CONTABILITÀ
ECHO          CONTAB
ECHO  (VERS. 1.0)
ECHO  -----
ECHO  -----
ECHO  Con questa installazione CONTAB
ECHO  viene creato (o sostituito)
ECHO  nella directory del disco fisso C:\CONTAB
ECHO  se non vuoi proseguire premi CTRL + C
ECHO  -----
PAUSE
ECHO      prego attendere ....
C:
CD\
COPY D:con*.* C:\contab >NUL
COPY D:Inizio.bat C:\ >NUL
CD\
ECHO  -----
ECHO      Installazione terminata
ECHO  -----
ECHO  -----
ECHO  PER INIZIARE, DIGITARE:  Inizio
ECHO  ON
```

La ridirezione **>NUL** nel comando *Copy* serve ad eliminare la visualizzazione dei nomi dei file che vengono via via copiati dal CD.



MATERIALI ONLINE

3. Comandi avanzati per i file batch

4. File batch per le operazioni aritmetiche

PROBLEMI

Comandi

- 1 Ritornare alla directory root trovandosi nella sottodirectory CLIENTI.
- 2 Elencare i file che stanno sul disco F: nella directory BETA e che hanno il nome di 4 caratteri ed estensione qualsiasi.
- 3 Scrivere i comandi per visualizzare sul video:
 - l'elenco dei file il cui nome inizia con la lettera A
 - l'elenco dei file che hanno estensione TXT
 - l'elenco dei file che hanno il carattere 1 al terzo posto nel nome.
- 4 Controllare se il file CONTI è contenuto nella sottodirectory AZ00 del disco C:, indicando il suo pathname assoluto.
- 5 Creare una directory NUOVA nella root trovandosi nella sottodirectory ANAGRA.
- 6 Cancellare dalla directory COPIA del disco C: i file che iniziano per PRO e hanno l'estensione che inizia con P.
- 7 Cancellare tutti i file che iniziano con P3 e hanno estensione BAK.
- 8 Spostarsi dalla root del disco C: alla sottodirectory ARCHIVI della directory CONTAB; indicare poi due modi possibili per tornare alla root.
- 9 Cambiare il nome al file ARTICOLI.DAT nel nuovo nome PRODOTTI.OLD dopo averlo copiato dalla directory MAGAZ del disco C: sull'unità E:.
- 10 Copiare il file PROVA1 della directory COPIA del disco C: sulla chiavetta USB, denominata con F:, nella directory TRASF.
- 11 Copiare tutti i file con estensione DOC, che stanno nella directory TESTI del disco C:, sul disco D: nella directory T01.
- 12 Scrivere il comando per copiare nella directory PROVA2 tutti i documenti Word presenti nella directory \DOCUMENTI del CD identificato come unità D:.
- 13 Controllare la data e l'ora del computer, e correggerle nel caso siano errate.
- 14 Cancellare la directory DOCUMENTI del disco F:.
- 15 Con l'opportuno comando, attivare l'aiuto in linea.
- 16 Scegliere un file di testo dal disco fisso del computer e visualizzare il suo contenuto sul video.
- 17 Elencare i file che hanno nel nome in seconda posizione la lettera A e nell'estensione in terza posizione la lettera M.
- 18 Cambiare il nome al file INTESTA della directory DISCO nel nuovo nome PRIMA.

- 19 Nell'albero delle directory di un disco G: ci sono le directory \ALFA e \ALFA\BETA. Quale comando occorre dare per cancellare la directory ALFA dal disco G:?
- 20 Creare una nuova directory sull'unità F: di nome PROVA2 e dentro di essa la directory ARCHIV2.
- 21 Cancellare i file VIDEO2 e STUD2, che si trovano rispettivamente nelle sottodirectory TERZAA e TERZAB, usando i pathname assoluti.
- 22 Verificare che i file del problema precedente siano stati cancellati e controllare lo spazio disponibile su disco dopo questa operazione.
- 23 Scegliere un file dal disco fisso del computer e copiarlo nella directory principale di una chiavetta USB attribuendogli lo stesso nome. Con l'opportuno comando, cambiare poi il nome al file in PROVA.DOC.
- 24 Scelta una directory del disco fisso, con un solo comando e facendo uso dei caratteri jolly, copiare su un'unità esterna G: i file presenti nella directory e che hanno estensione EXE.
- 25 Scrivere il comando per ottenere sul video il contenuto del file CLIENTI.TXT, 24 righe per volta.

Ridirezioni e pipeline

- 26 Elencare i file con estensione .DOC e con la seconda lettera del nome uguale a R; si vuole ottenere l'output del comando nel file DocumentiR.
- 27 Produrre in output l'elenco dei file creati o modificati nell'anno scorso.
- 28 Elencare i file modificati o creati nel mese di febbraio, visualizzandoli 24 righe per volta.
- 29 Un file di nome STUD2 nella directory ARCHIV2 contiene un elenco di nomi di studenti con le rispettive età: verificare se nel file ci sono studenti di 26 anni e, se ci sono, elencarli in ordine alfabetico.
- 30 Scrivere la linea di comandi che consente di visualizzare i nomi dei file della directory corrente che sono stati creati nell'anno corrente e, all'interno di questi, quelli con estensione SYS.
- 31 Scrivere il comando per memorizzare in un file l'elenco alfabetico dei file creati nel giorno 2 febbraio contenuti nella directory \LAVORI.
- 32 Ridirigere la visualizzazione della data di oggi su un file di nome GIORNO.
- 33 Visualizzare i file del disco C: che sono stati creati o modificati due anni fa.
- 34 Un file CLIENTI contiene un elenco di cognomi e nomi: ordinare alfabeticamente il file creando un nuovo file ordinato di nome CLIORD.
- 35 Controllare se tra i nomi dei clienti del problema precedente esiste il cliente di Cognome = ROSSI.

- 36** Visualizzare la lista ordinata dei file presenti nella directory UTILITY.
- 37** Scrivere una pipeline che consenta di visualizzare soltanto il numero dei byte ancora disponibili su un disco.
- 38** Visualizzare l'elenco ordinato dei soli nomi dei file contenuti nella directory F:\LAVORI.
- 39** Scrivere il comando che consente di ottenere l'elenco alfabetico dei file con estensione .TXT creati o modificati il 4 maggio.
- 40** Scrivere su un nuovo file l'elenco dei file che hanno una B nella quarta posizione del nome e hanno estensione EXE.
- 41** Produrre la lista ordinata dei file che iniziano con R ed estensione qualsiasi creati nello scorso anno (utilizzare l'opzione /O del comando *DIR*).
- 42** Comunicare il contenuto del file NOMI.DAT su standard output.
- 43** Registrare la data di oggi sul file di nome ODIERNA.
- 44** Cercare in un file le righe che contengono la parola NAPOLI visualizzandole 24 righe per volta.
- 45** Elencare in modo ordinato la lista dei file che iniziano con C e con estensione qualsiasi, creati nel mese di novembre.

File batch

- 46** Creare il file batch C:\PROVA.BAT che soddisfi alle seguenti richieste:
 - disabilitare l'output su video delle righe del file
 - creare nella root dell'unità G: due sottodirectory TESTI e VARIE
 - copiare nella sottodirectory TESTI i file con estensione TXT contenuti nella sottodirectory C:\ESERC.
 - creare nella sottodirectory VARIE un file di nome RADICE.TXT contenente la lista dei file attualmente presenti nella root del disco C:.
- 47** Creare il file batch VIDEO2 che consenta di posizionarsi nella directory ARCHIV2, da qualsiasi directory ci si trovi, e di fare l'elenco dei file in essa contenuti prima su video e poi su un file ELENC2.
- 48** Scrivere un file batch che produca la lista dei file contenuti nella directory C:\PROGRAM che hanno una certa estensione in byte fornita dall'utente al momento dell'esecuzione del comando batch.
- 49** Costruire il file batch che consenta di fare le seguenti operazioni:
 - copiare il file FORNIT.DAT, che si trova nella directory ANAGRA, nella directory COPIA del disco D:.
 - cambiare il nome del file CONTAB.DAT della directory AZ101 in CONTI.DAT
 - cercare nel file MOVIM.DAT le righe che contengono la parola fornita dall'utente al momento dell'esecuzione del file batch, visualizzandole 24 righe per volta.

4

parte terza

La programmazione

Progettazione degli algoritmi

OBIETTIVI DI APPRENDIMENTO

In questo capitolo conoscerai il concetto di algoritmo e saprai riconoscere le caratteristiche fondamentali delle istruzioni che compongono un algoritmo.

Applicherai i principi della programmazione strutturata per costruire algoritmi ben ordinati attraverso le strutture di controllo. Imparerai inoltre a rappresentare gli algoritmi utilizzando

la pseudocodifica e i diagrammi a blocchi. Sarai in grado di organizzare algoritmi complessi attraverso sottoprocedure e funzioni e potrai riutilizzare gli stessi sottoprogrammi assegnando diversi valori ai parametri.

Modello del problema
Dati e azioni
La metodologia di lavoro
Definizione di algoritmo
Algoritmo ed esecutore
Acquisire e comunicare i dati
Gli operatori
Strumenti per la stesura di un algoritmo
L'individuazione dei dati di un problema
La macchina di Turing
Le strutture di controllo
Teorema di Böhm-Jacopini
La struttura alternativa
La ripetizione
Le strutture derivate
La ripetizione precondizionale
La ripetizione con contatore
La struttura di scelta multipla
Organizzazione degli algoritmi
Procedure con parametri
Funzioni
Funzioni ricorsive

1 Modello del problema

Nel primo capitolo è stato introdotto il concetto di **modello** come rappresentazione schematica di un particolare aspetto della realtà. In particolare un modello permette, attraverso meccanismi di formalizzazione, di individuare gli elementi principali della realtà osservata e di trattare tali elementi come entità astratte.

Queste entità devono avere due caratteristiche fondamentali:

- devono essere registrabili in una memoria
- devono essere messe in relazione tra loro.

Il trasferimento di un problema sul sistema di elaborazione viene indicato nel linguaggio informatico con il termine **implementazione**.

Si parla di implementazione di un problema quando la sua soluzione viene affrontata attraverso la costruzione di un modello formalizzato, in modo tale che il problema possa essere rappresentato e gestito con un sistema di elaborazione automatica.

L'automazione di un'attività implica la necessità di studiare un sistema, rappresentare la realtà osservata in modo semplificato attraverso un modello, individuare, classificare e selezionare i dati.

Le previsioni del tempo atmosferico richiedono, per esempio, la disponibilità dei dati riguardanti la temperatura e la pressione, rilevate non solo in un determinato luogo, ma in molte località geograficamente distanti.

All'interno del problema considerato vengono quindi individuate le **entità** che sono importanti: ciascuna di esse può essere presente nella realtà osservata con diversi *esemplari*, ai quali associamo i diversi valori che possono essere assunti dall'entità.

Per esempio, nell'attività di fatturazione entrano diverse entità:

- il cliente al quale è intestata la fattura
- il prodotto che viene venduto
- la fattura.

Gli esemplari di queste entità sono le diverse fatture che un'azienda emette in date diverse, i vari prodotti che l'azienda vende e i clienti che fanno parte dell'apposito archivio.

Per fornire una rappresentazione più precisa, ma soprattutto più formalizzata, nel modello vengono introdotte le **proprietà** delle entità.

Per esempio sono proprietà del cliente: il nome, l'indirizzo, la città dove risiede, il numero di partita IVA; per il prodotto: la descrizione, il prezzo; per la fattura: la quantità venduta, l'imponibile e l'imposta calcolata secondo una certa aliquota.

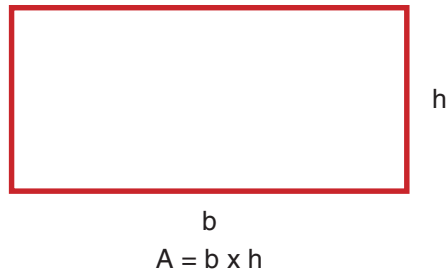
Queste proprietà possono assumere valori diversi per fatture diverse: quindi diciamo che sono *variabili*, cioè stanno al posto dei valori specifici che verranno considerati in relazione a una determinata fattura.

Le **variabili** sono proprietà delle entità che possono assumere valori diversi per diversi esemplari della stessa entità.

Nelle formule della matematica l'uso delle variabili è una prassi comune: per calcolare l'area di un rettangolo si utilizza la formula

$$A = b \times h$$

dove *A*, *b* e *h* sono nomi che stanno al posto dei valori rispettivamente dell'area, della base e dell'altezza del rettangolo. Questi nomi verranno poi sostituiti dai valori effettivi delle misure riferite al particolare esemplare di rettangolo del quale si vuole calcolare l'area.



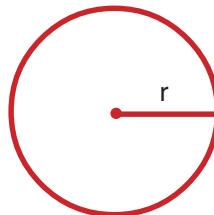
Considerando poi problemi dello stesso tipo, è possibile spesso identificare delle proprietà che assumono sempre gli stessi valori, o comunque mantengono lo stesso valore per un ragionevole periodo di tempo.

Queste proprietà vengono definite **costanti**, cioè proprietà che non cambiano considerando problemi con variabili diverse.

Esempi di *costanti in senso stretto* sono sicuramente il valore di π (pi-greco), che rappresenta il rapporto tra la circonferenza e il diametro di un cerchio, e la costante g nella formula del moto uniformemente accelerato:

$$s = \frac{1}{2} g t^2.$$

Quindi nel problema del calcolo dell'area del cerchio, la misura del raggio e dell'area sono variabili, pi-greco è una costante.



Esempi di *costanti in senso lato* possono essere le aliquote IVA applicate agli importi di una fattura o le percentuali di imposta sui redditi: questi valori possono cambiare nel tempo, ma si mantengono costanti per un periodo di durata rilevante all'interno di una classe di problemi dello stesso tipo.

Nel problema del calcolo del totale di una fattura, sono variabili l'imponibile, l'imposta e il totale della fattura ed è costante l'aliquota per calcolare l'imposta.

I nomi che diamo alle variabili o alle costanti, per distinguerle all'interno del modello considerato, si chiamano, in modo generico, **identificatori**.

2 Dati e azioni

La soluzione di un problema richiede da una parte la disponibilità dei *dati* necessari per effettuare i calcoli e dall'altra un insieme di *azioni* da svolgere per ottenere il risultato finale.

I **dati** sono i valori assunti dalle proprietà degli elementi che caratterizzano il problema, rappresentate con variabili o costanti.

Le **azioni** sono le attività che, mettendo i dati in relazione tra loro, consentono di ottenere i risultati desiderati.

PROGETTO 1

Esaminare un elenco di persone, con nome e anno di nascita, contare le persone che hanno più di 20 anni di età, e fornire alla fine il risultato del conteggio.

Nome	Anno
.....	
.....	
.....	
.....	
.....	
.....	
.....	

Una buona regola pratica, per individuare quali sono i dati e quali le azioni presenti nel problema, consiste nel trattare i sostantivi come dati e i verbi come azioni.

Nell'esempio, i dati sono:

- nome
- anno di nascita
- età delle persone
- età minima (20 anni)
- risultato del conteggio.

In un problema i **dati** possono essere di tipo diverso:

- *numerici*, come nel caso di stipendi dei dipendenti, età delle persone, importi da pagare, dimensioni di un oggetto;
- *alfabetici*, come il nome di una persona, il titolo di un libro, la descrizione della ragione sociale di un'azienda, il nome di un prodotto commerciale;
- *alfanumerici*, detti comunemente anche **stringhe**, cioè dati che sono rappresentati con cifre e lettere, quali il codice fiscale di un cittadino o la targa di un'automobile.

Una regola di carattere generale è la seguente: se un dato viene utilizzato in procedimenti di calcolo, allora è un dato di tipo numerico, altrimenti è alfabetico o alfanumerico.

Per esempio la partita IVA di un'azienda o il numero di telefono di un abbonato sono dati rappresentati con cifre numeriche, ma non vengono mai usati per fare addizioni o prodotti, e pertanto possono sicuramente essere considerati dati alfanumerici.

L'elenco di persone, pur essendo un sostantivo, è un dato di tipo *non elementare* che comprende un insieme di dati elementari, che sono il nome e l'anno di nascita di ciascuna persona. Il problema richiede di operare sui dati *elementari*, nel senso che le persone devono essere esaminate una ad una per controllare la loro età.

Si deve anche notare che il problema richiede di controllare l'età delle persone, che non è un dato immediatamente disponibile, in quanto nell'elenco compare l'anno di nascita e non l'età: quindi abbiamo bisogno di un ulteriore dato, l'anno in corso, che può essere considerato, in questo contesto, come una *costante*, in quanto non varia esaminando persone diverse.

Lo schema seguente serve a documentare i dati individuati all'interno del problema:

Identificatore	Variabile o costante	Descrizione	Tipo
Nome	variabile	nome della persona	alfabetico
AnnoNascita	variabile	anno di nascita della persona	numerico
Età	variabile	età della persona	numerico
EtàMinima	costante	età minima per il controllo	numerico
AnnoAttuale	costante	anno in corso	numerico
Contatore	variabile	risultato del conteggio	numerico

Rileggendo il testo del problema si possono identificare i verbi che corrispondono alle azioni da svolgere per trovare i risultati:

- esaminare l'elenco;
- contare le persone;
- fornire il risultato.

A queste azioni va aggiunto il calcolo dell'età, conoscendo l'anno in corso e l'anno di nascita della persona, attraverso la formula:

$$\text{Età} = \text{AnnoAttuale} - \text{AnnoNascita}$$

Si noti come, in questa azione di calcolo, più dati, variabili e costanti, vengono combinati tra loro per formare un'espressione matematica.

Le **azioni** possono essere riconducibili ad operazioni:

- di tipo *aritmetico*, quali i calcoli;
- di tipo *logico*, quali il confronto tra due valori o il controllo del valore di verità di un enunciato.

Per esempio:

- il calcolo del 3% di un totale è un'operazione di tipo aritmetico;
- la somma di N numeri è un'operazione di tipo aritmetico;
- il confronto $A > B$ è un'operazione di tipo logico;
- il controllo per stabilire se è stata raggiunta la fine dell'elenco nella lettura di una lista di nomi è un'operazione di tipo logico;
- il controllo per stabilire se il numero di telefono di una persona è stato trovato in una rubrica è un'operazione di tipo logico.

L'azione *Esaminare* indicata precedentemente è un'azione complessa che corrisponde all'insieme di tante piccole azioni, tutte dello stesso tipo, che controllano i dati delle persone dell'elenco, una ad una.

Quindi, in generale, l'azione di *Esaminare un elenco* significa in pratica, e anche nella formalizzazione dei problemi, prendere in considerazione i dati scritti nelle singole righe che compongono l'elenco, una per volta dalla prima riga fino alla fine dell'elenco.

3 La metodologia di lavoro

Quando, nella vita di tutti i giorni, ci troviamo di fronte alla necessità di risolvere un problema, seguiamo in genere una sequenza di passi precisa. Infatti, indipendentemente dalla dimensione e dall'importanza che assume il problema, il processo di risoluzione può essere diviso in fasi ben distinte tra loro.

Nella prima fase si definisce il problema, cercando di descrivere con la maggior precisione possibile quali sono i dati che in esso sono coinvolti, quali sono quelli che abbiamo a disposizione e quali sono i risultati che ci aspettiamo dalla soluzione del problema. Si deve inoltre pensare dove si possono rintracciare e come si possono raccogliere tutte le risorse che potranno essere utili.

Nella seconda fase vanno definite quali azioni intraprendere e viene stabilita, tenuto conto dei dati a disposizione, l'esatta sequenza delle azioni da compiere per ottenere la soluzione del problema e per raggiungere il tipo di risultati attesi.

Infine si procede all'esecuzione delle azioni definite nella fase precedente.

Vediamo qualche esempio.

- *Ricetta di cucina per la preparazione dell'arrosto di vitello*

Dosi per 4 persone: 700 gr. di polpa di vitello, 50 gr. di burro, mezzo bicchiere di vino bianco, un mestolo di brodo, rosmarino e sale. Mettere nella pentola due rametti di rosmarino e rosolare la carne in burro ed olio ben caldi. Bagnare con il vino e lasciare evaporare. Aggiungere il brodo, cuocere a fuoco lento per circa un'ora. Alla fine restringere il brodo a fuoco vivace. La prima cosa che è specificata nella ricetta di cucina è il numero di persone, seguono le dosi e gli ingredienti: tutto ciò costituisce l'insieme dei dati iniziali del problema.

Vanno poi elencate le risorse che, nel nostro caso, sono i tipi di pentole da utilizzare e, se è necessario, il forno e quale tipo di fuoco. Da ultimo è indicato il processo di preparazione della pietanza.

- *Tenuta del bilancio familiare*

I dati iniziali sono costituiti dalla lista dettagliata delle entrate e delle uscite, i risultati attesi sono il rendiconto dei tipi di spesa effettuati e del tipo di entrate godute. Il criterio di calcolo e di aggregazione delle entrate e delle uscite costituisce il procedimento risolutivo del problema.

I medesimi passi, descritti sopra per la soluzione di problemi della vita quotidiana, devono essere seguiti quando si utilizzano le risorse dell'elaborazione automatica dei dati.

La procedura per affrontare una situazione complessa, da districare attraverso l'utilizzo delle metodologie informatiche è, infatti, scomponibile in almeno due passi fondamentali che possono essere descritti nel seguente modo.

Passo 1: Descrizione del problema

In questa prima fase occorre definire con precisione: i dati che abbiamo a disposizione, i confini delle soluzioni adottate, i risultati attesi e le risorse da utilizzare per elaborare i dati iniziali e per ottenere la soluzione del problema.

Schematicamente gli elementi che caratterizzano un problema sono:

- i dati che servono ovvero i dati iniziali sui quali basare la soluzione del problema da affrontare. I dati iniziali vengono anche detti dati di ingresso, o **dati di input**. Tali dati non devono essere né sovrabbondanti né incompleti.
- i risultati che si vogliono ottenere, detti anche dati di uscita o **dati di output**.
- le risorse a disposizione sia dal punto di vista logico (tabelle, criteri di calcolo, schemi logici, ecc.) sia dal punto di vista fisico o hardware (strumenti di calcolo quali macchine calcolatrici, personal computer o calcolatori di grosse dimensioni).

- le soluzioni adottate, ovvero il procedimento e il percorso che permettono di passare dai dati iniziali ai risultati attesi.

L'individuazione di questi elementi costituisce l'**analisi del problema**.

Passo 2: Stesura dell'algoritmo

La soluzione del problema, così come è stata descritta nella prima fase, deve essere ora organizzata e sviluppata in una serie di operazioni da attuare secondo un ordine ben definito, che permette di giungere ai risultati attesi a partire dai dati iniziali. La scomposizione del procedimento risolutivo di un problema, in una sequenza di operazioni elementari da seguire per ottenere i risultati attesi, costituisce la stesura dell'algoritmo risolutivo.

4 Definizione di algoritmo

Il termine **algoritmo** necessita di una definizione precisa e dettagliata per evidenziare quali siano le sue caratteristiche principali.

Nell'accezione più diffusa il termine algoritmo indica un procedimento, in generale per il calcolo di grandezze o di quantità, basato sull'applicazione di un numero finito di operazioni, che specificano in modo rigido e meccanico ogni singolo passo del procedimento risolutivo.

Il termine algoritmo proviene dalla matematica e sta a indicare le regole, o in generale le operazioni da compiere, per fare un calcolo o risolvere un problema. Deriva dal nome di un algebrista arabo del IX secolo di nome **Al-Khuwarizmi**.

L'**algoritmo** è la descrizione di un insieme finito di istruzioni, che devono essere eseguite per portare a termine un dato compito e per raggiungere un risultato definito in precedenza.

Esempi di algoritmi possono essere, oltre a quelli visti nel paragrafo precedente, le istruzioni per l'utilizzo di un elettrodomestico, le regole per eseguire la divisione tra due numeri, le indicazioni per la consultazione di un orario ferroviario, ecc.

Va notato, da subito, che nella definizione è implicita la presenza di un **esecutore**, in altre parole di un oggetto, macchina, ente o persona alla quale affidare l'interpretazione e l'esecuzione delle istruzioni contenute nell'algoritmo.

PROGETTO 2

Una segretaria deve contattare telefonicamente una serie di persone a cui deve lasciare un messaggio.

La segretaria deve procurarsi la lista delle persone da contattare e il numero di telefono di ciascuna. Scorrendo l'elenco, per ogni persona deve comporre il numero. Se risponde la persona desiderata, deve lasciare il messaggio e spuntare la persona dall'elenco. Negli altri casi (telefono occupato o numero errato) deve annotare sull'elenco la situazione incontrata.

Dati di input (*dati che servono*):

l'elenco delle persone con il nome e il numero di telefono,
il messaggio da comunicare.

Dati di output (*risultati che si vogliono ottenere*):

l'elenco delle persone con l'indicazione per ciascuna dell'esito della telefonata.

Algoritmo

Per ciascuna persona dell'elenco:

leggi il numero telefonico

componi il numero

a seconda della situazione che si presenta:

scrivi sull'elenco: "occupato", "numero errato" o "ok".

Ripeti le operazioni precedenti finché le persone sono finite.

L'algoritmo è presentato secondo un linguaggio volutamente privo di qualsiasi formalismo. Più avanti verranno presentate le regole fondamentali e gli strumenti di base che servono per l'organizzazione di un algoritmo ben strutturato.



CARATTERISTICHE DELL'ALGORITMO

Esaminando l'esempio di algoritmo presentato nel paragrafo precedente si possono cogliere le caratteristiche principali che devono avere le istruzioni e i requisiti minimi di un algoritmo.

Riprendiamo la prima parte della definizione di algoritmo fornita nel paragrafo precedente: in essa si parla di "istruzioni che devono essere eseguite". Per rendere meno generica la definizione data, è necessario specificare le caratteristiche fondamentali delle istruzioni che possono comporre un algoritmo.

1. Ogni istruzione deve essere concretamente **realizzabile** dall'esecutore, a cui è affidato l'algoritmo. Le istruzioni, viste nell'esempio della ricetta di cucina descritte all'inizio, sono tutte di questo tipo. Esempi di istruzioni irrealizzabili, invece, possono essere: "vola sulla luna", oppure "calcola la radice quadrata con approssimazione alla quarta cifra decimale del numero 43", se questa istruzione è affidata ad un alunno delle elementari.
2. Le istruzioni che compongono un algoritmo devono essere **precise e non ambigue** in modo che non lascino dubbi nell'interpretazione da parte dell'esecutore. Se si dice "calcola l'imposta del 23% sul reddito di una persona", viene data un'indicazione precisa e completa, mentre l'istruzione "calcola l'imposta" è generica e imprecisa.
3. Ogni istruzione, se pur concretamente eseguibile e non ambigua, deve avere una **durata limitata nel tempo**. Per esempio "calcola il quadrato dei primi diecimila numeri" comporta un notevole impiego di tempo, ma presenta un punto di arresto. L'istruzione "calcola tutte le cifre decimali di pi-greco", invece, non avrà mai fine poiché tali cifre decimali sono infinite.
4. Ogni istruzione deve produrre, se eseguita, un **risultato osservabile**. Per esempio "dipingi la parete" dà luogo ad un effetto oggettivamente riscontrabile, mentre "pensa" o "immagina" non sono effettivamente osservabili.
5. Ogni istruzione deve avere carattere **deterministico**, in altre parole deve produrre sempre il medesimo effetto se eseguita a partire dalle stesse condizioni iniziali. Per esempio l'istruzione "tira una freccia contro il bersaglio" può avere risultati diversi anche se è rivolta al medesimo arciere munito del suo solito arco, mentre l'istruzione "moltiplica 3 per 4" produce sempre il medesimo risultato.

6. Le istruzioni devono essere **elementari**, cioè non ulteriormente scomponibili rispetto alle capacità dell'esecutore. Per esempio "calcola la radice quadrata di 5" è elementare per un esecutore che abbia a disposizione una calcolatrice tascabile dotata della funzione di radice quadrata, mentre deve essere specificato tutto il processo di calcolo se il destinatario del comando ha a disposizione soltanto carta e penna.

Se le istruzioni devono soddisfare le caratteristiche fondamentali descritte sopra, anche gli algoritmi, perché siano costruiti in modo corretto, devono seguire determinate regole.

Prima di iniziare la descrizione dei criteri fondamentali che devono essere seguiti per la stesura di un algoritmo, vengono presentate alcune caratteristiche basilari che devono essere rispettate in ogni caso.

- a. Un algoritmo deve essere composto da un numero **finito** di istruzioni, e deve presentare un punto di inizio dove comincia il procedimento risolutivo e un punto di fine, raggiunto il quale si interrompe l'esecuzione delle istruzioni.
- b. Deve essere completo ed **esaustivo** nel senso che per tutti i casi che si possono verificare durante l'esecuzione deve essere indicata la soluzione da seguire.
- c. Deve essere **riproducibile**: ogni successiva esecuzione dello stesso algoritmo con i medesimi dati iniziali deve produrre sempre i medesimi risultati finali.

Dal punto di vista dell'esecutore, l'algoritmo è la sequenza delle operazioni da compiere. È chiaro che l'esecuzione dell'algoritmo deve terminare dopo una sequenza finita di operazioni.

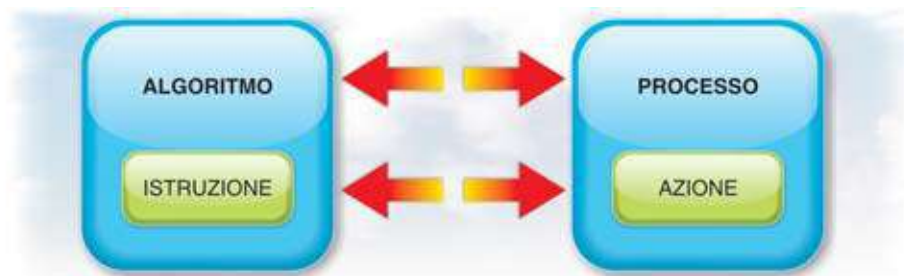
5 Algoritmo ed esecutore

Un'**azione** è un evento che si compie in un intervallo di tempo finito e che produce un risultato, un effetto, previsto e ben determinato. Ogni azione modifica lo stato di qualche oggetto, e il suo effetto può essere riconosciuto dal cambiamento di stato dell'oggetto in questione. Per descrivere le azioni è necessario disporre di un **linguaggio** o di una notazione formale: le descrizioni delle azioni sono dette **istruzioni**.

Un'istruzione elementare è un'istruzione che non può essere scomposta in istruzioni più semplici. Un'istruzione non elementare è detta **algoritmo**.

Un'istruzione composta descrive un'azione composta, vale a dire un'azione che può essere frammentata in azioni più semplici. Un'azione composta è chiamata **esecuzione** (o **processo**).

Un *algoritmo* è dunque composto da un certo numero di *istruzioni*, ad esso corrisponde un *processo* composto da un certo numero di *azioni*; in generale, non vi è corrispondenza tra la successione temporale delle azioni e la disposizione delle istruzioni all'interno dell'algoritmo.



L'**esecutore** (o **processore**) è l'ente che esegue le azioni secondo le istruzioni di un algoritmo.

Esecutore è un nome generico, che non indica se si tratta di un esecutore umano oppure automatico. In pratica l'esecutore è da intendersi come l'ente che procede a compiere l'esecuzione del processo risolutivo.

Inoltre, per una descrizione generale dell'ambiente, in cui si opera con l'informatica, vanno definite due figure fondamentali:

- il **programmatore** ovvero colui che organizza, prepara e scrive l'algoritmo
- l'**utente** ovvero la persona che attiva l'esecuzione dell'algoritmo e che interagisce con l'esecutore per fornire ad esso i dati iniziali del problema e per utilizzare i risultati finali.



L'esecutore può essere una persona, una macchina o un insieme di persone e macchine. Nei capitoli successivi di questo testo, per esecutore si intenderà principalmente un computer. In tal caso l'algoritmo dovrà essere scritto secondo un linguaggio definito, con regole rigorose in modo da essere comprensibile da parte del computer.

In questo caso l'algoritmo diventa un **programma**.



6 Acquisire e comunicare i dati

La risoluzione di un problema comporta l'individuazione di tre elementi ben definiti e indispensabili per una corretta definizione dell'algoritmo risolutivo: si tratta dei dati iniziali, della soluzione adottata e dei risultati finali. Le relazioni che intercorrono tra questi tre elementi possono essere schematizzate nel seguente modo:



Ne consegue che un esecutore che provvede a compiere le azioni richieste dalle istruzioni di un algoritmo deve essere in grado di acquisire i dati iniziali, su cui attivare il processo di elaborazione, e deve essere in grado di comunicare all'esterno i risultati finali ottenuti: in altre parole deve essere in grado di leggere dall'esterno i dati in ingresso (o *dati di input*) e deve essere in grado di scrivere verso l'esterno i dati di uscita (o *dati di output*).



I **dati di input** sono quelli che vengono forniti dall'esterno per poter risolvere il problema; i **dati di output** sono quelli che vengono comunicati all'esterno, come risultato della soluzione del problema.

Ci possono essere poi altre variabili, che non sono né di input né di output, ma che sono comunque necessarie all'attività di elaborazione per ottenere risultati parziali, e che vengono chiamate **variabili di lavoro**, temporanee o di calcolo.

PROGETTO 3

Date le misure dei due cateti di un triangolo rettangolo, calcolare la misura del perimetro del triangolo.

Le misure dei due cateti devono essere fornite dall'esterno perché sono necessarie per il calcolo richiesto; l'ipotenusa è una variabile di lavoro, perché non viene fornita dall'esterno, ma per calcolare il perimetro serve la sua misura; il perimetro è il valore da comunicare come risultato finale, ed è perciò un dato di output.

Il tutto può essere schematizzato in questo modo:

Dati	Input	Output	Lavoro
Cateto1	✓		
Cateto2	✓		
Ipotenusa			✓
Perimetro		✓	

PROGETTO 4

Calcolare l'importo totale di una fattura con applicazione dell'IVA.

Dall'esterno devono essere acquisiti i valori dell'imponibile e dell'aliquota IVA da applicare, che sono quindi dati di input, l'imposta è una variabile di lavoro, e il totale della fattura è il risultato da comunicare al termine dell'elaborazione, e quindi è il dato di output.

Dati	Input	Output	Lavoro
Imponibile	✓		
Aliquota IVA	✓		
Imposta			✓
TotaleFattura		✓	

Per fare in modo che l'esecutore acquisisca i dati e comunichi i risultati, nell'algoritmo che richiede tali azioni, per le operazioni di *input* vengono indicate istruzioni del tipo:

- immetti
- leggi
- acquisisci
- accetta
- read
- accept
- ecc.

e per le operazioni di *output*:

- scrivi
- comunica
- mostra
- write
- display
- ecc.

7 Gli operatori

Un'altra azione fondamentale che risulta comunque presente in un processo eseguito da un calcolatore è l'**assegnamento di un valore a una variabile**. Possiamo paragonare una variabile a una porzione di una lavagna: è sempre possibile leggervi quanto scritto ed è possibile cancellare e riscrivervi dell'altro.

Per esempio, l'assegnamento del valore 9 a una variabile *v* può essere descritto in questo modo:

assegna $v = 9$

che significa: "v prende il valore 9" oppure "9 viene assegnato a v", cioè l'assegnazione è verso sinistra.

Ad una variabile può essere assegnato un dato costante, come nell'esempio precedente, oppure il risultato di un'espressione:

calcola $v = E$

Tale istruzione indica che viene calcolato il valore dell'espressione *E* e che il risultato viene assegnato a *v*. Un'espressione è una formula (o regola di calcolo) che specifica sempre un valore (o risultato). Ogni espressione è composta da operatori e operandi. Gli operandi possono essere costanti (per esempio numeri), espressioni o variabili.

Gli operatori possono essere di tre tipi: aritmetici, di relazione e logici.

Gli **operatori aritmetici** sono:

- + per l'addizione
- per la sottrazione
- * per la moltiplicazione
- / per la divisione
- ^ per l'elevamento a potenza

Gli **operatori di relazione** (o *di confronto*) sono utilizzati per confrontare il contenuto di due variabili e sono indicati con i simboli:

=	uguale
<	minore di
<=	minore o uguale di
>	maggiore di
>=	maggiore o uguale di
<>	diverso

Gli **operatori logici** sono:

AND per il prodotto logico (coniunzione)

OR per la somma logica (disgiunzione)

NOT per la negazione

XOR per la disgiunzione esclusiva

**AUTOVERIFICA**

Problemi da 1 a 12 pag. 203-204

8 Strumenti per la stesura di un algoritmo

Scrivere o comprendere algoritmi è un'operazione non semplice che comporta la conoscenza di regole ben precise e di linguaggi specifici. Nei progetti precedenti è stato usato un metodo descrittivo ricorrendo in larga misura al linguaggio comune in quanto si trattava di risolvere problemi abbastanza semplici. Occorre però fornire strumenti di lavoro validi per risolvere situazioni complesse e variamente articolate.

La conoscenza di linguaggi utili per descrivere un algoritmo è, pertanto, il primo e fondamentale compito di chi vuole conoscere l'arte di inventare e comunicare algoritmi.

Vengono descritti, di seguito, i due linguaggi più comunemente utilizzati: la pseudocodifica e i diagrammi a blocchi.

Come in tutti i linguaggi occorre definire l'alfabeto (i simboli ammessi dal linguaggio), le parole (le combinazioni dei simboli) e la sintassi (le regole che permettono di associare tra loro le parole in modo coerente).

Il linguaggio di pseudocodifica

Si intende per **pseudocodifica** (o *pseudolinguaggio* o *linguaggio di progetto*) la descrizione di un algoritmo ottenuta utilizzando termini e parole del linguaggio comune, ma applicando una serie di regole che permettono di organizzare un tipo di testo formalmente rigoroso e strettamente orientato alla stesura degli algoritmi; non si tratta di un vero linguaggio di programmazione, ma solo di un linguaggio sintetico composto di un vocabolario e di una sintassi molto ristretti.

Utilizzando la pseudocodifica, il programmatore esprime le proprie idee in una forma naturale e si avvale di frasi ed espressioni elementari della lingua italiana; ciò permette di concentrarsi sulla logica della soluzione del problema, senza vincolare il lavoro al particolare tipo di linguaggio di programmazione effettivamente utilizzato, e senza essere distratto dai vincoli formali richiesti nel lavoro di stesura del programma.

Di seguito vengono presentate le regole principali.

- Le parole chiave che aprono e chiudono il testo di un algoritmo sono
inizio fine

Ogni istruzione è indicata con una frase del linguaggio corrente e può contenere un'espressione di tipo aritmetico o logico. Le espressioni possibili sono quelle eseguibili da un elaboratore come indicato nelle pagine precedenti.

- Le istruzioni

immetti variabile

scrivi variabile

scrivi messaggio

vengono utilizzate per descrivere rispettivamente le operazioni di input (*immetti*) e output (*scrivi*) dei dati.

- Le istruzioni

assegna variabile = valore

calcola variabile = espressione

indicano operazioni di assegnazione a una variabile di un valore o del risultato di un'espressione.

- Le variabili, le costanti e, in generale, le risorse utilizzate nell'algoritmo vengono indicate da parole dette **identificatori**.

In pseudocodifica vengono utilizzate altre **parole chiave** che permettono di strutturare la logica della soluzione adottata e che corrispondono ai passaggi fondamentali dell'algoritmo. Le parole chiave utilizzate in pseudocodifica sono:

se, allora, altrimenti, fine se, esegui, finché, mentre, ripeti, per, da, a, passo, caso di, fine caso e verranno illustrate nelle pagine successive.

Diagramma a blocchi

Il metodo del diagramma a blocchi consiste in una descrizione grafica dell'algoritmo; esso permette una visione immediata dell'intero procedimento e dell'ordine di esecuzione delle varie istruzioni.

I diagrammi a blocchi sono formati da simboli di forma diversa, ciascuna con un proprio significato; all'interno di ogni simbolo è presente un breve testo sintetico ma non ambiguo. Linee orientate con frecce, che uniscono tra loro i vari simboli, indicano il flusso delle operazioni.

I diagrammi vengono comunemente chiamati **diagrammi a blocchi** (o *diagrammi logici* o *flow-chart* o *diagrammi di flusso*).

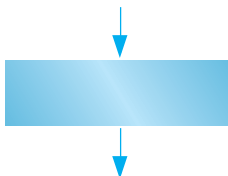
I simboli



indicano il punto di partenza e quello di terminazione dell'algoritmo.

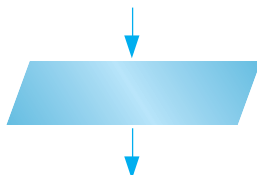
Da *Inizio* parte una sola freccia che raggiunge la prima istruzione e verso *Inizio* non arriva alcuna freccia. Verso *Fine* arrivano una o più frecce, ma da esso non ne parte nessuna.

Il simbolo



è detto di **elaborazione** e contiene al suo interno l'istruzione da eseguire (solitamente un'assegnazione di un valore o di un'espressione ad una variabile); può avere una o più frecce in entrata, ma una sola in uscita.

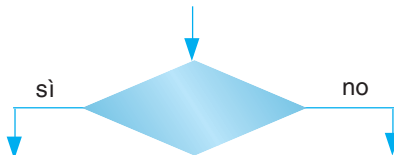
Il simbolo



viene utilizzato per rappresentare operazioni di immissione di dati (**input**) o di emissione di dati (**output**).

In mancanza di indicazioni specifiche si assume che l'immissione dei dati avvenga da tastiera (**standard input**) e che i dati di output siano presentati sullo schermo (**standard output**). Il simbolo è associato a una o più frecce di entrata e ad una sola freccia di uscita.

Il simbolo



viene detto **simbolo di decisione** e, nella maggior parte dei casi, serve per rappresentare un'operazione di confronto tra due dati. Il simbolo di decisione è usato per stabilire se una proposizione è vera o falsa e in corrispondenza delle frecce in uscita si trovano indicazioni del tipo sì/no, vero/falso, V/F.

Viene così definito il valore di un'espressione di tipo logico, cioè di una variabile a due valori a uno dei quali corrisponde il significato di condizione verificata, vero, e all'altro quella di condizione non verificata, falso.

Il rombo ha un punto di entrata e due punti di uscita.

Infine, è possibile, all'interno di un diagramma a blocchi, inserire alcuni **commenti** illustrativi che servono per dare maggiore chiarezza allo schema. Allo scopo si usa il simbolo



PROGETTO 5

Calcolo dell'area di un triangolo.

Per calcolare l'area del triangolo occorre conoscere le misure della base e dell'altezza; l'esecutore deve quindi chiedere all'utente tali misure, attendere che l'utente comunichi tali misure e leggere i valori assegnandoli ad altrettante variabili. Dopo aver calcolato l'area, l'esecutore deve provvedere a scrivere (comunicare all'utente) il risultato.

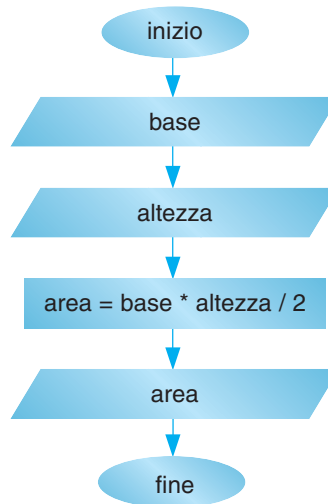
Dati di input: base e altezza del triangolo.

Dati di output: area del triangolo.

Algoritmo in pseudocodifica

```
inizio
  immetti base
  immetti altezza
  calcola area = base * altezza / 2
  scrivi area
fine
```

Diagramma a blocchi



Per meglio comprendere il senso di quanto scritto nell'algoritmo si osservi che il programmatore scrive una sequenza di comandi che sono espressi da verbi in modo imperativo (per esempio: *immetti*, *assegna*, *scrivi*) e che sono rivolti all'esecutore (computer).

L'esecutore, oltre a procedere nello svolgimento dei calcoli (per esempio: $\text{area} = \text{base} * \text{altezza} / 2$), interagisce con l'utente in tre modi:

- riceve dei dati (i dati di input),
- comunica dei risultati (i dati di output).

PROGETTO 6

Calcolo della media di tre numeri.

Il procedimento deve acquisire tre valori numerici, calcolarne la somma e dividere quest'ultima per 3.

Dati di input: i tre numeri a , b , c .

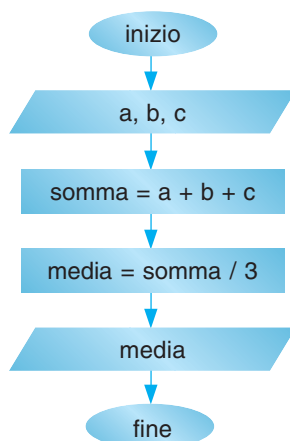
Dati di output: media dei numeri.

Variabili di lavoro: somma dei numeri.

Algoritmo in pseudocodifica

```
inizio
  immetti a, b, c
  calcola somma = a + b + c
  calcola media = somma / 3
  scrivi media
fine
```

Diagramma a blocchi



9 L'individuazione dei dati di un problema

L'aspetto più importante, e più difficile, nella formalizzazione dei problemi, consiste nel saper individuare con precisione quali sono i dati da controllare per risolvere il problema. Spesso nel testo del problema questi dati non sono esplicitati, ma presentati sotto forma di aggettivi che descrivono le caratteristiche di un'entità:

- per sapere se una persona è giovane o anziana, occorre controllare la sua *età*;
- per sapere se una persona è alta o bassa, occorre controllare la sua *statura*;
- per sapere se un prodotto è costoso o conveniente, occorre controllare il suo *prezzo*;
- per sapere se una città è vicina o lontana, occorre controllare la sua *distanza*;
- per sapere quanto tempo occorre per uno spostamento, occorre conoscere la *distanza* e la *velocità* del mezzo;
- per sapere se una città è grande o piccola, occorre controllare la sua *estensione*;
- per sapere se una nazione è popolosa, occorre conoscere il *numero di abitanti* e la *superficie*;
- per sapere se la prova di uno studente è sufficiente, occorre controllare il *voto*.

Da questo punto di vista, spesso la formalizzazione dei problemi richiede espressioni che sono diverse dal modo consueto di esprimersi del linguaggio naturale.

Per esempio, nel caso in cui il problema richieda di controllare se una persona abita a *Roma*, nella formalizzazione del problema la variabile da controllare è la *residenza* della persona, cioè il modello di soluzione conterrà un'espressione di tipo logico:

$\text{residenza} = \text{Roma}$

Come esercizio, si provi a completare le seguenti frasi:

1. Per sapere se un cliente è a debito o a credito, occorre conoscere
2. Per sapere se una persona è più alta o più bassa di m. 1,80, occorre controllare
3. Per sapere se una città è fredda o calda, occorre conoscere
4. Per sapere se una persona è diplomata o laureata, occorre controllare
5. Per sapere se una squadra ha vinto o ha perso, occorre controllare



AUTOVERIFICA

Domande da 1 a 8 pag. 199-200
Problemi da 13 a 20 pag. 204



MATERIALI ONLINE

1. Diagrammi a blocchi con Word

LA MACCHINA DI TURING

Nel Capitolo 2 è stato introdotto il concetto di *automa* e in questo capitolo è stato definito quello di *algoritmo*. Dopo aver descritto questi due concetti, si presentano ora due aspetti da risolvere:

- a) stabilire se esiste sempre un procedimento risolutivo da formalizzare con un algoritmo in corrispondenza di un determinato problema;
- b) stabilire le caratteristiche dell'automa in grado di eseguire l'algoritmo.

La risposta al primo aspetto verrà data di seguito con la *Tesi di Church-Turing*.

Alla seconda esigenza risponde il modello di macchina astratta proposto nel 1936 dal matematico inglese **Alan M. Turing** (1912-1954), che prende perciò il nome di **Macchina di Turing** (MdT).

Il particolare automa ideato da Turing fa riferimento alla comune attività mentale quando è impegnata nella risoluzione di algoritmi di calcolo: di solito si usano un foglio e una penna con la quale segnare sul foglio i dati e i simboli delle operazioni; il lavoro è controllato dalla mente umana, per la quale la carta costituisce il supporto di memoria (il foglio per i calcoli, oppure il foglio del libro che contiene le regole di calcolo).

La **Macchina di Turing**, in modo analogo, è costituita da un meccanismo di controllo, da un nastro di lunghezza infinita nei due sensi e da una testina di lettura/scrittura che può effettuare due tipi di movimento a sinistra o a destra.



Il nastro è diviso in celle, in ciascuna delle quali può essere letto o scritto un simbolo appartenente all'alfabeto dei simboli trattati dalla macchina.

Durante il suo funzionamento la macchina evolve da una configurazione all'altra, cioè da uno **stato** all'altro: in corrispondenza del simbolo letto sul nastro e dello stato in cui si trova, viene determinato il simbolo che viene scritto sul nastro, lo stato successivo della macchina e il movimento della testina.

All'inizio del processo di evoluzione, sul nastro si trova la sequenza dei simboli di input e al verificarsi della terminazione si trova l'output del procedimento eseguito.

La testina di lettura/scrittura può accedere ad una cella per volta, e si può spostare a destra o a sinistra.

Da un punto vista formale la MdT viene definita come una sestupla

$$\text{MdT} = (A, I, S, s_0, F, d)$$

dove:

- A è l'**alfabeto** dei simboli utilizzati dalla macchina; l'alfabeto comprende anche il simbolo *blank* che corrisponde alla cella vuota, cioè alla parte di nastro non utilizzato, e serve per indicare l'assenza di simboli.
- I è l'insieme dei **simboli di input** (sottoinsieme di A) che possono essere letti dal nastro.
- S è l'insieme degli **stati** della macchina.
- s_0 è lo **stato iniziale** della macchina.
- F è l'insieme degli **stati finali** della macchina (sottoinsieme di S).
- d indica la **funzione di transizione**, che associa ad ogni coppia (simbolo di input, stato) una terna (stato, simbolo di output, movimento della testina):

$$(i_t, s_{t-1}) \rightarrow (s_t, u_t, m_t)$$

e sta quindi ad indicare il simbolo che viene scritto sul nastro, lo stato successivo e il movimento della testina, quando la macchina, trovandosi in un determinato stato, legge un determinato simbolo di input dal nastro.

Nella **matrice di transizione**, che schematizza con una tabella la *funzione di transizione*, sono indicati sulle righe i simboli che vengono letti dal nastro (sottoinsieme dell'alfabeto A) e sulle colonne gli stati della macchina: all'incrocio tra una riga e una colonna vengono specificati il simbolo che viene scritto in output sul nastro, lo stato successivo e il movimento della testina (separati da una virgola):

stati input	s_1	s_2	s_3
i_1	s_{11}, u_{11}, m_{11}	s_{12}, u_{12}, m_{12}	s_{13}, u_{13}, m_{13}
i_2	s_{21}, u_{21}, m_{21}	s_{22}, u_{22}, m_{22}	s_{23}, u_{23}, m_{23}
i_3	s_{31}, u_{31}, m_{31}	s_{32}, u_{32}, m_{32}	s_{33}, u_{33}, m_{33}
i_4	s_{41}, u_{41}, m_{41}	s_{42}, u_{42}, m_{42}	s_{43}, u_{43}, m_{43}

In ogni istante la MdT assume una configurazione che può essere definita con la quintupla:

- simbolo letto
- stato
- nuovo stato
- simbolo scritto
- movimento della testina.

Il passaggio da una configurazione a quella dell'istante successivo si chiama **mossa** della MdT.

Vediamo ora l'esempio di un problema risolto mediante una MdT.

PROGETTO 7

Calcolare il quoziente e il resto della divisione di un numero intero per 2.

L'alfabeto dei simboli è composto dalle 10 cifre decimali con l'aggiunta del simbolo *b*, che indica il *blank*, e il simbolo # che indica la situazione di fine input sul nastro:

$$A = (b, \#, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9)$$

L'insieme I dei simboli di input è costituito dalle 10 cifre decimali e dal simbolo #.

L'insieme degli stati è composto da:

$$S = (R0, R1, \text{FINE})$$

Lo stato iniziale è $R0$, lo stato finale è FINE .

All'inizio sul nastro si trovano scritte le cifre che compongono il dividendo; alla fine dell'elaborazione si possono trovare le cifre del quoziente e, immediatamente alla loro destra, quella del resto (che può essere ovviamente 0 oppure 1).

La funzione di transizione opera in questo modo:

$(R0, \text{cifra pari}) \rightarrow (R0, \text{cifra pari}:2, \text{DESTRA})$
 $(R1, \text{cifra pari}) \rightarrow (R0, (\text{cifra pari}+10):2, \text{DESTRA})$

 $(R0, \text{cifra dispari}) \rightarrow (R1, \text{cifra dispari}:2, \text{DESTRA})$
 $(R1, \text{cifra dispari}) \rightarrow (R1, (\text{cifra dispari}+10):2, \text{DESTRA})$

 $(R1, \#) \rightarrow (\text{FINE}, 1, \text{DESTRA})$
 $(R0, \#) \rightarrow (\text{FINE}, 0, \text{DESTRA}).$

La MdT opera scandendo le cifre del dividendo a partire da quella più significativa (più a sinistra) e scrivendo al posto di ogni cifra il quoziente della cifra diviso 2, tenendo conto dell'eventuale resto precedente.

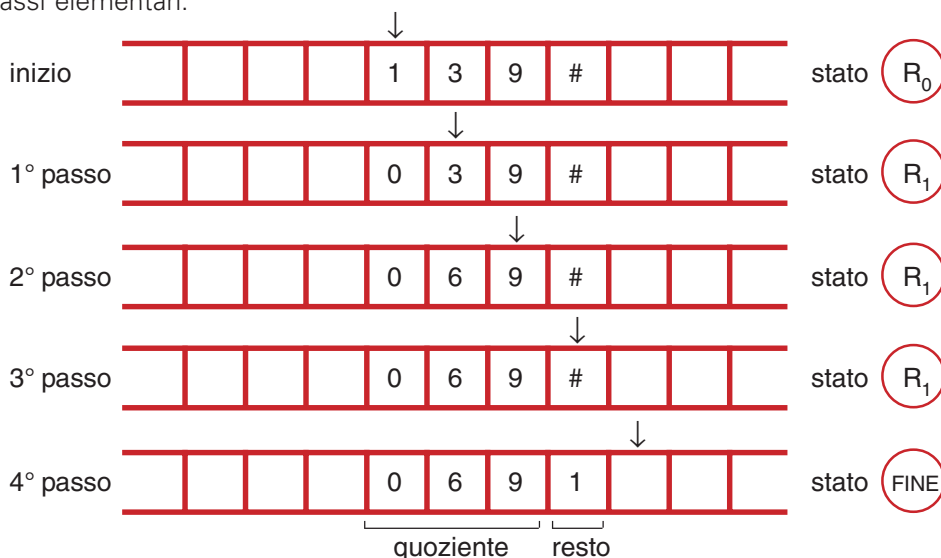
Gli stati $R0$ e $R1$ servono a ricordare rispettivamente le situazioni con resto uguale a 0 oppure a 1.

La funzione di transizione può essere descritta con la seguente tabella:

	R0	R1
#	FINE,0,D	FINE,1,D
0	R0,0,D	R0,5,D
1	R1,0,D	R1,5,D
2	R0,1,D	R0,6,D
3	R1,1,D	R1,6,D
4	R0,2,D	R0,7,D
5	R1,2,D	R1,7,D
6	R0,3,D	R0,8,D
7	R1,3,D	R1,8,D
8	R0,4,D	R0,9,D
9	R1,4,D	R1,9,D

La lettera D indica il movimento della testina verso destra e il simbolo # sta ad indicare la fine dell'input sul nastro.

Il funzionamento della MdT così definita può essere descritto secondo questa sequenza di passi elementari:



L'uso della Macchina di Turing introduce in modo intuitivo l'idea che un problema sia risolubile attraverso un procedimento quando sia possibile costruire una MdT in grado di eseguire un algoritmo che descrive, attraverso passi elementari, il procedimento risolutivo. D'altra parte appare intuitiva anche una coincidenza tra il concetto di algoritmo e il concetto di MdT, nel senso che l'algoritmo è il procedimento risolto dalla MdT astratta.

La risposta all'esigenza presentata (e rimasta aperta) all'inizio, di stabilire quando un problema è risolubile attraverso un procedimento, viene fornita attraverso la sistemazione teorica delle idee intuitive sopra descritte.

Nel seguito si userà il termine **computabile** come sinonimo di calcolabile o risolubile. Il termine **funzione computabile** indica un procedimento che trasforma i dati iniziali in dati finali.

La **Tesi di Church-Turing**, elaborata da Turing e dallo scienziato americano **Alonzo Church** (1903-1995), risolve a livello teorico l'esigenza posta e afferma che:

La classe delle funzioni computabili, secondo il concetto intuitivo di algoritmo, coincide con la classe delle funzioni Turing-computabili, cioè computabili con una MdT.

Sulla base della Tesi di Church-Turing possiamo quindi individuare all'interno dell'insieme di tutti i possibili problemi il sottoinsieme dei problemi che sono computabili: sono quelli ai quali può essere associata una MdT che risolve il problema.

Inoltre l'algoritmo risolutivo di un problema computabile può essere identificato con la sequenza delle azioni che vengono eseguite da una MdT.



AUTOVERIFICA

Domande da 9 a 10 pag. 200
Problemi da 21 a 23 pag. 205



MATERIALI ONLINE

- 2. Esempi di Macchina di Turing
- 3. Teoria della complessità algoritmica

10 Le strutture di controllo

Il lavoro di organizzazione di un algoritmo consiste, in prima approssimazione, nella stesura della sequenza di operazioni che devono essere eseguite per ottenere uno scopo prefissato. Dal punto di vista informatico la stesura di un algoritmo è uno dei passi necessari all'interno di un lavoro più complesso che costituisce la realizzazione di un programma.

La programmazione è infatti un'attività che può essere suddivisa in almeno quattro fasi:

- definizione del problema, dei dati di input e di output;
- organizzazione dell'algoritmo risolutivo;
- stesura del programma, cioè la traduzione dell'algoritmo nel linguaggio di programmazione;
- prove di esecuzione del programma.

Poiché la stesura dell'algoritmo risolutivo è una delle fasi fondamentali del lavoro di programmazione, si tratta di definire un insieme di regole che devono essere seguite per una corretta organizzazione del lavoro e per il raggiungimento di un buon livello di qualità dei programmi. Questo metodo di lavoro va sotto il nome di **programmazione strutturata** e ha come obiettivo quello di rendere più facile la costruzione, la lettura e la manutenzione di un programma.

La programmazione strutturata è la progettazione, la realizzazione e il collaudo di un programma costituito di parti che dipendono l'una dall'altra secondo un ben definito modello organizzativo.

Per individuare i modelli organizzativi con cui si possono strutturare gli algoritmi, per mettere a fuoco le strutture fondamentali che compongono un programma e per mostrare gli aspetti principali dei formalismi usati per la descrizione di un algoritmo, viene riproposto, a titolo di esempio, il problema del giro di telefonate, presentato in precedenza nel Progetto 2, scrivendo l'algoritmo risolutivo con le tecniche della pseudocodifica e dei diagrammi a blocchi.

PROGETTO 8

Una segretaria deve contattare telefonicamente una serie di persone a cui deve lasciare un messaggio.

La segretaria deve procurarsi la lista delle persone da contattare e il numero di telefono di ciascuna. Scorrendo l'elenco, per ogni persona deve comporre il numero. Se risponde la persona desiderata, deve lasciare il messaggio e spuntare il nome della persona dall'elenco. Negli altri casi (telefono occupato o numero errato) deve annotare sull'elenco la situazione incontrata.

Dati di input:

l'elenco delle persone con il nome e il numero di telefono,
il messaggio da comunicare.

Dati di output:

l'elenco delle persone con l'indicazione per ciascuna dell'esito della telefonata.

Algoritmo in pseudocodifica

inizio

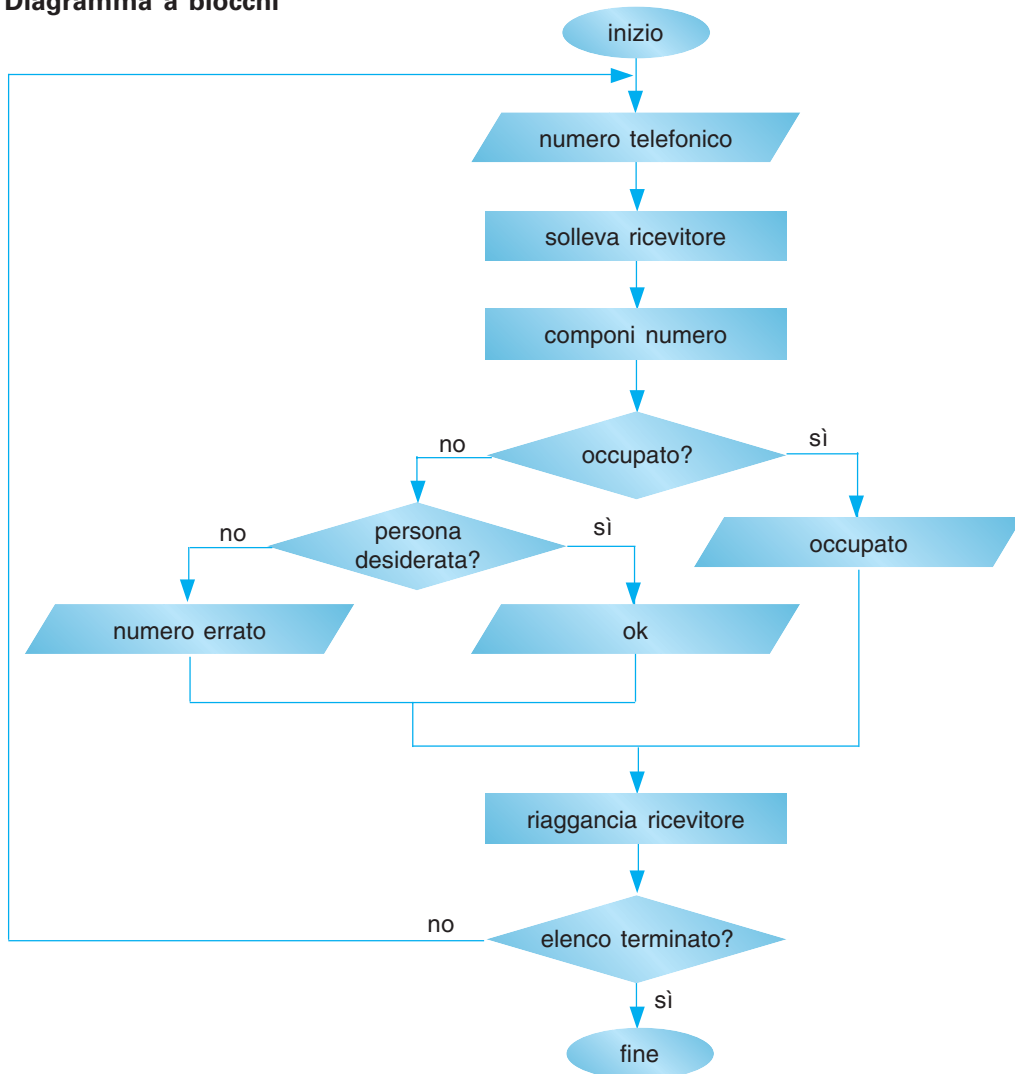
 esegui

 leggi numero telefonico

 solleva il ricevitore

```
componi il numero
se occupato
allora
    scrivi 'occupato'
altrimenti
    se risponde la persona desiderata
    allora
        scrivi 'ok'
    altrimenti
        scrivi 'numero errato'
    fine se
fine se
riaggancia il ricevitore
ripeti finché elenco terminato
fine
```

Diagramma a blocchi



Osservando con attenzione l'algoritmo precedente, si può notare che le istruzioni sono organizzate secondo schemi classificabili nel modo seguente.

1) Istruzioni organizzate in **sequenza**:

.....
leggi numero telefonico
solleva il ricevitore
componi il numero
.....

vale a dire istruzioni che devono essere eseguite una dopo l'altra secondo l'ordine con cui sono state scritte.

2) Istruzioni che vengono eseguite in **alternativa** con altre:

se risponde la persona desiderata
allora
 scrivi 'ok'
altrimenti
 scrivi 'numero errato'
fine se

l'esecutore deve fare una scelta tra un certo gruppo di istruzioni e un altro gruppo a seconda di quello che succede in quel momento durante l'elaborazione.

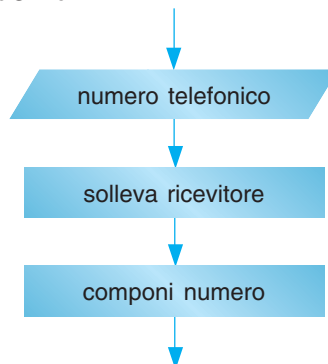
3) Istruzioni che devono essere eseguite in **ripetizione**:

esegui
.....
.....
ripeti finché elenco terminato

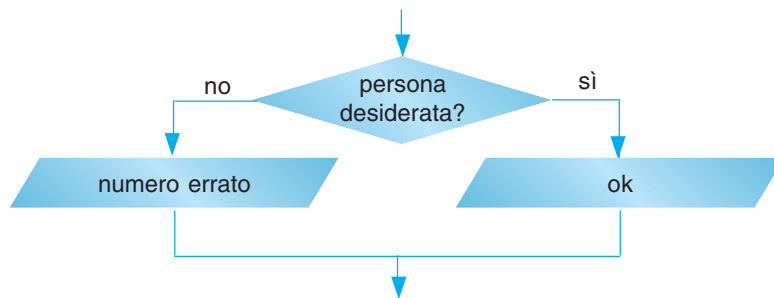
Le istruzioni comprese tra *esegui* e *ripeti* devono essere eseguite più volte, tante quante sono le telefonate da fare.

Le medesime strutture sono ovviamente presenti anche nel diagramma a blocchi; varia solamente la forma con cui si presentano, poiché è differente la modalità di rappresentazione degli algoritmi.

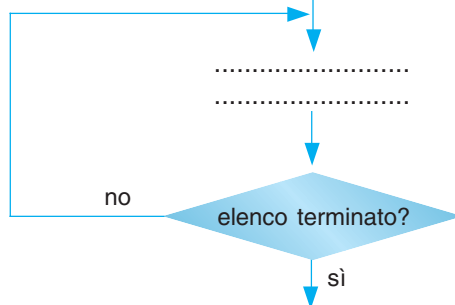
1) Istruzioni organizzate in **sequenza**:



2) Istruzioni che vengono eseguite in **alternativa** con altre:



3) Istruzioni che devono essere eseguite in **ripetizione**



Per quanto visto sopra, si può dire che è possibile individuare all'interno di un algoritmo alcune strutture tipiche.

Nel progetto precedente sono state messe in risalto tre strutture, vale a dire sono stati riconosciuti degli schemi particolari secondo cui sono organizzate le istruzioni all'interno dell'algoritmo. Si tratta ora di stabilire se le strutture riconosciute nell'esempio sono un caso particolare strettamente legato al problema esaminato, oppure se quanto è stato enucleato trova una sua giustificazione e una conferma nella teoria generale della programmazione strutturata. In altre parole si tratta di stabilire quali e quanti sono gli schemi organizzativi che possono essere utilizzati per le istruzioni all'interno di un algoritmo, se ve ne sono altri rispetto a quelli incontrati, e se quelli visti sono veramente necessari.

Questi ragionamenti sono stati affrontati da numerosi studi teorici che permettono di rispondere alle domande emerse nelle righe precedenti: qui viene semplicemente illustrato l'enunciato fondamentale, che sta alla base dell'utilità e dell'importanza della teoria della programmazione strutturata, per un corretto approccio al lavoro di creazione degli algoritmi.

Sostanzialmente la teoria della programmazione strutturata si può riassumere nella seguente affermazione.

Qualsiasi algoritmo appropriato può essere scritto utilizzando soltanto tre strutture di base: **sequenza, alternativa, ripetizione**.

Questi tre modelli organizzativi di base si chiamano **strutture di controllo**, perché servono a controllare il percorso all'interno del procedimento risolutivo per ottenere i risultati desiderati.

La struttura alternativa si chiama anche **selezione** o **struttura condizionale**, e la struttura di ripetizione viene indicata anche con il termine **iterazione**.

All'interno dell'enunciato precedente sono presenti due concetti che vanno definiti con chiarezza come segue.

Per **struttura** si deve intendere il modello organizzativo secondo cui si presentano le istruzioni all'interno di un algoritmo.

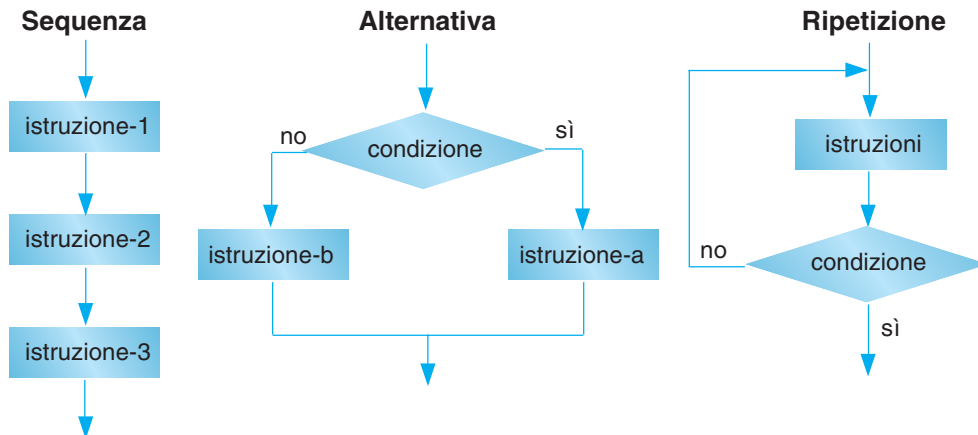
Un **algoritmo appropriato** consiste in un algoritmo che ha le seguenti caratteristiche essenziali:

- non presenta istruzioni che sono ripetute infinite volte,
- non ha percorsi non sfruttati,
- ha un solo punto di entrata e un solo punto di uscita.

Riassumendo, se si utilizza la pseudocodifica, le tre strutture di controllo fondamentali sono rappresentate nel seguente modo:

Sequenza	Alternativa	Ripetizione
..... istruzione-1 istruzione-2 istruzione-3	se condizione allora istruzione-a altrimenti istruzione-b fine se	esegui istruzioni ripeti finché condizione

Se si utilizzano i diagrammi a blocchi le strutture sono rappresentate dai seguenti schemi:



La programmazione strutturata rappresenta il punto di arrivo di un processo di riflessione sviluppatosi nella seconda metà degli anni '60, nato dall'esigenza di portare ordine e criteri di lavoro più efficienti nella produzione di programmi e dall'urgenza di colmare la distanza che si stava delineando tra le prestazioni dei computer (hardware) e i programmi che li facevano funzionare (software).

Da quegli anni sono iniziate poi le definizioni di altre metodologie di lavoro che fanno riferimento alla programmazione strutturata, secondo un'evoluzione che va sotto il nome di **ingegneria del software**.

Per quanto riguarda l'uso delle strutture di controllo per la stesura degli algoritmi, il riferimento classico è il **Teorema di Böhm-Jacopini** (1966) che può essere così formulato:

Un qualsiasi algoritmo può essere espresso usando esclusivamente le strutture di sequenza, di selezione e di iterazione.

AUTOVERIFICA

Problemi da 24 a 26 pag. 205



MATERIALE ONLINE

4. Approfondimento sul Teorema di Böhm-Jacopini

11 La struttura di alternativa

La struttura alternativa o **selezione** viene rappresentata secondo lo schema:

```
se condizione
allora
    istruzione-a
altrimenti
    istruzione-b
fine se
```

Se la *condizione* è vera, viene eseguita l'*istruzione-a*, altrimenti viene eseguita l'*istruzione-b*. *Istruzione-a* e *istruzione-b* possono indicare, come accade nella maggior parte dei casi, non una sola istruzione, ma un gruppo di istruzioni.

La condizione è un'espressione booleana di cui viene valutata la verità: vengono quindi utilizzati i segni del confronto: <, >, =, >=, <=, <>, e gli operatori booleani AND, NOT, OR, XOR per costruire espressioni logiche combinando tra loro più condizioni.

Il progetto che segue mostra un algoritmo risolutivo con la struttura di alternativa.

PROGETTO 9

Dati due numeri disporli in ordine crescente.

Dati di input: due numeri, primo e secondo.

Dati di output: i due numeri ordinati, minore e maggiore.

Algoritmo in pseudocodifica

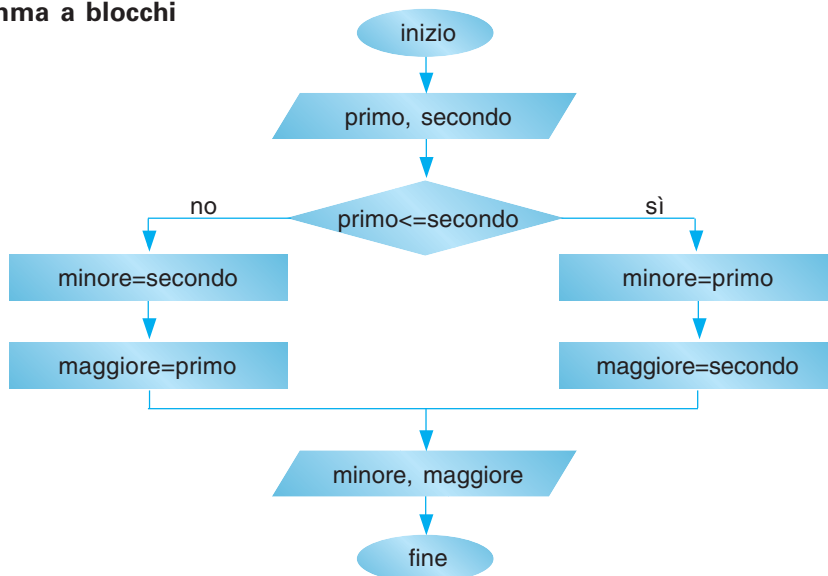
```
inizio
    immetti primo, secondo
    se primo <= secondo
    allora
        assegna minore = primo
        assegna maggiore = secondo
    altrimenti
        assegna minore = secondo
        assegna maggiore = primo
    fine se
    scrivi minore, maggiore
fine
```

Nella struttura di alternativa viene controllata la condizione indicata dopo la parola *se*: se la condizione è vera viene eseguita l'istruzione o il gruppo di istruzioni scritte sotto la parola *allora*; se la condizione è falsa vengono eseguite le istruzioni sotto la parola *altrimenti*.

Si noti che le istruzioni da eseguire quando la condizione è vera sono disposte in modo rientrato (si dice anche *indentato*) rispetto ad *allora*; inoltre, l'insieme di operazioni che devono essere eseguite, se si prende la strada di *allora*, termina quando termina l'indentazione; altrettanto vale per le istruzioni sotto *altrimenti*.

L'uso di una corretta indentazione, quando si descrivono gli algoritmi con la pseudocodifica, consente di facilitare la lettura e la comprensione del procedimento risolutivo.

Diagramma a blocchi



OSSERVAZIONI SULLA STRUTTURA DI ALTERNATIVA

Per la struttura alternativa sono possibili due casi: l'alternativa a due vie e l'alternativa a una via. Nel primo caso è prevista un'azione sia per l'uscita *allora...* sia per l'uscita *altrimenti...*, nel secondo caso sono previste operazioni solo per *allora* . La tipologia delle possibili strutture di alternativa può essere schematizzata nel seguente modo:

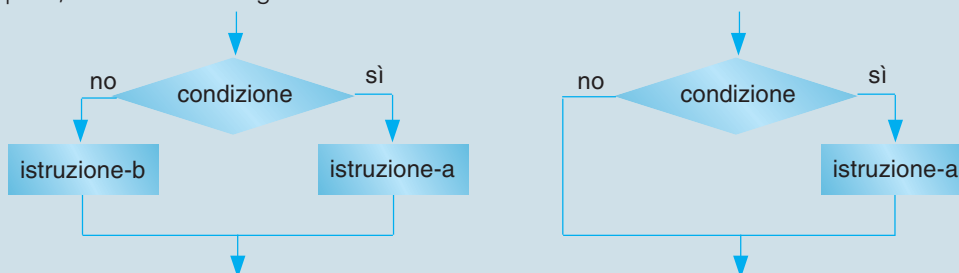
alternativa a due vie

se condizione
allora
 istruzione-a
altrimenti
 istruzione-b
fine se

alternativa a una via

se condizione
allora
 istruzione-a
fine se

Oppure, utilizzando i diagrammi a blocchi:



Si può notare come nella vita quotidiana siano frequenti situazioni di questo genere. Esempi:

“Se piove, vado al lavoro in autobus, altrimenti vado in bicicletta” (2 vie)
 “Se non posso venire, ti telefono” (1 via)
 “Se sono promosso vado in vacanza, altrimenti vado a lavorare” (2 vie)
 “Se prevedo di essere interrogato, mi preparo studiando” (1 via).

All'interno di una struttura alternativa è possibile inserire qualsiasi gruppo di istruzioni, in particolare un'ulteriore struttura alternativa, in questo caso si parla di **se annidati** e si può distinguere tra *se interni* e *se esterni*.

PROGETTO 10

**Su una somma di denaro si vuole applicare un'imposta progressiva secondo lo schema:
 da 0 a 5000 euro: imposta del 10%
 dopo i 5000 euro e fino a 15000 euro: imposta del 20%
 oltre i 15000 euro: imposta dell'30%.**

Dopo aver acquisito in input l'importo a cui applicare l'imposta, si considerano tre possibilità mutuamente esclusive. Se l'importo è minore o uguale a 5000 euro, si applica l'imposta del 10%; in caso contrario se l'importo è minore o uguale a 15000 euro, si applica il 10% sui primi 5000 e si applica il 20% sulla parte restante; come ultima possibilità, resta il caso in cui l'importo superi i 15000 euro: si calcola allora il 10% sui primi 5000, più il 20% sui successivi 10000, più l'30% sulla parte eccedente i 15000 euro.

Dati di input: importo su cui applicare l'imposta progressiva.

Dati di output: imposta da pagare.

Costanti:

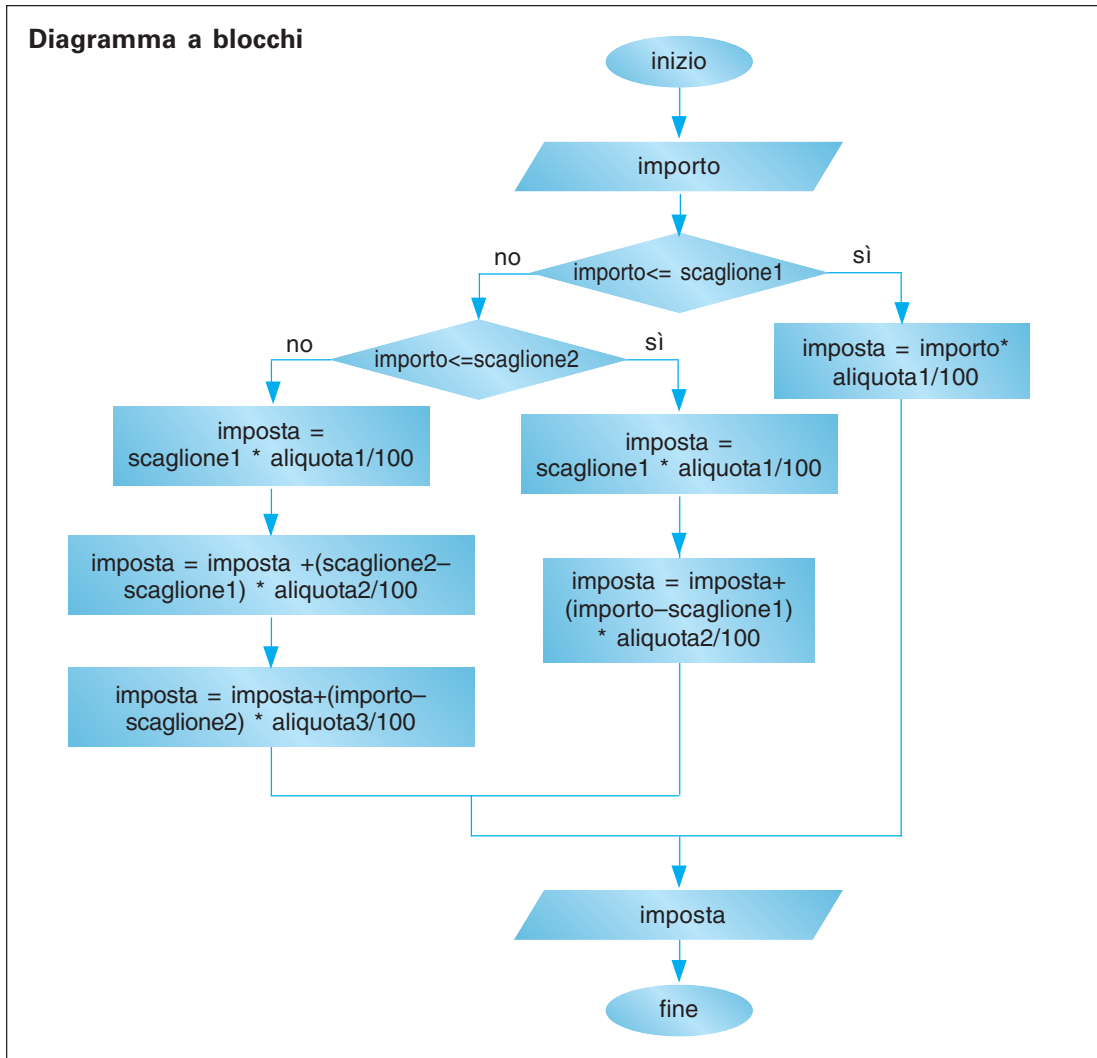
scaglione1 = 5000
 scaglione2 = 15000
 aliquota1 = 10
 aliquota2 = 20
 aliquota3 = 30

Algoritmo in pseudocodifica

```

inizio
  immetti importo
  se importo <= scaglione1
    allora
      calcola imposta = importo * aliquota1 / 100
    altrimenti
      se importo <= scaglione2
        allora
          calcola imposta = scaglione1 * aliquota1 / 100
          calcola imposta = imposta + (importo - scaglione1) * aliquota2 / 100
        altrimenti
          calcola imposta = scaglione1 * aliquota1 / 100
          calcola imposta = imposta + (scaglione2 - scaglione1) * aliquota2 / 100
          calcola imposta = imposta + (importo - scaglione2) * aliquota3 / 100
      fine se
    fine se
  scrivi imposta
fine
  
```

Diagramma a blocchi



AUTOVERIFICA

Problemi da 27 a 36 pag. 205-206

12 La ripetizione

La **ripetizione**, come terza struttura di controllo fondamentale della programmazione, si rappresenta con la schema:

esegui

istruzioni

ripeti finché condizione

La *condizione* deve essere un'espressione che rappresenta un valore *Vero* o *Falso*.

Le istruzioni comprese tra *esegui* e *ripeti* vengono eseguite una prima volta, dopo di che viene verificata la condizione scritta dopo *finché*: se la condizione risulta vera si prosegue con l'istruzione successiva, altrimenti si ripete l'esecuzione delle istruzioni a partire dalla prima istruzione dopo *esegui*.

PROGETTO 11

Dato un elenco di prodotti, con l'indicazione per ciascuno di essi del nome e del prezzo, scrivere il nome dei prodotti che hanno un prezzo superiore a un valore prefissato e fornito all'inizio del procedimento.

Per ogni prodotto dell'elenco si devono richiedere il nome e il prezzo; si deve controllare se il prezzo è superiore al valore fornito e, in questo caso, deve essere scritto il nome del prodotto. Il controllo viene ripetuto finché l'elenco è finito.

Dati di input:

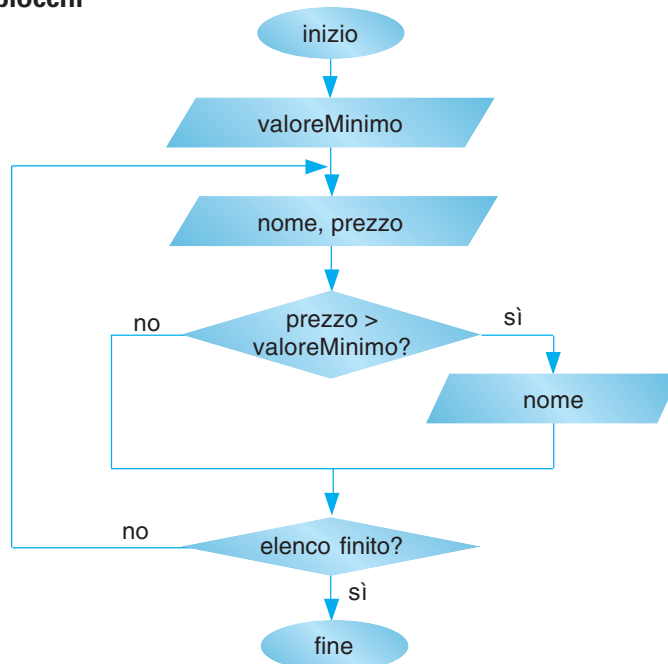
valore minimo prefissato per i prezzi
nome del prodotto
prezzo del prodotto.

Dati di output:

nome dei prodotti con prezzo superiore.

Algoritmo in pseudocodifica

```
inizio
  immetti valoreMinimo
  esegui
    immetti nome, prezzo
    se prezzo > valoreMinimo
      allora
        scrivi nome
      fine se
  ripeti finché elenco finito
fine
```

Diagramma a blocchi



LE STRUTTURE DERIVATE

L'utilizzo delle tre strutture di base è sufficiente, secondo il Teorema di Böhm-Jacopini, per la soluzione di qualsiasi problema.

Per facilitare il lavoro di programmazione si ricorre, però, anche ad altri tipi di strutture che risultano di notevole utilità. Dal punto di vista teorico, però, va notato che le strutture che vengono presentate ora non sono indispensabili, ma sono una derivazione delle strutture precedenti.

Per quanto riguarda la ripetizione, è definita, oltre alla struttura *esegui ... ripeti finché*, anche la struttura *esegui ... ripeti mentre*. Nella prima la ripetizione continua quando la condizione indicata dopo *finché* risulta falsa e si arresta non appena la condizione diventa vera. Nella seconda le istruzioni comprese tra *esegui* e *ripeti mentre* vengono ripetute se la condizione dopo *mentre* è vera e l'iterazione si arresta non appena la condizione diventa falsa.

Le due strutture si rappresentano in pseudocodifica con:

esegui	esegui
istruzioni	istruzioni
ripeti finché condizione	ripeti mentre condizione

Per quanto detto sopra la struttura *esegui...ripeti mentre* è una struttura derivata, cioè può essere realizzata usando la struttura *esegui...ripeti finché*.

Infatti la struttura a sinistra è equivalente alla struttura di destra:

esegui	esegui
istruzioni	istruzioni
ripeti mentre condizione	ripeti finché NON condizione

Inversamente un segmento di algoritmo con *esegui...ripeti finché* può essere realizzato usando la struttura *esegui...ripeti mentre*.

Le strutture di ripetizione viste fino ad ora prendono il nome di **ripetizione postcondizionale**. Il nome sta ad indicare che la condizione, che verifica se si devono ripetere le istruzioni comprese nella struttura, è posta alla fine delle istruzioni. Ne consegue che le istruzioni da ripetere vengono eseguite almeno una volta.

13 Ripetizione precondizionale

È possibile definire anche **ripetizioni precondizionali**, cioè strutture che prima verificano se le istruzioni si devono eseguire e, solo in caso affermativo, procedono all'iterazione delle istruzioni stesse.

In pseudocodifica la ripetizione precondizionale si rappresenta con:

esegui mentre condizione
istruzioni
ripeti

Il segmento di algoritmo precedente significa che si deve prima controllare la condizione: se questa è verificata si procede all'esecuzione delle istruzioni comprese tra *esegui mentre* e *ripeti*. Le istruzioni vengono ripetute quando la condizione è verificata, la ripetizione si arresta quando la condizione diventa falsa.

Se la condizione risulta subito falsa, le istruzioni non vengono eseguite nemmeno una volta.

PROGETTO 12

Calcolo del prodotto tra interi utilizzando la sola operazione di somma.

Presi in considerazione due numeri interi, si tratta di sommare il primo numero con se stesso per un numero di volte pari al secondo numero. Per sapere quante volte deve essere eseguita l'operazione di somma e per sapere quando fermarsi, ogni volta che viene fatta la somma si decrementa di 1 il valore del secondo numero e si interrompe la ripetizione della somma quando questo diventa 0.

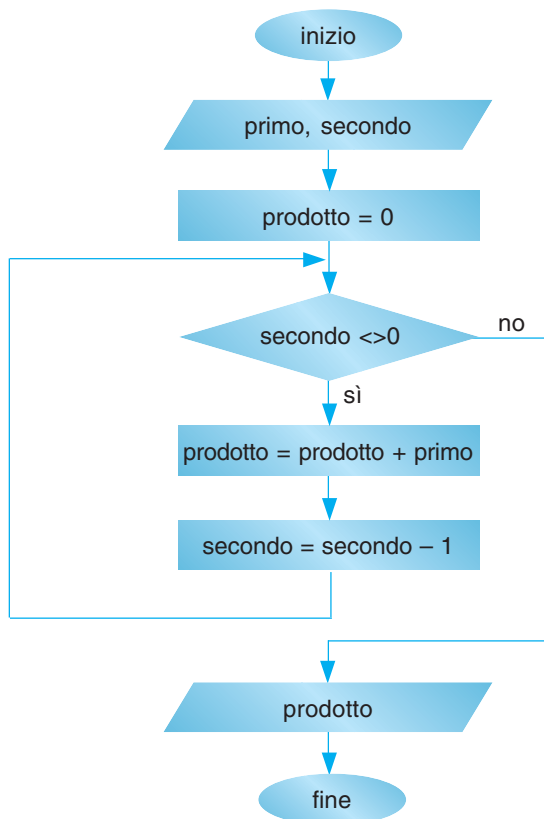
Dati di input: i due numeri da moltiplicare, primo e secondo fattore.

Dati di output: il prodotto dei due numeri.

Algoritmo in pseudocodifica

```
inizio
  immetti primo, secondo
  assegna prodotto = 0
  esegui mentre secondo <> 0
    calcola prodotto = prodotto + primo
    calcola secondo = secondo - 1
  ripeti
  scrivi prodotto
fine
```

Diagramma a blocchi





CONFRONTO TRA LE STRUTTURE DI RIPETIZIONE

Così come per la struttura postcondizionale esistono le due varianti *esegui...ripeti finché* e *esegui...ripeti mentre*, anche per la ripetizione precondizionale esistono sia la struttura *esegui mentre...ripeti* sia la struttura *esegui finché...ripeti*.

In pseudocodifica si indicano con:

esegui finché condizione
 istruzioni
ripeti

esegui mentre condizione
 istruzioni
ripeti

Anche per la ripetizione precondizionale vale l'equivalenza tra le due strutture:

esegui mentre condizione
 istruzioni
ripeti

esegui finché NON condizione
 istruzioni
ripeti

L'iterazione precondizionale può essere considerata una struttura derivata, in quanto un frammento di algoritmo che contenga tale struttura può essere sostituito da un segmento di algoritmo equivalente (cioè che svolga le medesime funzioni e che dia luogo ai medesimi risultati) in cui sia presente soltanto la struttura di iterazione postcondizionale.

Infatti la struttura:

esegui mentre condizione
 istruzioni
ripeti

può essere sostituita dalla struttura del tutto equivalente:

se condizione
 allora
 esegui
 istruzioni
 ripeti finché non condizione
 fine se

AUTOVERIFICA

Problemi da 37 a 42 pag. 206



MATERIALE ONLINE

5. Ripetizione precondizionale per il controllo di fine input

14 La ripetizione con contatore

La struttura di **ripetizione con contatore** (o **enumerativa**) è un'altra struttura derivata dalla ripetizione. Questa nuova struttura permette di ripetere un certo gruppo di istruzioni, non in base al valore di verità di una condizione, ma in base a un numero di volte prefissato.

Se il numero di ripetizioni che si vogliono eseguire è noto a priori, è possibile organizzare una ripetizione precondizionale dove la condizione di arresto è indicata dal valore massimo che può raggiungere una variabile utilizzata come contatore.

La rappresentazione della ripetizione enumerativa avviene in pseudocodifica nel modo seguente:

```
per variabile-contatore da iniziale a finale  
    istruzioni  
ripeti
```

Quando, all'interno di un algoritmo, viene incontrata tale struttura, la *variabile-contatore*, che serve da controllo per la ripetizione, assume il valore indicato con *iniziale*. Si procede poi all'esecuzione del gruppo di istruzioni indicate sotto la parola *per* ed indentate rispetto ad essa. L'esecuzione di queste istruzioni viene ripetuta e, ad ogni ripetizione, viene incrementato il valore della *variabile-contatore*. Mentre tale valore è compreso nell'intervallo che va dal valore *iniziale* al valore *finale*, le istruzioni vengono rieseguite; la ripetizione si arresta quando la *variabile-contatore* diventa maggiore di *finale*.

Una variante rispetto alla versione precedente consiste nell'inserimento dell'incremento (passo) che deve subire la *variabile-contatore* ad ogni ripetizione del gruppo di istruzioni.

Tale variante in pseudocodifica si indica con:

```
per variabile-contatore da iniziale a finale passo incremento  
    istruzioni  
ripeti
```

In questo caso, ad ogni esecuzione delle istruzioni la *variabile-contatore* viene aumentata di un valore pari ad *incremento*.

In mancanza di indicazioni sull'incremento (per *default*, come si dice con linguaggio informatico) si sottintende che l'incremento sia unitario.

La struttura di ripetizione con contatore è derivata dalla struttura fondamentale di ripetizione, nel senso che il controllo della ripetizione delle istruzioni mediante un contatore può essere fatto in modo equivalente usando una struttura *esegui mentre... ripeti*.

Le due strutture riportate sotto sono funzionalmente equivalenti:

ripetizione con contatore	ripetizione precondizionale
per i da 1 a n	assegna i = 1
istruzioni	esegui mentre i <= n
ripeti	istruzioni
	calcola i = i + 1
	ripeti

La struttura di ripetizione con contatore, nel caso di ripetizione con un numero prefissato di volte, risulta più compatta e più semplice da rappresentare, in quanto dentro la frase *per* sono rappresentati l'assegnazione del valore iniziale del contatore e l'incremento del contatore stesso.

PROGETTO 13

Dati gli importi delle spese domestiche per ciascun mese dell'anno, calcolare la spesa totale annuale.

Il numero degli importi è prefissato a 12, quanti sono i mesi dell'anno. All'inizio viene azzerato il totale dell'anno; con una ripetizione con contatore si acquisiscono i valori delle spese di ciascun mese e si sommano al totale. Alla fine della ripetizione viene scritto il totale come risultato dell'elaborazione.

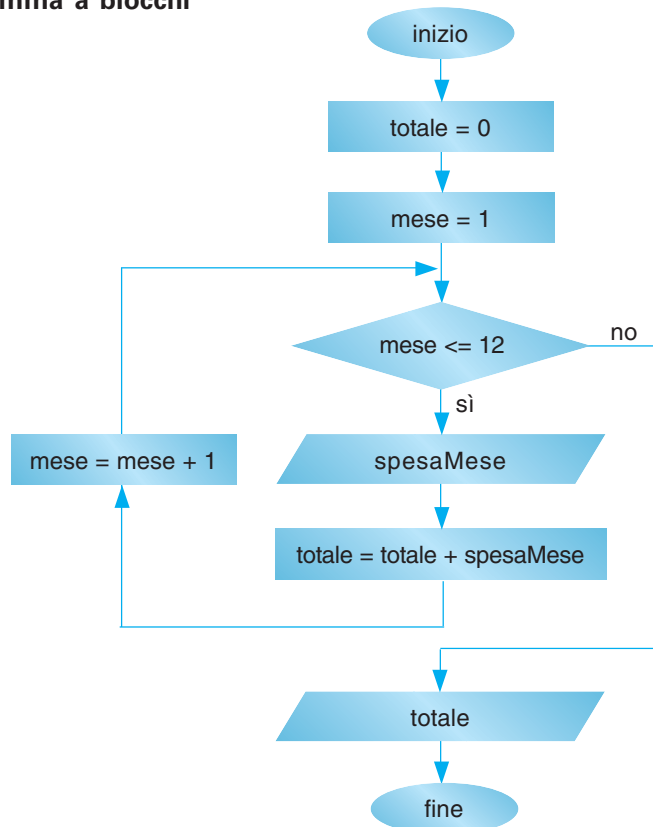
Dati di input: spesa di ciascun mese dell'anno.

Dati di output: totale delle spese nell'anno.

Algoritmo in pseudocodifica

```
inizio
    assegna totale = 0
    per mese da 1 a 12
        immetti spesaMese
        calcola totale = totale + spesaMese
    ripeti
    scrivi totale
fine
```

Diagramma a blocchi



AUTOVERIFICA

Problemi da 43 a 49 pag. 206-207



LA STRUTTURA DI SCELTA MULTIPLA

L'alternativa a due vie non sempre risponde alle necessità di risolvere situazioni più complesse: è stato quindi introdotto lo schema della **scelta multipla** (o *struttura di selezione multipla*).

In pseudocodifica tale struttura può essere rappresentata nel modo seguente:

caso di variabile =

 lista valori-1

 istruzioni-1

 lista valori-2

 istruzioni-2

 ...

 lista valori-n

 istruzioni-n

altrimenti

 istruzioni

fine caso

La presenza della struttura di selezione multipla all'interno di un algoritmo determina al momento dell'esecuzione le seguenti azioni: se il valore della *variabile* è presente in una delle liste di valori, allora viene eseguito il blocco di istruzioni corrispondente; successivamente si procede con l'esecuzione della prima istruzione che segue la struttura *caso di*; in caso contrario viene eseguito il blocco di istruzioni indicato in modo indentato sotto ad *altrimenti*, e poi si passa all'istruzione successiva.

La variabile da controllare si chiama anche **selettore**.

La struttura di selezione multipla è una struttura derivata dalla struttura fondamentale di selezione binaria, infatti può essere rappresentata utilizzando una successione di controlli nidificati del tipo *se.... allora ... altrimenti ... fine se*.

se variabile = valore-1

 allora

 istruzioni-1

 altrimenti se variabile = valore-2

 allora

 istruzioni-2

 altrimenti

 se variabile = valore-n

 allora

 istruzioni-n

 altrimenti

 istruzioni

 fine se

 fine se

fine se

Le due forme sono funzionalmente equivalenti, cioè producono gli stessi output con gli stessi input, ma la prima forma, con *caso di*, risulta più compatta e rende il testo dell'algoritmo più leggibile.

PROGETTO 14

Un volo aereo low-cost per sola andata con destinazione Londra ha prezzi diversi a seconda dei mesi dell'anno e precisamente:

da gennaio a giugno euro 26

luglio e agosto euro 54

da settembre a novembre euro 21

dicembre euro 48.

Fornita la data di partenza, si vuole ottenere il costo del volo.

La data inserita è nella forma gg/mm/aaaa: da essa si può estrarre il valore numerico del mese. Con una selezione multipla, si esaminano i diversi casi: a seconda del numero del mese si scrive il prezzo del volo, secondo la tabella contenuta nel testo del problema.

Dati di input:

data della partenza.

Dati di output:

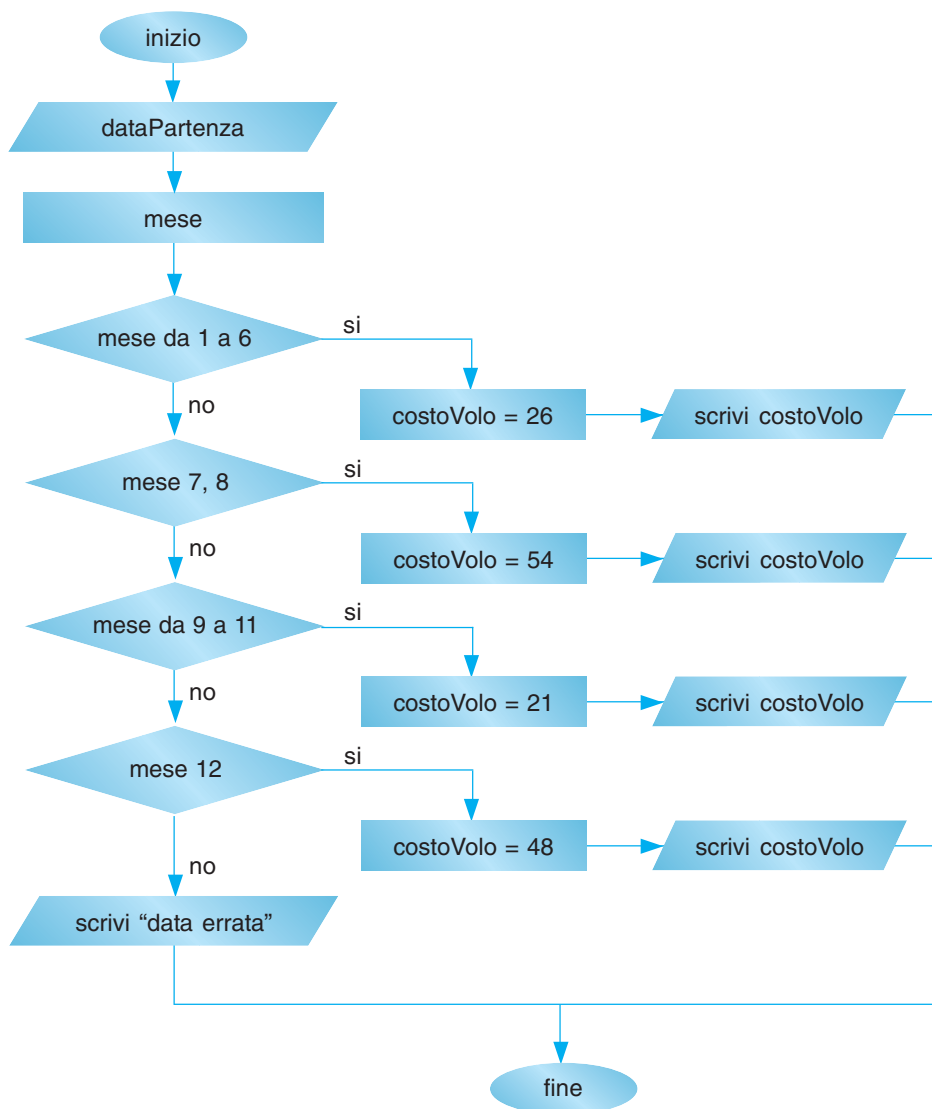
costo del volo.

Algoritmo in pseudocodifica

```
inizio
    immetti dataPartenza
    estrai mese
    caso di mese =
        1 .. 6
            assegna costoVolo = 26
            scrivi costoVolo
        7, 8
            assegna costoVolo = 54
            scrivi costoVolo
        9 .. 11
            assegna costoVolo = 21
            scrivi costoVolo
        12
            assegna costoVolo = 48
            scrivi costoVolo
    altrimenti
        scrivi "data errata"
    fine caso
fine
```

Si noti che la lista di valori con cui viene valutata la variabile *mese* può essere espressa in modi diversi: indicando un singolo valore, oppure valori separati con una virgola (per esempio: 7, 8), oppure un intervallo di valori possibili (per esempio, 1 .. 6).

Diagramma a blocchi



AUTOVERIFICA

Domande da 11 a 14 pag. 200-201

Problemi da 50 a 52 pag. 207

Problemi di riepilogo da 68 a 81 pag. 208-209



MATERIALI ONLINE

6. Algoritmi diversi per la soluzione dello stesso problema

15 Organizzazione degli algoritmi

Nei paragrafi precedenti sono stati presentati problemi relativamente semplici per l'inserimento di valori o la visualizzazione di alcuni dati.

Quando il livello di complessità aumenta, prima di iniziare l'attività vera e propria di programmazione, cioè prima di scrivere del codice, è necessario impostare i termini precisi del problema, ovvero scrivere un **algoritmo**.

Ogni istruzione deve risultare comprensibile e concretamente realizzabile da parte dell'esecutore a cui viene consegnato l'algoritmo. Se ci si riferisce ad un esecutore umano è ovvio che, a seconda delle capacità e del grado di istruzione del destinatario dell'algoritmo, si possono scrivere istruzioni contenenti un grado di difficoltà e di astrazione diversi. Per esempio, se si devono trovare le soluzioni di un'equazione di secondo grado e ci si rivolge a un laureato in matematica, l'algoritmo da eseguire può essere composto solamente di due istruzioni:

inizio

 immetti i coefficienti a, b, c

 risolvi l'equazione

fine

Se l'esecutore è invece uno studente di terza media che ancora non conosce le equazioni di secondo grado, l'algoritmo da eseguire deve contenere un livello di spiegazione molto più dettagliato. Pertanto l'istruzione *risolvi l'equazione* dell'esempio precedente deve essere sviluppata in un algoritmo ben articolato, contenente passo per passo tutte le indicazioni necessarie per chi non abbia mai visto un'equazione di secondo grado. Il livello di astrazione possibile nel definire le istruzioni dipende solo dall'autore dell'algoritmo e dalle capacità dell'esecutore. Quando il livello di difficoltà dei problemi aumenta, ci troviamo di fronte a situazioni in cui è difficile controllare, a un livello di dettaglio basso, i vari aspetti del problema. Si ricorre allora al cosiddetto metodo dei **raffinamenti successivi**.

Il problema principale, che deve essere risolto tramite un programma, viene suddiviso in sottoproblemi più semplici di dimensioni più ridotte che avranno minore complessità.

Si realizza così uno schema logico del tipo indicato nella figura seguente.



Ciascun sottoproblema viene affrontato separatamente e, se presenta ancora un grado di complessità notevole, si procede, come prima, attraverso la suddivisione del problema in sottoproblemi per diminuire ulteriormente il livello di astrazione.



Tale modo di procedere viene detto **sviluppo top-down** in quanto, nel realizzare gli schemi visti prima, si procede dall'alto verso il basso: ad ogni passaggio al livello inferiore si ottiene una suddivisione in sottoproblemi più semplici.

Si realizza una **struttura ad albero** dove ogni riquadro (detto **nodo**) contiene una descrizione del problema e dove viene indicata la **gerarchia delle funzioni**. In questo contesto per *funzione* si intende un segmento di algoritmo che è in grado di risolvere il problema a cui si riferisce. La struttura che si ottiene con lo sviluppo top-down è da considerarsi come un albero rovesciato. Il nodo più in alto viene detto **radice**: esso contiene solo la descrizione (o meglio il titolo) del problema affrontato. I nodi che non presentano diramazioni di livello più basso vengono detti **foglie**. Il livello di dettaglio a cui scendere dipende dalle capacità dell'esecutore. All'interno di ogni nodo va indicata una breve descrizione della funzione svolta: è buona norma descrivere la funzione attraverso un verbo e un oggetto a cui applicare l'azione del verbo. Si tratta in sostanza di definire quali sono le parti fondamentali di cui si compone l'applicazione software (*top*), e procedere poi al dettaglio di ogni singola parte (*down*). Ogni parte corrisponde ad un modulo che svolge una specifica funzione all'interno del problema, cioè l'applicazione deve essere scomposta in **moduli funzionalmente indipendenti**. Questo modo di operare consente di costruire applicazioni software come insiemi di moduli, con il vantaggio di avere sempre sotto controllo il funzionamento del programma, e nello stesso tempo poter avere in ogni istante una visione dettagliata di ogni singolo modulo.

Viene facilitato il lavoro di **manutenzione e aggiornamento del software**, perché si può intervenire con modifiche o correzioni su un solo modulo, avendo nel contempo cognizione di quello che fa l'intero programma.

Inoltre alcuni moduli, essendo funzionalmente indipendenti, possono essere riutilizzati, senza bisogno di molte modifiche, all'interno di altre applicazioni.

La metodologia top-down si realizza organizzando un algoritmo in sottoprocedure o in funzioni. Esse contraddistinguono diversi tipi di sottoprogramma: sia le procedure che le funzioni eseguono delle operazioni di tipo generale, ma solo le funzioni forniscono un valore in output. Vediamo le caratteristiche delle procedure attraverso il seguente progetto.

PROGETTO 15

Calcolo delle soluzioni di equazioni di primo e di secondo grado.

Dati di input:

tipo di equazione
coefficienti dell'equazione (i coefficienti sono due, a e b , per le equazioni di primo grado e tre, a , b , c , per le equazioni di secondo grado)

Dati di output:

numero di soluzioni
soluzioni dell'equazione.

Analisi del problema

Un'equazione di primo grado viene scritta nella forma $ax + b = 0$ mentre una di secondo grado viene scritta nella forma $ax^2 + bx + c = 0$. Le soluzioni sono determinate a partire dai coefficienti.

Nella risoluzione di un'equazione di primo grado si possono incontrare tre casi:

- $a \neq 0$, l'equazione è *determinata*, la soluzione è: $x = -\frac{b}{a}$
- $a = 0$ e $b \neq 0$, l'equazione è *impossibile*, non ha soluzioni.
- $a = 0$ e $b = 0$, l'equazione è *indeterminata*, ha infinite soluzioni.

Un'equazione di secondo grado, nel caso in cui il coefficiente a sia nullo, si risolve come un'equazione di primo grado. Se a non è nullo si calcola il discriminante (*delta*) con la formula $\Delta = b^2 - 4ac$.

Si hanno tre situazioni:

- $\Delta < 0$, l'equazione è *impossibile*, non esistono soluzioni reali;
- $\Delta = 0$, l'equazione ha una soluzione (due soluzioni coincidenti), $x = -\frac{b}{2a}$
- $\Delta > 0$, ci sono due soluzioni reali, distinte $x_1 = \frac{-b + \sqrt{\Delta}}{2a}$ e $x_2 = \frac{-b - \sqrt{\Delta}}{2a}$

Da questa analisi si ricava che la risoluzione del problema presenta due sottoprocedure per i due diversi tipi di equazione; inoltre la sottoprocedura per il calcolo delle soluzioni dell'equazione di secondo grado contiene un'ulteriore sottoprocedura per il calcolo del discriminante:



Come si vede, le sottoprocedure sono indicate solo con un nome e rappresentano le macroistruzioni del procedimento risolutivo, che vengono poi sviluppate in dettaglio.

Sviluppiamo ora in dettaglio le sottoprocedure:

Algoritmo *risolviPrimo* in pseudocodifica

```
inizio
  immetti a, b
  se a = 0
    allora
      se b = 0
        scrivi "indeterminata"
      altrimenti
        scrivi "impossibile"
      fine se
    altrimenti
      calcola x1 = -b/a
      scrivi x1
    fine se
fine
```

Si osservi anche che, nel caso di un'equazione di secondo grado, se l'utente fornisce come primo coefficiente a il valore zero, l'equazione diventa di primo grado e si può utilizzare la sottoprocedura *risolviPrimo*.

L'algoritmo risolutivo della sottoprocedura per l'equazione di secondo grado, a livello poco dettagliato, può essere quindi descritto nel seguente modo:

Algoritmo *risolviSecondo* in pseudocodifica

```

inizio
    immetti a, b, c
    se a = 0
    allora
        risolviPrimo
    altrimenti
        calcolaDelta
        calcolaSoluzioni
    fine se
fine

```

La procedura *risolviSecondo* utilizza le procedure *calcolaDelta* e *calcolaSoluzioni*, che devono essere ulteriormente sviluppate in modo più dettagliato:

Algoritmo *calcolaDelta* in pseudocodifica

```

inizio
    calcola delta =  $b^2 - 4 \cdot a \cdot c$ 
fine

```

Algoritmo *calcolaSoluzioni* in pseudocodifica

```

inizio
    se delta < 0
    allora
        scrivi "impossibile"
    altrimenti
        se delta = 0
        allora
            calcola  $x_1 = -b/(2 \cdot a)$ 
            scrivi  $x_1$ 
        altrimenti
            calcola  $x_1 = (-b + \sqrt{\text{delta}})/(2 \cdot a)$ 
            calcola  $x_2 = (-b - \sqrt{\text{delta}})/(2 \cdot a)$ 
            scrivi  $x_1$ 
            scrivi  $x_2$ 
        fine se
    fine se
fine

```

L'identificatore **sqrt** rappresenta la funzione predefinita per il calcolo della radice quadrata di un numero.

Nel diagramma a blocchi viene introdotto il nuovo simbolo



che serve a rappresentare, all'interno del programma principale, il richiamo a una procedura. Il simbolo rappresenta un algoritmo a cui è assegnato un nome: esso è composto da una o più istruzioni che sono specificate all'esterno del diagramma principale.

Diagramma a blocchi principale

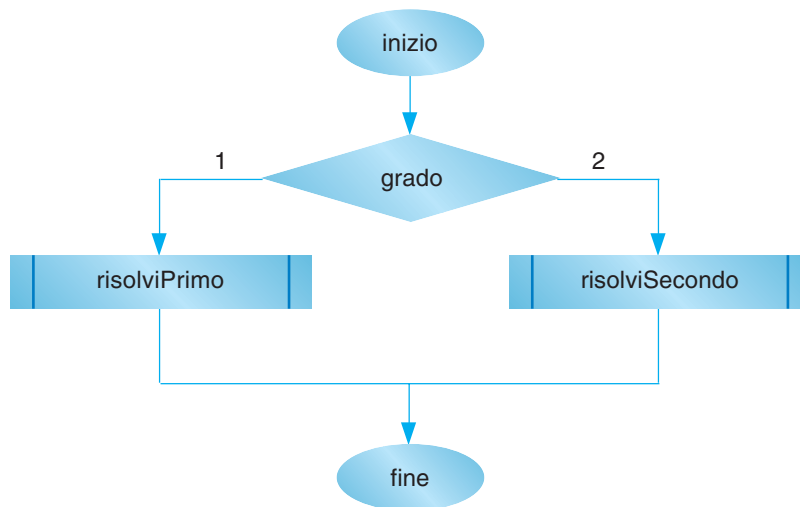


Diagramma a blocchi *risolviPrimo*

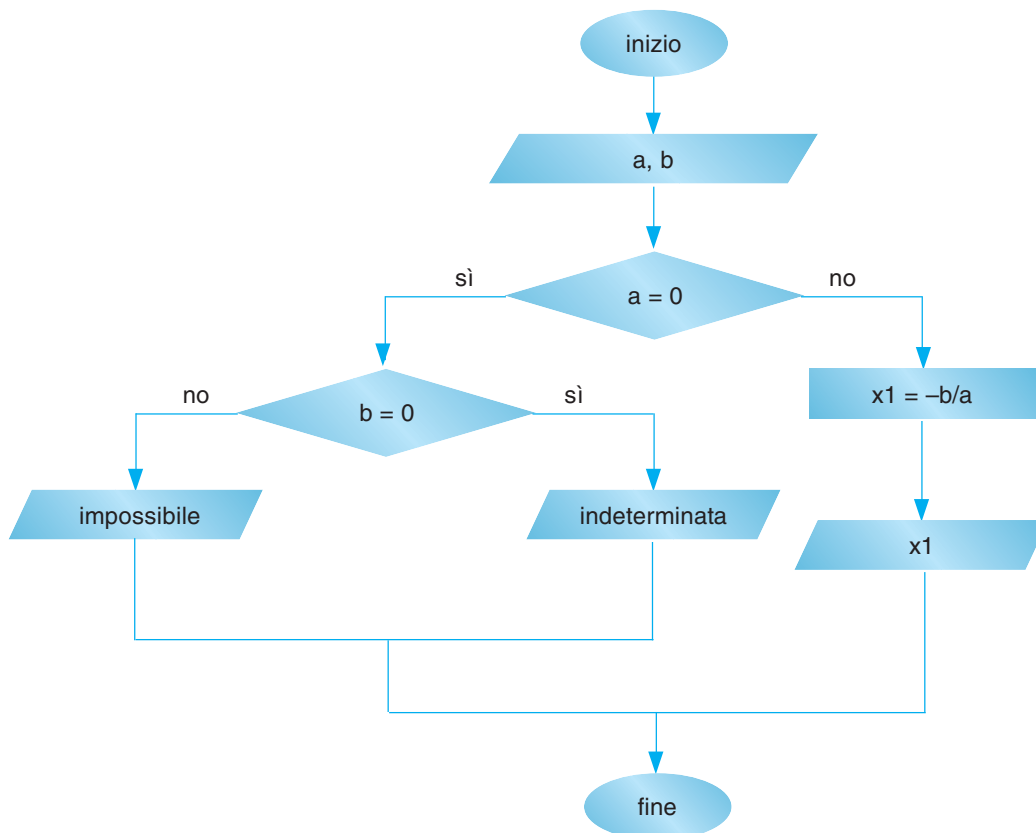


Diagramma a blocchi *risolviSecondo*

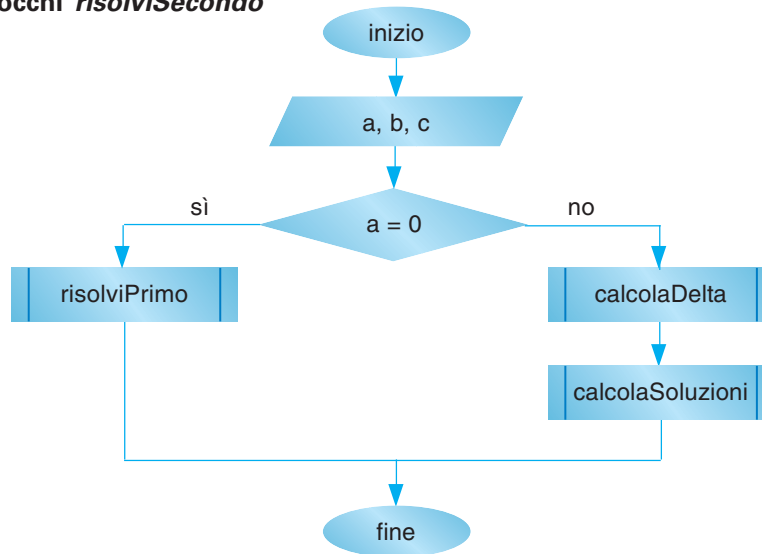


Diagramma a blocchi *calcolaDelta*

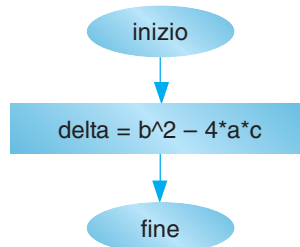
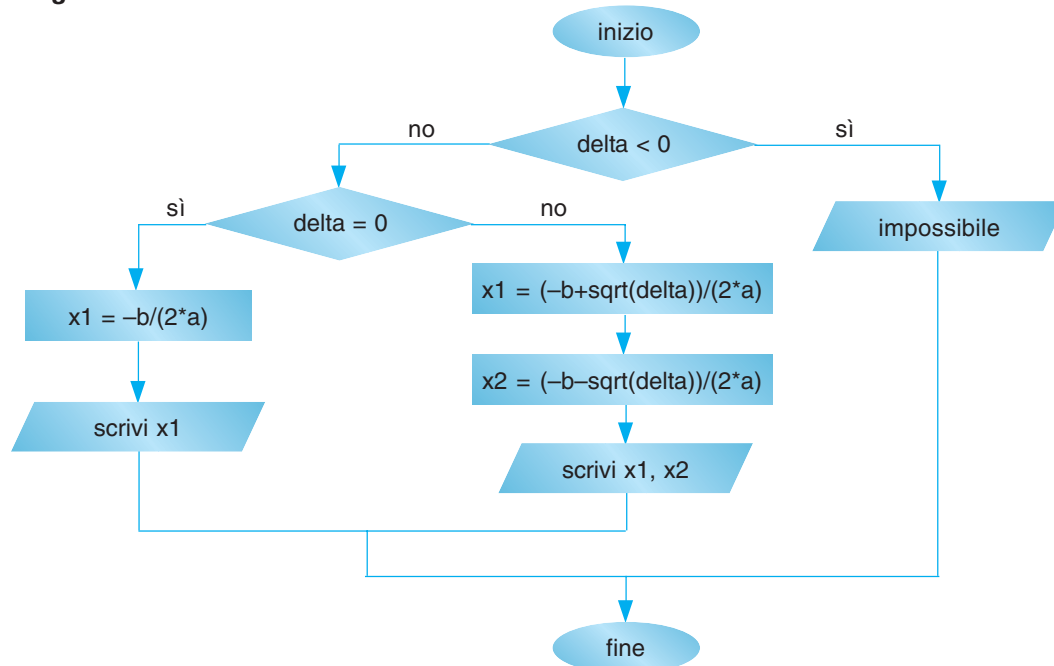


Diagramma a blocchi *calcolaSoluzioni*



Le procedure *risolviPrimo*, *calcolaDelta* e *calcolaSoluzioni* vengono richiamate dalla procedura *risolviSecondo*, che è quindi la **procedura chiamante** mentre le altre sono dette **procedure chiamate**.

La chiamata della procedura è rappresentata nella pseudocodifica scrivendo semplicemente il nome della procedura stessa.

16 Procedure con parametri

L'uso delle procedure risponde all'esigenza fondamentale di costruire programmi ben organizzati e strutturati secondo la metodologia *top-down*. Un'altra esigenza che viene risolta dalle procedure è rappresentata dalla possibilità di riutilizzare sottoprogrammi provenienti da altri progetti: in questi casi potrebbe accadere che le variabili della procedura non abbiano lo stesso nome delle variabili che devono corrispondere nella procedura chiamata.

Questo non è importante, in quanto è possibile, al momento della chiamata di una procedura, passare dei **parametri** che sono indicati tra parentesi dopo il nome della procedura.

L'operazione, con la quale il programma chiamante manda i valori alla procedura, assegnandoli ai parametri, si chiama **passaggio di parametri**.

Si consideri, per esempio, la procedura *risolviPrimo*, inserita nel Progetto 15 del paragrafo precedente, per il calcolo della soluzione di un'equazione di primo grado. Essa è stata utilizzata due volte: la prima per il caso dell'equazione di primo grado, la seconda per il caso particolare in cui un'equazione di secondo grado abbia il primo coefficiente nullo.

La soluzione presentata nel Progetto 15 è formalmente corretta, ma poco efficiente: infatti, nel secondo caso, si deve effettuare una nuova assegnazione dei due coefficienti *a* e *b* per poter riutilizzare la procedura *risolviPrimo*.

In modo più opportuno conviene riscrivere la procedura *risolviPrimo* con due parametri, indicati per esempio con le lettere *p* e *q*, e utilizzare due volte la procedura passando coppie di valori diversi nei due casi.

Il codice della procedura viene così modificato, sostituendo le variabili *a* e *b* con i parametri *p* e *q* che sono indicati nell'intestazione della procedura, nella parentesi dopo il nome.

Algoritmo risolviPrimo (p, q)

inizio

 se $p = 0$

 allora

 se $q = 0$

 allora

 scrivi "indeterminata"

 altrimenti

 scrivi "impossibile"

 fine se

 altrimenti

 calcola $x1 = -q / p$

 scrivi $x1$

 fine se

fine

Le chiamate della procedura deve ora essere modificata aggiungendo, tra parentesi, dopo il nome della procedura, i valori da passare ai parametri p e q :

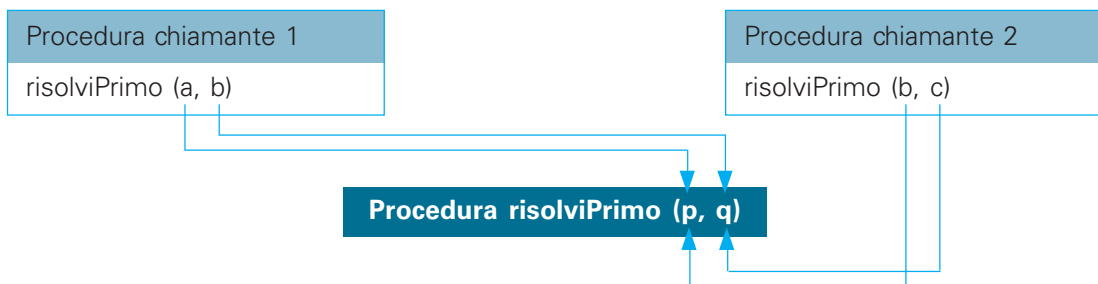
- per il caso dell'equazione di primo grado

```
...  
immetti a, b  
risolviPrimo(a, b)  
...
```

- per il caso dell'equazione di secondo grado che ha il primo coefficiente a uguale a 0

```
...  
immetti a, b, c  
se a = 0  
allora  
    risolviPrimo(b, c)  
altrimenti  
    ...
```

Come si vede, i valori del secondo e del terzo coefficiente sono passati ai parametri p e q della procedura *risolviPrimo*, in modo da riutilizzare la stessa procedura con due valori diversi.



L'uso delle procedure con i parametri risponde quindi all'esigenza di costruire sottoprogrammi che possono essere eseguiti più volte all'interno dello stesso progetto con dati diversi; inoltre le procedure così strutturate diventano moduli software che possono essere utilizzati in altri programmi.

Ovviamente i parametri passati devono essere in numero e di tipo corretto, in caso contrario vi sarà una situazione di errore.

Le variabili indicate nell'intestazione della procedura si chiamano **parametri formali**; le variabili che forniscono i valori ai parametri si chiamano **parametri attuali**.

La lista dei parametri formali, nell'intestazione della procedura, e quella dei parametri attuali, nella chiamata della procedura, devono quindi rispettare una coerenza basata su tre aspetti:

- **numero** dei parametri, cioè il numero dei parametri forniti dalla procedura chiamante deve essere uguale al numero dei parametri formali richiesti dalla procedura chiamata;
- **tipo** dei parametri, cioè parametri attuali e formali devono essere dello stesso tipo;
- **ordine** dei parametri, cioè il passaggio è posizionale, nel senso che il primo parametro attuale manda il proprio valore al primo parametro formale, e così via.

PROGETTO 16

Ordinamento crescente di tre numeri.

L'utente inserisce tre valori numerici. L'algoritmo deve ordinarli dal più piccolo al più grande.

Dati di input: i tre numeri.

Dati di output: i tre numeri ordinati.

Analisi del problema

L'algoritmo risolutivo confronta il primo numero con il secondo, scambiandoli tra loro se non sono in ordine crescente; successivamente confronta il primo con il terzo, scambiandoli tra loro se non sono in ordine crescente, e infine il secondo con il terzo, con eventuale scambio.

Algoritmo in pseudocodifica

```
inizio
    immetti a, b, c
    se a > b
    allora          scambia a con b
        assegna temp = a
        assegna a = b
        assegna b = temp
    fine se
    se a > c
    allora          scambia a con c
        assegna temp = a
        assegna a = c
        assegna c = temp
    fine se
    se b > c
    allora          scambia b con c
        assegna temp = b
        assegna b = c
        assegna c = temp
    fine se
    scrivi a, b, c
fine
```

Al termine dell'algoritmo la variabile *a* contiene il numero più piccolo e la variabile *c* il numero più grande. Si può notare che l'operazione di controllo e scambio del contenuto di due variabili viene eseguita più volte nel corso del programma su variabili diverse (prima *a* e *b*, poi *a* e *c* infine *b* con *c*), ma utilizzando istruzioni identiche.

È conveniente raggruppare quindi queste istruzioni, che vengono usate tre volte, in una singola procedura che verrà chiamata tre volte.

Procedura Ordina (x,y)

inizio

se $x > y$

allora

assegna temp = x

assegna x = y

assegna y = temp

fine se

fine

Algoritmo principale

inizio

Ordina (a,b)

Ordina (a,c)

Ordina (b,c)

fine

L'algoritmo chiama tre volte la procedura *Ordina*, chiedendo di operare su tre differenti coppie di dati. Ogni chiamata causa l'esecuzione della procedura, al termine della quale la coppia di dati forniti risulta ordinata.

Nel progetto precedente gli identificatori, sui quali opera la procedura per fare il confronto e l'eventuale scambio di valore, sono di tipo parametrico e ricevono i valori dall'algoritmo chiamante quando viene attivata la procedura. Nell'intestazione della procedura vengono indicati, tra parentesi dopo il nome, i nomi dei parametri.

**PASSAGGIO DI PARAMETRI**

L'operazione con la quale vengono associate le due liste di parametri attuali e formali, detta *passaggio di parametri*, può avvenire in due modi differenti:

- **Passaggio di parametri per indirizzo (o per variabile o per referenza)**, quando i due parametri fanno riferimento allo stesso *indirizzo* di memoria; questa tipologia di passaggio è da utilizzare quando si vuole che gli eventuali cambiamenti di valore dei parametri formali durante l'esecuzione della procedura si riflettano anche sui valori delle variabili corrispondenti nella procedura chiamante.
- **Passaggio di parametri per valore**, quando i valori passati vengono copiati nei parametri della procedura. In questo caso i cambiamenti sui parametri formali durante l'esecuzione della procedura non influenzano i valori delle variabili nella procedura chiamante.

Il passaggio di parametri per referenza è indicato in pseudocodifica con la parola chiave **Ref** e il passaggio per valore con la parola chiave **Val**, prima del nome del parametro formale nell'intestazione della procedura.

Le procedure descritte nel paragrafo precedente presentano due esempi di utilizzo di queste specifiche per il passaggio di parametri:

Procedura risolviPrimo (**Val** p, q)

Procedura Ordina (**Ref** x, y)

Nella prima procedura i cambiamenti sui parametri formali *p* e *q* durante l'esecuzione della procedura non influenzano i valori delle variabili nella procedura chiamante, mentre nel secondo caso i cambiamenti di valore dei parametri formali *x*, *y* durante l'esecuzione della procedura devono riflettersi sui valori delle variabili corrispondenti nella procedura chiamante.

In generale, il passaggio per indirizzo si usa quando l'elaborazione effettuata all'interno della procedura chiamata deve produrre valori per le variabili del programma chiamante necessari nelle successive elaborazioni del programma; si usa il passaggio per valore quando la procedura chiamata corrisponde ad un modulo che, per svolgere il suo lavoro, ha bisogno dei valori forniti dalla procedura chiamante (per esempio procedure che devono stampare intestazioni o righe di totali), senza la necessità di fornire valori alle variabili del programma chiamante. Nel passaggio di parametri per indirizzo la variabile di memoria utilizzata è la stessa, mentre con il passaggio per valore viene creata una copia del parametro attuale, e il parametro formale, passato per valore, diventa come una variabile locale della procedura chiamata; per far questo viene richiesto più spazio in memoria, che comunque viene rilasciato al termine dell'esecuzione della procedura stessa.

AUTOVERIFICA

Domande da 15 a 17 pag. 201-202
Problemi da 53 a 59 pag. 208

17 Funzioni

In matematica una **funzione** è una relazione tra un insieme detto degli **argomenti** e un altro detto dei **valori**. Analogamente nella progettazione degli algoritmi il termine **funzione** sta ad indicare un sottoprogramma, che ha una struttura simile alle procedure, ma che si comporta in modo del tutto differente.

La **funzione** è un sottoprogramma che riceve dalla procedura chiamante i valori assegnati ai parametri formali e le restituisce un valore. Diversamente dalle procedure che eseguono delle operazioni di tipo generale, le funzioni vengono usate per rappresentare un procedimento di calcolo o, in generale, un'elaborazione che deve fornire un valore alla procedura chiamante.

La struttura generale di una funzione è la seguente:

```
Funzione NomeFunzione (Val x)
inizio
    ...
    NomeFunzione = ...
fine
```

Poiché la funzione deve restituire un valore, all'interno del corpo comparirà sicuramente un'istruzione di assegnazione contenente a sinistra il nome della funzione. Normalmente una funzione riceve i parametri con un *passaggio per valore*.

Per esempio, la funzione per calcolare il risultato dell'addizione di due numeri può essere rappresentata in questo modo:

```
Funzione somma (Val a, b)
inizio
    somma = a + b
fine
```

La funzione riceve dalla procedura chiamante due valori assegnati ai parametri *a* e *b*, e restituisce in *somma* il risultato.

In alternativa è possibile utilizzare l'istruzione **ritorna** per indicare il valore restituito dalla funzione:

Funzione somma (Val a, b)

inizio

 ritorna a + b

fine

Le funzioni vengono usate nel programma come se fossero variabili o espressioni.

Per esempio, l'istruzione:

`totale = somma (x, y) * 2`

assegna alla variabile *totale* il doppio del valore restituito dalla funzione *somma*.

Oppure, la struttura di selezione:

...

se `somma(c, d) > 100`

allora

...

fine se

esegue le istruzioni scritte dopo *allora* nel caso in cui il valore restituito dalla funzione *somma* sia maggiore di 100.

PROGETTO 17

Calcolo del reciproco di un numero.

Dati di input: un numero.

Dati di output: il reciproco del numero.

Analisi del problema

La funzione riceve come parametro un numero e calcola il reciproco del numero. Controlla anche se il valore del numero è zero: in questo caso restituisce il messaggio "*impossibile*".

Algoritmo in pseudocodifica

Funzione reciproco (Val x)

inizio

 se `x=0`

 allora

 reciproco = "*impossibile*"

 altrimenti

 reciproco = $1/x$

 fine se

fine

18 Funzioni ricorsive

Con il termine **ricorsione** (o ricorsività) si indica la possibilità che le procedure e le funzioni hanno di chiamare se stesse, cioè la chiamata della procedura (o della funzione) è contenuta all'interno della procedura stessa.

In sostanza la procedura diventa essa stessa una risorsa per l'attività di elaborazione, insieme alle variabili, alle costanti, ai parametri e agli altri sottoprogrammi. In matematica la quasi totalità delle funzioni possono essere definite attraverso procedimenti di calcolo ricorsivo.

PROGETTO 18

Calcolo del fattoriale di un numero n ($n!$).

Dati di input: un numero intero.

Dati di output: il fattoriale del numero.

Analisi del problema

La notazione $n!$ si legge fattoriale di n e sta ad indicare la moltiplicazione di tutti gli interi minori o uguali ad n :

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 3 \cdot 2 \cdot 1$$

Per esempio:

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$$

Inoltre, per definizione, $0! = 1$ e $1! = 1$.

È possibile dare per il calcolo del fattoriale anche una definizione ricorsiva:

per $n = 0$, $n! = 1$

per $n > 0$, $n! = n \cdot (n-1)!$

Esattamente questa definizione viene utilizzata per realizzare una funzione che calcoli il fattoriale di n .

Nella funzione fattoriale è contenuta la chiamata della funzione stessa, passando come parametro il valore di $x-1$.

La ricorsione è possibile perché ad ogni chiamata della funzione, viene generata in memoria centrale una copia della funzione, come se fosse una funzione diversa.

Funzione fattoriale (Val x)

inizio

se $x = 0$

allora

fattoriale = 1

altrimenti

fattoriale = $x \cdot \text{fattoriale}(x - 1)$

fine se

fine

Algoritmo principale

inizio

immetti n

scrivi fattoriale(n)

fine

AUTOVERIFICA



Domande da 18 a 21 pag. 202

Problemi da 60 a 67 pag. 208

Problemi di riepilogo da 82 a 85 pag. 209



MATERIALI ONLINE

7. Esempi di procedure e funzioni ricorsive

DOMANDE

Caratteristiche degli algoritmi

- 1 Con quale termine viene indicato il trasferimento di un problema sul sistema di elaborazione?
 - a) Traduzione
 - b) Implementazione
 - c) Programmazione
 - d) Traslazione
- 2 Come si chiamano i nomi che diamo alle variabili o alle costanti, per distinguerle all'interno del modello considerato?
 - a) Nominatori
 - b) Operatori
 - c) Operandi
 - d) Identificatori
- 3 Quale tra le seguenti caratteristiche, riferite ai tipi di dato, è errata?
 - a) Alfabetico
 - b) Numerico
 - c) Descrittivo
 - d) Alfanumerico
- 4 Quali sono le attività che in un algoritmo consentono di ottenere i risultati desiderati, mettendo i dati in relazione tra loro?
 - a) Istruzioni
 - b) Controlli
 - c) Azioni
 - d) Trasformazioni
- 5 Completa le frasi seguenti scegliendo tra le parole elencate al termine della domanda.
 - a) Le descrivono le azioni che deve compiere un esecutore.
 - b) Un è formato da più azioni elementari.
 - c) Si dice l'ente che esegue le azioni secondo le istruzioni di un algoritmo.

controllo, processo, modello, algoritmo, processore, istruzioni, variabili
- 6 Completa la seguente tabella indicando per ciascuna istruzione se è di input o di output:

	Input	Output
Leggi		
Immetti		
Visualizza		
Read		
Scrivi		
Display		

- 7 Qual è il significato della frase "Le istruzioni presenti in un algoritmo devono essere elementari"?
- a) Le istruzioni devono essere semplici da capire.
 - b) Le istruzioni devono essere non ulteriormente scomponibili.
 - c) Le istruzioni devono essere formate da un testo breve.
 - d) Le istruzioni devono essere stabilite all'inizio del lavoro.
- 8 Quale tra le seguenti caratteristiche, riferite ad un algoritmo, è errata?
- a) Finito
 - b) Esaustivo
 - c) Traducibile
 - d) Riproducibile

Macchina di Turing

- 9 Quale tra le seguenti definizioni descrive meglio la Macchina di Turing?
- a) Un esempio di computer di piccole dimensioni.
 - b) Un automa in grado di eseguire solo le quattro operazioni aritmetiche.
 - c) Una macchina astratta in grado di eseguire un algoritmo.
 - d) Una macchina in grado di scrivere simboli su un nastro.
- 10 Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
- a) La tesi di Church-Turing determina qual è il procedimento risolutivo di un problema V F
 - b) La tesi di Church-Turing stabilisce che i problemi risolvibili hanno una MdT ad essi associata V F
 - c) Non esiste nessun rapporto tra la Macchina di Turing e il concetto di algoritmo V F
 - d) La sequenza delle azioni eseguite da una Macchina di Turing non è un algoritmo V F

Strutture di controllo

- 11 Che cosa viene scritto in output dopo l'esecuzione delle seguenti istruzioni?
- ```

assegna a=5
assegna b=3
se a>b
allora
 calcola c = a - b
altrimenti
 calcola c = b - a
fine se
scrivi c

```
- 12 Che cosa viene scritto in output eseguendo gli algoritmi a) e b)?
- a)

```

assegna x = 0
esegui
 calcola x = x + 1
 scrivi x
ripeti finché x > 0

```

b)

```

assegna x = 0
esegui mentre x <= 0
 calcola x = x + 1
 scrivi x
ripeti

```



**13** Quali delle seguenti affermazioni sono vere (V) e quali false (F)?

- a) Nella condizione dopo se si può usare l'operatore *And*
- b) La parola *ripeti* al termine della ripetizione si può omettere
- c) In una struttura alternativa, dopo *altrimenti* ci può essere una struttura se annidata
- d) In una struttura di iterazione *per ... ripeti* l'incremento deve essere sempre unitario



**14** L'esecuzione del seguente algoritmo può avere un punto di arresto?

```

assegna x = 1
esegui mentre x <= 3
caso di x
1
 calcola x = x + 1
2
 calcola x = x + 1
3
 calcola x = x - 2
fine caso
ripeti

```

### Procedure e passaggio di parametri

**15** Perché la seguente pseudocodifica non è scritta correttamente?

```

Procedura addizione
inizio
 somma = x + y
 scrivi somma
fine
Algoritmo principale
inizio
 immetti a, b
 addizione (a,b)
fine

```

- a) Le variabili *a*, *b* devono essere sostituite con *x* e *y*.
- b) Nella procedura non sono indicati i parametri formali.
- c) La variabile *somma* viene calcolata in maniera errata.
- d) La chiamata di *addizione* deve essere preceduta dalla parola *Procedura*.

**16** Data la procedura

```

Procedura varia (Ref a, b)
inizio
 a = a + 1
 b = b + 1
fine

```

quali valori assumono le variabili *x* e *y* dopo le seguenti istruzioni?

*x* = 0

*y* = 0

varia(*x*, *y*)

- a) *x*=0 *y*=0
- b) *x*=0 *y*=1
- c) *x*=1 *y*=0
- d) *x*=1 *y*=1

### 17 Data la procedura

Procedura varia (Val a, b)

inizio

a = a + 1

b = b + 1

fine

quali valori assumono le variabili x e y dopo le seguenti istruzioni?

x = 0

y = 0

varia(x, y)

a) x=0 y=0

b) x=0 y=1

c) x=1 y=0

d) x=1 y=1

### Funzioni

#### 18 Quali delle seguenti affermazioni riguardanti le funzioni sono vere (V) e quali false (F)?

a) Una funzione deve sempre restituire un risultato

V F

b) Una funzione è una procedura senza input

V F

c) È possibile utilizzare la parola chiave *output* per indicare il valore restituito

V F

d) È possibile utilizzare la parola chiave *ritorna* per indicare il valore restituito

V F

#### 19 Completa le seguenti frasi riguardanti le funzioni utilizzando una tra le parole elencate alla fine della domanda.

a) Rispetto alle ..... si ha sempre un valore restituito.

b) All'interno del corpo della funzione ci deve essere sicuramente una ..... per la restituzione del valore.

c) La funzione è un sottoprogramma che riceve i valori assegnati ai parametri formali e restituisce un .....

d) Le funzioni vengono usate per rappresentare un .....

**valore, numero, procedure, istruzione, procedimento di calcolo, variabile**

#### 20 Quali delle seguenti istruzioni è un esempio corretto di chiamata ricorsiva della funzione *Fattoriale*?

a)  $Fattoriale = x * Fattoriale$

b)  $Fattoriale(x-1) = x * Fattoriale(x)$

c)  $Fattoriale = x * Fattoriale(x-1)$

d)  $Fattoriale(x) = x * (x-1)$

#### 21 Quali delle seguenti affermazioni riferite al concetto di procedura ricorsiva sono vere (V) e quali false (F)?

a) Una procedura ricorsiva è una procedura che utilizza una ripetizione con contatore

V F

b) Una procedura ricorsiva è una procedura che chiama se stessa

V F

c) Una procedura ricorsiva permette di velocizzare i tempi di calcolo

V F

d) Il calcolo del fattoriale è un esempio di procedimento ricorsivo

V F

## PROBLEMI

### Modello del problema

- 1 Per calcolare il volume di una sfera, conosciuto il raggio, si utilizza la formula  $v = \frac{4}{3} \pi r^3$ . Distinguere in questa formula le variabili e le costanti.

- 2 Fornire tre esempi di variabili.

.....

.....

.....

- 3 Fornire tre esempi di costanti.

.....

.....

.....

*Per i seguenti problemi, produrre uno schema di documentazione delle variabili e costanti, secondo la traccia seguente:*

| Identificatore | Variabile o costante | Descrizione | Tipo  |
|----------------|----------------------|-------------|-------|
| .....          | .....                | .....       | ..... |
| .....          | .....                | .....       | ..... |
| .....          | .....                | .....       | ..... |
| .....          | .....                | .....       | ..... |
| .....          | .....                | .....       | ..... |

- 4 Data la colonna vincente del Totocalcio, si vuole sapere quanti segni sono stati indovinati nella colonna giocata.
- 5 Date le informazioni sui nomi degli studenti, di una classe composta da 27 alunni, e sui voti che essi hanno ottenuto in una prova, si vuole stabilire il voto medio della classe.
- 6 Date le temperature registrate nei giorni del mese di luglio in una città, si vuole sapere in quanti giorni del mese si è avuta una temperatura superiore ai 30°.

*Per i seguenti problemi, individuare dati e azioni:*

- 7 Quanti punti ha in classifica una squadra di calcio alla fine del campionato?
- 8 Calcolare la densità di popolazione delle 10 province di una Regione.
- 9 Si vuole sapere qual è la percentuale, per ogni taglia, delle vendite di una marca di jeans sul totale dei jeans venduti per quella marca, in un negozio di abbigliamento.

Per i seguenti problemi, individuare quali sono i dati di input, di output e di lavoro, utilizzando il seguente schema:

| Dati | Input | Output | Lavoro |
|------|-------|--------|--------|
|      |       |        |        |
|      |       |        |        |
|      |       |        |        |
|      |       |        |        |

- 10** Conoscendo il saldo iniziale di un conto corrente bancario, ed esaminando i movimenti effettuati durante l'anno (a debito o a credito), si vuole comunicare il saldo finale del conto corrente.
- 11** Si abbia a disposizione il valore che rappresenta la produzione industriale nell'anno precedente, e si supponga che sia aumentata del 5% nell'anno in corso: qual è il valore attuale?
- 12** Calcolare la statura media dei giovani selezionati per un gruppo sportivo.

### Pseudocodifica e diagrammi a blocchi

- 13** Disegnare il diagramma a blocchi corrispondente al seguente frammento di pseudocodifica:  
 assegna  $k = 5$   
 calcola  $k = k + 1$   
 scrivi  $k$

Per i seguenti problemi produrre:

- la descrizione delle costanti (se presenti)
- la descrizione dei dati di input, di output e delle eventuali variabili di lavoro
- la pseudocodifica
- il diagramma a blocchi.

- 14** Calcolare il doppio di un numero fornito da tastiera.
- 15** Dato il lato trovare il perimetro e l'area di un quadrato.
- 16** Scrivere il procedimento per scambiare il contenuto di due variabili (occorre usare una terza variabile temporanea).
- 17** Calcolare la misura dell'ipotenusa di un triangolo rettangolo, noti i cateti.
- 18** Date le età di tre persone, calcolare l'età media delle persone.
- 19** Stabilita la percentuale di sconto, dato il prezzo di un prodotto, calcolare lo sconto, il prezzo scontato e comunicare i risultati.
- 20** Dato lo stipendio lordo di un dipendente, calcolare la trattenuta previdenziale e la trattenuta fiscale secondo percentuali di trattenuta fissate come costanti. Calcolare poi lo stipendio netto (lordo - trattenute) e comunicare i risultati.

## Macchina di Turing

- 21 Costruire una semplice Macchina di Turing che legge sul nastro, a partire da sinistra, una sequenza di cifre 0 e 1 e cambia le cifre 0 in 1 e le cifre 1 in 0, finché legge il simbolo # che indica la fine dell'input sul nastro.
- 22 Costruire una semplice Macchina di Turing per il calcolo del successivo di un numero intero. Normalmente la testina scrive la cifra successiva a quella letta nella casella del nastro; quando legge la cifra 9 deve scrivere 0 e passare a uno stato che serve a ricordare il riporto. All'inizio la testina è posizionata sull'ultima cifra a destra, e il movimento avviene verso sinistra.
- 23 Costruire una semplice Macchina di Turing per il calcolo del doppio di un numero intero. All'inizio la testina si trova sull'ultima cifra a destra del numero, e si muove verso sinistra, ricordando il riporto se legge una cifra superiore a 4. Per esempio: se legge 7, scrive 4 e passa allo stato di riporto; se legge 7 e si trova in stato di riporto, scrive 5 e rimane nello stato di riporto, e così via.

## Struttura sequenziale

- 24 Di una città viene fornita la popolazione di due anni successivi. Calcolare l'incremento di popolazione.
- 25 Vengono forniti da tastiera i risultati di un referendum:
  - numero iscritti a votare
  - numero votanti
  - numero dei sì
  - numero dei no.
 Calcolare la percentuale di votanti sul totale degli iscritti, le percentuali dei sì e dei no rispetto al numero dei votanti.
- 26 La popolazione di un Comune due anni fa era di 6340 abitanti; l'anno scorso ha registrato un incremento del 5%. Calcolare la popolazione attuale del Comune.

## Struttura alternativa

- 27 Riscrivere la seguente struttura in modo più efficiente.
 

```

se a=0
allora
 scrivi "valore nullo"
altrimenti
 se a<>0
 allora
 scrivi "valore diverso da zero"
 fine se
fine se

```
- 28 Sul prezzo di un prodotto viene praticato lo sconto del 3% se costa meno di 500 euro e del 5% se il prezzo è superiore o uguale a 500 euro. Calcolare il prezzo da pagare.
- 29 Date le coordinate di due punti scrivere l'equazione della retta passante per essi (controllare che la retta non sia verticale).

- 30 Date le equazioni di due rette nella forma  $y = mx + q$ , trovare le coordinate del punto di intersezione (segnalare anche il caso di rette parallele e di rette coincidenti).
- 31 Data l'equazione di una retta nella forma  $y = mx + q$  e le coordinate di un punto, dire se il punto appartiene alla retta.
- 32 Scrivere un programma che acquisisca da tastiera la quantità e l'importo della merce venduta, lo sconto da applicare se la quantità venduta supera le 100 unità, lo sconto da applicare in caso contrario, l'aliquota IVA: calcolare l'importo da pagare.
- 33 Scrivere un programma per il calcolo dell'acconto di imposta: se il saldo di imposta è negativo non si deve pagare nulla, se è positivo si deve versare il 98% dell'importo come acconto. Se l'acconto è inferiore a 50 euro non si versa nulla.
- 34 La scuola rimborsa il 15% del costo dell'abbonamento se lo studente abita in provincia, usa l'autobus ed è lontano almeno 20 Km dalla scuola. Alle stesse condizioni, se usa il treno il rimborso è del 10%. Calcolare l'importo del rimborso.
- 35 Sul prezzo del biglietto di un treno viene applicato un supplemento del 7% se il treno è di tipo A e del 12% se è di tipo B; per gli altri tipi di treni non è previsto alcun supplemento. Si vuole calcolare il prezzo totale del biglietto, a seconda del tipo di treno.
- 36 Un'imposta sul reddito viene calcolata considerando come imponibile il 70% del reddito e applicando una percentuale del 20% se l'imponibile è minore di una cifra massima prefissata, o del 25% in caso contrario. Dato un reddito, calcolare imponibile e imposta da pagare.

### Struttura di ripetizione

- 37 L'utente deve inserire dei numeri interi, finché si è riusciti a sommare 100 numeri pari. Alla fine si deve comunicare il risultato della somma.
- 38 Data una sequenza di numeri, contare quelli positivi e quelli negativi. Comunicare il numero dei positivi e dei negativi, e comunicare la percentuale dei positivi e dei negativi rispetto al totale dei numeri.
- 39 Su un insieme di articoli di cui sono dati il nome e il prezzo, viene applicato il 15% di sconto. Calcolare la somma dei prezzi e lo sconto totale.
- 40 Si introduce da tastiera un elenco di libri formato da autore, titolo e prezzo. Calcolare il totale e applicare sul totale il 4% in più per le spese di spedizione.
- 41 Uno studente esamina le informazioni sulle verifiche che ha fatto durante l'anno scolastico, e vuole contare quante sono le prove sufficienti.
- 42 All'inizio si fornisce la giacenza iniziale di un prodotto in magazzino. Si introducono poi con una ripetizione: la data, il tipo di movimento (carico o scarico) e la quantità. Alla fine si vuole ottenere la giacenza finale.

### Struttura di ripetizione con contatore

- 43 Scrivere la struttura *esegui mentre* equivalente alla seguente:  
 per  $i$  da 1 a 10  
     calcola  $a = i * 2$   
     scrivi  $a$   
 ripeti

- 44** Calcolare la somma dei primi 10 numeri interi successivi a un numero N fornito all'inizio dell'algoritmo.
- 45** Calcolare la somma dei primi N numeri pari successivi a un numero dato A.
- 46** Calcolare la somma dei quadrati dei primi N numeri interi naturali.
- 47** Dopo aver inserito un numero intero N, sommare i primi N numeri dispari e verificare che tale somma è uguale al quadrato di N.
- 48** Dato un elenco di N numeri, si devono scegliere quelli che sono maggiori di 10 e minori di 100. Contare tali numeri e scrivere la loro somma.
- 49** Dati N Comuni, di ciascuno si conosce: il nome, il numero degli occupati e il numero dei lavoratori del settore agricolo. Si vuole calcolare il numero totale degli occupati in agricoltura e la percentuale degli occupati in agricoltura rispetto al totale degli occupati.

### Struttura di scelta multipla

- 50** Scrivere la struttura se equivalente alla seguente struttura di scelta multipla:

caso di  $n =$

2

calcola  $n = n + 1$

3

assegna  $n = 2$

4

calcola  $n = n - 1$

altrimenti

assegna  $n = 0$

fine caso

- 51** Si supponga che le tariffe postali per lettere raccomandate siano le seguenti:

| PESO            | EURO |
|-----------------|------|
| Fino a 25 gr.   | 1,00 |
| da 26 a 50 gr.  | 2,00 |
| da 51 a 100 gr. | 3,50 |
| oltre i 100 gr. | 4,50 |

Per ogni lettera si deve aggiungere anche un costo fisso di 1 euro. Dato il peso della lettera si calcoli il costo totale dell'affrancatura.

- 52** Sul prezzo di un prodotto viene applicato uno sconto variabile in base alla confezione di pezzi acquistati secondo la tabella seguente:

| Pezzi acquistati | Percentuale di sconto |
|------------------|-----------------------|
| 100              | 2%                    |
| 200              | 6%                    |
| 400              | 8%                    |
| 500              | 10%                   |
| più di 1000      | 15%                   |

Calcolare la spesa totale.

## Procedure e passaggio di parametri

- 53** Organizzare una procedura che calcoli il perimetro e l'area di un quadrato conoscendone il lato.
- 54** Definire una procedura che abbia come parametro un numero intero positivo e che visualizzi la sequenza dei primi 5 numeri successivi al numero dato.
- 55** Definire una procedura che calcoli la misura di una circonferenza conoscendone il raggio.
- 56** Definire una procedura che calcoli il raggio di un cerchio conoscendone la circonferenza.
- 57** Dato un elenco di reparti con l'indicazione del nome e dell'incasso giornaliero, calcolare l'incasso medio. Definire la procedura per l'acquisizione dei dati e la procedura per il calcolo della media.
- 59** Data la parabola  $y = ax^2 + bx + c$ , definire tre procedure per calcolarne il vertice, il fuoco e l'intersezione con gli assi. Le tre procedure ricevono come parametri i coefficienti  $a$ ,  $b$ ,  $c$ .
- 59** Data la retta  $y = mx + q$  e un punto di coordinate  $(x1, y1)$ , scrivere una procedura per controllare se il punto appartiene alla retta.

## Funzioni

- 60** Scrivere una funzione che restituisca l'età di una persona, conoscendo il suo anno di nascita.
- 61** Scrivere una funzione che abbia come parametri il prezzo di vendita e la percentuale di sconto e restituisca il prezzo scontato.
- 62** Scrivere la funzione *radice2* che calcoli la radice quadrata di un numero.
- 63** Scrivere una funzione per stabilire se due rette  $a_1x + b_1y + c_1 = 0$  e  $a_2x + b_2y + c_2 = 0$  sono perpendicolari.
- 64** Dato un numero, fornire la sua rappresentazione in esadecimale.
- 65** Dato un numero binario, acquisito da tastiera come una stringa formata da cifre 0 e 1, calcolare il suo valore esadecimale.
- 66** Calcolare il prodotto tra due numeri interi positivi  $a$  e  $b$  in modo ricorsivo e utilizzando la sola operazione di somma. Utilizzare la seguente definizione:  
se  $b = 0$  allora  $a*b = 0$   
se  $b > 0$  allora  $a*b = a + a*(b - 1)$
- 67** Calcolare la somma dei primi  $n$  interi utilizzando la definizione ricorsiva:

$$\sum_{i=1}^1 i = 1$$

$$\sum_{i=1}^n i = n + \sum_{i=1}^{n-1} i$$

## PROBLEMI DI RIEPILOGO

- 68** Dato il raggio del cerchio di base di un cilindro e la sua altezza, calcolare il peso del cilindro sapendo che la sostanza di cui è composto ha peso specifico =  $3,4 \text{ g/cm}^3$  (Volume = area della base x altezza; Peso = Volume x Peso specifico).



- 69** Una nuova compagnia telefonica ha promosso l'offerta "oltre80": alla cifra fissa di 0,10 euro (costo alla risposta) occorre aggiungere la cifra di 0,15 euro per ogni secondo del tempo della telefonata; però oltre gli 80 secondi la tariffa per ogni secondo è di 0,09. Fornito il numero dei secondi della telefonata, visualizzare il costo totale della chiamata.
- 70** Conoscendo il risultato di una partita di calcio ad eliminazione diretta (non è consentito il pareggio), si vuole visualizzare il nome della squadra vincente.
- 71** Dati i nomi di tre studenti e la distanza in Km. da casa a scuola, visualizzare il nome degli studenti che abitano in comuni distanti più di 30 Km.
- 72** Sul consumo di gas si paga 0,65 euro al mc per un consumo annuo inferiore a 500 mc, altrimenti 0,85 al mc. Quanto si deve pagare?
- 73** Dato il reddito di una persona, se è inferiore a 3000 euro l'imposta è uguale a 0, altrimenti fino a 30000 euro paga il 29% di imposta, sul resto il 35%. Calcolare l'imposta totale da pagare.
- 74** Ad un concorso pubblico hanno partecipato due candidati di cui si inseriscono i nomi e i punteggi conseguiti. Scrivere l'elenco dei candidati prima in ordine alfabetico e poi in ordine decrescente di punteggio.
- 75** Costruire un algoritmo che accetta un elenco di nomi e man mano li visualizza.
- 76** Costruire un algoritmo che accetta un elenco di nomi e li conta. Alla fine fornisce il numero dei nomi inseriti.
- 77** Dato un elenco di nomi con le rispettive città, conta quanti sono quelli di Milano.
- 78** Costruire un algoritmo che accetta da tastiera un elenco di N prezzi e li somma. Alla fine fornisce il totale dei prezzi inseriti.
- 79** Dato un elenco di prodotti con i rispettivi prezzi, conta quanti sono quelli che hanno il prezzo superiore a 100 euro.
- 80** Dato un elenco di studenti con nome e giudizio ottenuto in una prova scolastica (sufficiente, buono, distinto, ottimo), comunicare il numero degli studenti che hanno avuto il giudizio "ottimo".
- 81** Dato un elenco di computer con descrizione, tipo e prezzo in euro, scrivere la descrizione e il prezzo dei computer appartenenti a un tipo prefissato e fornito all'inizio.
- 82** I voti ottenuti da due candidati in un ballottaggio sono passati come parametri a una procedura che calcola le percentuali ottenute da ciascuno e visualizza i risultati in ordine decrescente.
- 83** Da tastiera vengono introdotti per N prodotti il nome del prodotto, la quantità venduta e il prezzo unitario. Calcolare il prezzo medio, la quantità media e il valore totale delle vendite.
- 84** Di N studenti iscritti in una scuola si introducono da tastiera il nome, il Comune di provenienza e la distanza in km dalla scuola. Si devono comunicare: il numero degli studenti residenti in comuni diversi dal capoluogo di provincia; la percentuale di questi rispetto al totale degli studenti considerati; la distanza media in km calcolata su tutti gli studenti.
- 85** Per ciascun giorno del mese si conoscono le temperature minima e massima di una città. Dopo aver inserito i dati, si calcoli il valore medio delle minime e delle massime.



## SCHEDA DI AUTOVALUTAZIONE

### CONOSCENZE

- ☐ Variabili e costanti, dati e azioni
- ☐ La metodologia di lavoro nella formalizzazione dei problemi
- ☐ Definizione e caratteristiche di algoritmo
- ☐ Operazioni di input e di output
- ☐ Gli operatori
- ☐ Strumenti per la stesura di un algoritmo
- ☐ L'individuazione dei dati di un problema
- ☐ La macchina di Turing
- ☐ La programmazione strutturata
- ☐ Le strutture di controllo
- ☐ Teorema di Böhm-Jacopini
- ☐ Le strutture derivate
- ☐ Lo sviluppo top-down e l'organizzazione dei programmi
- ☐ Procedure con parametri
- ☐ Passaggio di parametri
- ☐ Le funzioni
- ☐ Ricorsività dei sottoprogrammi

### ABILITÀ

- ☐ Saper distinguere all'interno di un problema tra variabili e costanti, tra dati ed azioni
- ☐ Utilizzare la pseudocodifica per rappresentare gli algoritmi
- ☐ Rappresentare graficamente gli algoritmi con i diagrammi a blocchi
- ☐ Definire una Macchina di Turing in modo formalizzato
- ☐ Costruire algoritmi strutturati
- ☐ Rappresentare le strutture di controllo
- ☐ Individuare le strutture di controllo più idonee per la soluzione di un problema
- ☐ Raffrontare le strutture di base e le strutture derivate in termini di efficienza e di utilità
- ☐ Utilizzare il metodo dei raffinamenti successivi per la soluzione di problemi complessi
- ☐ Utilizzare il passaggio di parametri per valore e per indirizzo
- ☐ Distinguere i problemi in cui è necessario utilizzare una funzione o una procedura
- ☐ Dichiarare e utilizzare una funzione
- ☐ Comprendere l'importanza di procedimenti ricorsivi
- ☐ Individuare la possibilità di utilizzare sottoprogrammi ricorsivi



**SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 662**

# 5

**parte terza**

La programmazione

## Il linguaggio C++

### **OBIETTIVI DI APPRENDIMENTO**

In questo capitolo imparerai la struttura generale di un programma in linguaggio C++ e le caratteristiche principali dei dati, delle istruzioni e degli operatori.

Sarai in grado di scrivere i programmi in C++ utilizzando in modo corretto la sintassi delle istruzioni di input/output e delle strutture di controllo.

Potrai riconoscere le diverse fasi del lavoro di programmazione per codificare e validare gli algoritmi. Imparerai anche a scomporre il programma in funzioni e a riutilizzare più volte le stesse funzioni assegnando diversi valori ai parametri.

Sarai in grado di organizzare dati dello stesso tipo o di tipo diverso, associando ad ogni situazione problematica la struttura di dati più idonea.

Le basi del linguaggio  
Gli statement  
Variabili e costanti  
I tipi di dati e il casting  
Le frasi di commento  
L'assegnazione dei valori alle variabili  
Operatori aritmetici e logici  
Le istruzioni di ingresso e uscita  
Le fasi della programmazione  
L'importanza della documentazione  
La programmazione strutturata  
Le strutture di sequenza, di alternativa e di ripetizione  
Lo sviluppo top-down  
Le funzioni  
Il passaggio di parametri  
Prototipi di funzioni  
Regole di visibilità  
Le funzioni predefinite  
Namespace e librerie di inclusione  
Function overloading  
La ricorsione  
Costanti ed enumerazioni  
Array  
Strutture  
Puntatori

# 1 Le basi del linguaggio

Come per i linguaggi naturali, in un linguaggio di programmazione si devono rispettare:

- il **lessico**, cioè l'insieme delle parole riconosciute dal linguaggio;
- le regole della **sintassi**, cioè le modalità per scrivere le frasi in modo corretto.

Oltre alle parole che fanno parte del linguaggio, in un programma compaiono gli **identificatori**, cioè i nomi che il programmatore assegna alle specifiche componenti da lui definite, quali costanti, variabili e sottoprogrammi.

Come avviene per molti linguaggi di programmazione oggi disponibili, il linguaggio C++ permette di costruire con precisione ed efficacia un programma ben strutturato.

Iniziamo con un esempio molto semplice, che ci consente di mettere in evidenza le operazioni più importanti da svolgere per preparare ed eseguire un programma.

Supponiamo di voler scrivere un programma C++ per calcolare la somma di due numeri.

Il procedimento da seguire consiste nel prendere in ingresso due numeri, che indichiamo con  $a$  e  $b$ , sommarli e produrre in uscita il risultato, che indichiamo con  $s$ : cioè

$$s = a + b.$$

$a$ ,  $b$  sono i dati di input,  $s$  è il dato di output, e l'addizione è l'attività di elaborazione richiesta.

L'algoritmo che risolve il problema in esame è il seguente:

## Algoritmo Somma

inizio

    immetti  $a$ ,  $b$

    calcola  $s = a + b$

    scrivi  $s$

fine

Si noti che nella descrizione del procedimento non si usano numeri, ma lettere che stanno al posto dei numeri, cioè **variabili**: questo consente di scrivere procedimenti di carattere generale, offrendo la possibilità di eseguire più volte lo stesso procedimento con dati diversi.

Per far eseguire questo semplice algoritmo a un computer, occorre indicare la sequenza delle istruzioni usando il linguaggio di programmazione.

In C++ il procedimento precedente viene scritto in questa forma:

```
// Somma.cpp: somma di due numeri
#include <iostream>
using namespace std;

int main()
{
 int a, b, s;
 cin >> a;
 cin >> b;
 s = a + b;
 cout << s;

 return 0;
}
```

La prima riga è un **commento**, non obbligatorio ma documentativo, che inizia con una coppia di barre //. In questo caso è stato scritto il nome con il quale il programma è stato salvato su disco (*Somma.cpp*), in un file di testo con l'estensione **.cpp** (abbreviazione di *C plus plus*) e una breve descrizione delle funzionalità del programma stesso.

Il software dei computer moderni si basa su una forte interazione della macchina con l'utente dei programmi, essenzialmente sull'acquisizione dei dati tramite l'*unità standard di input* (tastiera) e la visualizzazione di messaggi e di risultati sull'*unità standard di output* (video).

Il linguaggio C++ non possiede apposite istruzioni per la gestione dell'interazione con l'utente, ma si appoggia sulle funzioni delle librerie standard del linguaggio, dette **librerie di I/O** (Input/Output). Il programma che utilizza operazioni di input e output deve quindi contenere all'inizio un'apposita dichiarazione (direttiva **include**) per includere in esso la libreria di I/O:

```
#include <iostream>
```

Il file *iostream* contiene la definizione delle risorse e la descrizione delle funzioni (**prototipi**) per le operazioni di I/O, utilizzabili poi in un qualsiasi punto del programma, in particolare la definizione di **cin** per rappresentare il flusso dei dati acquisiti dalla tastiera e di **cout** per il flusso dei risultati visualizzati sul video.

Un programma complesso può contenere e utilizzare un grande numero di nomi e identificatori per i dati e le istruzioni. Risulta quindi comodo, per evitare ambiguità sull'uso degli identificatori, raggruppare i nomi in contenitori, detti **namespace** (*spazio dei nomi*), predefiniti nel linguaggio o definiti dai programmatori.

In particolare gli identificatori standard del linguaggio C++ sono contenuti nel *namespace* **std** (abbreviazione di *standard*).

Quando poi si usa un identificatore nel programma occorre scrivere il nome del *namespace* e il nome dell'identificatore separati da una coppia di due punti ::, detta **risolutore di visibilità** (in inglese **scope resolution operator**).

Per esempio, un'istruzione di lettura di un dato dalla tastiera si può scrivere:

```
std::cin >> a;
```

Per evitare di ripetere ogni volta il nome del *namespace*, si può scrivere una dichiarazione iniziale nel programma, con la quale si specifica che si intende usare un determinato *namespace* e i nomi in esso contenuti. La dichiarazione viene fatta con la clausola **using namespace**, per esempio:

```
using namespace std;
```

come nel programma precedente. È possibile usare più di un *namespace* in un programma.

Gli aspetti del linguaggio C++ riguardanti le librerie di inclusione e i *namespace* saranno ripresi nei capitoli successivi. Queste note iniziali servono a spiegare brevemente il significato delle prime righe del programma, che da ora in poi saranno presenti in tutti i progetti.

Il programma vero e proprio inizia con la parola **main()**, seguita da una parentesi graffa aperta, che indica l'inizio della sequenza delle istruzioni da eseguire. La sequenza termina con una parentesi graffa chiusa.

Per inserire le parentesi graffe, usando una tastiera che non riporta esplicitamente il tasto, si può digitare una tra le seguenti combinazioni di tasti:

|   |                                 |                        |                   |
|---|---------------------------------|------------------------|-------------------|
| { | Alt + 123 (tastierino numerico) | Ctrl + Alt + Shift + [ | Shift + AltGr + [ |
| } | Alt + 125 (tastierino numerico) | Ctrl + Alt + Shift + ] | Shift + AltGr + ] |

Subito dopo la parola *main* c'è una coppia di parentesi tonde, perché, come vedremo meglio in seguito, il programma può essere considerato una funzione che può ricevere argomenti: essi, se presenti, vengono indicati tra le parentesi tonde.

Una funzione, in generale, oltre a ricevere gli argomenti necessari all'elaborazione, restituisce (o, come si dice correntemente in informatica, *ritorna*) un valore come risultato dell'elaborazione, proprio come accade per le funzioni matematiche.

La funzione *main* restituisce al sistema operativo un valore 0 (*Falso*) o diverso da 0 (*Vero*) per indicare, rispettivamente, l'assenza o la presenza di uno stato di errore al termine dell'esecuzione. Questo spiega l'istruzione finale del programma precedente:

```
return 0;
```

Il *main* ritorna il valore 0, cioè errore = Falso, come dire "non si sono verificati errori". Quindi il valore restituito dalla funzione *main* è un numero intero: il tipo di dato restituito è indicato prima della parola *main* con la specificazione **int**, che significa appunto numero intero. Occorre precisare che, secondo lo standard ISO C++, l'esplicitazione dell'istruzione *return 0* non è obbligatoria: nei progetti successivi sarà comunque scritta per una maggiore chiarezza nella lettura del codice dei programmi e per compatibilità con versioni precedenti del linguaggio. La prima riga dopo *main* specifica quali sono le variabili utilizzate nel programma e di che tipo sono (**int**, cioè numeri interi).

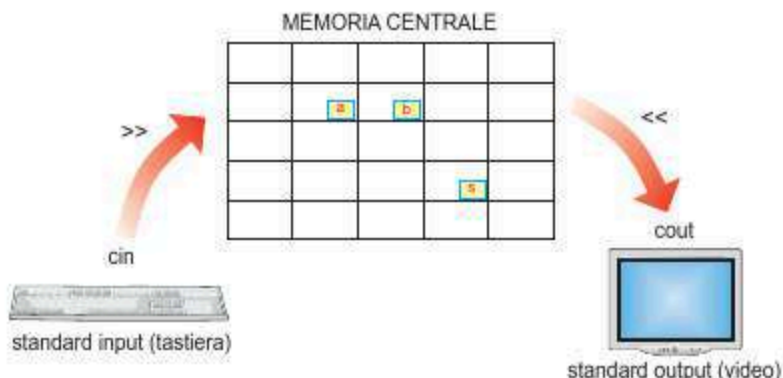
```
int a, b, s;
```

Nelle righe successive vengono scritte le istruzioni da eseguire, nella stessa sequenza prevista dall'algoritmo:

```
cin >> a;
cin >> b;
s = a + b;
cout << s;
```

**cin>>** indica la lettura dei dati dalla tastiera (*input*), l'istruzione  $s = a + b$  specifica che il risultato del calcolo  $a + b$  deve essere assegnato alla variabile *s*, e infine **cout<<** ordina la visualizzazione del risultato di *output* (variabile *s*) sullo schermo del computer.

I simboli **>>** per *cin* e **<<** per *cout* rappresentano le frecce di direzione (verso) del flusso dei dati dalle unità standard di I/O alle variabili in memoria centrale e viceversa. In particolare il valore acquisito dal flusso *cin* sullo **standard input** (tastiera) viene inviato alla memoria centrale in una cella identificata con il nome simbolico *a*. Viceversa il valore contenuto nella cella di memoria identificata con il nome simbolico *s* viene inviato al flusso *cout* sullo **standard output** (video).



Tutte le righe terminano con il *punto e virgola*, come se fossero le frasi di un testo scritto in un linguaggio naturale, ad eccezione delle righe di commento e della dichiarazione di inclusione *#include*.

Nel programma C++ precedente si possono quindi riconoscere principalmente tre sezioni:

- intestazione
- zona delle dichiarazioni
- sezione esecutiva.

Il linguaggio C++ è un linguaggio **case-sensitive**, nel senso che distingue tra lettere maiuscole e minuscole, per cui la variabile *numero* è diversa dalla variabile *Numero* e dalla variabile *NUMERO*.

Nella consuetudine della programmazione nel linguaggio C++, si usano normalmente i caratteri minuscoli sia per le parole riservate del linguaggio sia per le variabili e i caratteri maiuscoli per le costanti; i nomi composti sono indicati spesso con l'uso delle maiuscole nelle iniziali (per esempio, *TriangoloRettangolo*).

Un programma è composto di dati e istruzioni: le istruzioni sono le azioni che trasformano l'insieme dei dati di input nell'insieme dei dati di output.

Nei prossimi paragrafi vedremo più in dettaglio come si rappresentano in un programma C++ i dati e le istruzioni.

I dati utilizzati all'interno di un programma possono essere:

- **costanti**, se non cambiano il loro valore durante l'esecuzione del programma
- **variabili**, se cambiano il valore.

I dati trattati in un programma possono essere:

- **numerici**, quali età, importi, stipendi, misure
- **alfanumerici** (o **stringhe**), quali nomi, descrizioni, codici.

Sui dati di tipo numerico si possono effettuare le usuali operazioni aritmetiche, che vengono rappresentate con gli **operatori**:

- + per l'addizione
- per la sottrazione
- \* per la moltiplicazione
- / per la divisione
- % per il calcolo del resto della divisione tra interi.



MATERIALE ONLINE

## 1. I linguaggi C e C++

## 2 Gli statement

Un programma si compone di istruzioni (**statement**), cioè righe che rappresentano dichiarazioni di oggetti e di risorse che verranno usate nel programma (istruzioni dichiarative), e righe che contengono le istruzioni che governano l'elaborazione del programma (istruzioni esecutive). In particolare le **istruzioni dichiarative** comprendono:

- le direttive del linguaggio
- l'inizio e la fine del programma
- la dichiarazione dei moduli software esterni al programma
- le dichiarazioni dei dati e del loro tipo
- le frasi di commento, che servono a documentare le varie parti del programma,



mentre le **istruzioni esecutive** comprendono:

- le operazioni di tipo aritmetico, il calcolo di espressioni e le assegnazioni di valori alle variabili
- il confronto tra grandezze e il controllo di verità delle proposizioni
- le istruzioni per l'input dei dati dalla tastiera o dagli archivi di dati su disco
- le istruzioni per l'output sul video, sulla carta della stampante o sul disco
- le frasi per rappresentare controlli all'interno dell'algoritmo (se...allora...altrimenti...)
- le frasi per rappresentare gruppi di istruzioni che devono essere ripetute (ripeti mentre...).

### 3 La dichiarazione delle variabili e delle costanti

Le **variabili** utilizzate nel programma devono essere dichiarate, all'inizio del programma o comunque prima del loro impiego, indicando il tipo e il nome assegnato alla variabile (con il punto e virgola finale):

*tipo* nome;

Si può anche utilizzare un'unica dichiarazione per diverse variabili dello stesso tipo, separando con la virgola i loro nomi:

*tipo* nome1, nome2, nome3;

Una **dichiarazione** introduce un **nome** nel programma specificandone il tipo. Il **tipo** definisce l'uso appropriato del nome e ne sottintende la quantità di spazio occupato in memoria.

I principali **tipi standard** delle variabili in C++ sono:

| Tipo             | Descrizione                                                              | Numero di bit utilizzati |
|------------------|--------------------------------------------------------------------------|--------------------------|
| <b>short int</b> | numeri interi compresi tra -32768 e +32767                               | 16                       |
| <b>int</b>       | numeri interi compresi tra -2.147.483.648 e 2.147.483.647                | 32                       |
| <b>long int</b>  | numeri interi compresi tra -2.147.483.648 e 2.147.483.647                | 32                       |
| <b>float</b>     | numeri in virgola mobile ( <i>floating point</i> ) in singola precisione | 32                       |
| <b>double</b>    | numeri in virgola mobile ( <i>floating point</i> ) in doppia precisione  | 64                       |
| <b>char</b>      | un carattere                                                             | 8                        |
| <b>bool</b>      | può assumere solo i valori <i>true</i> (vero) o <i>false</i> (falso)     | 8                        |

Esempi di dichiarazione di variabili:

```
int eta;
int anni;
float statura;
double AreaCerchio;
double AreaTriangolo;
char risposta;
bool trovato;
```

L'identificatore di una variabile è una sequenza qualsiasi di caratteri alfabetici e cifre, che inizia comunque con una lettera; si può usare anche il segno `_` per definire nomi composti, per esempio `area_cerchio`.



È possibile anche, in fase di dichiarazione, assegnare alla variabile un valore iniziale, usando la seguente sintassi:

*tipo nome = valore;*

Per esempio:

```
int contatore = 0;
char risposta = 's';
```

Questa operazione si chiama **inizializzazione** di una variabile. Si noti che i valori di tipo carattere sono racchiusi tra apici. Si può scrivere anche:

```
int i(10);
```

per indicare l'assegnazione del valore 10 alla variabile *i*. Questa inizializzazione è equivalente a:

```
int i = 10;
```

Le **costanti** utilizzate nel programma vengono dichiarate attraverso la parola chiave **const**, secondo la seguente sintassi:

*const tipo nome = valore;*

con l'identificatore della costante separato con il segno uguale dal valore assegnato. Anche per le costanti occorre specificare il tipo di dato, come per le variabili.

Per esempio:

```
const double PIGRECO = 3.14;
const char OK = 's';
const int SCONTO = 20;
```

La costante formata da un singolo carattere è racchiusa tra apici ('s'), mentre per i numeri la separazione tra cifre intere e decimali è indicata dal carattere . (punto decimale). Nella consuetudine della programmazione in C++, i nomi delle costanti sono scritte in maiuscolo.

I valori costanti espressi in esadecimale sono preceduti dai caratteri **0x** (zero x) e quelli espressi in ottale dal carattere **0** (zero); i valori costanti di tipo *long* sono rappresentati scrivendo la lettera **L** dopo il numero:

```
const int VAL1 = 0xAF;
const int VAL2 = 077;
const long MISURA = 25L;
```

Si può osservare, nella dichiarazione delle costanti, un'analogia con l'inizializzazione delle variabili vista in precedenza. In realtà l'uso delle costanti garantisce che il valore assegnato all'inizio venga mantenuto inalterato per tutta l'esecuzione del programma, evitando modifiche indesiderate dei valori.

I dati di tipo alfanumerico formati da più caratteri si chiamano **stringhe** e possono essere considerati come una sequenza di dati di tipo *char*. Per facilitare la rappresentazione di questi dati, che sono di uso comune nella programmazione, il linguaggio C++ mette a disposizione il tipo **string**.

Per utilizzare questo tipo occorre includere nel programma la libreria **string** con la dichiarazione:

```
#include <string>
```

Con questa specificazione si possono poi scrivere dichiarazioni di variabili come le seguenti:

```
string nome;
string siglaprov = "TO";
```

I valori alfanumerici formati da più caratteri devono essere racchiusi tra virgolette.



## I TIPI DI DATI DEL LINGUAGGIO C++

La seguente tabella riassume tutti i tipi predefiniti di dati del linguaggio C++, per i dati di tipo non numerico e per i tipi numerici, interi e non interi.

La parola **unsigned** indica il tipo senza segno, cioè il tipo che non può assumere valori negativi.

|                  | Tipo                      | Intervallo                                                                        | Bit utilizzati |
|------------------|---------------------------|-----------------------------------------------------------------------------------|----------------|
| <b>booleani</b>  | <b>bool</b>               | true/false                                                                        | 8 bit          |
| <b>caratteri</b> | <b>unsigned char</b>      | Da 0 a 255                                                                        | 8 bit          |
|                  | <b>char</b>               | Da -128 a 127                                                                     | 8 bit          |
|                  | <b>wchar_t</b>            | carattere esteso                                                                  | 16 bit         |
| <b>interi</b>    | <b>unsigned short int</b> | Da 0 a 65.535                                                                     | 16 bit         |
|                  | <b>short int</b>          | Da -32.768 a 32.767                                                               | 16 bit         |
|                  | <b>int</b>                | Da -2.147.483.648 a 2.147.483.647                                                 | 32 bit         |
|                  | <b>unsigned long int</b>  | Da 0 a 4,294,967,295                                                              | 32 bit         |
|                  | <b>long int</b>           | Da -2.147.483.648 a 2.147.483.647                                                 | 32 bit         |
| <b>reali</b>     | <b>float</b>              | Da $1,17549435 \cdot 10^{-38}$ a $3,40282347 \cdot 10^{+38}$                      | 32 bit         |
|                  | <b>double</b>             | Da $2,2250738585072014 \cdot 10^{-308}$<br>a $1,7976931348623157 \cdot 10^{+308}$ | 64 bit         |
|                  | <b>long double</b>        | Da $3,4 \cdot 10^{-4932}$ a $1,1 \cdot 10^{+4932}$                                | 80 bit         |

Il tipo **wchar\_t** (*wide character*) si riferisce ai **caratteri estesi**, cioè i caratteri, rappresentati con due byte, della codifica internazionale **Unicode**.

Il seguente programma in linguaggio C++ consente di visualizzare la dimensione dei principali tipi di dati.

```
// Tipi.cpp: dimensione dei tipi
#include <iostream>
using namespace std;

int main()
{
 cout << "Dimensione di int: " << sizeof(int) << " byte \n";
 cout << "Dimensione di short int: " << sizeof(short) << " byte \n";
 cout << "Dimensione di long int: " << sizeof(long) << " bytes \n";
}
```

```
cout << "Dimensione di float: " << sizeof(float) << " byte \n";
cout << "Dimensione di double: " << sizeof(double) << " byte \n";
cout << "Dimensione di char: " << sizeof(char) << " byte \n";
cout << "Dimensione di bool: " << sizeof(bool) << " byte \n";

return 0;
}
```

I risultati riportati dal programma variano a seconda del compilatore utilizzato e dell'elaboratore con il quale il programma viene eseguito. L'operatore **sizeof()** permette di conoscere il numero di byte di memoria occupati da una variabile o da un tipo di variabile. Un singolo carattere, per esempio, occupa, di norma, un byte di memoria. L'istruzione **cout <<** visualizza sullo schermo i dati e i messaggi specificati come argomenti dopo l'operatore **<<**. La sequenza **\n** indica il ritorno a capo alla fine di una riga di output.

## 4 Le frasi di commento

All'interno del programma possono essere inserite righe, o paragrafi di più righe, contenenti **commenti** o annotazioni del programmatore, con le quali è possibile documentare il significato delle variabili o delle costanti utilizzate, oppure la funzionalità svolta dall'intero programma o da una sua parte.

Le frasi di commento iniziano con i delimitatori **//**.

I commenti possono essere posti all'inizio di una nuova riga oppure accanto ad un'istruzione: tutto quello che viene scritto a partire dai caratteri **//** fino alla fine della riga viene considerato commento.

```
int eta; // età di una persona
```

I commenti possono essere collocati in un punto qualsiasi del programma.

Si noti l'assenza del carattere accentato nell'identificativo della variabile *eta*, in quanto sono consentiti i soli caratteri del codice ASCII standard (da 0 a 127).

L'uso frequente delle frasi di commento deve diventare un'abitudine del programmatore fin dai primi semplici esercizi: in questo modo possono essere costruiti programmi che si autodocumentano, che possono essere compresi e letti facilmente anche a distanza di tempo e anche da persone diverse dal programmatore che ha scritto il programma.

## 5 L'assegnazione dei valori alle variabili

I valori vengono assegnati alle variabili all'interno del programma principalmente con due modalità:

- acquisendo i valori dalla tastiera, come visto nel primo esempio di questo capitolo e meglio spiegato più avanti
- con un'istruzione di assegnazione.

L'istruzione di **assegnazione** ha una sintassi del tipo:

variabile = valore;

Il valore viene assegnato alla variabile scritta a sinistra del simbolo **=**.

Per *valore* si intende un numero o il risultato di un'espressione.

Per esempio, il calcolo dell'area di un cerchio viene indicato con l'istruzione:

```
area = raggio * raggio * 3.14;
```

Nelle espressioni di tipo numerico si usano gli operatori aritmetici già visti **+**, **-**, **\***, **/**, per le quattro operazioni elementari e **%** per il resto della divisione tra interi.

Per esempio:

```
minuti = tempo_in_sec % 60;
velocita = spazio / tempo;
```

Per una variabile *nome* di tipo *string*:

```
nome = "Rossi";
```

Nei linguaggi di programmazione sono, in genere, consentite anche istruzioni del tipo:

```
pag_stampate = pag_stampate + 1;
```

con la variabile *pag\_stampate* che compare sia a sinistra sia a destra del segno **=**, intendendo che il nuovo valore di *pag\_stampate* si ottiene aggiungendo 1 al precedente valore.

Nel linguaggio C++ si può anche scrivere:

```
pag_stampate++;
```

per indicare, con l'**operatore ++**, un incremento unitario del valore di una variabile.

Esiste anche l'**operatore --** per il decremento unitario:

```
giorni_a_scadenza--;
```

che è equivalente all'istruzione di assegnazione:

```
giorni_a_scadenza = giorni_a_scadenza - 1;
```

Per incrementi e decrementi non unitari si possono usare gli operatori **+=** e **-=**.

Per esempio, l'istruzione:

```
tot_vendite += scontrino;
```

significa che il valore di *tot\_vendite* + *scontrino* viene memorizzato nella variabile *tot\_vendite* ed è equivalente all'istruzione di assegnazione:

```
tot_vendite = tot_vendite + scontrino;
```

L'istruzione

```
prezzo -= sconto;
```

significa che il valore di *prezzo* - *sconto* viene memorizzato nella variabile *prezzo* ed è equivalente all'istruzione di assegnazione:

```
prezzo = prezzo - sconto;
```

Si possono usare in modo analogo anche gli operatori `*=`, `/=` e `%=` per la moltiplicazione, per la divisione e per il resto della divisione tra interi.

La tabella seguente riassume le equivalenze tra i vari operatori di assegnamento:

| Operatore       | Utilizzo                         | Esempio             | Istruzione equivalente  |
|-----------------|----------------------------------|---------------------|-------------------------|
| <code>++</code> | Incremento unitario              | <code>x++;</code>   | <code>x = x + 1;</code> |
| <code>--</code> | Decremento unitario              | <code>x--;</code>   | <code>x = x - 1;</code> |
| <code>+=</code> | Incremento                       | <code>x+= y;</code> | <code>x = x + y;</code> |
| <code>-=</code> | Decremento                       | <code>x-= y;</code> | <code>x = x - y;</code> |
| <code>*=</code> | Moltiplicazione                  | <code>x*= y;</code> | <code>x = x * y;</code> |
| <code>/=</code> | Divisione                        | <code>x/= y;</code> | <code>x = x / y;</code> |
| <code>%=</code> | Resto della divisione tra interi | <code>x%=y;</code>  | <code>x = x % y;</code> |

Nelle istruzioni di assegnazione e nelle espressioni di calcolo possono poi comparire le **funzioni**, ossia strumenti predefiniti (*built-in*) del linguaggio che, ricevendo un valore, restituiscono un valore calcolato.

Tra le più usate, possono essere citate le seguenti:

- **`pow(x, y)`**, per indicare il calcolo della potenza di `x` con esponente `y` (`x` e `y` devono essere di tipo *double*, così come il risultato ottenuto dal calcolo)
- **`sqrt(x)`**, per indicare il calcolo della radice quadrata del numero `x`
- **`ceil(x)`**, per indicare il valore di un numero arrotondato all'intero superiore
- **`floor(x)`**, per ottenere un numero senza parte decimale.

Le ultime due necessitano di un esempio chiarificatore:

`ceil(3.7)` produce 4  
`floor(3.7)` produce 3.

Per calcolare l'ipotenusa di un triangolo rettangolo, si può scrivere un'istruzione del tipo:

```
ipotenusa = sqrt(pow(cateto1,2.0) + pow(cateto2,2.0));
```

I numeri 2.0 come esponenti dei cateti sono costanti di tipo *double* come previsto dalla funzione *pow*.

Tutte le funzioni illustrate in precedenza appartengono alla libreria delle funzioni matematiche predefinite del linguaggio. Pertanto i programmi che usano queste funzioni devono contenere all'inizio la direttiva:

```
#include <cmath>
```

per l'inclusione della libreria **`cmath`** con i prototipi delle funzioni matematiche.

L'uso delle parentesi (sempre tonde) determina l'ordine di esecuzione delle operazioni indicate in un'espressione aritmetica, come nell'algebra elementare:

```
a = (b + c) * (d - e);
```

Si noti che nel linguaggio di programmazione non è possibile indicare le frazioni con la consueta notazione:

$$x = \frac{(a - b)^2}{c * d}$$

Questa frazione deve essere indicata in un'istruzione di assegnazione nel seguente modo:

```
x = pow((a-b),2.0)/(c*d);
```

Nelle istruzioni di assegnazione, occorre fare attenzione all'uso dei tipi per le variabili, considerando i problemi di incompatibilità tra tipi che si possono verificare.

Per esempio, se si hanno le variabili:

```
int classe;
char sezione;
string istituto;
```

le istruzioni di assegnazione *classe = sezione*, *sezione = classe* oppure *istituto = classe* non sono considerate errate nel linguaggio C++, ma possono produrre valori indesiderati per le variabili.

Invece l'istruzione *classe = istituto* è considerata errata perché non è possibile la conversione da stringa a numero.



## OPERATORE ++

L'operatore unario ++ incrementa il valore della variabile di una unità ed aggiorna il valore della variabile stessa. Per esempio l'istruzione:

```
i++;
```

equivale all'istruzione:

```
i = i + 1;
```

Quando questo operatore viene utilizzato all'interno di un'espressione più complessa, esso assume un significato diverso a seconda che preceda o segua la variabile cui si riferisce: se segue la variabile (*suffisso*), l'incremento viene effettuato dopo la valutazione dell'espressione; se invece la precede (*prefisso*), prima viene aggiornato il valore della variabile e poi viene valutata l'espressione.

Per esempio:

```
i = 5;
x = 3;
y = x + i++;
```

l'ultima istruzione equivale alle due istruzioni:

```
y = x + i;
i = i + 1;
```

quindi ad *y* viene assegnato il valore 8 e *i* assume il valore 6.

Se invece si scrive:

```
i = 5;
x = 3;
y = x + ++i;
```

l'ultima istruzione equivale alle due istruzioni:

```
i = i + 1;
y = x + i;
```

quindi ad *y* viene assegnato il valore 9 e *i* assume il valore 6.

Il risultato ottenuto nei due casi è diverso. Occorre quindi prestare attenzione alla posizione dell'operatore, perché cambia il significato (la *semantica*) dell'istruzione.

Si deve anche notare che l'istruzione:

```
y = x + (++i);
```

è equivalente alla seguente:

```
y = x + (i++);
```

perché le parentesi tonde hanno priorità maggiore rispetto a qualsiasi altro simbolo di operazione.

Gli operatori di assegnamento hanno invece priorità inferiore a qualsiasi altro.

In ogni caso di incertezza, può essere quindi conveniente aggiungere le parentesi per forzare l'ordine voluto.

L'operatore di decremento unitario `--` opera in modo del tutto analogo.

## 6 Il casting per la conversione di tipo

Per **conversione di tipo** si intende un'operazione volta a trasformare un valore di un certo tipo in un valore di altro tipo.

Questa operazione è molto comune in tutti i linguaggi, anche se spesso il programmatore non se ne rende conto. Per esempio, quando in un'espressione si utilizzano dati dello stesso tipo, il tipo del risultato di calcolo è ben definito ed è lo stesso degli operandi. Il problema nasce quando in un'operazione gli operandi sono di diverso tipo.

Per esempio, nel calcolo dell'area del seguente frammento di codice, si tratta di decidere di che tipo sia il valore della variabile *area*:

```
int base;
float altezza;

area = base * altezza;
```

Non sempre una conversione di tipo preserva il valore: per esempio nella conversione da *float* a *int* in generale si riscontra una perdita di precisione. In effetti in una conversione da *float* a *int* il compilatore non fa altro che scartare la parte frazionaria; se poi il valore non è rappresentabile, il risultato è indefinito.

È opportuno sottolineare che le operazioni di conversione riguardano il valore della variabile in quel particolare contesto e non il tipo della variabile stessa.

Le conversioni eseguite direttamente dal compilatore si chiamano **conversioni implicite**. Per esempio, si consideri il programma seguente nel quale un valore non intero viene assegnato ad una variabile di tipo *int*.

```
#include <iostream>
using namespace std;
```

```
int main()
{
 int a = 5;
 float b = 3.56;
 a = b;
 cout << a;

 return 0;
}
```

L'output del programma è 3.

Nel seguente programma, invece, un valore intero è assegnato ad una variabile di tipo *float*.

```
#include <iostream>
using namespace std;

int main()
{
 int a = 5;
 float b = 3.56;
 b = a;
 cout << b;

 return 0;
}
```

In questo caso l'output del programma è 5.

Il linguaggio C++ consente al programmatore stesso di richiedere le conversioni di tipo: in questo caso si parla di **conversioni esplicite** (cioè decise dal programmatore) o *cast*.

Il **casting** è l'azione che consente di trasformare la rappresentazione di un dato dal suo tipo originale a un altro.

Seguendo la sintassi tradizionale, l'operatore **cast** è rappresentato da una coppia di parentesi tonde che racchiudono il tipo di dato: è un operatore a tutti gli effetti e viene rappresentato nel seguente modo:

(tipo) espressione;

Per esempio:

```
b = (int) a;
```

Nelle versioni più recenti del linguaggio C++ l'operazione di casting è rappresentata con la parola chiave **static\_cast** secondo la seguente sintassi:

static\_cast <tipo> (espressione)

il nuovo tipo è indicato tra due parentesi angolari < > e la variabile, o l'espressione, da convertire è racchiusa tra parentesi tonde.

Per esempio:

```
b = static_cast <int> (a)
```



I due programmi seguenti evidenziano l'importanza del casting nelle operazioni con numeri di tipo diverso.

```
#include <iostream>
using namespace std;

int main()
{
 int a = 5;
 int b = 2;
 float c = 3.56;
 float p;
 p = a/b + c;
 cout << p;

 return 0;
}
```

```
#include <iostream>
using namespace std;

int main()
{
 int a = 5;
 int b = 2;
 float c = 3.56;
 float p;
 p = static_cast<float>(a)/b + c;
 cout << p;

 return 0;
}
```

Con il programma di sinistra si ottiene in output 5.56, perché la divisione intera  $a/b$  produce il valore 2 che viene poi sommato a 3.56.

Con il programma di destra si ottiene invece in output il valore 6.06, perché il casting, applicato alla divisione  $a/b$ , produce correttamente il valore 2.5 che viene poi sommato a 3.56.

Lo **static\_cast** si chiama così perché la conversione viene effettuata nella fase di traduzione del programma in linguaggio macchina (*compilazione*), ma non viene effettuato alcun controllo sulle compatibilità tra tipi durante l'esecuzione del programma.

Quindi è una modalità di *cast* che deve essere usata quando si è sicuri della validità della conversione, cioè che la conversione possa essere fatta senza perdita di informazione.

Negli altri casi si può usare il *cast dinamico*, che effettua un controllo sulla conversione durante l'esecuzione del programma e che viene rappresentato con la parola chiave **dynamic\_cast**.

Sono disponibili poi altri due tipi di cast:

- **reinterpret\_cast**: converte la variabile nel nuovo tipo interpretando bit per bit secondo la specifica implementazione del linguaggio;
- **const\_cast**: converte una costante in variabile (non costante) mantenendo lo stesso tipo.

L'uso delle parole chiave dei diversi tipi di *cast* è identico a quanto visto per *static\_cast*.



## AUTOVERIFICA

Domande da 1 a 12 pag. 295-296

## 7 Gli operatori di relazione e logici

Nel programma C++, gli **operatori di confronto** si indicano con i simboli seguenti:

|    |                   |
|----|-------------------|
| == | uguale            |
| >  | maggiore          |
| <  | minore            |
| <= | minore o uguale   |
| >= | maggiore o uguale |
| != | diverso           |

Con questi simboli si possono rappresentare espressioni booleane del tipo:

```
a >= c
b < 5
f != g
```

I **connettivi logici** che possono essere utilizzati in C++ sono:

**&&** indica l'operazione di congiunzione (*And*)  
**||** indica l'operazione di disgiunzione (*Or*)  
**!** indica la negazione (*Not*).

Utilizzando poi le operazioni di congiunzione, disgiunzione e negazione, si possono costruire espressioni booleane del tipo:

```
(a > b) && (c == 3)
(a == 5) || (a == 7)
!(a > 4)
```

Il valore di queste espressioni può essere *true* (vero) o *false* (falso) e può essere assegnato a variabili di tipo *bool*.

Il linguaggio C++, oltre agli operatori logici *And* (&&), *Or* (||) e *Not* (!), possiede anche gli **operatori logici** che operano sui dati **bit a bit** anziché globalmente. Essi sono:

|                 |                                                                                                          |
|-----------------|----------------------------------------------------------------------------------------------------------|
| <b>&amp;</b>    | per la congiunzione logica bit a bit ( <i>And</i> )                                                      |
| <b> </b>        | per la disgiunzione logica bit a bit ( <i>Or</i> )                                                       |
| <b>^</b>        | per l'Or esclusivo bit a bit ( <i>Xor</i> )                                                              |
| <b>~</b>        | per il complemento a uno dell'operatore, nega cioè bit a bit il dato                                     |
| <b>&gt;&gt;</b> | per lo scorrimento a destra ( <i>shift</i> ) del primo operando di tanti bit quanti indicati dal secondo |
| <b>&lt;&lt;</b> | per lo scorrimento a sinistra ( <i>shift</i> ).                                                          |

Per esempio:

| Espressione | Valore in bit prima della valutazione | Valore in bit dopo la valutazione | Valore decimale dopo la valutazione |
|-------------|---------------------------------------|-----------------------------------|-------------------------------------|
| X = 11      |                                       | 00001011                          | 11                                  |
| X = X & 3   | 00001011                              | 00000011                          | 3                                   |
| X = X   12  | 00000011                              | 00001111                          | 15                                  |
| X = X ^ 10  | 00001111                              | 00000101                          | 5                                   |
| X = ~X      | 00000101                              | 11111010                          | 250                                 |
| X = X >> 3  | 11111010                              | 00011111                          | 31                                  |
| X = X << 2  | 00011111                              | 01111100                          | 124                                 |

Si noti il diverso impiego degli operatori >> e << rispetto a quanto visto nelle operazioni di immissione ed emissione.

## 8 Le istruzioni di ingresso e uscita

La lettura di dati (*input*) è rappresentata dall'**operatore >>**.

L'orientamento dell'operatore indica la direzione in cui si muovono i flussi di dati, da sinistra verso destra.

Allo stesso modo l'**operatore <<** è impiegato nella scrittura dei dati (*output*) ed è orientato verso sinistra per indicare il passaggio di informazioni da destra verso sinistra.

In generale, a sinistra degli operatori >> e << troviamo lo **stream** (flusso), cioè l'identificativo della periferica, o *unità di I/O*, interno al programma.

A destra degli operatori >> e << occorre indicare il nome della variabile utilizzata nell'operazione di I/O.

La sintassi generale di un'istruzione di input è la seguente:

```
unità_input >> variabile;
```

La sintassi generale di un'istruzione di output è invece la seguente:

```
unità_output << variabile;
unità_output << costante;
```

L'*unità di input standard* è la tastiera.

Nel linguaggio C++ questa periferica viene indicata con la parola **cin**, abbreviazione di *console input*. Per esempio:

```
cin >> raggio;
```

fa entrare da tastiera un numero e lo assegna alla variabile di nome *raggio*.

L'istruzione `cin >>` indica quindi un'operazione sullo **standard input**, cioè l'unità standard del computer per acquisire dati dall'esterno (tastiera).

L'operazione di lettura con *cin*, permette di leggere con un'unica operazione tutti i caratteri digitati fino alla pressione del tasto *Invio*. Se si vogliono leggere più dati, si devono indicare tante variabili quanti sono i dati da inserire separandole con i caratteri >>:

```
cin >> base >> altezza;
```

L'utente del programma può inserire i dati richiesti sulla stessa riga separandoli con la barra spaziatrice o con il tasto di tabulazione oppure con il ritorno a capo.

L'*unità di output standard* è il video.

Nel linguaggio C++ questa periferica è chiamata **cout**, abbreviazione di *console output*. Per esempio:

```
cout << area;
```

scrive sul video il valore contenuto nella variabile *area*.

L'istruzione `cout <<` indica quindi un'operazione sullo **standard output**, cioè l'unità standard del computer per comunicare i risultati (video).

Le definizioni della tastiera (*cin*) e del video (*cout*) sono memorizzate nel file **iostream** delle librerie di I/O del linguaggio. Questo significa che per poter utilizzare gli operatori di I/O sulle periferiche standard è necessario includere tale file all'inizio del programma:

```
#include <iostream>
```

Oltre a *cin* e *cout* sono disponibili due altre periferiche di output: **cerr** e **clog**. La prima di queste rappresenta l'unità standard per l'invio dei messaggi di errore (*standard error*), mentre la seconda prevede la gestione dei file di tracciamento (*file di log*), cioè la registrazione delle attività svolte dagli utenti di un sistema di elaborazione.

È compito del sistema operativo distinguere i diversi tipi di output. Quando questa distinzione non è stata definita o non è prevista, l'output è inviato sul video.

Con un'unità di output è possibile stampare un valore costante (per esempio, 123 o 3.14) oppure un messaggio o una stringa di caratteri.

Eventuali messaggi, che servono a illustrare i risultati, devono essere inseriti tra le virgolette, nella posizione dove devono comparire, separandoli dalle variabili con i caratteri <<.

Per esempio, se la variabile *area* possiede il valore 10, l'istruzione:

```
cout << "Risultato del calcolo = " << area;
```

produce in output la seguente riga:

```
Risultato del calcolo = 10
```

Le costanti rappresentate da un singolo carattere sono delimitate da apici, mentre le stringhe di caratteri sono racchiuse tra una coppia di virgolette.

Per esempio, per separare due valori sulla stessa riga si può inserire il carattere spazio (*space*) indicato nell'istruzione tra due apici:

```
cout << "Dimensioni della figura: " << b << ' ' << a;
```

L'istruzione `cout <<` visualizza il messaggio o il valore lasciando il cursore sulla stessa riga del video; l'output di una successiva istruzione `cout <<` verrà visualizzato subito dopo.

Per portare il cursore all'inizio della riga successiva del video (cioè per andare a capo nella visualizzazione), occorre aggiungere alla fine dell'istruzione il carattere di ritorno a capo rappresentato nel linguaggio C++ con la parola **endl** (*end of line*); l'output successivo comparirà all'inizio di una nuova riga. Per esempio:

```
cout << "Risultato del calcolo = " << area << endl;
```

In alternativa si può usare la sequenza di caratteri **\n**, detta **sequenza di escape** per il ritorno a capo, racchiusa tra due apici:

```
cout << "Risultato del calcolo = " << area << '\n';
```

Le due modalità, pur con una leggera differenza, sono del tutto equivalenti.

Un altro esempio di *sequenza di escape* è rappresentato dal carattere di tabulazione **\t** che può servire per disporre in modo ordinato i dati all'interno di una riga di output. Per esempio:

```
cout << "Dimensioni della figura: " << b << '\t' << a << '\n';
```

Per facilitare l'inserimento dei dati da parte dell'utente del programma, è buona norma far precedere all'istruzione *cin >>* un'istruzione `cout <<`, che comunichi un messaggio per ricordare quali dati occorre inserire, di quale tipo (numeri o stringhe) ed eventualmente in quale formato. Per esempio:

```
cout << "Introduci il lato: ";
cin >> lato;
```

Il seguente esempio mostra l'uso delle istruzioni di input e di output per richiedere dati all'utente:

```
#include <string>

string nome,cognome;

cout << "Inserisci il nome: ";
cin >> nome;
cout << "Inserisci il cognome: ";
cin >> cognome;
```

Il linguaggio C++ non possiede funzioni o istruzioni standard specifiche per l'invio dell'output di un programma alla **stampante** per ottenere i risultati su carta anziché sul video: il problema sarà risolto nel Capitolo 8 con l'introduzione delle istruzioni per la gestione, in generale, degli *stream*.

Una soluzione intermedia consiste nell'eseguire il programma C++ dalla linea comandi del sistema operativo effettuando la **ridirezione dell'output** su un file temporaneo o sulla stampante collegata al computer.

Per esempio, il seguente comando manda in esecuzione il programma *calcolo.exe*, che si trova nella directory *esercizi*, ridirigendo l'output sul file *temp.txt*:

```
C:\esercizi>calcolo > temp.txt
```

Tutto l'output diretto al flusso *cout* viene scritto sul file *temp.txt*, anziché sul video. Il file *temp.txt*, essendo un file di testo, può poi essere letto o stampato con un qualsiasi programma di editing, per esempio *Blocco note* di Windows.

Invece, con la linea di comandi seguente:

```
C:\esercizi>calcolo > lpt1:
```

l'output diretto al flusso *cout* viene inviato alla stampante parallela, che ha il nome riservato **lpt1:**.

Viene presentato di seguito un esempio di programma C++ completo, che si riferisce ad un semplice problema, per dimostrare le modalità di utilizzo delle basi del linguaggio C++ viste nei paragrafi precedenti.

## PROGETTO 1

**Un gruppo di ragazzi vince un certo numero di gettoni in una sala giochi e decide di dividersi i gettoni in parti uguali, lasciando il resto della divisione come riserva per l'intero gruppo.**

Si tratta di costruire un programma che richieda all'utente il numero dei gettoni e il numero dei ragazzi, che esegua la divisione intera tra gettoni e ragazzi e che scriva la quota di gettoni spettanti per ragazzo e il numero dei gettoni che avanzano.

### Programma C++

```
// Salagiochi.cpp: divisione della vincita
#include <iostream>
using namespace std;
```

```
int main()
{
 // input
 int gettoni, ragazzi;
 // output
 int quota, avanzo;

 cout << "Numero di gettoni e di ragazzi: ";
 cin >> gettoni >> ragazzi;
 quota = gettoni / ragazzi;
 avanzo = gettoni % ragazzi;
 cout << "Ad ogni ragazzo spettano " << quota
 << " gettoni " << endl;
 cout << "e ne avanzano " << avanzo << endl;

 return 0;
}
```

In questo esempio si può osservare l'uso dei commenti con i caratteri `//`, per documentare il titolo del programma e il significato delle variabili utilizzate per le operazioni di input e di output. L'istruzione `cin >>` è preceduta dall'istruzione `cout <<` per facilitare l'inserimento dei dati da parte dell'utente del programma. Per calcolare il quoziente e il resto sono stati utilizzati gli operatori `/` e `%`.

#### AUTOVERIFICA

Domande da 13 a 14 pag. 297  
Problemi da 1 a 6 pag. 302



#### MATERIALI ONLINE

- 2. Manipolatori per l'Input/Output formattato
- 3. Utilizzo degli output formattati

## 9 Le fasi della programmazione

La programmazione comprende un insieme articolato di attività che va dall'organizzazione del programma alla sua prova di esecuzione e non è composta soltanto dalla stesura del programma in linguaggio C++.

I passi da compiere per ottenere l'esecuzione di un programma scritto in un linguaggio come il C++ sono fondamentalmente tre: **codifica**, **compilazione**, **linking** (collegamento).

L'attività con la quale il procedimento risolutivo viene tradotto in un programma, usando il linguaggio di programmazione, si chiama **codifica**, in quanto le istruzioni vengono rappresentate in una forma (**codice**) che il computer è in grado di comprendere.

Il testo del programma, così progettato, deve essere comunicato al computer e immesso nella memoria centrale: per far questo si usa un programma chiamato **editor**, cioè uno strumento software che consente di scrivere e modificare il testo del programma utilizzando tastiera e video. Si può usare, per esempio, il programma **Blocco note** degli *Accessori* di Windows.

In alternativa, si possono usare ambiente integrati di sviluppo del software (**IDE**, *Integrated Development Environment*) formati da diversi strumenti per le diverse fasi della programmazione: editing del codice sorgente, compilazione, prove di esecuzione.

Spesso questi ambienti utilizzano un'**interfaccia grafica** con icone, menu e finestre per rendere più facile il lavoro del programmatore.

Il risultato della fase di codifica si chiama **programma sorgente**, cioè il testo del programma, scritto usando le regole del linguaggio, che viene collocato nella memoria centrale. Esso deve essere anche salvato su disco per utilizzi successivi, assegnando un nome significativo e un'estensione che ne ricordi il contenuto.

I file che contengono un programma C++ hanno l'estensione **.cpp**.

La seconda fase del lavoro di programmazione consiste nel tradurre il programma sorgente in un **programma oggetto**, cioè un programma scritto nel linguaggio interno al computer: per far questo si utilizza un altro strumento software, il **compilatore**.

Se il testo sorgente è stato scritto rispettando il lessico e la sintassi del linguaggio C++, sul disco viene creato il file **eseguibile**, cioè un file con lo stesso nome del sorgente e con l'estensione **.exe**. In caso contrario occorre ritornare all'*editor* per le correzioni degli errori riscontrati dal compilatore.

Gli errori segnalati dal compilatore sono essenzialmente di due tipi:

- **warning**, cioè avvertimenti, errori che non impediscono la creazione del programma oggetto, ma che potrebbero creare problemi in fase di esecuzione;
- **error**, errori che bloccano il lavoro di traduzione del compilatore e non consentono di creare il programma oggetto.

Per comprendere come funziona il compilatore si provi a modificare il testo del programma C++, visto all'inizio di questo capitolo.

```
// Somma.cpp: somma di due numeri
#include <iostream>
using namespace std;

int main()
{
 int a, b, s;
 cin >> a;
 cin >> b;
 s = a + b;
 cout << s ;

 return 0;
}
```

Si introducano artificialmente degli errori: per esempio, si tolga il carattere ; (punto e virgola) alla fine della riga  $s = a + b$  e si cambi la parola *cout* con *caut*.

Quando si lancia la compilazione, compare un primo messaggio di errore in corrispondenza della riga  $s = a + b$ : a seconda del compilatore utilizzato si ottiene il messaggio

Statement missing ;

oppure

expected ';' before caut

cioè il compilatore si aspettava il carattere *punto e virgola* alla fine dell'istruzione di assegnazione.

Questo può essere considerato un **errore sintattico**, in quanto non sono state rispettate le regole per la costruzione corretta delle frasi del linguaggio.

Un secondo errore, riferito all'uso della parola *caut*, viene segnalato con un messaggio, che, a seconda del compilatore utilizzato, può essere:

```
Undefined symbol 'caut'
```

oppure

```
'caut' was not declared
```

cioè il compilatore non trova un riferimento alla funzione *caut* nelle direttive del programma per le librerie di funzioni del linguaggio.

In sostanza il compilatore ha incontrato una parola che non appartiene al linguaggio, e che non è neppure il nome di un identificatore definito dal programmatore. Possiamo dire che questo è un **errore lessicale**, in quanto è stata utilizzata una parola che non esiste nel dizionario del linguaggio C++.

Dopo aver corretto gli errori con l'*editor*, si ricompila e si ottiene il programma eseguibile senza segnalazioni di errori.

In effetti, se il *programma sorgente* è corretto, la creazione del *programma eseguibile* è preceduta da un'ulteriore operazione di **linking**, nella quale un apposito programma, detto **linker**, effettua il collegamento tra il *programma oggetto*, cioè la traduzione del testo sorgente in linguaggio macchina, e le librerie del linguaggio necessarie per l'esecuzione del programma.

Il programma può essere poi avviato scrivendo il nome del programma eseguibile. Facendo riferimento all'esempio visto in precedenza, il programma si ferma sulla prima istruzione *cin >>*, perché aspetta che l'utente da tastiera introduca due numeri che verranno assegnati nell'ordine alle variabili *a* e *b*; dopo aver eseguito la seconda istruzione con la somma *a + b*, il risultato viene assegnato alla variabile *s*, e, per finire, con l'esecuzione dell'istruzione *cout <<*, sul video compare il valore calcolato di *s*.

Oltre agli errori di tipo lessicale e sintattico, che vengono riscontrati nella fase di compilazione, nell'attività di programmazione si possono incontrare errori che vengono segnalati durante l'esecuzione del programma (errori di **run-time**), e che bloccano l'avanzamento del processo di elaborazione.

Scriviamo per esempio il seguente programma per il calcolo del quoziente:

```
// Quoziente.cpp: divisione di due numeri
#include <iostream>
using namespace std;

int main()
{
 int a, b, q;
 cin >> a >> b;
 q = a / b;
 cout << q;

 return 0;
}
```

La riga con *cin >>* serve per richiedere all'utente l'inserimento da tastiera di due numeri, che vengono assegnati alle variabili *a* e *b*. La successiva istruzione *cout <<* produce sul video il risultato della divisione tra i due numeri.



Si provi ad eseguire il programma una prima volta inserendo da tastiera due numeri, entrambi diversi da zero, per esempio 8 e 4: il programma viene eseguito correttamente, producendo il quoziente dei due numeri. Introducendo invece, con una seconda esecuzione, il valore zero come secondo operando (divisore), sul video compare il messaggio di errore “*Division by zero*” (divisione per zero). Il programma, pur essendo scritto secondo le regole del linguaggio C++, si blocca perché ha incontrato una situazione di errore durante l’esecuzione.

L’esecuzione di un programma può anche produrre **errori logici**, che sono gli errori che riguardano la correttezza dell’algoritmo, cioè gli errori che producono risultati diversi da quelli attesi, a causa di un’errata costruzione del procedimento risolutivo.

Questi errori non sono rilevabili dal compilatore, che controlla soltanto la correttezza formale del codice: è compito del programmatore controllare che l’output dell’elaborazione corrisponda a risultati corretti. Questa fase del lavoro si chiama **validazione** del codice.

Il programma di utilità che aiuta il programmatore nell’individuazione degli errori logici si chiama **debugger** (letteralmente, *spulciatore*) e l’attività di controllo del programma si chiama **debugging** (togliere i *bug*). Nel linguaggio corrente dell’informatica, infatti, gli errori di un programma si chiamano **bug**.

Il debugger può consentire di:

- eseguire il programma da correggere un’istruzione per volta, così da verificarne la corretta evoluzione;
- stabilire punti di interruzione (**breakpoint**) durante l’esecuzione per effettuare controlli;
- controllare i valori assunti dalle variabili in certi punti del programma.

## 10 L’importanza della documentazione

La **documentazione** prodotta durante il lavoro di programmazione permette di raggiungere due obiettivi importanti:

- far comprendere l’analisi del procedimento e l’algoritmo risolutivo a una persona diversa da quella che lo ha progettato;
- riuscire a capire il procedimento e le attività da svolgere anche a distanza di tempo dal momento in cui il progetto è stato realizzato.

Questa seconda esigenza può nascere anche dalla necessità di dover modificare, ampliare o migliorare in momenti successivi il lavoro realizzato.

Una buona documentazione si realizza attraverso alcuni accorgimenti, spesso anche semplici:

- usare per le variabili, definite nel programma, nomi significativi, cioè che servono a ricordare l’entità che rappresentano.

Per esempio:

*nome* è più significativo di *n*

*abbonato* è più significativo di *a*

*lato* è più significativo di *l*

e così via.

- nell’analisi del problema, esplicitare con precisione quali sono i tipi previsti per i dati (di input e di output) che verranno utilizzati nel programma: stringhe di caratteri, numeri interi, numeri reali, ecc.;
- scrivere i programmi utilizzando le tre strutture di controllo, sequenza, selezione e iterazione e le strutture derivate (come vedremo nei prossimi paragrafi);

- organizzare i programmi in modo strutturato anche attraverso l'individuazione delle parti principali di cui si compone il procedimento, che corrispondono ai moduli che svolgono una determinata funzionalità all'interno del programma complessivo;
- inserire frequenti frasi di commento, per illustrare i diversi passaggi del programma, oppure il significato di particolari variabili, oppure il titolo dei sottomoduli funzionali individuati;
- validare la correttezza del programma attraverso prove di esecuzione, con dati verosimili rispetto al problema trattato, alternati a dati estremi per la verifica di casi eccezionali;
- fornire istruzioni per l'uso del programma, rivolte all'utente finale del prodotto realizzato (*manuale per l'utente*).

#### AUTOVERIFICA



Domande da 15 a 17 pag. 297  
Problemi da 7 a 12 pag. 302-303



#### MATERIALI ONLINE

### 4. Namespace e file di intestazione

## 11 La programmazione strutturata

Le istruzioni del procedimento risolutivo di un problema (**algoritmo**), sono organizzate secondo schemi classificabili in tre costrutti fondamentali, chiamati **strutture di controllo**.

- Istruzioni organizzate in **sequenza**:

istruzione-1  
istruzione-2  
istruzione-3  
istruzione-4

.....

vale a dire istruzioni che devono essere eseguite una dopo l'altra secondo l'ordine con cui sono state scritte.

- Istruzioni che vengono eseguite in **alternativa** con altre:

SE condizione  
ALLORA  
    istruzioni-a  
ALTRIMENTI  
    istruzioni-b  
FINE SE

l'esecutore (il computer) deve fare una scelta tra un certo gruppo di istruzioni e un altro gruppo a seconda di quello che succede in quel momento durante l'elaborazione.

Si parla in questo caso di **selezione**.

- Istruzioni che devono essere eseguite in **ripetizione**:

ESEGUI  
    istruzioni  
RIPETI MENTRE condizione

Le istruzioni comprese tra ESEGUI e RIPETI devono essere eseguite più volte mentre si verifica una condizione.

Si chiamano *strutture di controllo* perché organizzano le istruzioni in modo ordinato e perché controllano il percorso all'interno dell'algoritmo per arrivare ad ottenere i risultati.

Vediamo di seguito alcuni semplici esempi di programmi in linguaggio C++ nei quali sono utilizzate le tre strutture di base.

## 12 La sequenza

La **sequenza** si rappresenta costruendo un blocco di istruzioni, ognuna terminante con il punto e virgola, delimitato all'inizio e alla fine da una coppia di parentesi graffe aperta e chiusa { ... }. Per esempio, la parte di programma C++ che calcola la differenza tra due numeri viene codificata in questo modo:

```
#include <iostream>
using namespace std;

int main()
{
 int num1, num2, differenza;
 cout << "Due numeri: ";
 cin >> num1 >> num2;
 differenza = num1 - num2;
 cout << "Risultato = " << differenza << endl;

 return 0;
}
```

Le istruzioni comprese tra le due parentesi graffe possono comparire all'interno di un'altra struttura di controllo (selezione o ripetizione), oppure possono essere le istruzioni che compongono la parte esecutiva del programma C++.

### PROGETTO 2

#### Calcolare lo sconto del 20% sul prezzo di un articolo.

Occorre acquisire in input la descrizione dell'articolo e il suo prezzo. Viene calcolato lo sconto e sottratto dal prezzo iniziale. Da ultimo si devono scrivere la descrizione dell'articolo e il nuovo prezzo scontato.

#### Programma C++

```
// CalcoloSconto.cpp: calcolo del prezzo scontato
#include <iostream>
#include <string>
using namespace std;
int main()
{
 // variabili di input-output
 string descrizione;
 float prezzo;
 // variabili di lavoro
 float sconto;
```

```
cout << "Descrizione e prezzo: ";
cin >> descrizione >> prezzo;
sconto = prezzo * 20 / 100;
prezzo = prezzo - sconto;
cout << descrizione << ": " << prezzo << endl;

return 0;
}
```

Si noti che le variabili *prezzo* e *descrizione* sono sia di input sia di output.

I programmi C++ visti nei paragrafi precedenti sono esempi di utilizzo della struttura sequenziale.

## 13 La struttura di alternativa

Per la **selezione** nel linguaggio C++ si usa l'istruzione **if** che ha la sintassi seguente:

```
if (condizione) {
 istruzioni-a;
}
else {
 istruzioni-b;
}
```

Si noti che la condizione è racchiusa tra due parentesi tonde.

Se la condizione è vera, viene eseguita la sequenza *istruzioni-a*, altrimenti viene eseguita la sequenza *istruzioni-b*.

### PROGETTO 3

#### Scrivere in ordine crescente due numeri.

Si tratta di scegliere tra due numeri il minore e di scrivere prima il minore e poi il maggiore.

#### Programma C++

```
// Ordina.cpp: due numeri in ordine crescente

#include <iostream>
using namespace std;

int main()
{
 // input
 int a, b;

 cout << "Due numeri: ";
 cin >> a >> b;
 if (a < b) {
 cout << a << endl;
 cout << b << endl;
 }
}
```

```
else {
 cout << b << endl;
 cout << a << endl;
}

return 0;
}
```

Nel caso in cui la condizione  $a < b$  sia verificata, devono essere eseguite in sequenza le istruzioni

```
cout << a << endl;
cout << b << endl;
```

Pertanto, in accordo con quanto detto nel paragrafo precedente, le due istruzioni sono state raggruppate tra due parentesi graffe.

Analoghe considerazioni vanno fatte per la sequenza

```
cout << b << endl;
cout << a << endl;
```

Nel caso in cui dopo la condizione o dopo *else* ci sia un'unica istruzione, le parentesi graffe possono essere omesse:

```
if (condizione) istruzione-a;
else istruzione-b;
```

Se si ha una selezione a una sola via, in C++ si omette il ramo *else*:

```
if (condizione) {
 istruzioni-a;
}
```

La condizione è un'espressione booleana di cui viene valutata la verità: vengono quindi utilizzati i segni del confronto:  $=$ ,  $<$ ,  $>$ ,  $>=$ ,  $<=$ ,  $!=$ , e gli **operatori booleani**  $\&\&$  (And),  $\|$  (Or),  $!$  (Not) per costruire espressioni logiche combinando tra loro più condizioni.

Per esempio, nel seguente frammento di programma, la struttura di selezione:

```
if (classe == 5 && anni > 18) cout << nome << endl;
```

produce in output il nome di uno studente, soltanto nel caso in cui si verifichino entrambe le condizioni, perché viene usato l'operatore  $\&\&$  (And): frequenta la 5.a classe e ha più di 18 anni di età.

## PROGETTO 4

### Prodotti in offerta 3 x 2.

Se di un prodotto vengono acquistati 3 pezzi, si deve pagare l'importo pari a 2 pezzi. Per le quantità inferiori a 3 il prezzo resta invariato. Si ipotizza che il prodotto non venga acquistato in quantità maggiori di 3.

### Programma C++

```
// TreXDue.cpp: offerta 3 x 2

#include <iostream>
using namespace std;

int main()
{
 // input
 float prezzo;
 int qta;
 // output
 float importo;

 cout << "Prezzo e quantita': ";
 cin >> prezzo >> qta;
 if (qta == 3) qta = 2;
 importo = prezzo * qta;
 cout << "Importo da pagare = " << importo << endl;

 return 0;
}
```

In sostituzione della struttura di controllo *if* è possibile utilizzare l'**operatore ?**. Si tratta di un operatore ternario che è preceduto da una condizione e seguito da due istruzioni separate dal carattere due punti:

```
(condizione) ? istruzione1: istruzione2;
```

Quando la condizione risulta vera viene eseguita la prima istruzione, in caso contrario è la seconda istruzione ad essere eseguita.  
L'istruzione seguente:

```
(qta > 2) ? cout << qta - 1 : cout << qta;
```

è equivalente all'istruzione:

```
if (qta > 2) cout << qta - 1;
else cout << qta;
```

Questo operatore è solitamente utilizzato quando le istruzioni da eseguire sotto condizione sono semplici: per blocchi di istruzioni si predilige la versione estesa che fa uso dell'istruzione *if*.

Esiste un impiego particolare di questo operatore in combinazione con l'assegnazione:

```
qtatot = (qta > 2)? qta - 1 : qta;
```

La precedente istruzione è equivalente a:

```
if (qta > 2) qtatot = qta - 1;
else qtatot = qta;
```



## LE STRUTTURE ANNIDATE DI ALTERNATIVA

In una struttura di alternativa, dopo la condizione o dopo *else*, può essere presente un gruppo di istruzioni, contenente a sua volta altre strutture *if* interne alla struttura più esterna. Si dice, in questo caso, che si creano **strutture annidate**.

### PROGETTO 5

**Su una somma di denaro si vuole applicare un'imposta progressiva secondo lo schema:**

**da 0 a 5000 euro: imposta del 5%**

**sopra i 5000 euro e fino a 15000 euro: imposta del 7%**

**oltre i 15000 euro: imposta dell'8%.**

Dopo aver acquisito in input l'importo a cui applicare l'imposta, si considerano tre possibilità mutuamente esclusive. Se l'importo è minore o uguale a 5000 euro, si applica l'imposta del 5%; in caso contrario se l'importo è minore o uguale a 15000 euro, si applica il 5% sui primi 5000 e si applica il 7% sulla parte rimanente; come ultima possibilità, resta il caso in cui l'importo superi i 15000 euro: si calcola allora il 5% sui primi 5000, più il 7% sui successivi 10000, più l'8% sulla parte eccedente i 15000 euro (calcolo dell'imposta a scaglioni).

#### Programma C++

```
// Imposta.cpp: imposta a scaglioni

#include <iostream>
using namespace std;

const float SCA1 = 5000; // primo scaglione
const float SCA2 = 15000; // secondo scaglione
const float AL1 = 5; // aliquote
const float AL2 = 7;
const float AL3 = 8;

int main()
{
 // input
 float impo; // imponibile
 // output
 float tassa; // imposta

 cout << "Importo: ";
 cin >> impo;
 if (impo <= SCA1)
 tassa = impo * AL1 / 100;
 else
 if (impo <= SCA2) {
 tassa = SCA1 * AL1 / 100;
 tassa += (impo - SCA1) * AL2 / 100;
 }
}
```

```
else {
 tasso = SCA1 * AL1 / 100;
 tasso += (SCA2 - SCA1) * AL2 / 100;
 tasso += (impo - SCA2) * AL3 / 100;
}

cout << "Imponibile = " << impo << endl;
cout << "Imposta = " << tasso << endl;

return 0;
}
```

L'utilizzo di costanti all'interno di un programma è molto utile soprattutto per facilitare un eventuale intervento di aggiornamento del programma qualora cambiassero i criteri di calcolo dell'imposta: in questo caso basta modificare i valori definiti come costanti, senza modificare le altre istruzioni.

#### AUTOVERIFICA

Domande da 18 a 21 pag. 297-298  
Problemi da 13 a 18 pag. 303

## 14 La ripetizione

Nel linguaggio C++ la **struttura di ripetizione** si rappresenta con l'istruzione **do ... while**:

```
do {
 istruzioni;
} while (condizione);
```

La sequenza di istruzioni compresa tra *do* e *while* viene ripetuta tante volte, mentre la condizione scritta dopo *while* si mantiene vera: in altre parole la ripetizione termina quando la condizione scritta dopo *while* diventa falsa.

Per esempio, si consideri il seguente frammento di programma:

```
do {
 cout << "il mese:";
 cin >> mese;
} while (mese < 1 || mese > 12);
```

Solo quando il mese digitato dall'utente è compreso tra 1 e 12 l'esecuzione del programma procede con l'istruzione successiva a *while*.

Si noti che le istruzioni da ripetere sono raggruppate tra una coppia di parentesi graffe e che la condizione scritta dopo *while* è racchiusa tra le parentesi tonde. La condizione è composta e utilizza l'operatore **||** (Or).



## PROGETTO 6

### Calcolo del prodotto tra interi utilizzando la sola operazione di somma.

Presi in considerazione due numeri interi positivi, si tratta di sommare il primo numero con se stesso un numero di volte pari al secondo numero. Per sapere quante volte deve essere eseguita l'operazione di somma e per sapere quando fermarsi, ogni volta che viene fatta la somma si decrementa di 1 il valore del secondo numero e si interrompe la ripetizione della somma quando questo diventa 0.

#### Programma C++

```
// Prodotto.cpp: prodotto di due numeri

#include <iostream>
using namespace std;

int main()
{
 // input
 int a, b;
 // output
 int prod = 0;

 cout << "Due numeri: ";
 cin >> a >> b;
 do {
 prod += a;
 b--;
 } while (b > 0);

 cout << "Prodotto = " << prod << endl;

 return 0;
}
```

## PROGETTO 7

### Dato un elenco di persone, con l'indicazione per ciascuna di esse del nome e dell'età, si devono contare i maggiorenni.

Si deve organizzare un programma che, per ogni persona dell'elenco, ripeta la richiesta dei dati (nome ed età), controlli se l'età è maggiore o uguale a 18 anni e, in questo caso, incrementi il contatore dei maggiorenni.

Alla fine della ripetizione viene scritto il numero dei maggiorenni registrato nel contatore. Dopo la richiesta dei dati di una persona e il controllo sull'età, viene posta all'utente la domanda se l'elenco è finito.

Si ripetono le operazioni finché la risposta è uguale a 1 per 'sì'; la ripetizione continua se la risposta è 0 per 'no'.

### Programma C++

```
// Elenco1.cpp: elenco di persone

#include <iostream>
#include <string>
using namespace std;

int main()
{
 // input
 string nome;
 int eta;
 int risp;
 // output
 int conta = 0;

 do {
 cout << "Nome: ";
 cin >> nome;
 cout << "Eta': ";
 cin >> eta;
 if (eta >= 18) conta++;
 cout << "Elenco finito: 0=no, 1 = si ? ";
 cin >> risp;
 } while (risp == 0);

 cout << "I maggiorenni sono = " << conta << endl;

 return 0;
}
```

## 15 La ripetizione precondizionale

Nel linguaggio C++ la **ripetizione precondizionale**, cioè con controllo iniziale della condizione, si rappresenta con l'istruzione **while**:

```
while (condizione) {
 istruzioni;
}
```

Mentre la *condizione* si mantiene vera, viene eseguita la sequenza di istruzioni (racchiuse tra le parentesi graffe).

Per esempio se si vuole far entrare da tastiera un elenco di numeri, segnalando la fine dell'elenco con il numero 0, si può utilizzare il seguente frammento di programma C++.

```
int numero;

cout << "Inserire un numero (0=fine): ";
cin >> numero;
while (numero != 0) {
 ;
 ;
 cout << "Inserire un numero (0=fine): ";
 cin >> numero;
}
```

Se, invece, si vuole inserire un elenco di nomi, usando il carattere \* per segnalare la fine dell'elenco, si può scrivere:

```
string nome;

cout << "Nome (*=fine): ";
cin >> nome;
while (nome != "*") {
 ;
 ;
 cout << "Nome (*=fine): ";
 cin >> nome;
}
```

Per controllare la fine dell'inserimento, il nome acquisito da tastiera viene confrontato con il carattere \*: essendo *nome* una stringa, il carattere \* è racchiuso tra una coppia di virgolette. In alternativa, considerando l'asterisco \* come singolo carattere, si può scrivere

```
while (nome[0] != '*') {

}
```

Poiché la stringa può essere considerata come una sequenza di caratteri, il confronto del carattere \*, che segnala la fine dell'inserimento, deve essere fatto con il primo carattere della stringa (carattere della posizione 0 della stringa) indicato con *nome[0]*. In questo caso il carattere \* è racchiuso tra una coppia di apici.

## PROGETTO 8

### Divisione tra interi usando le sottrazioni successive.

Presi in considerazione due numeri interi, si tratta di sottrarre al primo numero il secondo e di controllare quando il primo numero diventa minore del secondo. Le sottrazioni ripetute vanno contate e il numero di queste fornisce il quoziente. La parte rimanente del primo numero dopo le sottrazioni ripetute è il resto.

#### Programma C++

```
// Divisione.cpp: divisione tra interi con sottrazioni successive

#include <iostream>
using namespace std;

int main()
{
 // input
 int a, b;
 // output
 int quoz = 0; // quoziente della divisione tra interi

 cout << "Due numeri: ";
 cin >> a >> b;
 while (a >= b) {
```

```
 a -= b;
 quoz++;
}

cout << "Quoziente = " << quoz << endl;
cout << "Resto = " << a << endl;

return 0;
}
```

## PROGETTO 9

**Modificare il programma presentato nel Progetto 7, per contare i maggiorenni presenti in un elenco di persone, utilizzando una struttura di ripetizione precondizionale.**

Viene riproposto il problema del Progetto 7 per mostrare come si presenta la soluzione quando si utilizza la struttura di ripetizione precondizionale.

Alla richiesta dei dati di ogni persona viene suggerito all'utente di segnalare la fine dell'elenco digitando un carattere speciale (per esempio un asterisco) al posto del nome. Il nome della prima persona viene richiesto all'esterno del ciclo e le operazioni di scansione dell'intero elenco sono eseguite *mentre* il nome è diverso dal carattere speciale (asterisco).

### Programma C++

```
// Elenco2.cpp: elenco di persone

#include <iostream>
#include <string>
using namespace std;

int main()
{
 // input
 string nome;
 int eta;
 // output
 int conta = 0;

 cout << "Nome (*=fine): ";
 cin >> nome;
 while (nome != "*") {
 cout << "Eta': ";
 cin >> eta;
 if (eta >= 18) conta++;
 cout << "Nome (*=fine): ";
 cin >> nome;
 }

 cout << "I maggiorenni sono = " << conta << endl;

 return 0;
}
```

## 16 La ripetizione con contatore

Nel linguaggio C++, la **struttura di ripetizione con contatore** (o *enumerativa*) viene rappresentata con la struttura **for** nel modo seguente:

```
for (int i=min; i<=max; i++) {
 istruzioni;
}
```

Le istruzioni da eseguire in modo iterato possono essere una o più istruzioni. In presenza di più istruzioni è necessario racchiuderle tra una coppia di parentesi graffe.

Le istruzioni vengono ripetute tante volte, quante ne occorrono per portare il valore del contatore *i* dal valore iniziale (*min*) al valore finale (*max*).

Il contatore deve essere una variabile dichiarata di tipo *int* e la dichiarazione del tipo può essere posta direttamente all'interno della struttura *for*.

Per esempio, il frammento di programma

```
for (int i=1; i<=20; i++) {
 cin >> numero;
 cout << numero*2 << endl;
}
```

rappresenta un'iterazione per ottenere il doppio di 20 numeri inseriti da tastiera.

La struttura di ripetizione con contatore è una struttura derivata dalla struttura fondamentale di ripetizione, nel senso che il controllo della ripetizione delle istruzioni mediante un contatore può essere realizzato in modo equivalente usando una struttura di ripetizione postcondizionale *do ... while* o una struttura precondizionale *while*, come si vede dal seguente esempio.

Queste tre strutture sono funzionalmente equivalenti:

struttura **for**

```
for(int i=1;i<=n;i++) {
 cout << "xxxxxxxx" << endl;
}
```

struttura **do ... while**

```
int i;
i = 1;
do {
 cout << "xxxxxxxx" << endl;
 i++;
} while (i<=n);
```

struttura **while**

```
int i;
i = 1;
while (i<=n) {
 cout << "xxxxxxxx" << endl;
 i++;
}
```

La struttura *for*, nel caso di ripetizione con un numero prefissato di volte, risulta più compatta e più semplice da rappresentare rispetto alla struttura *while*, in quanto dentro il *for* sono rappresentati l'assegnazione del valore iniziale del contatore e l'incremento del contatore stesso.

Si noti che, nel secondo esempio con la struttura di ripetizione postcondizionale, le istruzioni vengono eseguite almeno 1 volta, anche nel caso in cui il valore iniziale di *i* sia maggiore del valore finale.

Il programma seguente calcola il doppio dei primi 30 numeri naturali:

```
#include <iostream>
using namespace std;

int main()
{
 for (int i=1; i<=30; i++) {
 cout << i*2 << endl;
 }

 return 0;
}
```

## PROGETTO 10

### Tavola pitagorica.

Il programma seguente produce su video la tavola pitagorica: *r* indica il numero di riga, *c* il numero di colonna. I prodotti di ciascuna *r* per i diversi *c* vengono visualizzati sulla stessa riga all'interno di 5 posizioni di stampa (si usa il manipolatore *setw(5)* spiegato nei *Materiali on line* n. 2 di questo capitolo).

Alla fine di ogni riga c'è un ritorno a capo con l'istruzione *cout << endl;*

Il programma mostra l'uso di un ciclo *for* interno con contatore *c*, e di un ciclo *for* esterno con contatore *r*.

### Programma C++

```
// TavolaPitagorica.cpp
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
 for (int r=1; r<=10; r++) {
 for (int c=1; c<=10; c++)
 cout << setw(5) << r*c ;
 cout << endl;
 }

 return 0;
}
```

Per ogni valore di *r* vengono eseguite due istruzioni: *for* e *cout << endl;*; per questo motivo occorre usare le parentesi graffe.

Per ogni valore di *c* viene invece eseguita la sola istruzione *cout <<* e pertanto non occorrono le parentesi graffe.

L'uso del manipolatore **setw** richiede l'inserimento della dichiarazione di inclusione:

```
#include <iomanip>
```



## LA STRUTTURA DI RIPETIZIONE *FOR* E L'OPERATORE VIRGOLA

Il ciclo *for* del linguaggio C++ possiede caratteristiche più generali rispetto ai cicli enumerativi degli altri linguaggi di programmazione. Si tratta infatti di un ciclo *while* combinato con una parte dedicata all'assegnazione e una all'incremento.

È pertanto corretto scrivere:

```
for (; a<b;)
 a += 2;
```

che risulta essere equivalente a

```
while (a<b)
 a += 2;
```

La situazione limite è:

```
for (;;) {
}
```

che rappresenta una ripetizione senza fine.

Le caratteristiche di generalità di questo ciclo *for* permettono, inoltre, combinazioni di questo tipo:

```
for (int i=0, j=1; a < 4; i++, j+=3)
 a = i * j / 2;
```

In questo esempio vengono dichiarate e inizializzate due variabili, *i* e *j*; la ripetizione viene eseguita per *a* minore di 4 e ad ogni iterazione si incrementa *i* di 1 e *j* di 3. L'**operatore virgola** all'interno del *for* esegue una valutazione sequenziale delle operazioni da sinistra verso destra.

Si osservi che la condizione non deve necessariamente dipendere dal contatore, ma, al contrario, può fare riferimento ad altri contesti.

Il seguente frammento di codice calcola la somma dei doppi e dei tripli dei primi 10 numeri interi:

```
tot1 = 0;
tot2 = 0;
for (int i=1; i<=10; i++) {
 tot1 += i*2;
 tot2 += i*3;
}
```

Esso può essere riscritto nel seguente modo abbreviato:

```
for (int i=1, tot1=0, tot2=0; i<=10; tot1+=i*2, tot2+=i*3, i++);
```

producendo gli stessi risultati per le variabili *tot1* e *tot2*.

## PROGETTO 11

### Data in input una serie di $n$ numeri, determinare il massimo tra essi.

Poiché si suppone noto il numero  $n$  dei dati da inserire, la soluzione più conveniente è quella di organizzare un programma che richieda all'utente tale numero e di inserire come secondo passo del programma una ripetizione con contatore per acquisire  $n$  volte i numeri da analizzare.

Nessuna ipotesi, invece, è fatta sui numeri entro i quali cercare il **massimo**: si può, pertanto, ipotizzare in modo del tutto arbitrario che il primo numero inserito sia il massimo. Successivamente, per ogni numero della serie inserito, si confronta tale numero con l'ipotetico massimo precedente. Se il numero inserito risulta maggiore del massimo, il massimo viene aggiornato, altrimenti resta invariato.

### Programma C++

```
// Massimo.cpp: massimo di n numeri

#include <iostream>
using namespace std;

int main()
{
 // input
 int n; // numero dei dati inseriti
 int dato; // dato inserito
 // output
 int max; // massimo

 cout << "Quanti sono i dati? ";
 cin >> n;
 for (int i=1; i<=n; i++) {
 cout << "Inserisci il "<< i << ". dato: ";
 cin >> dato;
 if (i == 1) max = dato;
 if (dato > max) max = dato;
 }

 cout << "Il massimo valore e' " << max << endl;

 return 0;
}
```

Si osservi che il precedente algoritmo per il calcolo del massimo, può essere usato con poche modifiche anche per calcolare il **minimo**: basta indicare con *min* il valore minimo e scambiare il segno del confronto da  $>$  a  $<$  nella struttura di selezione:

```
if (dato < min) min = dato;
```

### AUTOVERIFICA



Domande da 22 a 25 pag. 298  
Problemi da 19 a 30 pag. 303-304



## LA STRUTTURA DI SCELTA MULTIPLA

Nel linguaggio C++ la **struttura di scelta multipla** è realizzata dall'istruzione **switch** e ha il seguente formato:

```
switch(variabile) {
 case valore1:
 istruzioni1;
 break;
 case valore2:
 istruzioni2;
 break;

 case valoren:
 istruzionin;
 break;
 default:
 istruzioni;
 break;
}
```

Dopo la parola *switch*, tra parentesi tonde, è indicato il nome della variabile (**selettore**) di cui si deve controllare il valore per decidere quale strada seguire tra quelle possibili.

Accanto ai valori previsti (valori costanti), per ogni **case**, devono essere scritti l'istruzione o il blocco di istruzioni da eseguire nel caso in cui la variabile assuma quei valori.

Se nessuno dei valori elencati corrisponde al valore di variabile, vengono eseguite le istruzioni scritte dopo **default**.

Di solito, uno solo dei casi tra quelli previsti deve essere eseguito: è possibile quindi interrompere il controllo degli altri casi inserendo l'istruzione **break** che provoca l'uscita dalla struttura *switch*.

La struttura di selezione multipla è una **struttura derivata** dalla struttura fondamentale di selezione binaria, in quanto potrebbe essere rappresentata utilizzando una successione di controlli nidificati del tipo *if (condizione) ... else ...*

Per esempio:

```
if (variabile == valore1)
 istruzioni1;
else
 if (variabile == valore2)
 istruzioni2;
 else
 if (variabile == valore3)
 istruzioni3;
 else
 istruzioni4;
```

## PROGETTO 12

Per la vendita di un prodotto si deve applicare uno sconto progressivo in base al numero dei pezzi ordinati secondo la tabella:

| Pezzi     | Sconto |
|-----------|--------|
| fino a 3  | 5%     |
| fino a 5  | 10%    |
| fino a 10 | 20%    |
| più di 10 | 30%    |

Le informazioni necessarie per risolvere il problema sono: il numero dei pezzi e il prezzo dell'articolo. In base al numero dei pezzi si determina la percentuale dello sconto e si calcola l'importo totale scontato.

### Programma C++

```
// ScontoProgressivo.cpp: sconto progressivo sui prodotti

#include <iostream>
using namespace std;

int main()
{
 // input
 float prezzo; // prezzo del prodotto
 int pezzi; // pezzi acquistati
 // output
 float importo; // importo da pagare
 // lavoro
 int sconto; // percentuale di sconto

 cout << "Pezzi acquistati: ";
 cin >> pezzi;
 cout << "Prezzo del prodotto: ";
 cin >> prezzo;
 switch(pezzi) {
 case 1:
 case 2:
 case 3:
 sconto = 5;
 break;
 case 4:
 case 5:
 sconto = 10;
 break;
 case 6:
 case 7:
 case 8:
 case 9:
 case 10:
```

```
sconto = 20;
break;
default:
 sconto = 30;
}

importo = pezzi * prezzo * (100.0-sconto)/100.0;
cout << "Importo da pagare = " << importo << endl;

return 0;
}
```

Si noti che le stesse istruzioni possono essere eseguite in corrispondenza di più valori possibili per la variabile: i diversi valori sono elencati con la parola *case* uno di seguito all'altro.



#### AUTOVERIFICA

Problemi di riepilogo da 63  
a 65 pag. 307



MATERIALE ONLINE

### 5. Programmi eseguibili e videate dei progetti precedenti

## 17 Lo sviluppo top-down

I programmi visti in precedenza riguardavano problemi relativamente semplici. Quando il livello di complessità aumenta, conviene suddividere il programma in sottoprogrammi secondo la metodologia di **sviluppo top-down**.

Si tratta in sostanza di definire quali siano le parti fondamentali di cui si compone il programma (*top*), e procedere poi al dettaglio di ogni singola parte (*down*). Ogni parte, che compone il programma, corrisponde ad un modulo che svolge una specifica funzionalità per la risoluzione del problema, cioè il programma viene scomposto in **moduli funzionalmente indipendenti**.

Viene facilitato il lavoro di **manutenzione del software**, perché si può intervenire con modifiche o correzioni su un solo modulo, avendo nel contempo cognizione di quello che fa l'intero programma.

Inoltre alcuni moduli, essendo funzionalmente indipendenti, possono essere riutilizzati, senza bisogno di grandi modifiche, all'interno di altri programmi.

Nel linguaggio C++ i sottoprogrammi sono rappresentati attraverso le **funzioni**.

Nei paragrafi precedenti sono già state presentate le **funzioni** di calcolo, per esempio la funzione *sqrt* per il calcolo della radice quadrata di un numero.

Esse sono esempi di funzioni predefinite nel linguaggio C++ o, come si dice nel linguaggio informatico, **funzioni built-in**, cioè moduli software già disponibili al programmatore che quindi non richiedono la loro implementazione. Questi moduli vengono utilizzati indicandone semplicemente il nome.

## 18 Le funzioni

Il termine **funzione** (in inglese *function*) è utilizzato nella programmazione con lo stesso significato che ha, in generale, nella matematica, cioè un procedimento che riceve valori come argomenti in ingresso e *restituisce* un valore come risultato. Si dice anche che la funzione *ritorna* un valore.

La sintassi generale di una funzione è:

```
 tipo di dato restituito nome della funzione (elenco dei parametri)
 {
 istruzioni;

 return valore restituito;
 }
```

Poiché la funzione restituisce un valore, occorre specificare, prima del nome che identifica la funzione, il **tipo** del valore restituito. Dopo il nome della funzione, le parentesi tonde servono a contenere l'elenco degli argomenti passati alla funzione, detti **parametri**. Le istruzioni che formano la funzione sono delimitate dalle parentesi graffe e rappresentano il codice che viene eseguito alla chiamata della funzione.

Il corpo della funzione contiene come ultima istruzione, la parola **return** seguita dal valore o dalla variabile contenente il risultato restituito dalla funzione. L'istruzione *return* provoca il ritorno del controllo alla funzione principale o, in generale, alla funzione chiamante.

In una funzione ci possono essere anche più *return*, in corrispondenza di uscite per diversi percorsi di elaborazione nella funzione.

Se il tipo restituito dalla funzione non è specificato, si assume, in mancanza (per *default*), che il tipo sia **int**.

Se si vuole invece che la funzione non restituisca alcun valore, bisogna specificare, come tipo di dato restituito, il tipo **void** (in italiano *vuoto* o *privo*):

**void** nome della funzione (elenco dei parametri)

L'elenco dei parametri segue la sintassi della dichiarazione delle variabili, vista nei progetti precedenti; i parametri sono separati dalla virgola e per ciascun parametro deve essere indicato il tipo (diversamente dalla dichiarazione delle variabili):

*tipo* nome della funzione (*tipo1* nome1, *tipo2* nome2, ...)

Il tipo *void* può essere usato anche nella dichiarazione dei parametri: il tipo *void* scritto tra parentesi tonde indica che nessun parametro viene passato alla funzione. Questo è il valore di *default*; infatti:

**void f (void)**

si può anche scrivere:

**void f ( )**

L'esempio seguente mostra la codifica di una funzione che esegue la somma di due numeri interi, ricevuti come parametri, e restituisce il valore calcolato.

```
int Somma (int a, int b)
{
 int s;

 s = a + b;
 return s;
}
```

Quando la funzione ha, come tipo di ritorno, il tipo *void*, l'istruzione *return* non è seguita da alcun valore o espressione, in quanto tale tipo indica proprio il fatto che la funzione non restituisce nulla alla funzione chiamante; in questo caso l'istruzione *return*, alla fine della funzione, può essere omessa.

L'esempio seguente mostra la codifica di una funzione che esegue la somma di due numeri e visualizza il risultato, senza restituire nulla alla funzione chiamante.

```
void StampaSomma (int a, int b)
{
 int s;

 s = a + b;
 cout << s << endl;
}
```

La seguente funzione di stampa è un esempio di funzione che non restituisce alcun valore e che non riceve alcun parametro come argomento: il tipo restituito è *void* e l'assenza del tipo tra parentesi tonde indica che nessun parametro viene passato alla funzione.

```
void Stampa ()
{
 cout << "stampa di prova" << endl;
 cout << "fine della funzione" << endl;
}
```

Vediamo ora come si possa effettuare la **chiamata di una funzione** da un qualunque punto del programma, dopo averla dichiarata e definita. Occorre specificare il nome della funzione seguito dall'elenco, tra parentesi tonde, dei valori da assegnare ai parametri della funzione:

nome della funzione (*elenco dei valori da passare ai parametri*);

Al momento della chiamata, il compilatore esegue il controllo di corrispondenza tra i parametri specificati nella definizione della funzione e i valori passati alla funzione.

Un esempio di chiamata della funzione *Somma* definita in precedenza è il seguente:

```
int totale;
int subtot1;
int subtot2;
...
totale = Somma(subtot1, subtot2);
...
```

Il **main** è a tutti gli effetti una funzione: questo è il motivo della presenza della coppia di parentesi tonde dopo la parola *main*, come visto nei progetti precedenti. Tuttavia la funzione *main* ha una caratteristica specifica: è la funzione che viene eseguita per prima all'avvio del programma e dalla quale direttamente o indirettamente vengono chiamate tutte le altre funzioni.

L'istruzione *return* può essere presente anche nella funzione *main()*, in tal caso il valore viene restituito direttamente al sistema operativo, che è il programma chiamante della funzione.

Questo spiega la presenza, in tutti i progetti precedenti, della parola **int** prima di *main()*, per indicare il tipo del valore restituito, così come la presenza dell'istruzione **return 0**. L'uso di *return* con il valore 0 è il modo più comune per indicare la terminazione di un programma che non ha incontrato errori durante l'esecuzione.

## PROGETTO 13

### Calcolo delle soluzioni di un'equazione di secondo grado.

Un'equazione di secondo grado scritta nella forma  $ax^2 + bx + c = 0$  è caratterizzata dai coefficienti  $a, b, c$ . Dopo aver calcolato il discriminante  $\Delta$  (delta) con la formula

$$b^2 - 4 * a * c$$

si possono riconoscere tre situazioni:

- $\Delta < 0$  non esistono soluzioni reali,
- $\Delta = 0$  le due soluzioni reali sono coincidenti,
- $\Delta > 0$  ci sono due soluzioni reali, distinte.

Se esistono soluzioni reali, queste si ottengono dalla formula  $(-b \pm \sqrt{\Delta}) / (2 * a)$ .

Perché il programma comprenda tutte le situazioni possibili, occorre prevedere anche che  $a$  sia uguale a zero. Infatti, se  $a = 0$  si deve procedere alla soluzione dell'equazione di primo grado:

$$bx + c = 0.$$

In questa eventualità bisogna poi riconoscere i sottocasi:

- $b = 0$  e  $c = 0$  equazione indeterminata,
- $b = 0$  equazione impossibile.

Da questa analisi sono state individuate tre sottofunzioni richiamate dal *main*:

- calcolo di *delta*
- risoluzione dell'equazione di primo grado nel caso  $a = 0$
- visualizzazione delle soluzioni quando *delta* è maggiore o uguale a 0.

### Programma C++

```
// Equazioni.cpp: soluzioni di un'equazione di secondo grado
```

```
#include <iostream>
#include <cmath>
using namespace std;

// input
float a, b, c; // coefficienti equazione
// output
float x1, x2; // soluzioni
// lavoro
float delta; // discriminante

void RisolviPrimoGrado()
{
 if ((b==0) && (c==0)) {
 cout << "Equazione indeterminata" << endl;
 }
 else {
 if (b==0) cout << "Equazione impossibile" << endl;
 else cout << "x = " << -c/b << endl;
 }
} // RisolviPrimoGrado
```

```
float CalcolaDelta()
{
 delta = b*b - 4*a*c;
 return delta;
} // CalcolaDelta

void ScriviSoluzioni()
{
 delta = CalcolaDelta();
 if (delta < 0) {
 cout << "Non esistono soluzioni reali" << endl;
 }
 else {
 x1 = (-b-sqrt(delta)) / (2*a);
 x2 = (-b+sqrt(delta)) / (2*a);
 cout << "x1 = " << x1 << endl;
 cout << "x2 = " << x2 << endl;
 }
} // ScriviSoluzioni

// funzione principale
int main()
{
 cout << "Tre coefficienti: ";
 cin >> a >> b >> c;

 if (a != 0) ScriviSoluzioni();
 else RisolviPrimoGrado();

 return 0;
}
```

Nel progetto precedente tutte le funzioni sono senza parametri. La funzione *CalcolaDelta* restituisce un valore di tipo *float*, le altre non restituiscono alcun valore e quindi sono di tipo *void*. Nel programma inoltre le variabili sono dichiarate all'inizio, dopo le direttive e al di fuori del *main*, perché esse sono utilizzate anche dalle funzioni: le variabili dichiarate all'esterno del *main* e delle funzioni si chiamano **variabili globali**, in contrapposizione a quelle dichiarate all'interno delle funzioni o all'interno del *main*, che prendono il nome di **variabili locali**.

Si noti infine l'uso della direttiva **#include <cmath>**, perché il problema richiede il calcolo della radice quadrata, realizzato dalla funzione predefinita **sqrt**.

Per rendere più efficace la lettura e l'interpretazione del codice, può essere utile ripetere, dopo la parentesi graffa che chiude il corpo della funzione, il nome della funzione stessa come commento (preceduto dai caratteri *//*): in questo modo appare più evidente dove inizia e dove termina la funzione.

## 19 Funzioni con parametri

L'uso delle funzioni risponde all'esigenza fondamentale di costruire programmi ben organizzati e strutturati secondo la metodologia top-down.

Un'altra esigenza che viene risolta dalle funzioni è rappresentata dalla possibilità di poter utilizzare uno stesso gruppo di istruzioni in programmi diversi, proprio come accade con le funzioni predefinite. In questo caso conviene rendere parametrici i valori utilizzati dalle funzioni, in modo da ricevere come argomenti i valori passati dalla funzione *main* o da un'altra funzione chiamante.

## PROGETTO 14

### Calcolo delle soluzioni di un'equazione di secondo grado.

Viene ripreso il progetto del paragrafo precedente per il calcolo delle soluzioni di un'equazione di secondo grado, introducendo i parametri nelle funzioni.

#### Programma C++

```
// Equazioni2.cpp: soluzioni di un'equazione di secondo grado

#include <iostream>
#include <cmath>
using namespace std;

void RisolviPrimoGrado(float c2, float c3)
{
 if ((c2==0) && (c3==0)) {
 cout << "Equazione indeterminata" << endl;
 }
 else {
 if (c2==0) cout << "Equazione impossibile" << endl;
 else cout << "x = " << -c3/c2 << endl;
 }
} // RisolviPrimoGrado

float CalcolaDelta(float c1, float c2, float c3)
{
 return c2*c2 - 4*c1*c3;
} // CalcolaDelta

void CalcolaSoluzioni(float c1, float c2, float c3)
{
 float delta; // discriminante
 float x1, x2; // soluzioni

 delta = CalcolaDelta(c1, c2, c3);
 if (delta < 0) {
 cout << "Non esistono soluzioni reali" << endl;
 }
 else {
 x1 = (-c2-sqrt(delta)) / (2*c1);
 x2 = (-c2+sqrt(delta)) / (2*c1);
 cout << "x1 = " << x1 << endl;
 cout << "x2 = " << x2 << endl;
 }
} // CalcolaSoluzioni

// funzione principale
int main()
{
 float a, b, c; // coefficienti equazione

 cout << "Tre coefficienti: ";
 cin >> a >> b >> c;
 if (a != 0) CalcolaSoluzioni(a, b, c);
 else RisolviPrimoGrado(b, c);

 return 0;
}
```



In questa seconda versione del programma *Equazioni*, le variabili *a*, *b*, *c* sono variabili interne al *main*, o come si dice più correttamente, **variabili locali** del *main*, mentre le funzioni operano sui parametri. Le variabili *x1*, *x2* e *delta* che sono utilizzate solo all'interno della funzione per il calcolo delle soluzioni, sono **variabili locali** della funzione stessa.

Il vantaggio di questa modalità d'uso delle variabili consiste nel fatto che le funzioni così strutturate diventano moduli software che possono essere utilizzati trasportando senza modifiche il loro codice (con un semplice *copia e incolla*) in altri programmi che richiedano le stesse funzionalità.

L'operazione, con la quale il *main* (o un'altra funzione chiamante) invia i valori alla funzione, assegnandoli ai parametri, si chiama **passaggio di parametri**.

Le variabili indicate nell'intestazione della funzione (nel progetto precedente, le variabili *c1*, *c2*, *c3*) si chiamano **parametri formali**; le variabili che forniscono i valori ai parametri si chiamano **parametri attuali** (nel progetto precedente le variabili *a*, *b*, *c*).

## 20 Il passaggio di parametri

L'operazione, con la quale vengono associate le due liste dei parametri attuali e formali, detta *passaggio dei parametri*, può avvenire in due modi diversi:

- **passaggio di parametri per valore**, quando i valori delle variabili della funzione principale vengono ricopiati nei parametri della funzione; i cambiamenti effettuati sui parametri formali durante l'esecuzione della funzione non influenzano i valori delle variabili nella funzione chiamante (*main* o altra funzione).
- **passaggio di parametri per referenza** (o **per indirizzo**), quando i parametri attuali e formali fanno riferimento alla stessa cella di memoria centrale, cioè allo stesso **indirizzo** di memoria; questo è il caso in cui il programmatore vuole che i cambiamenti di valore ai parametri, durante l'esecuzione della funzione, influenzino i valori delle variabili corrispondenti nella funzione chiamante.

Nel linguaggio C++ il passaggio di parametri **per valore** avviene indicando nell'intestazione della funzione il tipo e il nome di ciascun parametro formale.

Il programma, presentato in precedenza per il calcolo delle soluzioni di un'equazione di secondo grado, fornisce alcuni esempi di funzioni che utilizzano il passaggio di parametri *per valore*. Consideriamo ora il caso di un problema nel quale è necessario usare il passaggio di parametri *per referenza*.

### PROGETTO 15

#### Ordinamento crescente di tre numeri.

Per disporre in ordine crescente tre numeri, si può procedere nel seguente modo: si confronta il primo numero con il secondo, scambiandoli tra loro se non sono in ordine crescente; successivamente si confronta il primo con il terzo, scambiandoli tra loro se non sono in ordine crescente, e infine il secondo con il terzo, scambiandoli tra loro se non sono in ordine crescente.

Si noti che nel programma viene (volutamente) ripetuta per tre volte la struttura *if*.

### Programma C++

```
// TreNumeri.cpp: ordinamento crescente di tre numeri

#include <iostream>
using namespace std;

// funzione principale
int main()
{
 // input-output
 int a, b, c;
 // lavoro
 int temp;

 cout << "Tre numeri: ";
 cin >> a >> b >> c;
 if (a > b) {
 temp = a;
 a = b;
 b = temp;
 }
 if (a > c) {
 temp = a;
 a = c;
 c = temp;
 }
 if (b > c) {
 temp = b;
 b = c;
 c = temp;
 }
 cout << "Numeri ordinati: " << endl;
 cout << a << endl;
 cout << b << endl;
 cout << c << endl;

 return 0;
}
```

In questo programma l'operazione di controllo e scambio del contenuto di due variabili viene eseguita più volte nel corso del programma su variabili diverse, ma usando le stesse istruzioni. Queste istruzioni, che vengono usate tre volte, in un programma meglio organizzato possono essere convenientemente raggruppate in una sola funzione *Ordina*, che verrà richiamata dal *main* ad operare tre volte con variabili diverse.

Quindi l'uso della funzione risponde all'esigenza di poter utilizzare più volte uno stesso gruppo di istruzioni in momenti diversi del programma, e magari con dati diversi.

La funzione *Ordina*, per lo scambio di due numeri che non sono in ordine crescente, richiede il passaggio di parametri **per referenza**. Infatti si vuole che lo scambio, effettuato sui parametri formali *x* e *y* della funzione *Ordina*, provochi lo scambio sulle corrispondenti variabili passate dal *main* nella chiamata alla funzione.

Vediamo come si realizza nel linguaggio C++ il passaggio di parametri per referenza, in modo che le modifiche apportate all'interno della funzione permangano anche dopo il ritorno al programma chiamante.

Per far questo è necessario che la funzione non operi su una copia locale dei dati, ma direttamente sui dati originali.

Per attivare questo tipo di comportamento è necessario passare, anziché la variabile direttamente, il suo **indirizzo**, in modo che la funzione veda la locazione di memoria dove è contenuta la variabile e possa quindi modificarla.

Per indicare il passaggio di un parametro **per referenza**, si deve aggiungere il carattere **&** subito dopo la dichiarazione del tipo del parametro.

### Programma C++

```
// TreNumeri2.cpp: ordinamento crescente di tre numeri

#include <iostream>
using namespace std;

void Ordina (int& x, int& y)
{
 int temp;

 if (x > y) {
 temp = x;
 x = y;
 y = temp;
 }
}

// funzione principale
int main()
{
 // input-output
 int a, b, c;

 cout << "Tre numeri: ";
 cin >> a >> b >> c;

 Ordina(a, b);
 Ordina(a, c);
 Ordina(b, c);

 cout << "Numeri ordinati: " << endl;
 cout << a << endl;
 cout << b << endl;
 cout << c << endl;

 return 0;
}
```

La funzione principale chiama tre volte la funzione *Ordina*, chiedendo di operare su tre diverse coppie di dati. Ogni chiamata causa l'esecuzione della funzione, al termine della quale la coppia di dati forniti risulta ordinata.

L'eventuale scambio, effettuato sui parametri formali *x* e *y*, provoca lo scambio sulle corrispondenti variabili identificate dagli indirizzi.

La scrittura **int&** indica che i parametri, di tipo intero, sono passati per referenza.

La chiamata della funzione nel *main* contiene, tra parentesi dopo il nome della funzione, le variabili che mandano i valori ai parametri:

```
Ordina(a, b);
```

I due esempi seguenti possono chiarire meglio la differenza tra passaggio per valore e passaggio per indirizzo.

Nel primo programma il passaggio del parametro è *per valore*:

```
// Parametri1.cpp: per valore

#include <iostream>
using namespace std;

void Modifica (int y)
{
 y += 2;
}

int main()
{
 int x;

 x = 1;
 cout << x << endl;
 Modifica(x);
 cout << x << endl;

 return 0;
}
```

La variabile *x* passa il suo valore al parametro *y*: il parametro *y* viene incrementato all'interno della funzione, ma l'operazione *y += 2* non influenza il valore originale di *x*. Sul video quindi compariranno i numeri 1 e 1, in corrispondenza delle due istruzioni *cout <<*.

Nel secondo programma il passaggio è *per referenza* (o indirizzo):

```
// Parametri2.cpp: per referenza

#include <iostream>
using namespace std;

void Modifica (int& y)
{
 y += 2;
}

int main()
{
 int x;

 x = 1;
 cout << x << endl;
 Modifica(x);
 cout << x << endl;

 return 0;
}
```

La variabile *x* passa il suo indirizzo al parametro *y*: il valore di *y* viene incrementato all'interno della funzione, e quello che succede alla variabile *y* è ciò che succede alla *x*, perché *x* e *y* fanno riferimento allo stesso indirizzo di memoria centrale. Sul video quindi compaiono i numeri 1 e 3, in corrispondenza delle due istruzioni di output.

Riassumendo, il passaggio per referenza si usa quando l'elaborazione effettuata all'interno della funzione deve produrre effetti alle variabili passate, necessari nelle successive elaborazioni del *main* (o della funzione chiamante). Si usa il passaggio per valore quando la funzione corrisponde ad un modulo che, per svolgere il suo lavoro, ha bisogno dei valori forniti dalla funzione chiamante, per esempio funzioni che devono stampare intestazioni o righe di totali.

Per rafforzare la differenza tra parametri passati per indirizzo e per valore, vediamo anche il seguente esempio.

```
// Incrementa.cpp: per valore e per referenza

#include <iostream>
using namespace std;

void Aggiungi (int x, int& y)
{
 x++;
 y++;
 cout << "Nella funzione chiamata: " << x << ", " << y << endl;
}

int main()
{
 int a, b;

 a = 0;
 b = 0;
 Aggiungi (a, b);
 cout << "Nella funzione chiamante: " << a << ", " << b << endl;

 return 0;
}
```

Nel programma presentato il valore di *a* viene passato al parametro *x* per valore e la variabile *b* viene passata al parametro *y* per referenza. Per controllare il diverso comportamento dei parametri, i valori di *x* e *y* sono visualizzati all'interno della funzione, mentre i valori di *a* e di *b* sono visualizzati all'interno del *main* dopo che è stata eseguita la funzione *Aggiungi*.

Al termine dell'esecuzione, in output compaiono nell'ordine le coppie di numeri:

```
Nella funzione chiamata: 1 1
Nella funzione chiamante: 0 1
```

perché l'incremento di *x* (parametro per valore) all'interno della funzione non influenza il valore di *a*, mentre l'incremento della variabile *y* (parametro per referenza) modifica il valore iniziale di *b*.

## 21 Dichiarazione delle funzioni con i prototipi

Le funzioni di un programma C++ possono comparire in qualsiasi ordine nel file sorgente. Le funzioni devono però essere definite prima che esse vengano utilizzate dal *main* o da altre funzioni, come è stato fatto negli esempi precedenti.

L'ordine in cui le funzioni sono definite è quindi importante.

Una funzione non ne può richiamare un'altra se questa non è stata ancora definita.

Inoltre il programma sorgente può essere suddiviso in più file con estensione *.cpp* contenenti diverse dichiarazioni di variabili e di funzioni. In fase di compilazione dovranno poi essere specificati tutti i file sorgente che devono essere tradotti nel programma oggetto.

Per queste ragioni, e per una migliore organizzazione dei programmi, è opportuno inserire le funzioni nel programma C++ attraverso due fasi distinte tra loro:

- dichiarazione delle funzioni per mezzo dei loro **prototipi**;
- definizione (o *implementazione*) delle funzioni.

La **dichiarazione della funzione**, che si intende utilizzare nel programma, di norma va posta in testa al programma stesso, immediatamente dopo la sezione dedicata alle direttive e alla dichiarazione delle variabili globali. Tramite la dichiarazione il programmatore si limita a comunicare al compilatore la sua intenzione di utilizzare una funzione che restituisce un dato di un certo tipo e che utilizza determinati parametri, riservandosi di specificarne il codice in seguito, nella fase di definizione. Questa dichiarazione si chiama **prototipo** della funzione.

Tutte le funzioni devono essere dichiarate, fatta eccezione per la funzione *main* che non necessita di dichiarazione, in quanto, essendo la prima funzione ad essere eseguita, non richiede controlli alla chiamata.

La sintassi generale per la dichiarazione di una funzione è:

*tipo di dato restituito* nome della funzione (*elenco dei parametri*);

La **definizione** (o **implementazione**) **della funzione**, con le istruzioni che la compongono, viene normalmente posta dopo la funzione *main*.

### PROGETTO 16

**Scrivere il programma per calcolare l'area e il perimetro di un quadrato, di cui è fornita da tastiera la misura del lato.**

Il programma può essere scomposto in tre parti fondamentali:

- inserimento da tastiera della lunghezza del lato
- calcolo dell'area
- calcolo del perimetro.

A queste tre parti corrispondono tre funzioni che vengono codificate in C++ nel seguente modo:

```
float Tastiera()
{
 float misura;
 cout << "Introduci il lato: ";
 cin >> misura;
 return misura;
}
```

```
void CalcoloArea (float l)
{
 float area;
 area = l * l;
 cout << area << endl;
}

void CalcoloPerimetro (float l)
{
 float perim;
 perim = l * 4;
 cout << perim << endl;
}
```

Le variabili *misura*, *area* e *perim* vengono usate solo all'interno delle rispettive funzioni e sono quindi variabili **locali**. Il programma completo può essere organizzato in questo modo:

*// Quadrato.cpp: perimetro e area del quadrato*

```
#include <iostream>
using namespace std;

float Tastiera();
void CalcoloArea (float l);
void CalcoloPerimetro (float l);

// funzione principale
int main(void)
{
 float lato;

 lato = Tastiera();
 CalcoloArea(lato);
 CalcoloPerimetro(lato);

 return 0;
}

float Tastiera()
{
 float misura; // variabile locale
 cout << "Introduci il lato: ";
 cin >> misura;

 return misura;
} // Tastiera

void CalcoloArea (float l)
{
 float area; // variabile locale
 area = l * l;
 cout << area << endl;
} // CalcoloArea

void CalcoloPerimetro (float l)
{
 float perim; // variabile locale
 perim = l * 4;
 cout << perim << endl;
} // CalcoloPerimetro
```

All'inizio del programma le funzioni vengono soltanto dichiarate, specificandone per ciascuna il tipo restituito, il nome e i parametri richiesti. In questo modo si ottiene una semplice elencazione dei prototipi che facilita la comprensione del programma e realizza in pratica la metodologia top-down.

Se si vuole poi vedere in dettaglio le modalità di elaborazione delle singole funzioni, si può analizzare l'implementazione delle funzioni scritta dopo la funzione *main*.

Come abbiamo già visto in precedenza, nelle direttive *#include* sono specificati i file che contengono i prototipi delle funzioni predefinite del linguaggio.

Si osservi che, per i progetti di questa parte del testo, nella dichiarazione dei prototipi delle funzioni, sono indicati per ogni parametro il tipo e il nome, per rendere più comprensibile la lettura del codice. In effetti il linguaggio C++ consente la dichiarazione delle funzioni con i prototipi indicando solo il tipo, senza il nome, dei parametri.

## PROGETTO 17

**Scrivere un programma per acquisire da tastiera un numero e comunicare all'esterno un messaggio che indichi se il numero è pari o dispari.**

La funzione booleana *Pari* riceve come parametro il numero e calcola il resto della divisione per 2. Se il resto è 0, restituisce il valore *true*, altrimenti ritorna il valore *false*.

### Programma C++

```
// PariDispari.cpp: numero pari o dispari

#include <iostream>
using namespace std;

bool Pari(int x);

// funzione principale
int main()
{
 int y;

 cout << "Introduci un numero: ";
 cin >> y;
 if (Pari(y)) cout << y << " numero pari" << endl;
 else cout << y << " numero dispari" << endl;

 return 0;
}

bool Pari(int x)
{
 if (x%2 == 0) return true;
 else return false;
}
```

La scrittura

```
if (Pari(y))
```



significa

```
if (Pari(y) == true)
```

cioè rappresenta un controllo sul valore logico *true* della funzione.

Si osservi anche che la funzione *Pari* possiede due uscite con le istruzioni *return*, in corrispondenza dei due valori *true* e *false*.



## REGOLE DI VISIBILITÀ

L'idea di organizzare i programmi in moduli funzionalmente indipendenti porta con sé la necessità di definire, all'interno della funzione, tutte le variabili necessarie al suo funzionamento: in realtà spesso, in un programma, una funzione può condividere variabili con altre funzioni.

In un programma si possono quindi distinguere:

- variabili **globali**, che vengono dichiarate all'inizio del programma e che possono essere usate dal programma principale e da tutte le funzioni;
- variabili **locali**, che vengono dichiarate all'interno della funzione che le utilizza e che non sono visibili alle altre funzioni (*main* compreso).

Anche le variabili dichiarate nel *main* sono di tipo locale, ossia visibili solo dal *main* stesso.

```
#include prototipi delle funzioni predefinite

tipo nome; variabili globali
.....
.....

tipo nomefunzione(parametri); prototipi delle funzioni
 del programmatore
.....
.....

int main() funzione principale
{
 tipo nome; variabili locali

 return 0;
}

tipo nomefunzione(parametri)
{
 tipo nome; variabili locali

 return;
}
```

Sulla base del ragionamento fatto in precedenza, appare chiaro che un programma ben strutturato (ossia scritto in modo chiaro, efficace ed efficiente) avrà poche variabili globali e molte variabili locali, proprio per consentire una facile manutenzione del software nel tempo e maggiori possibilità di riutilizzare moduli software in altri programmi.

Un esempio tipico di variabile locale è costituito dal contatore (di tipo *int*) di un ciclo *for* all'interno di una funzione: la funzionalità di tale contatore si esaurisce all'interno della funzione ed è quindi inutile dichiararlo come variabile globale.

Tutto questo è molto importante perché significa che per le variabili locali vengono riservate posizioni nella memoria centrale solo al momento dell'esecuzione della funzione. Lo spazio viene creato all'inizio e rilasciato al termine della funzione, liberando quindi spazio nella memoria utente: una ragione in più per usare molte variabili locali e poche variabili globali.

Questo rende inoltre possibile l'utilizzo dello stesso nome di variabile locale in due funzioni diverse.

Le **regole di visibilità** (in inglese **scope rules**) sono i criteri con i quali si stabilisce se un programma o una funzione *vedono* un elemento, nel senso che possono utilizzare l'elemento.

Il termine **elemento** sta ad indicare una qualsiasi risorsa presente nel programma: costanti, variabili, funzioni definite dal programmatore, funzioni di libreria del linguaggio C++.

Il termine **ambiente** indica tutto ciò che è racchiuso tra parentesi graffe { }.

Le regole di visibilità possono essere riassunte nel modo seguente:

Ogni ambiente vede

- le proprie risorse locali
- quelle definite all'esterno e che precedono l'ambiente stesso.

In sostanza una funzione (compresa la funzione *main*) può utilizzare tutte le variabili dichiarate localmente e le variabili globali, dichiarate prima dell'implementazione della funzione, che hanno nomi diversi dalle variabili locali.

La stessa regola vale anche per la dichiarazione delle funzioni.

Si consideri infine il codice del seguente programma che visualizza il valore di una variabile globale e di una variabile locale ad una funzione.

```
#include <iostream>
using namespace std;

float x; // variabile globale

void A ();

// funzione principale
int main()
{
 x = 10.32;
 cout << x << endl;
 A();
 cout << x << endl;

 return 0;
}
```

```
void A()
{
 int x; // variabile locale

 x = 10;
 cout << x << endl;
}
```

Si noti l'uso dello stesso nome di variabile per una variabile globale e una locale di tipo diverso: questa situazione viene indicata in inglese con il termine **shadowing**, nel senso che la variabile locale *x* di tipo *int*, durante l'esecuzione della funzione *A*, mette in ombra la variabile globale *x* di tipo *float*, impedendone la visibilità alla funzione *A*. La funzione *A*, infatti, non altera il contenuto della variabile globale.

Coerentemente con le regole di visibilità, inoltre, è possibile utilizzare lo stesso nome per una variabile della funzione chiamante e per il parametro formale che le corrisponde nella funzione chiamata: i parametri formali sono infatti risorse locali della funzione. Questa soluzione può essere utile nei programmi complessi per rendere più leggibile il codice.



#### AUTOVERIFICA

Domande da 26 a 29 pag. 299  
Problemi da 31 a 43 pag. 304-305

## 22 Le funzioni predefinite

Il linguaggio C++ possiede alcune **funzioni predefinite** (*built-in*), che il programmatore può usare nel programma senza dichiarazione. Queste funzioni sono raggruppate in **librerie di funzioni**, che sono l'insieme dei file contenenti i prototipi delle funzioni. Le funzioni di libreria sono richiamabili dai programmi C++, inserendo all'inizio del programma la direttiva **#include** che specifica il file che contiene la libreria stessa.

Esempi di file di inclusione per le librerie, già utilizzati nei programmi precedenti, sono:

- **iostream** per le funzioni standard di input/output
- **cmath** per le funzioni matematiche.

Per esempio la funzione **sqrt** calcola la radice quadrata di un numero, mentre la funzione **fabs** restituisce il valore assoluto di un numero in virgola mobile (*double*).

Se la funzione *fabs* non fosse già predefinita nel linguaggio, il programmatore la dovrebbe creare, dichiarandola e usandola nel programma come nel progetto seguente.

### PROGETTO 18

**Scrivere un programma che calcoli il valore assoluto di un numero reale.**

La funzione per il calcolo del valore assoluto ritorna l'opposto del numero ricevuto come parametro, se il numero è negativo, altrimenti ritorna il numero stesso.

#### Programma C++

```
// ValoreAssoluto.cpp: valore assoluto di un numero

#include <iostream>
using namespace std;

double fabs(double x);
```

```
// funzione principale
int main()
{
 double y;

 cout << "Introduci un numero: ";
 cin >> y;
 cout << fabs(y) << endl;

 return 0;
}

double fabs(double x)
{
 double ValAss;
 if (x < 0) ValAss = -x;
 else ValAss = x;
 return ValAss;
}
```

Per poter usare la funzione predefinita *fabs*, occorre dichiarare, all'inizio del programma, il file di inclusione **cmath**:

```
#include <cmath>
```

La funzione predefinita **pow** per il calcolo della potenza riceve due argomenti (base e esponente) di tipo *double* e restituisce un valore di tipo *double*.

Anche per questa funzione occorre inserire all'inizio del programma la direttiva *#include <cmath>* per rendere disponibili i prototipi delle funzioni matematiche.

## PROGETTO 19

### Calcolare la potenza di un numero reale con esponente reale.

#### Programma C++

```
// Potenza.cpp: potenza di un numero

#include <iostream>
#include <cmath>
using namespace std;

// funzione principale
int main()
{
 double base, esponente;
 double risultato;

 cout << "Base: ";
 cin >> base;
 cout << "Esponente: ";
 cin >> esponente;
 risultato = pow(base, esponente);
 cout << "Potenza = " << risultato << endl;

 return 0;
}
```

Le funzioni *built-in* della libreria **cmath** più utilizzate nel linguaggio C++ sono:

- **pow** (*base, esponente*) utilizzata per l'elevamento a potenza (il primo argomento costituisce la base, il secondo l'esponente; entrambi gli argomenti devono essere di tipo *double*)
- **exp**(*esponente*), eleva il numero di Nepero *e* all'esponente passato come parametro
- **sqrt**(*argomento*), calcola la radice quadrata del numero passato come argomento
- **log**(*argomento*), calcola il logaritmo in base naturale dell'argomento
- **log10**(*argomento*), calcola il logaritmo in base 10
- **sin**(*argomento*), **cos**(*argomento*), **tan**(*argomento*), per il calcolo di seno, coseno e tangente aventi come argomento l'angolo espresso in radianti (l'argomento deve essere di tipo *double*).

Rispettando le regole di visibilità, per poter utilizzare le funzioni predefinite, occorre includere i loro prototipi all'inizio del programma. Quando il programma è complesso, per rendere il codice più leggibile, è buona abitudine includere i file necessari prima di ciascuna funzione che li richiede, tramite la clausola *#include*.

I file di inclusione sono realizzati in modo tale che non si creino conflitti qualora venissero richiamati più volte.



## NAMESPACE E LIBRERIE DI INCLUSIONE

Il linguaggio C++ offre la possibilità di dividere lo spazio globale di definizione delle variabili (e delle funzioni) in diverse parti, dette **namespace** (spazio dei nomi), ad ognuna delle quali è associato un nome univoco definito dal programmatore. Questa possibilità ha permesso agli sviluppatori di programmi software di sfruttare l'utilizzo di *risorse globali*, ossia definite globalmente e visibili a tutte le funzioni, senza entrare in conflitto con altre librerie che impiegano lo stesso nome di variabile.

In pratica il *namespace* è un raggruppamento di dati e di funzioni che hanno tra loro un nesso logico per il fatto di riferirsi alla gestione di una determinata entità utile alla programmazione.

La definizione di un *namespace* utilizza una sintassi come nell'esempio seguente:

```
namespace spaziotglobale1 {
 // esempi di variabili e funzioni globali
 int totale;
 int Calcolo(int a, int b);
}
```

Le componenti di un *namespace* possono essere richiamate antepoendo alla componente (variabile o funzione) il nome del *namespace* di appartenenza separato con i doppi due punti (::). I caratteri :: si chiamano **scope resolution operator** (*operatore di risoluzione della visibilità*). Per esempio:

```
cout << spaziotglobale1::totale << endl;
```

Per un impiego permanente di tutte le componenti di un *namespace* si deve utilizzare il comando **using namespace**. Per esempio:

```
using namespace spaziotglobale1;
.....
cout << totale << endl;
```

Quest'ultima istruzione è del tutto equivalente alla precedente, perché la clausola *using* rende visibile la variabile senza l'utilizzo dell'operatore ::.

Il progetto seguente mostra un semplice uso pratico del *namespace*.

## PROGETTO 20

### Definire un *namespace* per gestire un rettangolo e le operazioni di calcolo su di esso.

Il *namespace* definisce la parte dati costituita dalle dimensioni del rettangolo e la parte funzioni per il calcolo dell'area e del perimetro.

Il *namespace* contiene solo il nome delle funzioni e i parametri necessari, mentre il dettaglio del codice delle funzioni, per maggiore chiarezza di lettura, è scritto dopo la definizione del *namespace*.

L'uso dell'operatore con i doppi due punti, ::, specifica il riferimento delle funzioni al *namespace* di appartenenza.

```
namespace Rettangolo {
 // dati
 int base, altezza;
 // funzioni
 int Area(int, int);
 int Perimetro(int, int);
}

int Rettangolo::Area(int b, int h)
{
 return b * h;
}

int Rettangolo::Perimetro(int b, int h)
{
 return (b + h) * 2;
}
```

Questo file, che è un normale file di testo, viene salvato su disco con il nome *rettangolo* nella sottodirectory **include** all'interno della directory di installazione del compilatore C++, in modo che possa essere ritrovato durante la compilazione.

Quando viene richiamato nel programma, il suo nome è scritto tra due parentesi angolari < >. In alternativa può essere registrato nella directory di lavoro dell'utente; in questo secondo caso viene incluso indicandone il nome tra virgolette: `#include "rettangolo"`.

Di fatto esso costituisce a tutti gli effetti una libreria disponibile per il programmatore, che può essere inclusa all'interno del programma di calcolo con la direttiva **#include**, proprio come abbiamo fatto nei progetti precedenti con la libreria *iostream*.

```
#include <iostream>
#include <rettangolo>
using namespace std;

int main()
{
 cout << "Base: ";
 cin >> Rettangolo::base;
 cout << "Altezza: ";
 cin >> Rettangolo::altezza;
 cout << Rettangolo::Area(Rettangolo::base, Rettangolo::altezza) << endl;
 cout << Rettangolo::Perimetro(Rettangolo::base, Rettangolo::altezza) << endl;

 return 0;
}
```

Come si vede, il programmatore può utilizzare le variabili e le funzioni che si riferiscono al rettangolo, senza dichiararle nel programma. Il riferimento ad esse è specificato dall'operatore *Rettangolo::* che indica il nome del *namespace*.

Volendo evitare di dover scrivere ogni volta questo riferimento, si deve specificare la clausola

```
using namespace Rettangolo;
```

Con questa dichiarazione il programma precedente può essere riscritto nel seguente modo:

```
#include <iostream>
#include <rettangolo>
using namespace std;
using namespace Rettangolo;

int main()
{
 cout << "Base: ";
 cin >> base;
 cout << "Altezza: ";
 cin >> altezza;
 cout << Area(base, altezza) << endl;
 cout << Perimetro(base, altezza) << endl;

 return 0;
}
```

Il progetto precedente, pur essendo molto semplice, fornisce un'idea dell'importanza del *namespace* e delle librerie disponibili per il programmatore, il quale può utilizzare nei suoi programmi dati e funzioni senza conoscerne l'effettiva implementazione con il codice C++. In questo modo si realizza l'**information hiding** (nascondere le informazioni), con il vantaggio di poter modificare in tempi successivi le funzioni (e in generale le librerie) per renderle più efficienti, senza dover cambiare i programmi che le utilizzano.

Per esempio, il programma precedente utilizza le variabili *base*, *altezza* e le funzioni *Area*, *Perimetro*, ma la loro implementazione non è presente nel programma perché è definita nel *namespace Rettangolo*. In pratica le funzioni rappresentano le **interfacce** che il programmatore utilizza per effettuare le elaborazioni sul rettangolo.

Il nome del *namespace* delle principali librerie del linguaggio C++ è **std** (*standard*).



MATERIALE ONLINE

## 6. File di intestazione e file di inclusione

## 23 Function overloading

Il linguaggio C++ offre la possibilità di definire due funzioni con lo stesso nome che differiscono tra loro solamente per l'elenco dei parametri. Tale diversità può riguardare sia il numero di parametri passati sia il loro tipo. In base al tipo e al numero di parametri passati alla funzione il compilatore riconosce la funzione che deve essere richiamata.

Questo meccanismo di ridefinizione delle funzioni si chiama **function overloading** (sovraccarico di una funzione).

Il programma seguente contiene tre funzioni con lo stesso nome *f*. La prima riceve un solo parametro di tipo intero, la seconda un parametro di tipo carattere e la terza due parametri, l'uno intero e l'altro carattere.

```
#include <iostream>
using namespace std;

void f(int i)
{
 cout << i << endl;
}

void f(char c)
{
 cout << c << endl;
}

void f(int i, char c)
{
 cout << i << '\t' << c << endl;
}

int main()
{
 f(10);
 f('A');
 f(100, 'z');

 return 0;
}
```

L'esecuzione del programma produce il seguente output:

```
10
A
100 z
```

Nella presentazione della funzione predefinita **pow** del paragrafo precedente, si è detto che gli argomenti della funzione e il valore restituito devono essere di tipo *double*, secondo un vincolo derivato dal linguaggio C:

```
double pow (double, double);
```

In effetti, il linguaggio C++ consente anche il calcolo della potenza di un numero per i tipi *float* e *long double*, utilizzando lo stesso nome di funzione:

```
float pow (float, float);
long double pow (long double, long double);
```

Questi tre prototipi di funzione sono esempi di *function overloading* predefinito nel linguaggio C++.



Il programma seguente mostra un esempio di *function overloading* definito dal programmatore per poter calcolare la potenza con numeri di tipo *int*. Ovviamente il calcolo è possibile nell'ambito del range di valori previsto per il tipo *int* (tra -2.147.483.648 e 2.147.483.647).

## PROGETTO 21

### Calcolare la potenza di un numero intero.

Si tratta di organizzare un programma che richieda in input i valori della base e dell'esponente ed esegua l'elevamento a potenza. Per realizzare l'elevamento a potenza intera basta moltiplicare la base per se stessa un numero di volte pari all'esponente.

#### Programma C++

```
// PotenzaInt.cpp: overloading di pow

#include <iostream>
using namespace std;

int pow(int x, int y);

// funzione principale
int main()
{
 // input
 int base, esponente;

 cout << "Base: ";
 cin >> base;
 cout << "Esponente: ";
 cin >> esponente;
 cout << "Potenza = " << pow(base, esponente) << endl;

 return 0;
}

int pow(int x, int y)
{
 int potenza = 1;
 for(int i=1; i<=y; i++) potenza *= x;

 return potenza;
}
```

#### AUTOVERIFICA

Domande da 30 a 32 pag. 300

Problemi da 44 a 46 pag. 305

Problemi di riepilogo da 66 a 71 pag. 307



#### MATERIALE ONLINE

### 7. Scomposizione dei programmi con le funzioni

## LA RICORSIONE

Con il termine **ricorsione** (o *ricorsività*) si indica la possibilità che una funzione ha di chiamare se stessa, cioè la chiamata della funzione è contenuta all'interno della funzione stessa.

In sostanza la funzione diventa essa stessa una risorsa per l'attività di elaborazione, insieme alle variabili, ai parametri e altre funzioni.

In matematica possono essere definiti procedimenti di calcolo ricorsivo quali, per esempio, il calcolo del fattoriale di un numero o il calcolo della potenza di un numero.

### PROGETTO 22

#### Calcolare il fattoriale di un numero $n$ ( $n!$ ).

$n!$  si legge *fattoriale di  $n$*  e sta ad indicare la moltiplicazione di tutti gli interi minori o uguali ad  $n$ :

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 3 \cdot 2 \cdot 1$$

Per esempio:  $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$

Inoltre, per definizione,  $0! = 1$

È possibile, per il calcolo del fattoriale, dare anche una definizione ricorsiva:

per  $n = 0$ ,  $n! = 1$

per  $n > 0$ ,  $n! = n \cdot (n-1)!$

Questa definizione viene utilizzata per realizzare una funzione ricorsiva *Fatt( $n$ )* che calcola il fattoriale di  $n$ .

#### PROGRAMMA C++

```
// Fattoriale.cpp: fattoriale di un numero

#include <iostream>
using namespace std;

long Fatt(int x);

// funzione principale
int main()
{
 int n;

 cout << "Numero naturale: ";
 cin >> n;
 cout << "Fattoriale del numero " << n << " = "
 << Fatt(n) << endl;

 return 0;
}
```

```
long Fatt(int x)
{
 long f;
 if (x==0) f = 1;
 else f = x * Fatt(x-1);

 return f;
}
```

Nella funzione *Fatt* è contenuta la chiamata della funzione stessa, passando come parametro il valore di  $x-1$ .

## PROGETTO 23

### Calcolare la potenza $n$ -esima (intera) di un numero.

La potenza  $x^n$  è uguale a  $x * x * x * \dots * x$ , cioè si ottiene moltiplicando  $n$  volte il numero  $x$  per sé stesso.

Usando un procedimento ricorsivo, si può dire che:

$$x^0 = 1$$

$$x^n = x * x^{n-1}$$

Per esempio,  $8^5 = 8 * 8^4$

La funzione ricorsiva per il calcolo della potenza  $n$ -esima (intera) di un numero  $x$  si implementa in questo modo:

```
long potenza(int x, int n)
{
 long p;

 if (n==0) p = 1;
 else p = x * potenza(x, n-1);

 return p;
}
```

Il programma completo è il seguente.

### Programma C++

```
// PotenzaRicorsiva.cpp: calcolo della potenza intera con funzione ricorsiva
#include <iostream>
using namespace std;

long potenza(int x, int n);

// funzione principale
int main()
{
 // input
 int base, esponente;

 cout << "Base: ";
 cin >> base;
 cout << "Esponente: ";
 cin >> esponente;
 cout << "Potenza = " << potenza(base, esponente) << endl;
```

```
 return 0;
}

long potenza(int x, int n)
{
 long p;
 if (n==0) p = 1;
 else p = x * potenza(x, n-1);

 return p;
}
```

## PROGETTO 24

### Rappresentare in base 2 un numero intero.

Per la rappresentazione in base 2 di un numero intero, si deve procedere a successive divisioni del numero fino ad ottenere 0 come dividendo: la successione dei resti delle divisioni, presi nell'ordine inverso rispetto a come sono stati creati, rappresenta il numero in base 2. L'utilizzo ricorsivo di una funzione, che divida il numero e comunichi il resto della divisione, permette di ottenere la successione dei resti in ordine inverso.

#### Programma C++

```
// Binario.cpp: rappresentazione in base 2 di un numero

#include <iostream>
using namespace std;

void Binario(int x);

// funzione principale
int main()
{
 int n;

 cout << "Numero da convertire in binario: ";
 cin >> n;
 Binario(n);
 cout << endl;

 return 0;
}

void Binario(int x)
{
 if (x!=0) {
 Binario (x/2);
 cout << x%2 << " ";
 }
}
```

#### AUTOVERIFICA

Domanda 33 pag. 300  
Problemi da 47 a 48 pag. 305



#### MATERIALE ONLINE

### 8. Programmi eseguibili e videate dei progetti precedenti

## 24 Costanti ed enumerazioni

Come già visto in precedenza, le costanti vengono dichiarate attraverso la parola chiave **const**, secondo la sintassi:

```
const tipo nome = valore;
```

La dichiarazione termina con un punto e virgola. Per esempio:

```
const int ALIQUOTA = 20, TASSO = 0.08;
const char RISPOSTA = 's';
const string LINGUA = "inglese";
```

Le enumerazioni sono invece una serie di valori interi che vengono associati a nomi di costanti. La sintassi della dichiarazione **enum** che crea un'enumerazione è la seguente:

```
enum identificatore {lista delle costanti};
```

dove *identificatore* è un nome che ricorda il contenuto dell'enumerazione; le costanti sono inserite tra una coppia di parentesi graffe e separate da virgola.

Per esempio:

```
enum Lista {C1, C2, C3, C4};
```

Con questa dichiarazione alla costante *C1* risulta automaticamente assegnato il valore 0, a *C2* il valore 1, a *C3* il valore 2 e a *C4* il valore 3.

È possibile modificare la serie dei valori assegnandoli a ciascuna costante; per esempio, per rappresentare i mesi del secondo trimestre dell'anno, si può usare l'enumerazione seguente:

```
enum SecondoTrimestre {APR=4, MAG=5, GIU=6};
```

oppure, più semplicemente:

```
enum SecondoTrimestre {APR=4, MAG, GIU};
```

perché, se non specificato, il valore assegnato ad una costante è l'intero successivo della costante precedente.

Se non viene specificato, il primo valore dell'enumerazione viene assunto come uguale a 0. L'enumerazione consente al programmatore di definire un nuovo tipo di dato: è quindi un esempio di *programmer-defined type*.

### PROGETTO 25

#### Calcolare il giudizio finale di un test sulla base del punteggio conseguito.

Una prova scritta è formata da 20 quesiti. Le risposte ai diversi quesiti sono valutate in decimali:

0 per una risposta completamente errata;

0.5 per una risposta parzialmente corretta;

1 per una risposta esatta.

La valutazione complessiva non riporta il punteggio, ma raggruppa i punti totalizzati nella prova secondo la tabella:

PUNTI

| da | a  | risultato |
|----|----|-----------|
| 0  | 6  | negativo  |
| 7  | 13 | incerto   |
| 14 | 20 | positivo  |

Si tratta di trasformare il punteggio, espresso attraverso un numero reale, nel corrispondente valore scelto nell'insieme di valutazione (positivo, incerto, negativo) e di comunicare il risultato allo studente. Per il risultato viene utilizzata la seguente enumerazione:

Valutazione {POSITIVO, INCERTO, NEGATIVO}

### Programma C++

```
// Valuta.cpp: valutazione di un test

#include <iostream>
#include <string>
using namespace std;

enum Valutazione {POSITIVO, INCERTO, NEGATIVO};

void Presenta(Valutazione ris);

// funzione principale
int main()
{
 // input
 float punti;
 string nome;
 // lavoro
 Valutazione risultato;

 cout << "Nome dello studente: ";
 cin >> nome;
 cout << "Punti totalizzati: ";
 cin >> punti;
 if (punti <= 6.0) risultato = NEGATIVO;
 else
 if (punti <= 13.0) risultato = INCERTO;
 else risultato = POSITIVO;
 cout << "Lo studente " << nome
 << " ha conseguito il risultato: ";
 Presenta(risultato);

 return 0;
}

void Presenta(Valutazione ris)
{
 switch(ris) {
 case NEGATIVO:
 cout << "negativo " << endl;
 break;
 case INCERTO:
 cout << "incerto " << endl;
 break;
 case POSITIVO:
 cout << "positivo " << endl;
 break;
 }
} // Presenta
```

### AUTOVERIFICA

Domande da 34 a 35 pag. 300-301  
Problemi da 49 a 51 pag. 305

## 25 L'array

In generale i **dati** sono le informazioni rappresentate in una forma trattabile con il computer. In molti problemi si ha la necessità di aggregare molti dati di tipo semplice, per facilitarne la rappresentazione e rendere più veloce il loro ritrovamento.

I dati sono cioè organizzati in un insieme che prende il nome di **struttura di dati**.

Un importante esempio di struttura di dati è l'*array*.

L'**array** è un insieme di elementi omogenei tra loro. Con una variabile possiamo indicare solo un dato, con l'*array* possiamo indicare tanti dati dello stesso tipo con un solo nome collettivo di variabile: l'identificatore dell'*array*. Gli elementi si distinguono uno dall'altro attraverso l'indice che viene assegnato nell'*array*, e che viene posto accanto all'identificatore dell'*array*.

L'*array* è quindi un insieme omogeneo di dati. Nella terminologia matematica con il termine **vettore** si fa riferimento agli *array* a una dimensione.

L'*array* si ottiene in C++ aggregando variabili dello stesso tipo. Un *array* si definisce con la seguente dichiarazione:

```
tipo NomeArray[dimensione];
```

Alla normale dichiarazione di variabile si aggiunge semplicemente, tra parentesi quadre, il numero di elementi (**dimensione**) che compongono l'*array*.

Per esempio, la seguente dichiarazione crea un *array* di 10 coefficienti di tipo *double*:

```
double coeff[10];
```

Le componenti di un *array* possono essere non solo numeriche, ma di uno qualsiasi dei tipi standard del linguaggio C++.

Se una variabile è definita di tipo *array*, deve essere sempre usata all'interno del programma accompagnando l'identificatore della variabile con un indice.

L'indice è solitamente una variabile di tipo *int*.

Nel programma la componente dell'*array* viene indicata dal nome dell'*array*, seguito dall'indice della componente, racchiuso tra due parentesi quadre.

Per esempio, si può usare un'istruzione di assegnazione:

```
coeff[i] = 7;
```

oppure produrre in output i valori delle componenti di un *array* con un ciclo *for*:

```
for (int i=0; i<10; i++) {
 cout << coeff[i] << endl;
}
```

La numerazione degli indici nel linguaggio C++ **inizia da 0** e quindi, nel nostro esempio, gli elementi dell'*array* sono coeff[0], coeff[1], ..., coeff[9].

È possibile **inizializzare un array** assegnando i valori alle componenti in fase di dichiarazione, come nell'esempio seguente:

```
double numeri[3] = {2.5, 1.2, 3.0};
```

I valori sono indicati all'interno delle parentesi graffe e separati dalla virgola.

Per rendere più efficace la rappresentazione del codice si possono usare le costanti di enumerazione, come visto in precedenza.

Per esempio, la funzione principale del programma per calcolare l'incasso totale di una settimana può essere scritta usando un array per gli incassi e i valori dell'**enumerazione come indici** (si noti che l'enumerazione inizia da 0):

```
enum Giorni {LUN=0, MAR, MER, GIO, VEN, SAB, DOM};
int main()
{
 float incasso[6];
 float totale=0;

 for(Giorni i=LUN; i<=SAB;i++) {
 cout << "Incasso del giorno " << i << ": ";
 cin >> incasso[i];
 totale += incasso[i];
 }
 cout << "Incasso totale = " << totale << endl;

 return 0;
}
```

Nei paragrafi precedenti, sono state utilizzate variabili stringa come la seguente:

```
string nome;
```

Oltre al tipo *string*, in C++ le variabili che devono contenere delle stringhe possono essere realizzate con un **array di caratteri**. Questo, infatti, era il modo di trattare le stringhe nel linguaggio C e che continua ad essere disponibile in C++.

Volendo dichiarare una variabile alfanumerica come un array avente le componenti di tipo *char*, si può utilizzare una dichiarazione di array, come quella mostrata nel seguente programma:

```
#include <iostream>
using namespace std;

char stringa[10]="abcdefghi";

int main()
{
 for(int i=0;i<9;i++) {
 cout << "Stringa[" << i << "] = " << stringa[i] << endl;
 }

 return 0;
}
```

Esso produce in output il valore di ogni componente dell'array *stringa*:

```
Stringa[0] = a
Stringa[1] = b
Stringa[2] = c
Stringa[3] = d
Stringa[4] = e
Stringa[5] = f
Stringa[6] = g
Stringa[7] = h
Stringa[8] = i
```



Come si vede dal programma, l'array ha dimensione 10, ma la stringa è formata da 9 caratteri (dall'indice 0 all'indice 8). Infatti il compilatore, nella rappresentazione interna, aggiunge automaticamente al termine di una stringa costante il carattere **Nul** del codice ASCII che corrisponde alla sequenza di escape **\0** (carattere **terminatore di stringa**), in modo che il programma possa trovarne la fine.

Pertanto la dimensione effettiva dell'array è superiore di una unità al numero di caratteri che formano la stringa.

Con una certa semplificazione anche il tipo **string** può essere visto come una struttura di tipo array le cui componenti sono di tipo *char*.

Il passaggio del parametro di tipo *string* si riconduce però a quanto visto per gli altri tipi di dati, in cui è possibile passare un parametro sia per valore sia per indirizzo. In quest'ultimo caso il parametro formale è di tipo *string&*.

Un'ultima considerazione sugli array riguarda proprio il **passaggio di parametri** alle funzioni, perché si discosta da quanto visto sinora, pur essendo un passaggio per indirizzo.

Quando un array viene passato come parametro ad una funzione, in realtà viene passata la locazione (cioè l'*indirizzo*) della prima componente dell'array.

All'interno della funzione, il parametro diventa una variabile come le altre: il nome dell'array è a tutti gli effetti una variabile contenente un indirizzo, cioè un **puntatore**, come vedremo nel Paragrafo 28.

Quindi un array viene passato ad una funzione sempre per referenza, per esempio:

```
int x[10];
.
.
f(x);
.
.
```

e la dichiarazione del parametro formale che riceve il suo indirizzo è composto dal nome dell'array seguito dalle parentesi quadre e, opzionalmente, dal numero che indica la dimensione dell'array, che nel nostro esempio diventa:

```
f (int v[10]) {

}
```

Il numero indicato tra parentesi quadre non ha alcun significato per il compilatore: infatti è possibile utilizzare anche la seguente notazione:

```
f (int v[]) {

}
```

Di conseguenza quando occorre effettuare il passaggio per referenza di un array alla funzione, basta indicare il nome dell'array: *f(x)*, come indicato sopra. Si deve anche osservare che il passaggio per referenza è il solo modo di passare un array come parametro.

## PROGETTO 26

### Acquisire da tastiera un array e sommare le sue componenti.

Il progetto si divide in due parti. Nella prima parte si deve affrontare il problema di acquisire da tastiera gli elementi del vettore.

Per memorizzare tali elementi si utilizza una struttura di tipo array.

La seconda parte è relativa alla somma degli elementi del vettore: per ottenere il risultato è necessario accumulare il valore di ciascun componente del vettore.

#### Programma C++

```
// SommaArray.cpp: somma le componenti di un array

#include <iostream>
using namespace std;

const int MAX = 100;

// prototipi delle funzioni
int ChiediDimensione();
void CaricaVettore(int v[], int d);
int Somma(int v[], int d);

// funzione principale
int main()
{
 int n;
 int v[MAX];
 int tot;

 n = ChiediDimensione();
 cout << "Carica gli elementi del vettore " << endl;
 CaricaVettore(v,n);
 tot = Somma(v,n);
 cout << "Somma del vettore = " << tot << endl;

 return 0;
}

// dimensione dell'array
int ChiediDimensione()
{
 int d;
 do {
 cout << "Dimensione del vettore: ";
 cin >> d;
 } while (d<1 || d>MAX);

 return d;
} // ChiediDimensione

// caricamento delle componenti
void CaricaVettore(int v[], int d)
{
 for (int i=0; i<d; i++) {
 cout << "Elemento di posto " << i << ": ";
 cin >> v[i];
 }
} // CaricaVettore
```

```
// somma delle componenti
int Somma(int v[], int d)
{
 int s = 0;
 for(int i=0; i<d; i++) s+=v[i];
 return s;
} // Somma
```

Nel programma precedente si può osservare che la specificazione della dimensione dell'array *v*, quando è un parametro formale della funzione, può essere tralasciata.

## 26 L'array a due dimensioni

La **matrice** (o **array a due dimensioni**) è un insieme di elementi dello stesso tipo che sono in corrispondenza biunivoca con un insieme di coppie ordinate di numeri interi, che rappresentano rispettivamente il numero della riga e il numero della colonna della matrice.

Le matrici in linguaggio C++ vengono definite in modo analogo agli array:

*tipo* NomeMatrice[N][M];

indicando il tipo delle sue componenti, il nome della matrice, il numero N di righe, il numero M di colonne.

Se la matrice è quadrata la definizione può essere scritta in questo modo:

```
const int MAX = 20;
int matrice1[MAX][MAX];
```

Per esempio, per organizzare gli incassi nei 20 reparti di un grande magazzino nei 6 giorni della settimana, la struttura dati più adatta è una matrice di 20 righe e 6 colonne, di numeri reali, che viene dichiarata in C++ con il seguente frammento di codice:

```
const int NR = 20;
const int NG = 6;
double incassi[NR][NG];
```

Volendo calcolare l'incasso totale di tutti i reparti nel quarto giorno della settimana (giorno di indice 3), occorre utilizzare una struttura *for* all'interno del seguente segmento di programma:

```
double somma = 0;
for (int i=0; i<NR; i++) {
 somma += incassi[i][3];
}
cout << somma << endl;
```

La struttura dati più adatta ad organizzare le temperature massime registrate in 50 città nei 31 giorni del mese è una matrice di 50 righe e 31 colonne:

```
const int NCIT = 50;
const int NGG = 31;
int temper[NCIT][NGG];
```

La temperatura registrata nel giorno 18 nella 27esima città si ottiene con la seguente istruzione di output, ricordando che gli indici partono da 0:

```
cout << temper[26][17] << endl;
```

## PROGETTO 27

### Calcolare i totali di riga e di colonna degli elementi di una matrice.

Il problema va scomposto in due parti: la prima riguarda il caricamento dei dati in memoria, la seconda è relativa ai calcoli e alla scrittura dei risultati.

La struttura dati definita è un *array a due dimensioni*.

Per il caricamento dei dati si consideri ogni riga della matrice come un vettore e si proceda come negli esempi dei paragrafi precedenti al caricamento di un array attraverso una ripetizione con contatore che va da 0 al numero delle colonne – 1. Questa ripetizione viene annidata in una ripetizione più esterna che permette di ripetere l'operazione di acquisizione dei dati per tutte le righe della matrice.

Nella seconda parte relativa al calcolo dei totali di riga e di colonna, per ogni riga si azzeri la variabile di accumulo, si totalizzano in questa variabile i valori della riga con una ripetizione con contatore che va da 0 al numero di colonne – 1 e alla fine si scrive il contenuto della variabile di totalizzazione. Per ottenere i totali di colonna il procedimento si sviluppa in modo analogo al precedente sostituendo il termine "righe" con il termine "colonne".

### Programma C++

```
// Totali.cpp: totali di riga e colonna in una matrice

#include <iostream>
#include <iomanip>
using namespace std;

const int MAX = 100;

// prototipi delle funzioni
int ChiediRighe();
int ChiediColonne();
void CaricaMatrice(int m[MAX][MAX], int r, int c);

// funzione principale
int main()
{
 // input
 int righe, colonne; // dimensioni indicate dall'utente
 int mat[MAX][MAX]; // matrice
 // output
 int TotRiga; // totale di riga
 int TotColonna; // totale di colonna

 righe = ChiediRighe();
 colonne = ChiediColonne();
 cout << "Carica gli elementi della matrice" << endl;
 CaricaMatrice(mat, righe, colonne);
 cout << endl;
 for (int i=0; i<righe; i++) {
 TotRiga = 0;
 for (int j=0; j<colonne; j++) {
 cout << setw(5) << mat[i][j];
```

```

 TotRiga += mat[i][j];
 }
 cout << setw(7) << TotRiga << endl;
}
cout << endl;
for (int j=0; j<colonne; j++) {
 TotColonna = 0;
 for (int i=0; i<righe; i++) {
 TotColonna += mat[i][j];
 }
 cout << setw(5) << TotColonna;
}
cout << endl;

return 0;
}

// prima dimensione della matrice
int ChiediRighe()
{
 int d;
 do {
 cout << "Numero di righe: ";
 cin >> d;
 } while (d<1 || d>MAX);

 return d;
} // ChiediRighe

// seconda dimensione della matrice
int ChiediColonne()
{
 int d;
 do {
 cout << "Numero di colonne: ";
 cin >> d;
 } while (d<1 || d>MAX);

 return d;
} // ChiediColonne

// caricamento delle componenti
void CaricaMatrice(int m[MAX][MAX], int r, int c)
{
 for (int i=0; i<r; i++) {
 for (int j=0; j<c; j++) {
 cout << "Elemento di posto " << i << ", "
 << j << ": ";
 cin >> m[i][j];
 }
 }
} // CaricaMatrice

```

Si osservi che il passaggio di una matrice, avvenendo per referenza, richiede anche il passaggio delle sue dimensioni. È infatti molto importante non superare la dimensione dichiarata, in quanto il compilatore non effettua alcun controllo, provocando la generazione di errori durante l'esecuzione.



#### AUTOVERIFICA

Domande da 36 a 38 pag. 301  
Problemi da 52 a 54 pag. 306

## 27 Le strutture

La struttura che serve a contenere dati di tipo diverso viene detta **record** (in inglese record significa *registrazione*) e lo spazio per una singola informazione che lo costituisce si chiama **campo** del record.

Una tale struttura si diversifica dagli array, visti in precedenza, perché contiene dati tra loro non omogenei, cioè numeri, parole, codici alfanumerici e addirittura vettori come nel caso di un libro con molti autori.

La dichiarazione di un record nel linguaggio C++ è ottenuta con la parola chiave **struct** seguita dal nome assegnato alla struttura e dall'elenco dei campi tra parentesi graffe:

```
struct NomeStruttura {
 tipo1 campo1;
 tipo2 campo2;
 tipo3 campo3;
};
```

Da qui in poi useremo il termine **struttura**, anziché *record*, perché più aderente al linguaggio C++. Per esempio, la struttura adatta a contenere le informazioni anagrafiche di una persona è definita, come un nuovo tipo di dato, in questo modo:

```
struct Persona {
 int matricola;
 string nome;
 string indirizzo;
 string DataNascita;
 string CodiceFiscale;
 bool coniugato;
};
```

A questo punto occorre dichiarare una variabile, avente come tipo la struttura definita sopra:

```
struct Persona dipendente;
```

o più semplicemente:

```
Persona dipendente;
```

All'interno di un programma i campi sono indicati con l'identificatore della variabile e con l'identificatore del campo separati dal punto.

Per esempio, in un'istruzione di assegnazione si scrive:

```
dipendente.coniugato = true;
```

e in un'istruzione di output si scrive:

```
cout << dipendente.matricola;
```

Il campo *DataNascita* nella struttura precedente può essere definito, a sua volta, come una struttura con i tre campi giorno, mese, anno, anteponendo alla dichiarazione di *Persona* la definizione di un nuovo tipo *Giorno*.

```
struct Giorno {
 int gg;
 int mm;
 int aa;
};

struct Persona {
 int matricola;
 string nome;
 string indirizzo;
 Giorno DataNascita;
 string CodiceFiscale;
 bool coniugato;
};

Persona dipendente;
```

La visualizzazione dell'anno di nascita della persona si ottiene con la seguente istruzione:

```
cout << dipendente.DataNascita.aa << endl;
```

Il tipo *struct* rappresenta un esempio di tipo definito dal programmatore.

## PROGETTO 28

### Calcolo dei giorni che separano due date.

Per ciascuna data inserita si calcoli il giorno progressivo a partire dal primo gennaio 1900. La differenza dei giorni progressivi corrispondenti alle due date è il numero dei giorni che le separano.

Per calcolare il giorno progressivo dell'anno a partire dal mese e dal giorno del mese, viene definito un array di dodici elementi, ciascuno dei quali contiene il numero dei giorni trascorsi prima del mese che forma la data, mentre quest'ultimo è utilizzato come indice per accedere all'array.

L'array descritto è un **vettore costante** e viene definito in testa al programma nel seguente modo:

```
const int trascorsi[] = {0,31,59,90,120,151,181,212,243,273,304,334};
```

Per i giorni trascorsi fino all'inizio dell'anno indicato nella data, in prima approssimazione si moltiplicano i 365 giorni dell'anno per il numero degli anni della data. Al risultato del prodotto vanno aggiunti tanti giorni quanti sono gli anni bisestili intercorsi prima dell'anno indicato. Poiché gli anni bisestili sono quelli che sono divisibili per quattro, il numero dei giorni da aggiungere è dato dalla divisione intera che ha come dividendo il numero degli anni -1 e come divisore il numero 4.

Per l'anno della data va aggiunto un giorno, solo se l'anno è bisestile e il mese è successivo a febbraio.

Si rammenti che l'anno 1900 non è bisestile mentre l'anno 2000 lo è (gli anni divisibili per 100 non sono bisestili, a meno che siano divisibili per 400). Questi vincoli non sono esplicitati nell'algoritmo, ma sono rispettati nelle formule di calcolo.

### Programma C++

```
// DistanzaDate.cpp: distanza tra date

#include <iostream>
using namespace std;

const int trascorsi[] = {0,31,59,90,120,151,181,212,243,273,304,334};
struct data {
 int gg;
 int mm;
 int aa;
};

// prototipi delle funzioni
long Calcolo(data dt);

// funzione principale
int main()
{
 // input
 data data1, data2; // date di input
 // lavoro
 long int progr1, progr2; // giorni progressivi
 long int diff; // differenza

 cout << "Prima data (gg mm aaaa): ";
 cin >> data1.gg >> data1.mm >> data1.aa;
 cout << "Seconda data (gg mm aaaa): ";
 cin >> data2.gg >> data2.mm >> data2.aa;
 progr1 = Calcolo(data1);
 progr2 = Calcolo(data2);
 diff = progr2 - progr1;
 cout << "Giorni di differenza = " << diff << endl;

 return 0;
}

// calcolo giorno progressivo
long int Calcolo(struct data dt)
{
 long int anno; // anno dal 1900
 long int progr; // progressivo

 anno = dt.aa - 1900;
 progr = anno*365 + (anno-1)/4 + trascorsi[dt.mm-1] + dt.gg;
 if (dt.mm>2 && dt.aa%4==0) progr++;

 return progr;
} // Calcolo
```



## 28 I puntatori

Una variabile di tipo **puntatore** contiene l'indirizzo di memoria di un'altra variabile.

L'utilizzo dei puntatori è molto importante, ma richiede molta attenzione in quanto può essere facilmente causa di errori di compilazione e di esecuzione.

Per i tipi di variabile, sia predefiniti nel linguaggio che definiti dall'utente, è possibile creare un puntatore scrivendo il carattere **\*** (asterisco o *star*) accanto al tipo nella dichiarazione della variabile.

Per esempio:

```
int* puntatore;
char* s;
struct data* ricorrenza;
```

Il primo è un puntatore in grado di contenere l'indirizzo di una variabile intera e per tale ragione è detto *puntatore di tipo intero*; il secondo è un *puntatore di tipo carattere* e il terzo è un *puntatore per una struttura* definita dall'utente.

Il carattere **\*** significa "*puntatore a*".

La scelta del tipo del puntatore dipende dal tipo di variabile alla quale si vuole puntare, ossia la variabile della quale si vuole memorizzare l'indirizzo di memoria. Per conoscere l'indirizzo di memoria di una variabile si usa l'**operatore &** (*ampersand* o *e commerciale*).

Per esempio, nel seguente frammento di codice:

```
int num;
int* pInteri;

pInteri = #
```

*pInteri* è un puntatore di tipo intero, *&num* restituisce l'indirizzo di memoria dove è memorizzata la variabile *num*. Dopo l'assegnazione *pInteri* punta alla memoria occupata da *num*. L'uso della lettera *p* come iniziale del nome del puntatore serve ad evidenziare meglio le variabili di questo tipo all'interno del codice.

L'operatore **&** sottintende la frase "*indirizzo di*". Supponiamo che l'indirizzo della variabile *num* sia 126 e contenga il numero 5; il puntatore *pInteri*, dopo l'istruzione di assegnamento, contiene il valore 126.

Per accedere alla memoria puntata da un puntatore è necessario l'utilizzo dell'**operatore \***. Questo operatore è già stato utilizzato nella dichiarazione dei puntatori, ma non va confuso con il suo impiego nelle istruzioni del programma.

Si consideri, per esempio, il seguente frammento di programma C++:

```
int a, b;
int* pInteri;

a = 5;
pInteri = &a;
b = *pInteri;
cout << b;
```

Il puntatore *pInteri* punta alla variabile *a*; nella penultima istruzione la variabile *b* acquisisce il valore della variabile puntata da *pInteri*, quindi, con l'ultima istruzione, il programma visualizza il valore 5.

L'istruzione di assegnazione  $b = *pInteri$  è esattamente equivalente all'istruzione  $b = a$ . Se infatti  $a$  è il nome simbolico per utilizzare la memoria allocata per quella variabile,  $pInteri$ , puntando ad  $a$ , può essere utilizzato in modo equivalente ad  $a$ , in quanto l'operatore  $*$  consente di utilizzare la memoria puntata dal puntatore. Il seguente programma serve a chiarire la differenza tra puntatore e variabile puntata e, quindi, tra valore della variabile e valore dell'indirizzo di memoria.

```
#include <iostream>
using namespace std;

int main()
{
 int num;
 int* pInteri;

 num = 4;
 pInteri = #
 cout << "valore della variabile: " << num << endl;
 cout << "indirizzo della variabile: " << &num << endl;
 cout << "valore del puntatore: " << pInteri << endl;
 cout << "indirizzo del puntatore: " << &pInteri << endl;
 cout << "valore della variabile puntata: "
 << *pInteri << endl;

 return 0;
}
```

L'esecuzione del programma produce in output un risultato simile al seguente:

```
valore della variabile: 4
indirizzo della variabile: 0042049C
valore del puntatore: 0042049C
indirizzo del puntatore: 004204A0
valore della variabile puntata: 4
```

I valori degli indirizzi sono rappresentati nella notazione esadecimale. La figura seguente illustra quanto accade con l'esecuzione del programma precedente: il valore numerico associato alla variabile *num* si trova all'indirizzo 0042049C della memoria centrale e ha il valore 4; il puntatore è memorizzato nella cella di indirizzo 004204A0 e contiene come valore 0042049C, cioè l'indirizzo dove si trova la variabile *num*.

|         | Indirizzo di memoria | Contenuto |                 |
|---------|----------------------|-----------|-----------------|
| num     | 0042049A             |           | &num = 0042049C |
|         | 0042049B             |           |                 |
|         | 0042049C             | 4         |                 |
|         | 0042049D             |           |                 |
|         | 0042049E             |           |                 |
| pInteri | 0042049F             |           | *pInteri = 4    |
|         | 004204A0             | 0042049C  |                 |
|         | 004204A1             |           |                 |
|         | 004204A2             |           |                 |

Come abbiamo visto nel Paragrafo 20, nel **passaggio di parametri per indirizzo** dalla funzione chiamante ad una funzione chiamata, le modifiche apportate all'interno di quest'ultima devono permanere anche dopo il ritorno alla chiamante: per far questo è necessario che la funzione non operi su una copia locale dei dati, ma direttamente sui dati originali. Per attivare questo tipo di comportamento la funzione chiamante passa, anziché la variabile direttamente, il suo **indirizzo**, in modo che la funzione chiamata veda la locazione di memoria dove è contenuta la variabile e possa quindi modificarla. Per questo motivo i parametri formali nell'instanziazione della funzione hanno il simbolo & accanto al tipo.

Per esempio, per l'ordinamento crescente di due numeri, la funzione chiamante contiene queste righe di codice:

```
cout << "Due numeri: ";
cin >> a >> b;
Ordina(a, b);
```

La funzione che effettua l'ordinamento è la seguente:

```
void Ordina (int& x, int& y)
{
 int temp;

 if (x > y) {
 temp = x;
 x = y;
 y = temp;
 }
}
```

In questo esempio *x* e *y* sono dei puntatori mascherati da variabili, in quanto ogni modifica effettuata nella funzione è applicata direttamente ai parametri attuali *a* e *b*.

Un utilizzo particolare dei puntatori è legato alla struttura dati di tipo array.

Se si vuole assegnare ad un puntatore l'indirizzo di un vettore, basta specificare il nome del vettore stesso, in quanto, come già visto nel Paragrafo 25, il nome di un vettore è un puntatore costante al primo elemento del vettore.

Per esempio:

```
int v[10];
int* pInteri;

pInteri = v;
```

Si potrebbe anche scrivere:

```
pInteri = &v[0];
```

in quanto *v* è uguale a *&v[0]* per definizione.

È possibile quindi utilizzare un puntatore come sinonimo del nome di array e accedere ai vari elementi del vettore stesso tramite *l'aritmetica dei puntatori*.

L'**aritmetica dei puntatori** consiste nel fatto che nel momento in cui si incrementa o decrementa un puntatore di un numero intero, questa operazione non viene effettuata aumentando di uno stesso numero il valore dell'indirizzo contenuto nella variabile puntatore, ma spostando il puntatore in avanti o indietro di un numero di posizioni pari a quello indicato. Il compilatore è in grado di spostare in modo corretto il puntatore poiché questi è tipizzato alla variabile a cui può puntare.

Per esempio:

```
int a[10];
int* p;

p = a;
p++;
```

Dopo l'ultima istruzione il puntatore *p* punta ad *a[1]*.

Se, per esempio, l'indirizzo di memoria di *a* è uguale a 126, supponendo che la dimensione in byte di una variabile di tipo *int* sia uguale a 4, *p* viene inizializzato a 126 e poi incrementato a 130, che è proprio l'indirizzo di *a[1]*, in quanto in memoria gli elementi dell'array sono contigui.

| a[0] | a[1] | a[2]  | a[3]  | a[4] | a[5]  | a[6]  | a[7]  | a[8]  | a[9]  |
|------|------|-------|-------|------|-------|-------|-------|-------|-------|
| 126  | 130  | 134   | 138   | 142  | 146   | 150   | 154   | 158   | 162   |
| p    | p+1  | ..... | ..... | p+4  | ..... | ..... | ..... | ..... | ..... |

Se *p* fosse dichiarato puntatore a un dato di tipo carattere, verrebbe incrementato di un solo byte, quindi non punterebbe ad *a[1]*, ma ad un valore indefinito. Per questo motivo è molto importante controllare la coerenza di tipo tra puntatori e oggetti puntati.

Il seguente esempio serve ad illustrare quanto detto sopra sull'uso degli array e dei puntatori. Supponiamo di avere il seguente array e un puntatore a interi:

```
int v[7] = {10, 20, 30, 40, 50, 60, 70};
int* pInteri;
```

Con l'istruzione

```
pInteri = v;
```

*pInteri* punta al primo elemento dell'array.

Le componenti dell'array possono quindi essere visualizzate con il metodo utilizzato nei progetti precedenti:

```
for (int i=0; i<7; i++) {
 cout << i << '\t' << v[i] << endl;
}
```

Oppure con il seguente ciclo *for* che, sfruttando l'aritmetica dei puntatori, produce gli stessi risultati:

```
for (int i=0; i<7; i++) {
 cout << i << '\t' << *pInteri << endl;
 pInteri++;
}
```

Oltre a dati di tipo elementare, i puntatori possono puntare anche a dati di tipi definiti dall'utente; per esempio, si può dichiarare un puntatore per una struttura nel seguente modo:

```
struct data {
 int giorno;
 string mese;
 int anno;
};

data* p;
```

Nelle istruzioni del programma si può fare riferimento ai campi della struttura utilizzando l'**operatore ->** (operatore *freccia* formato da un segno - e un segno di >) come separatore tra il nome del puntatore e il nome del campo.

Con riferimento alla struttura dell'esempio precedente, per assegnare il valore 14 al campo *giorno*, si scrive:

```
data d;
p = &d;
p->giorno = 14;
```

La sola dichiarazione di un puntatore non crea l'area di memoria per i dati: è necessario puntare ad un'area di memoria.

Il seguente programma mostra un esempio per il corretto uso dei puntatori ad una struttura definita dal programmatore.

```
// PStruct.cpp: puntatori e strutture

#include <iostream>
#include <string>
using namespace std;

// dichiarazione della struttura
struct Merce {
 int codice;
 string descrizione;
 float prezzo;
 int reparto;
};

Merce articolo;
Merce* pArticolo;

int main()
{
 // inserisce i dati nella struttura
 cout << "Codice: ";
 cin >> articolo.codice;
 cout << "Descrizione: ";
 cin >> articolo.descrizione;
 cout << "Prezzo: ";
```

```
cin >> articolo.prezzo;
cout << "Reparto: ";
cin >> articolo.reparto;
pArticolo = &articolo;

// visualizza i dati della struttura usando il puntatore
cout << pArticolo->codice << endl;
cout << pArticolo->descrizione << endl;
cout << pArticolo->prezzo << endl;
cout << pArticolo->reparto << endl;

return 0;
}
```

Quando abbiamo incontrato gli array, nel Paragrafo 25, si è notato che il parametro formale di un vettore è un puntatore. Infatti, la dichiarazione di parametro:

```
int v[]
```

è equivalente a

```
int* v
```

La funzione che riceve un array, infatti, ha bisogno solamente di un puntatore per memorizzare l'indirizzo del primo elemento dell'array passato (&v[0]). Questo puntatore può poi essere utilizzato come se si trattasse dell'array passato, per esempio in un'istruzione di assegnazione come la seguente:

```
v[i] = n;
```

L'uso delle parentesi quadre che racchiudono l'indice *i* coincide con la memoria che si trova *i* posizioni a partire dall'indirizzo puntato da \*v, cioè

```
v[i]
```

è equivalente a

```
*(v+i).
```

Si osservi l'uso delle parentesi per effettuare prima l'operazione di somma.

Anche le parentesi quadre dei vettori possono quindi essere considerate parte dell'aritmetica dei puntatori.

#### AUTOVERIFICA

Domande da 39 a 42 pag. 301-302

Problemi da 59 a 62 pag. 306



MATERIALI ONLINE

## 9. Programmi eseguibili e videate dei progetti precedenti

## DOMANDE

### Dichiarazioni e assegnazioni

- 1 Completa la seguente tabella fornendo esempi di variabili del tipo indicato:

| Tipo   | Esempio di variabile |
|--------|----------------------|
| int    |                      |
| double |                      |
| char   |                      |

- 2 Completa la seguente tabella indicando il tipo più appropriato per ciascuno dei dati relativi a una persona:

| Dato                                     | Tipo |
|------------------------------------------|------|
| Cognome                                  |      |
| Nome                                     |      |
| Età                                      |      |
| Statura (espressa in metri e centimetri) |      |

- 3 Quali sono i valori minimo e massimo rappresentabili con i tipi numerici interi?

Minimo .....  
Massimo .....

- 4 Completa le frasi seguenti utilizzando una tra le parole elencate alla fine della domanda.

- a) Per rappresentare un numero di tipo float servono ..... bit.  
b) L'operazione di assegnazione si rappresenta nel linguaggio C++ con .....  
c) Le costanti vengono dichiarate attraverso la parola chiave .....  
d) L'istruzione `s = a + b;` si chiama .....

*var, cost, const, float, input, output, assegnazione, dichiarazione, 8, 16, 32, 64, =, \*, %*

- 5 Quale delle seguenti istruzioni è quella corretta per attribuire il risultato dell'espressione `n * m` alla variabile `p`?

- a) `n * m = p`  
b) `p << n * m`  
c) `p = n * m`  
d) `p >> n * m`

- 6 Inserisci le parole mancanti nel seguente frammento di programma:

```
_____ ()
{
 _____ a, b;
 cin >> a;
 b = a * 2;
 _____ b;
}
```

**7** Supponendo che la variabile *a* di tipo *float* assuma ogni volta il valore iniziale 5.0, quale valore assume alla fine di ogni gruppo di istruzioni?

- |                                                                                                     |                                                                                   |                                                                                                              |
|-----------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| a) <i>a</i> = <i>a</i> * 4;<br><i>a</i> = <i>a</i> * <i>a</i> ;<br><i>b</i> = - <i>a</i> ;<br>..... | b) <i>a</i> = <i>a</i> - 1;<br><i>b</i> = <i>a</i> / 2;<br><i>a</i> = 8;<br>..... | c) <i>b</i> = <i>a</i> + 2;<br><i>a</i> = ( <i>a</i> * <i>a</i> ) / 2;<br><i>a</i> = 1 / <i>a</i> ;<br>..... |
|-----------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|

**8** Supponendo che la variabile *a* di tipo intero assuma ogni volta il valore iniziale 3 e che la variabile *b* di tipo intero assuma il valore 0, quale valore assume ciascuna variabile alla fine di ogni gruppo di istruzioni?

- |                                                                                                                             |                                                                                                                             |                                                                                                   |
|-----------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|
| a) <i>a</i> = <i>a</i> + <i>b</i> ;<br><i>b</i> = ( <i>b</i> - 1) * <i>a</i> ;<br><i>c</i> = <i>b</i> - <i>a</i> ;<br>..... | b) <i>a</i> = <i>a</i> * <i>b</i> ;<br><i>c</i> = ( <i>a</i> + 1) * <i>b</i> ;<br><i>b</i> = <i>b</i> - <i>c</i> ;<br>..... | c) <i>b</i> = <i>b</i> + 1;<br><i>a</i> = <i>b</i> ;<br><i>c</i> = <i>a</i> + <i>b</i> ;<br>..... |
|-----------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|

**9** Qual è l'output dei seguenti programmi, supponendo che da tastiera venga assegnato ad *a* il valore 3 e a *b* il valore 5?

a)

```
int main()
{
 int a;
 cin >> a;
 a++;
 cout << a;
}
```

b)

```
int main()
{
 int a;
 cin >> a;
 a = a * a;
 cout << a;
}
```

c)

```
int main()
{
 int a, b, s;
 cin >> a >> b;
 s = a + b;
 cout << s;
}
```

**10** Associa ad ogni funzione predefinita della colonna di sinistra la sua descrizione scegliendo nella colonna a destra.

- |                               |                                                                   |
|-------------------------------|-------------------------------------------------------------------|
| a) pow( <i>x</i> , <i>y</i> ) | 1) Calcola il valore di <i>x</i> arrotondato all'intero superiore |
| b) sqrt( <i>x</i> )           | 2) Calcola la potenza <i>x</i> <sup><i>y</i></sup>                |
| c) ceil( <i>x</i> )           | 3) Arrotonda <i>x</i> all'intero inferiore o uguale               |
| d) floor( <i>x</i> )          | 4) Calcola la radice quadrata di <i>x</i>                         |

**11** Supponendo che *x* sia di tipo *int* e *y* di tipo *float*, quale delle seguenti istruzioni rappresenta un'operazione di *casting*?

- a) *y* = int *x*;  
b) (*y*) = int *x*;  
c) *y* = (int) *x*;  
d) (int) *y* = *x*;

**12** Quali tra i seguenti sono tipi utilizzabili per rappresentare i dati numerici non interi?

- a) long int  
b) float  
c) unsigned long int  
d) double  
e) long double



## Istruzioni di ingresso e uscita

**13** Completa le frasi seguenti utilizzando una tra le parole elencate alla fine della domanda.

- a) Per acquisire un dato si usa l'istruzione .....
- b) Per comunicare un risultato si usa l'istruzione .....
- c) La tastiera rappresenta lo standard .....
- d) Il video rappresenta lo standard .....

***include, input, output, int, main, float, double, cin >>, cout <<***

**14** Quali delle seguenti affermazioni sono vere (V) e quali false (F)?

- a) L'istruzione `cout<<a;` scrive il carattere 'a' sul video
- b) L'istruzione `cout<<a;` scrive il contenuto della variabile *a* sul video
- c) L'istruzione `cout<<a<<endl;` scrive il contenuto della variabile *a* e il cursore resta sulla medesima riga
- d) L'istruzione `cout<<a<<endl;` scrive il contenuto della variabile *a* e il cursore si sposta sulla riga successiva



## Programmazione e documentazione

**15** Associa a ciascun tipo di errore della colonna di sinistra la definizione corrispondente scegliendo dalla colonna a destra.

- |               |                                                                                                                 |
|---------------|-----------------------------------------------------------------------------------------------------------------|
| a) sintattico | 1) viene segnalato durante l'esecuzione del programma e blocca l'avanzamento nel processo di elaborazione       |
| b) lessicale  | 2) non sono state rispettate le regole per la costruzione corretta delle frasi del linguaggio di programmazione |
| c) run-time   | 3) è stata utilizzata una parola che non appartiene al linguaggio di programmazione                             |

**16** Completa le frasi seguenti utilizzando una tra le parole elencate alla fine della domanda.

- a) La codifica produce il programma .....
- b) Il linking produce il programma .....
- c) La compilazione produce il programma .....
- d) Un dato che cambia valore durante l'esecuzione del programma si dice .....
- e) Un dato non numerico, formato da più caratteri, si chiama .....

***sorgente, istruzione, eseguibile, utente, statement, oggetto, identificatore, variabile, stringa, costante***

**17** Quale tra le seguenti attività ha come obiettivi far comprendere l'analisi del procedimento e l'algoritmo risolutivo a una persona diversa da quella che li ha progettati e riuscire a capire il programma anche a distanza di tempo dal momento in cui il progetto è stato realizzato?

- a) Programmazione; b) Documentazione; c) Minutazione; d) Codifica

## Struttura di selezione

**18** Qual è l'istruzione inutile che si può togliere dal seguente frammento di codice per renderlo più efficiente?

```
if (a==0) cout << "valore nullo";
else
if (a!=0) cout << "valore diverso da zero";
```

**19** Qual è l'output prodotto dall'esecuzione dei due segmenti di codice a) e b) ?

a)

```
a = 5;
b = 3;
if (a > b) c = a - b;
else c = b - a;
cout << c;
```

b)

```
a = 0;
b = 10;
if (a < b)
 cout << a << endl << b << endl;
else
 cout << b << endl << a << endl;
```

**20** Che cosa viene scritto in output dall'esecuzione del seguente segmento di programma se x vale 2 e se x vale 9?

```
if (x > 5) {
 if (x > 7) x = 6;
}
else x = 0;
cout << x << endl;
```

**21** Che cosa viene scritto in output dall'esecuzione del seguente segmento di programma se x vale 2 e se x vale 9?

```
if (x > 5)
 if (x > 7) x = 6;
else x = 0;
cout << x << endl;
```

## Struttura di ripetizione

**22** Che cosa viene scritto in output dall'esecuzione del seguente segmento di programma?

```
a = 3;
do {
 a *= 2;
 cout << a << endl;
} while (a<=30);
```

**23** Che cosa viene scritto in output dall'esecuzione del seguente segmento di programma?

```
a = 3;
do {
 a *= 2;
 cout << a << endl;
} while (a>30);
```

**24** Che cosa viene scritto in output dall'esecuzione del seguente segmento di programma?

```
a = 1;
while (a<=10) {
 cout << a << endl;
 a++;
}
```

**25** Che cosa viene scritto in output all'esecuzione di questo segmento di programma?

```
a = 1;
while (a<=10) a++;
cout << a << endl;
```

## Funzioni

**26** Quali delle seguenti affermazioni sono vere (V) e quali false (F)?

- a) I parametri formali sono indicati nella riga di intestazione della funzione
- b) I parametri formali sono indicati quando viene richiamata la funzione
- c) I parametri attuali sono indicati nella riga di intestazione della funzione
- d) I parametri attuali sono indicati quando viene richiamata la funzione



**27** Quale valore scrivono con l'istruzione di output i seguenti tre programmi?

a)

```
#include <iostream>
using namespace std;
void Modifica (int y)
{
 y += 5;
}
int main()
{
 int x;
 x = 1;
 Modifica(x);
 cout << x << endl;
 return 0;
}
```

b)

```
#include <iostream>
using namespace std;
void Modifica (int& y)
{
 y += 5;
}
int main()
{
 int x;
 x = 1;
 Modifica(x);
 cout << x << endl;
 return 0;
}
```

c)

```
#include <iostream>
using namespace std;
void Modifica (int x, int& y)
{
 x--;
 y += 5;
 return;
}
int main()
{
 int a, b;
 a = 10;
 b = 20;
 Modifica(a, b);
 cout << a << " " << b << endl;
 return 0;
}
```

**28** Perché il seguente programma produce un errore di compilazione riferito alla funzione *CalcoloArea*?

```
#include <iostream>
using namespace std;
float CalcoloArea (float l);
// funzione principale
int main(void)
{
 float lato;
 lato = 10.0;
 CalcoloArea(lato);
 return 0;
}
void CalcoloArea (float l)
{
 float area;
 area = l * l;
 cout << area << endl;
}
```

**29** Quali delle seguenti affermazioni sono vere (V) e quali false (F)?

- a) Le variabili che vengono dichiarate all'inizio di un programma, e fuori dal *main()*, sono dette variabili locali
- b) In qualsiasi punto del programma si può utilizzare una variabile globale
- c) Una variabile locale a una funzione può essere utilizzata dalla funzione *main*
- d) In un programma ben strutturato è conveniente usare molte variabili globali



## Funzioni predefinite e overloading

- 30** Quale riga di codice occorre introdurre nel programma per utilizzare la funzione matematica predefinita *pow* per il calcolo della potenza?
- `#include cmath`
  - `#include pow`
  - `#include <cmath>`
  - `#include <pow>`
- 31** Associa ad ogni funzione della colonna di sinistra il corrispondente significato nella colonna di destra.
- |                      |                         |
|----------------------|-------------------------|
| a) <code>sqrt</code> | 1) elevamento a potenza |
| b) <code>pow</code>  | 2) radice quadrata      |
| c) <code>exp</code>  | 3) esponenziale         |
- 32** Quale delle seguenti frasi corrisponde alla definizione di *function overloading*?
- Due funzioni che differiscono per il nome ma hanno gli stessi parametri.
  - Due funzioni con lo stesso nome che differiscono tra loro solamente per il numero dei parametri passati.
  - Due funzioni con lo stesso nome che differiscono tra loro solamente per il tipo dei parametri passati.
  - Due funzioni con lo stesso nome che differiscono tra loro per il numero dei parametri passati o per il loro tipo o per entrambi.

## Ricorsione

- 33** Quale dei seguenti frammenti di codice rappresenta correttamente il calcolo ricorsivo del fattoriale di un numero?

a)

```
long Fatt(int x)
{
 long f;
 if (x==0) f = 1;
 else Fatt = x * f(x-1);
 return f;
}
```

b)

```
long Fatt(int x)
{
 long f;
 if (x==0) f = 1;
 else f = x * Fatt(x-1);
 return f;
}
```

c)

```
long Fatt(int x)
{
 long f;
 if (x==0) f = 1;
 else f = x * f(x-1);
 return f;
}
```

## Enumerazioni

- 34** Date le seguenti dichiarazioni

```
enum Giorni {LUN, MAR, MER, GIO, VEN, SAB, DOM};
Giorni i;
```

e le istruzioni

```
for(i=LUN;i<=VEN;i++) {
 Lavora();
}
for(i=SAB;i<=DOM;i++) {
 Riposa();
}
```

quante volte viene eseguita la procedura *Lavora* e quante la procedura *Riposa*?

35 Date le seguenti dichiarazioni

```
enum Colori {BIANCO, ROSSO, VERDE, ARANCIONE, VIOLA, GIALLO, BLU, MAGENTA};
Colori a, b, c;
```

Che cosa contengono le variabili a, b, c dopo le seguenti assegnazioni?

```
a = ARANCIONE;
b = VIOLA;
c = BIANCO;
```

## Array

36 Che cosa occorre specificare per indicare un elemento di un array?

- a) Un indice scritto in basso a destra rispetto al nome dell'array.
- b) Un indice racchiuso tra parentesi graffe.
- c) Un indice racchiuso tra parentesi quadre.
- d) Un indice formato da una sola cifra.

37 Quale tra le seguenti dichiarazioni è corretta per inizializzare un array?

- a) `double valori = {3.5, 0.2, 7.0};`
- b) `double valori[2] = {3.5, 0.2, 7.0};`
- c) `double valori[3] = {3.5, 0.2, 7.0};`
- d) `double valori[3] {3.5, 0.2, 7.0};`

38 Quale tra le seguenti affermazioni, riferite al passaggio di array a una funzione, è corretta?

- a) Per effettuare il passaggio per referenza di un array alla funzione, basta indicare il nome dell'array nella chiamata della funzione.
- b) Per effettuare il passaggio per referenza di un array alla funzione, occorre aggiungere il carattere & al tipo dell'array nell'intestazione della funzione.
- c) Per effettuare il passaggio per referenza di un array alla funzione, occorre aggiungere il carattere & al nome dell'array nella chiamata alla funzione.
- d) Per effettuare il passaggio per referenza di un array alla funzione, occorre aggiungere il carattere & alla prima componente dell'array nell'intestazione della funzione.

## Puntatori

39 Data l'istruzione

```
*pInteri = 2;
```

che cosa contiene il puntatore *pInteri*?

- a) Il valore presente nella cella di memoria numero 2.
- b) Il valore intero 2.
- c) L'indirizzo di una cella di memoria che contiene il valore intero 2.
- d) L'istruzione presenta un errore di sintassi perché *\*pInteri* non può essere usato in un'istruzione di assegnazione.

40 Quale delle seguenti affermazioni riferite all'istruzione

```
x = *p;
```

è corretta?

- a) Assegna a x il contenuto della memoria puntata da p.
- b) Assegna a p il contenuto della memoria puntata da x.
- c) Assegna a x il prodotto di x per p.
- d) Assegna a p il prodotto di x per p.

**41** Quale delle seguenti affermazioni riferite all'istruzione

```
p = &x;
```

è corretta?

- a) Assegna a x l'indirizzo della variabile p.
- b) Assegna a p l'indirizzo della variabile x.
- c) Assegna a p il contenuto della variabile x.
- d) Assegna a x il contenuto della variabile p.

**42** Data la definizione

```
struct Nodo {
 string nome;
 Nodo *succ;
};
Nodo* n, p, q;
```

dopo le istruzioni:

```
p = q;
q = n;
q->succ = p;
```

p e q puntano allo stesso nodo?

## PROBLEMI

### Programmi in C++

- 1** Calcolare il triplo di un numero fornito da tastiera.
- 2** Dato il lato, trovare il perimetro e l'area di un esagono regolare.
- 3** Dato il raggio, calcolare la circonferenza e l'area del cerchio.
- 4** Scrivere il programma per scambiare il contenuto di due variabili (occorre usare una terza variabile temporanea).
- 5** Calcolare l'area di un triangolo, noti i lati (usando la formula di Erone).
- 6** Dato il peso di tre oggetti, calcolare il peso medio.

### Programmazione e documentazione

- 7** Scrivere il seguente programma e determinare in quali casi si ha un errore di *run-time* nel calcolo della radice quadrata.

```
int main()
{
 int x;
 float y;
 cin >> x;
 y = sqrt(x);
 cout << y;
}
```

- 8 Scrivere il seguente programma e individuare le istruzioni che non sono corrette nel lessico o nella sintassi.

```
#includere <iostream>
int main()
{
 /* input
 int i;
 cout << Introduci un numero
 cin >> i;
 i*=2;
 scrivi << i;
 }
```

- 9 Scrivere un programma che richieda in input le coordinate di due punti e scriva le coordinate del punto medio del segmento che unisce i due punti. Effettuare poi il debugging del programma.
- 10 Scrivere un programma che richieda in input le coordinate di due punti e che scriva le coordinate dei punti simmetrici rispetto all'asse x. Effettuare poi il debugging del programma.
- 11 Dato il prezzo di un prodotto e la percentuale di sconto, calcolare il prezzo scontato. Scrivere il programma usando le regole per un'efficace documentazione.
- 12 Dato in input il valore di un deposito bancario e il tasso di interesse annuo, calcolare gli interessi maturati dopo 25 giorni. Scrivere il programma usando le regole per un'efficace documentazione.

### Struttura di sequenza e selezione

- 13 Scrivere il programma che, ricevute in input le misure delle diagonali di un rombo, calcoli il perimetro e l'area.
- 14 Scrivere il programma che, letto un numero intero, calcoli e scriva la sua metà.
- 15 Dato il raggio del cerchio di base di un cilindro e la sua altezza, calcolare il peso del cilindro sapendo che la sostanza di cui è composto ha peso specifico = 3.4 g/cm<sup>3</sup> (Volume = area della base · altezza : 3; Peso = Volume · Peso specifico).
- 16 Data la parabola di equazione  $y = ax^2 + bx + c$ , determinare le coordinate del vertice V (xv, yv), ricordando che la sua ascissa si ottiene con la formula  $xv = -b / (2 \cdot a)$ .
- 17 Una nuova compagnia telefonica ha promosso l'offerta "oltre80": alla cifra fissa di 0,10 euro (costo alla risposta) occorre aggiungere la cifra di 0,15 euro per ogni secondo del tempo della telefonata; però oltre gli 80 secondi la tariffa per ogni secondo è di 0,09. Fornito da tastiera il numero dei secondi della telefonata, visualizzare il costo totale della chiamata. (Le tariffe indicate nel testo del problema possono diventare più opportunamente costanti).
- 18 Le spese di spedizione di pacchi variano a seconda del peso e precisamente: fino a 3 kg, il costo è di 5 euro; dopo i 3 kg e fino a 10 kg, il costo è di 8 euro, oltre i 10 kg, il costo è di 10 euro. Dato il peso del pacco calcolare la spesa di spedizione.

### Struttura di ripetizione

- 19 Costruire un programma che accetta da tastiera dei numeri finché viene inserito il valore 0; man mano li visualizza.
- 20 Costruire un programma che accetta da tastiera dei nomi e li conta. L'inserimento si interrompe quando il numero dei nomi inseriti diventa 10.

- 21 Dato un elenco di nomi di persone con i comuni di residenza, contare quanti sono quelli di un comune fornito da tastiera.
- 22 Dato un elenco di prodotti con i rispettivi prezzi, contare quanti sono quelli che hanno il prezzo superiore a 100 euro.
- 23 Dato un elenco di prodotti con descrizione e prezzo, calcolare l'incremento del prezzo secondo una percentuale fornita da tastiera. Comunicare per ciascun prodotto la descrizione e il prezzo incrementato.
- 24 Calcolare il consumo medio di carburante di un elenco di veicoli utilizzando come dati di input i valori dei km percorsi e dei litri di carburante per ciascun veicolo.
- 25 Riscrivere il seguente frammento di codice utilizzando l'istruzione *for*:

```
a = 1;
while (a<=10) {
 cout << a << endl;
 a++;
}
```

- 26 Riscrivere il seguente frammento di codice utilizzando l'istruzione *for*:

```
a = 1;
while (a<=10) a++;
cout << a << endl;
```

- 27 Costruire un programma che accetta da tastiera un elenco di N distanze delle tappe di un percorso e le somma. Alla fine fornire la lunghezza totale del percorso.
- 28 Dato un elenco di 20 studenti della prima classe della scuola superiore, con nome e voto finale della scuola media, calcolare il valore medio dei voti.
- 29 Dati N studenti partecipanti a una gara sportiva di velocità (nome studente, categoria, tempo), visualizzare il tempo dello studente vincitore (tempo minimo) tra gli studenti della categoria "allievi".
- 30 Dati N modelli di ciclomotori e motocicli (nome del modello, cilindrata, prezzo), visualizzare il prezzo massimo tra i modelli di cilindrata superiore a 50 cc.

## Funzioni

- 31 Definire una funzione che ripeta sul video in tre righe diverse la stringa acquisita da tastiera.
- 32 Organizzare una funzione che acquisisca da tastiera il lato di un quadrato e ne calcoli la misura dell'area e del perimetro.
- 33 Data la parabola  $y = ax^2 + bx + c$ , definire tre funzioni per calcolarne i punti significativi: vertice, fuoco, intersezione con gli assi. Le tre funzioni ricevono come parametri i coefficienti  $a$ ,  $b$ ,  $c$ , e restituiscono il valore calcolato.
- 34 Definire una funzione che abbia come parametro un numero intero positivo e che visualizzi la sequenza dei primi 5 numeri successivi al numero dato.
- 35 Scrivere una funzione che abbia come parametri il numero dei valori e la loro somma e che calcoli la media aritmetica.
- 36 Dati in input le età di tre persone, scrivere i dati acquisiti in ordine decrescente.



- 37** Scrivere la funzione booleana che controlli se una persona può votare, conoscendo il suo anno di nascita (si consideri l'anno attuale come costante).
- 38** Scrivere una funzione che abbia come parametri il prezzo di vendita e la percentuale di sconto e che restituisca lo sconto.
- 39** Scrivere una funzione che restituisca un valore booleano Vero o Falso, controllando se due rette, di cui si conoscono le equazioni  $y = m_1 x + q_1$  e  $y = m_2 x + q_2$ , sono parallele.
- 40** Data la retta  $y = m x + q$  e il punto P di coordinate (xp, yp), controllare con una funzione booleana se il punto appartiene alla retta.
- 41** Dato un elenco di studenti con l'indicazione del nome e voto, calcolare il voto medio. Definire una funzione per l'acquisizione e la somma dei voti e una funzione per il calcolo della media.
- 42** Dato un elenco di prodotti con descrizione e prezzo, tramite una funzione viene calcolato l'incremento del prezzo secondo una percentuale fornita da tastiera. Comunicare per ciascun prodotto la descrizione e il prezzo incrementato.
- 43** Dopo aver acquisito da tastiera il testo delle due risposte di un sondaggio e il numero di preferenze per ciascuna, calcolare la percentuale delle preferenze sul totale e scrivere le risposte in ordine decrescente di percentuale.

### Funzioni predefinite

- 44** Scrivere un programma per acquisire in input le coordinate di due punti del piano cartesiano e per calcolare la distanza tra i due punti.
- 45** Scrivere il programma che calcola l'esponenziale, il logaritmo naturale, in base e, e il logaritmo in base 10 di un numero fornito da tastiera.
- 46** Scrivere il programma che calcola il seno, il coseno e la tangente di un angolo fornito da tastiera.

*(I problemi 47 e 48 richiedono la codifica degli algoritmi progettati per gli stessi problemi nel Capitolo 4)*

### Ricorsione

- 47** Calcolare il prodotto di due numeri interi positivi a, b utilizzando la sola operazione di somma e sfruttando la seguente definizione ricorsiva:

$$\begin{aligned} a*b &= 0 && \text{se } b = 0 \\ a*b &= a*(b-1)+a && \text{se } b > 0 \end{aligned}$$

- 48** Calcolare la somma di tutti i numeri che vanno da 1 a n utilizzando la definizione ricorsiva

$$\begin{aligned} \sum_{i=1}^1 i &= 1 && \sum_{i=1}^n i = n + \sum_{i=1}^{n-1} i \end{aligned}$$

### Enumerazioni

- 49** Definire un'enumerazione per contenere i nomi dei mesi.
- 50** Con riferimento al problema precedente, indicare i valori corrispondenti ai mesi di vacanza.
- 51** Definire un'enumerazione per contenere i colori del semaforo.

## Array

- 52** Dopo aver caricato in memoria un array di numeri interi con 10 componenti, calcolare il prodotto delle componenti.
- 53** Dopo aver caricato in memoria un array di numeri interi con 10 componenti, contare le componenti che hanno valore superiore a 5.
- 54** Dopo aver caricato in memoria un array di numeri interi con 10 componenti, raddoppiare il valore delle prime tre componenti.

## Strutture

- 55** Data la seguente definizione di dati

```
struct Persona {
 int matricola;
 string nome;
 string indirizzo;
 string DataNascita;
 string CodiceFiscale;
 bool coniugato;
};
Persona dipendente;
```

modificare la struttura *Persona* per rappresentare la data di nascita come una struttura formata da giorno, mese, anno.

- 56** Codificare la struttura che possa contenere i dati relativi al docente di una scuola.
- 57** Codificare la struttura che possa contenere i dati relativi al giocatore tesserato in una squadra di calcio.
- 58** Rappresentare con una struttura le informazioni di una città: nome, numero di abitanti attuale e numero di abitanti dello scorso anno. Calcolare poi l'incremento (o decremento) di abitanti.

## Puntatori

- 59** Dichiarare due variabili intere e due puntatori a interi. Dopo aver assegnato due valori alle variabili e gli indirizzi delle due variabili ai due puntatori, scrivere le istruzioni che permettono di visualizzare i valori delle variabili, gli indirizzi delle variabili, i valori dei puntatori e gli indirizzi dei puntatori.
- 60** Scrivere la funzione che, attraverso il passaggio di parametro per indirizzo, permette di aggiornare il prezzo di un prodotto incrementandolo del 3%.
- 61** Dato un array di 10 numeri interi e un puntatore ad esso, dopo aver acquisito da tastiera i valori delle sue componenti, scrivere il programma che visualizza i valori utilizzando l'aritmetica dei puntatori.
- 62** Data una struttura contenente i dati di un ordine per un prodotto (numero dell'ordine, codice prodotto, quantità, prezzo unitario), scrivere il programma che, dopo aver acquisito i dati, visualizza codice del prodotto e prezzo totale usando un puntatore alla struttura.

## PROBLEMI DI RIEPILOGO

### Strutture di controllo

- 63** Un grande magazzino ha 4 reparti, rappresentati con i numeri 1, 2, 3, 4. La Direzione decide di applicare sui prodotti di ciascun reparto sconti con percentuali differenziate e precisamente: reparto 1 = sconto 0% (nessuno sconto), reparto 2 = sconto 3%, reparto 3 = sconto 2%, reparto 4 = sconto 5%. Dati N prodotti con reparto di appartenenza e prezzo, si vuole calcolare e visualizzare per ciascuno il prezzo scontato (*utilizzare una struttura di scelta multipla*).
- 64** Dato un elenco di computer, di cui non si conosce a priori il numero, con descrizione, tipo e prezzo in euro, visualizzare i dati dei computer appartenenti a un tipo prefissato e fornito all'inizio da tastiera e alla fine anche il prezzo medio dei computer selezionati.
- 65** Dato un elenco di oggetti con il volume e il peso, calcolare il peso specifico (peso/volume) di ciascun oggetto e visualizzare il volume, il peso e il peso specifico calcolato. Alla fine visualizzare anche il numero di oggetti esaminati.

### Funzioni

- 66** Scrivere il programma per calcolare l'ipotenusa, il perimetro e l'area di un triangolo rettangolo, dopo aver acquisito da tastiera le misure dei due cateti. Organizzare il programma utilizzando le funzioni e individuando quali variabili possono essere dichiarate come locali delle funzioni.
- 67** Dato un elenco di N oggetti cilindrici: diametro di base, altezza, peso specifico della sostanza del cilindro. Determinare il peso degli oggetti aventi diametro superiore a un valore prefissato.
- 68** In un aeroporto sono stati rilevati il numero complessivo di passeggeri diretti verso ciascuna delle città che sono destinazione dei voli. Visualizzare il nome delle città con il numero di passeggeri superiore a un valore fornito da tastiera. Visualizzare anche il valore massimo tra i dati inseriti.
- 69** L'importo della tassa di possesso di un veicolo si calcola moltiplicando la potenza espressa in kW (*kiloWatt*), con eliminazione dei decimali, per la tariffa unitaria in euro. La tariffa unitaria è diversa tra automobili e motocicli (per esempio, auto: 2,58€ per kW; moto: 19,11€ per kW). Inoltre i veicoli elettrici sono esentati dal pagamento (importo = 0). Dati N veicoli identificati con la targa, calcolare per ciascuno l'importo della tassa e anche l'importo medio tra tutti i veicoli.
- 70** Dati due nomi di città con il numero di abitanti di ciascuna, scrivere le città prima in ordine alfabetico e poi in ordine crescente di abitanti.
- 71** Di una persona si inseriscono il cognome, il nome e l'anno di nascita. Il programma crea automaticamente un codice identificativo della persona prendendo i primi due caratteri del cognome, i primi due del nome e le cifre dell'anno di nascita.

## SCHEDA DI AUTOVALUTAZIONE

### CONOSCENZE

- ☐ struttura generale di un programma C++
- ☐ tipi di dati numerici e non numerici
- ☐ dichiarazione delle costanti e delle variabili
- ☐ istruzione di assegnazione e operatori
- ☐ casting
- ☐ istruzioni di I/O
- ☐ fasi del lavoro di programmazione
- ☐ errori sintattici, lessicali, di run-time e logici
- ☐ importanza della documentazione
- ☐ codifiche delle tre strutture fondamentali: sequenza, alternativa, ripetizione
- ☐ strutture annidate di alternativa
- ☐ ripetizione precondizionale e ripetizione con contatore
- ☐ struttura di scelta multipla
- ☐ funzioni
- ☐ funzioni con parametri
- ☐ passaggio di parametri per referenza e per valore
- ☐ dichiarazione dei prototipi di funzione
- ☐ definizione di risorse locali e globali
- ☐ regole di visibilità
- ☐ funzioni predefinite del linguaggio
- ☐ namespace e librerie di inclusione
- ☐ function overloading
- ☐ funzioni ricorsive
- ☐ enumerazioni
- ☐ array
- ☐ array a due dimensioni
- ☐ strutture
- ☐ puntatori

### ABILITÀ

- ☐ scrivere un programma C++ sintatticamente corretto
- ☐ saper scegliere il tipo di dato adatto a rappresentare le variabili
- ☐ validare un programma
- ☐ produrre programmi documentati
- ☐ individuare le strutture di controllo più idonee per la soluzione di un problema
- ☐ rappresentare la selezione
- ☐ annidare strutture di controllo
- ☐ esaminare un elenco di dati
- ☐ rappresentare le strutture derivate di ripetizione
- ☐ calcolare il valore massimo tra un insieme di valori
- ☐ utilizzare la struttura di scelta multipla
- ☐ sviluppare un programma introducendo le funzioni
- ☐ utilizzare il passaggio di parametri per referenza e per valore
- ☐ dichiarare le funzioni con i prototipi
- ☐ individuare le applicazioni pratiche delle regole di visibilità
- ☐ utilizzare le funzioni predefinite
- ☐ ridefinire una funzione con overloading
- ☐ individuare alcuni casi semplici di utilizzo di funzioni ricorsive
- ☐ creare un'enumerazione
- ☐ organizzare i dati in array a una o due dimensioni
- ☐ organizzare i dati in strutture
- ☐ utilizzare i puntatori



**SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 662**

## Ambienti di sviluppo in C++

Nel seguito vengono presentate due esempi di ambienti di sviluppo software per la programmazione nel linguaggio C++.

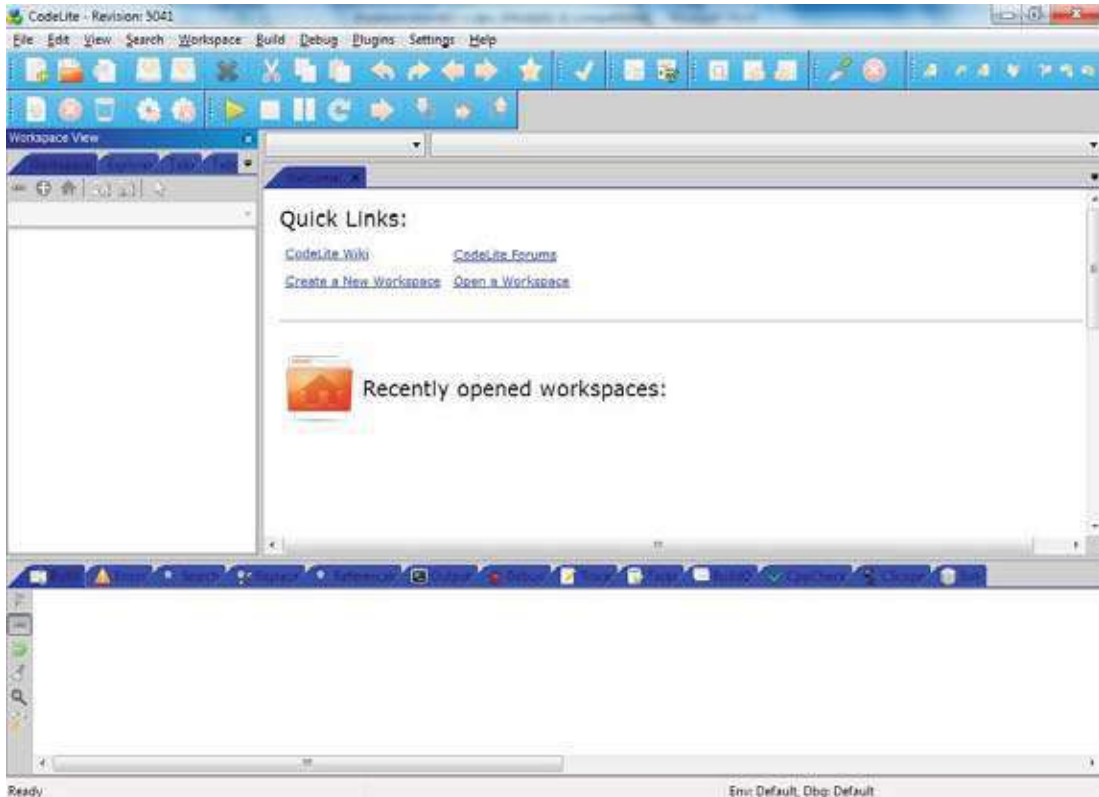
### CodeLite

**CodeLite** è un ambiente di sviluppo integrato (**IDE**, *Integrated Development Environment*) per la programmazione nei linguaggi C e C++. Comprende gli strumenti per lo sviluppo di software: editor, compilatore, linker, librerie, ecc. Può essere installato sui sistemi operativi Windows, Linux, Mac OSX.

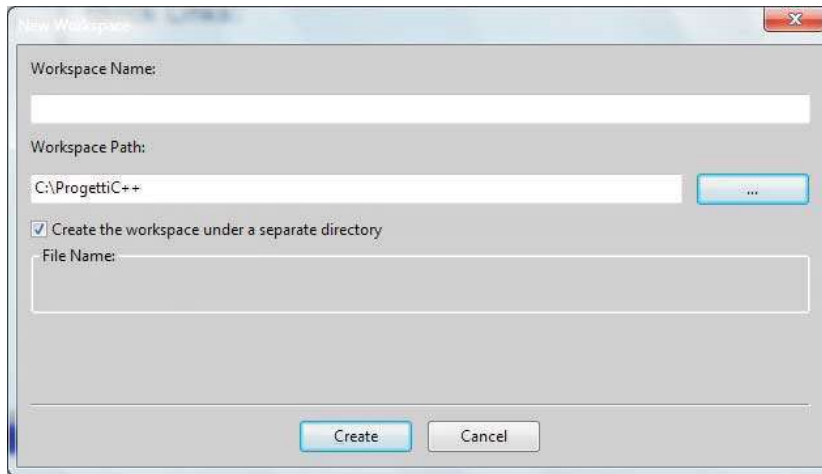
Per la presentazione delle note operative, nel seguito si farà riferimento a un'installazione per Windows.

CodeLite è una piattaforma **open source**, utilizzabile liberamente e distribuita sotto licenza **GPL** (*General Public License*), cioè la licenza per il software libero.

All'avvio del programma, si presenta la seguente videata, con la barra dei menu, la barra degli strumenti, il riquadro a sinistra per i *Workspace* e per la navigazione nelle cartelle del disco, la pagina di benvenuto al centro con i link rapidi e il riquadro inferiore per l'output e i messaggi di errore.

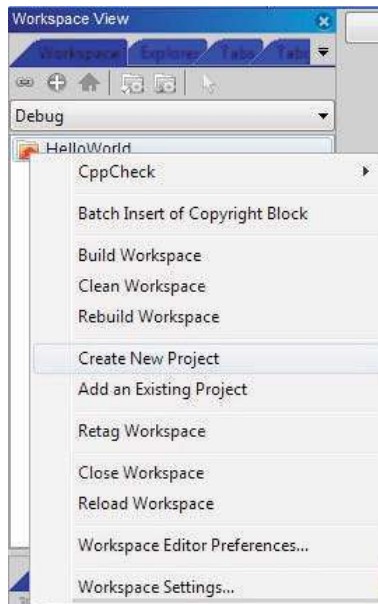


Per creare un nuovo *Workspace*, si deve fare clic sul link **Create a New Workspace**, oppure scegliere **New Workspace** dal menu **Workspace**. Se si vuole invece usare un *workspace* creato in precedenza, occorre scegliere **Open a Workspace**.



Viene richiesto il **nome** e il **percorso** con cui salvare il *workspace* e si può decidere se creare una nuova directory per il *workspace*. Si deve poi fare clic su **Create**.

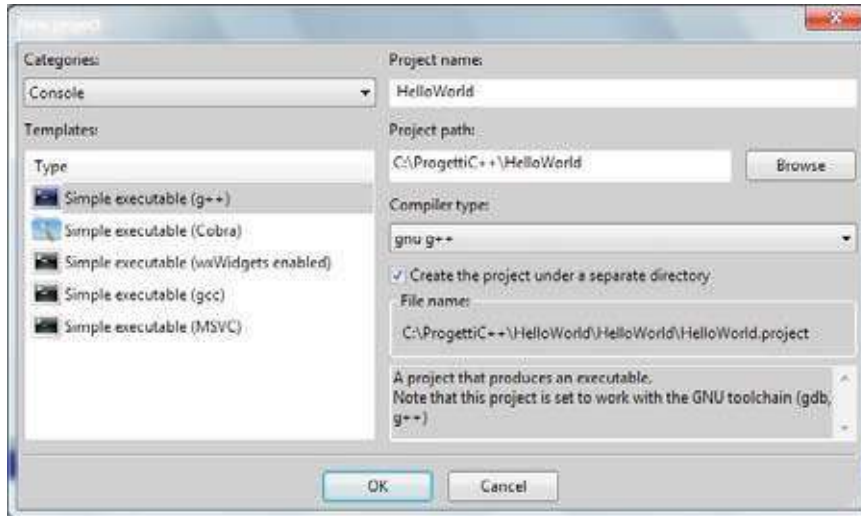
Viene creato un *workspace* vuoto all'interno del quale dobbiamo inserire uno o più **project**. Per creare un nuovo progetto, si deve fare clic con il tasto destro del mouse sul nome del *workspace* nel riquadro a sinistra e scegliere **Create New Project** (oppure selezionare **New Project** nel menu **Workspace**).



Attraverso una finestra di dialogo, per il nuovo progetto si deve scegliere il nome, il percorso e la categoria.

La categoria viene selezionata dalla casella combinata **Categories** in alto a sinistra.

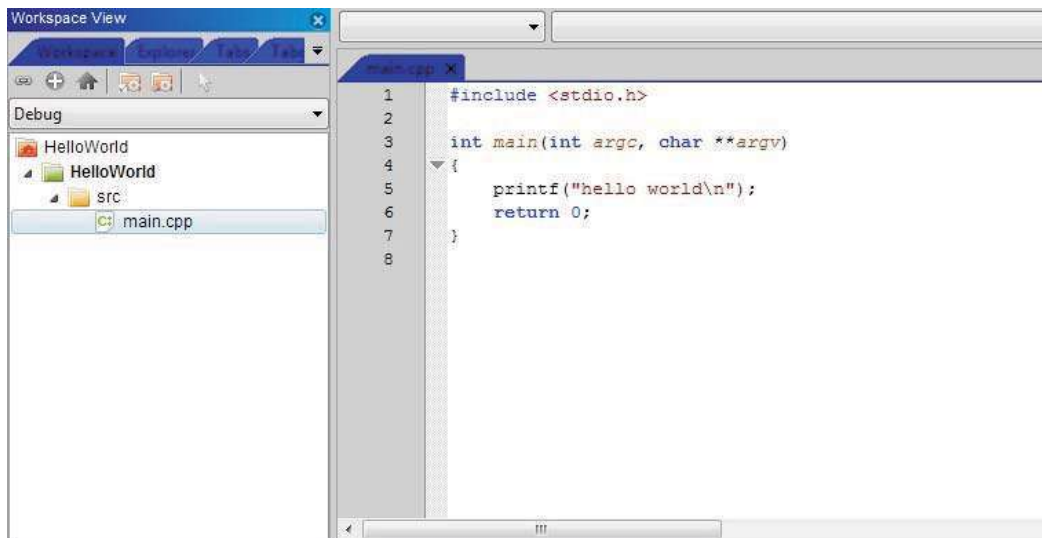
Per creare applicazioni in linguaggio C++ che producano programmi eseguibili (**.exe**) e utilizzabili direttamente dalla linea comandi del sistema operativo, occorre scegliere la categoria **Console** e, nei modelli (*Templates*) il tipo **Simple executable (g++)**. Confermare poi con un clic su **OK**.



Nel riquadro a sinistra (scheda *Workspace*) viene creato lo scheletro del progetto con il programma sorgente **main.cpp**. Il nome del file sorgente può essere cambiato (tasto destro sul nome, **Rename**).

Facendo doppio clic sul nome del file *.cpp*, nel riquadro centrale si apre la finestra di **editor** del codice.

In essa si possono aggiungere nuove istruzioni a quelle predefinite, oppure si può riscrivere il nuovo codice C++ dall'inizio.



Per eseguire il **debug** del programma, occorre fare clic sull'icona **Start Debugger** (icona con la freccia verde) nella barra degli strumenti, oppure si può scegliere **Start Debugger** dal menu **Debug** (la scorciatoia è il tasto **F5**).

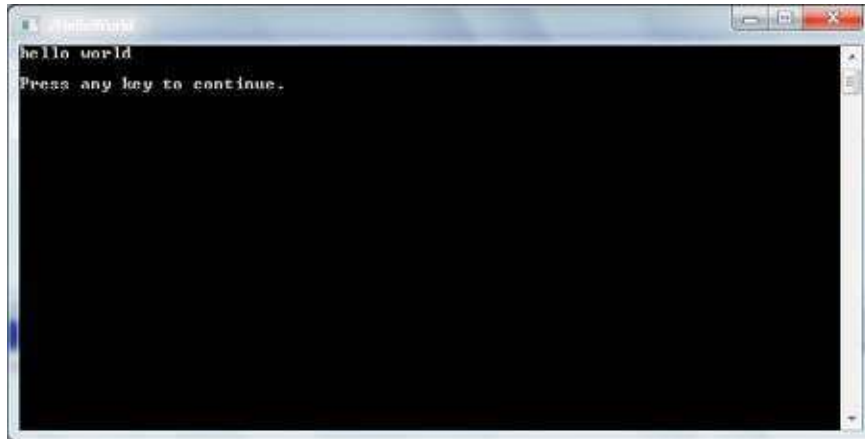
Il programma viene compilato: nella parte inferiore dello schermo (scheda **Debug**) vengono visualizzati i messaggi riguardanti le fasi della compilazione, del linking e della creazione dell'eseguibile (scheda **Build**). Se non ci sono errori, viene avviata una prova di esecuzione del programma.

Gli errori di compilazione, che impediscono la creazione del programma oggetto, sono segnalati nella scheda **Errors**.

Per creare l'eseguibile del programma, si deve fare clic sull'icona **Build Active Project** della barra degli strumenti, oppure scegliere **Build project** nel menu **Build** (la scorciatoia è il tasto **F7**). Il file **.exe** viene automaticamente salvato nella sottodirectory *Debug* della directory del *workspace*.

Se tutto è corretto, si può avviare l'esecuzione del programma: fare clic sull'icona **Run Active Project** nella barra degli strumenti, oppure scegliere **Run** nel menu **Build**. La scorciatoia da tastiera è la combinazione **Ctrl + F5**.

Viene aperta la finestra del *Prompt dei comandi* e l'esecuzione rimane in attesa che l'utente prema un tasto per uscire.



Per salvare il lavoro, si deve fare clic sull'icona **Save** della barra degli strumenti, oppure scegliere **Save all files** nel menu **File**.

Nell'individuazione ed eliminazione degli errori logici di un programma, si utilizza il debugger in *modo interattivo* e le funzionalità presenti nel menu **Debug**: impostazione di punti di interruzione (**Breakpoints**), esecuzione passo passo del programma (**Next**), debug anche nelle istruzioni all'interno delle funzioni (**Step Into**). Le funzionalità sono attivabili anche con le icone della barra degli strumenti oppure con le scorciatoie da tastiera (indicate accanto alle voci del menu *Debug*).

Per impostare (e disabilitare) velocemente un punto di interruzione su un'istruzione, basta fare clic con il mouse a sinistra della riga dell'istruzione (la scorciatoia è il tasto **F9**): viene visualizzato un cerchio rosso.





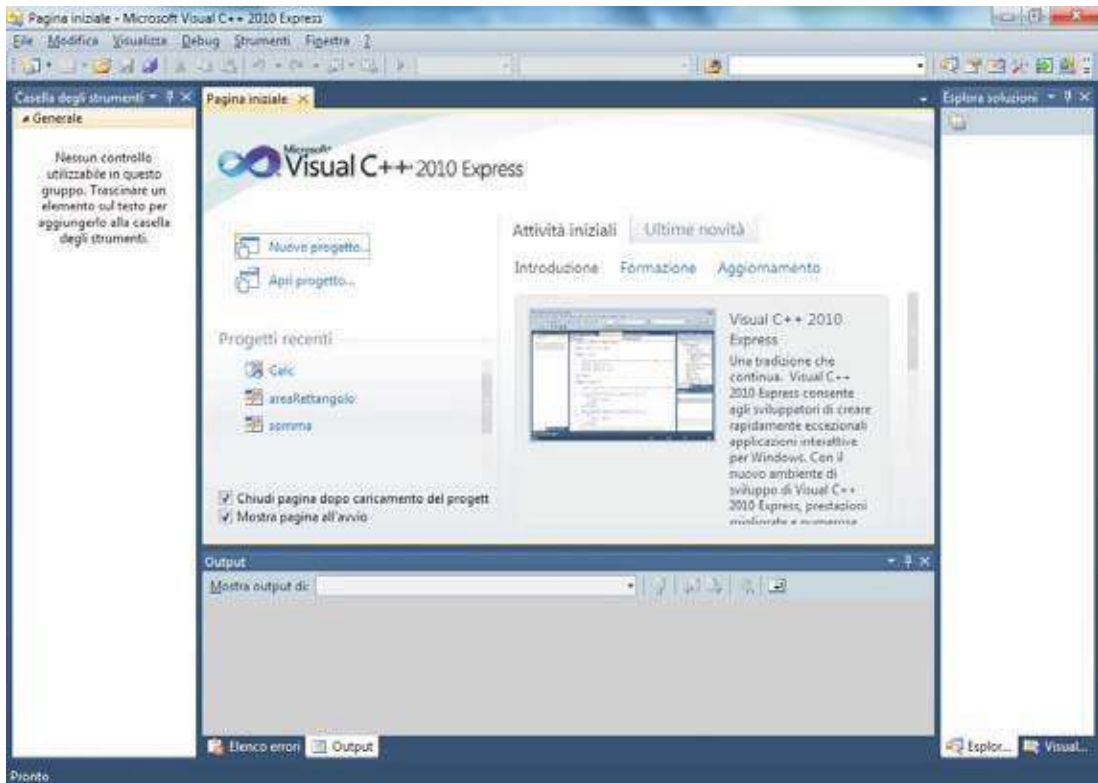
La creazione della versione definitiva dell'eseguibile viene creata dopo aver scelto la modalità **Release**, al posto di *Debug*, nel **Configuration Manager** del menu **Build**.

Con questa seconda modalità, l'operazione di *Build* sul programma crea la sottodirectory **Release** nella directory del *workspace* e, al suo interno, il file eseguibile (.exe), pronto per essere distribuito agli utenti finali.

## Visual C++

Il linguaggio **Visual C++** è parte di **Microsoft Visual Studio**, che denomina un insieme di strumenti di sviluppo software con diversi linguaggi. Visual Studio è un software con licenza commerciale; tuttavia Microsoft fornisce la versione **Express Edition** che è liberamente scaricabile da Internet e utilizzabile gratuitamente per fini di apprendimento e non commerciali. L'ambiente di sviluppo integrato (**IDE**, *Integrated Development Environment*) di Visual C++ è un insieme di strumenti che facilitano il lavoro del programmatore.

Per avviare Visual C++ occorre fare clic su *Start, Tutti i programmi, Microsoft Visual Studio, Visual C++ Express*.



Viene mostrata la **Pagina iniziale** del programma che, in zone differenti della finestra, elenca:

- i progetti aperti recentemente,
- le novità per il programmatore (gli argomenti presenti in questo riquadro si aggiornano automaticamente se è disponibile una connessione a Internet),
- alcuni capitoli della Guida in Linea.

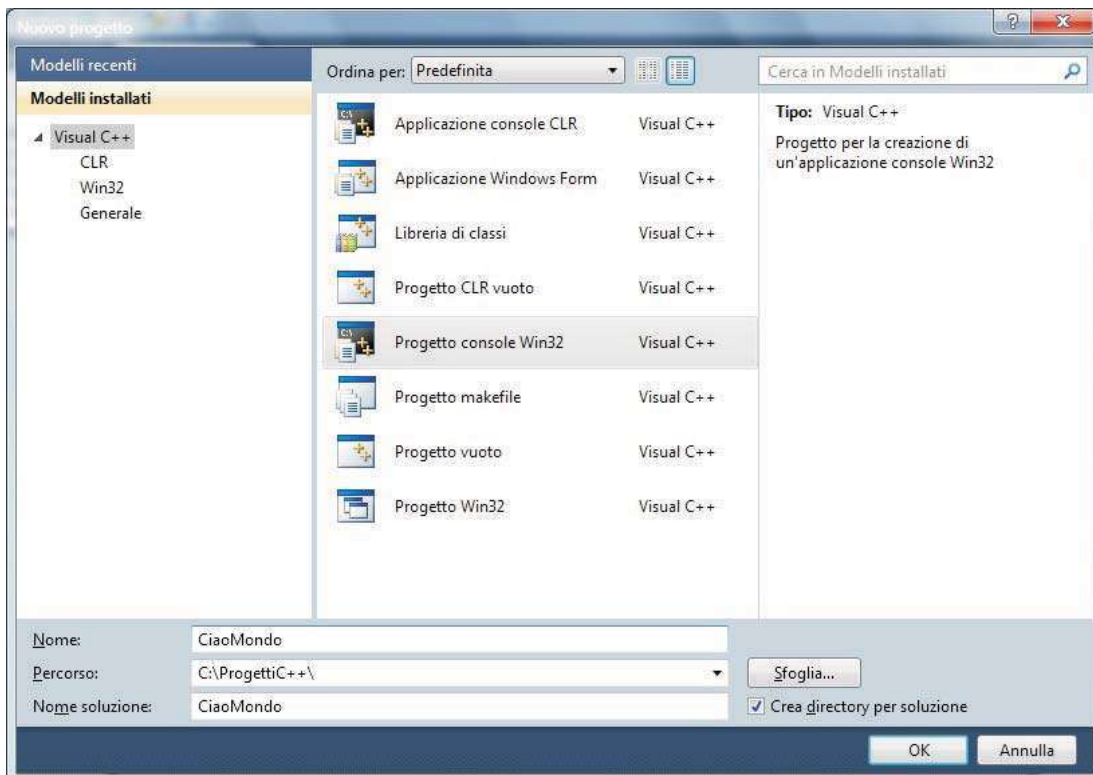
È importante sottolineare che il programmatore, che vuole apprendere l'uso di Visual C++, ha a disposizione un *help*, o **guida in linea**, con molti esempi di codice pronti da provare. Utilizzando la combinazione di tasti **CTRL + F1** si apre la **guida dinamica** che visualizza le informazioni della guida riferite all'attività che il programmatore sta svolgendo.

I programmi possono essere **progetti** o **soluzioni**, che comprendono molte componenti di tipo diverso. In un lavoro ordinato di programmazione, per ogni programma viene creata una cartella su disco: il programma contiene un file di soluzione avente l'estensione **.sln** e uno o più file di progetto che hanno estensione **.vcxproj**.

In generale una **soluzione** è un insieme di progetti, a ciascuno dei quali è assegnata una funzionalità di un'applicazione software complessa. Nei programmi semplici, formati da un solo progetto, l'apertura di un file di progetto equivale all'apertura di una soluzione.

Per creare un nuovo progetto si deve fare clic sul menu **File** e poi su **Nuovo Progetto**, oppure fare clic sulla scelta **Nuovo Progetto** nella *Pagina iniziale*. La scorciatoia da tastiera è la combinazione di tasti **CTRL + Maiuscolo + N**.

Viene aperta una finestra in cui è possibile scegliere tra vari tipi di progetto (**Modelli**) che hanno caratteristiche differenti.



Scegliendo il modello **Progetto console Win32** si può creare un'applicazione che utilizza la **riga di comando**, cioè un programma eseguibile che non fa uso dell'interfaccia grafica. In alternativa, per creare programmi con interfaccia grafica occorre scegliere il modello **Applicazione Windows Form**, che sarà utilizzato nei progetti presentati nel testo dopo il Capitolo 8.

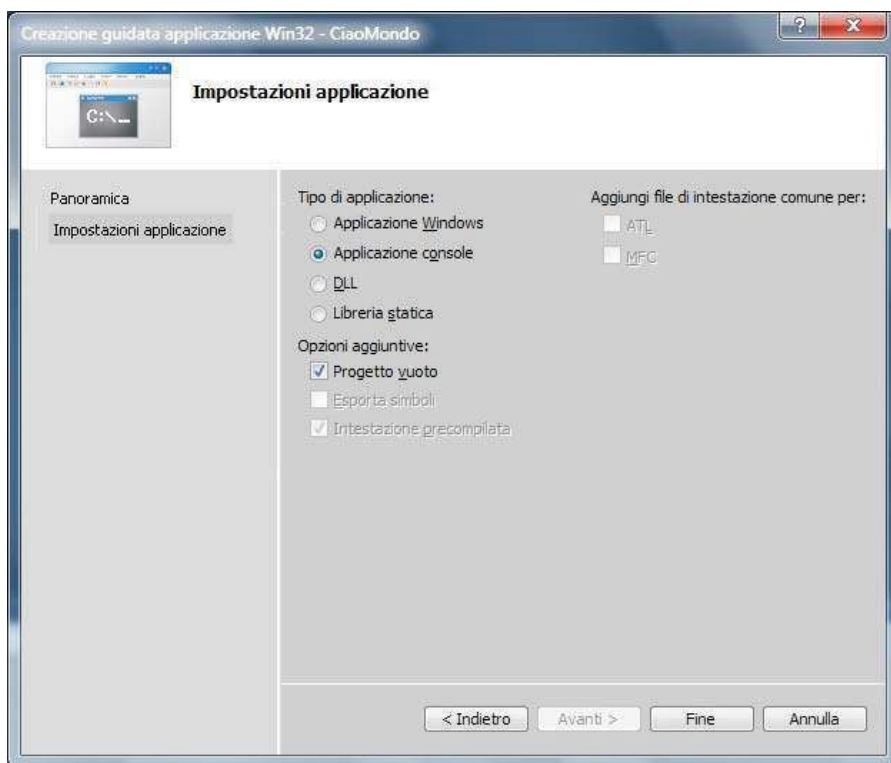
Dopo aver scelto il tipo di progetto, occorre inserire il nome con cui salvarlo. Scriviamo *CiaoMondo* nella casella **Nome**, perché si vuole realizzare un semplice esempio che visualizza il messaggio "Ciao, Mondo!". Selezioniamo anche il percorso dove salvare il progetto facendo clic sul pulsante **Sfoglia**.

Si noti che il nome scelto per il progetto viene assegnato, in modo predefinito, anche alla soluzione.

Il programma crea sul disco una cartella con lo stesso nome assegnato alla soluzione.

Facciamo infine clic su **OK**.

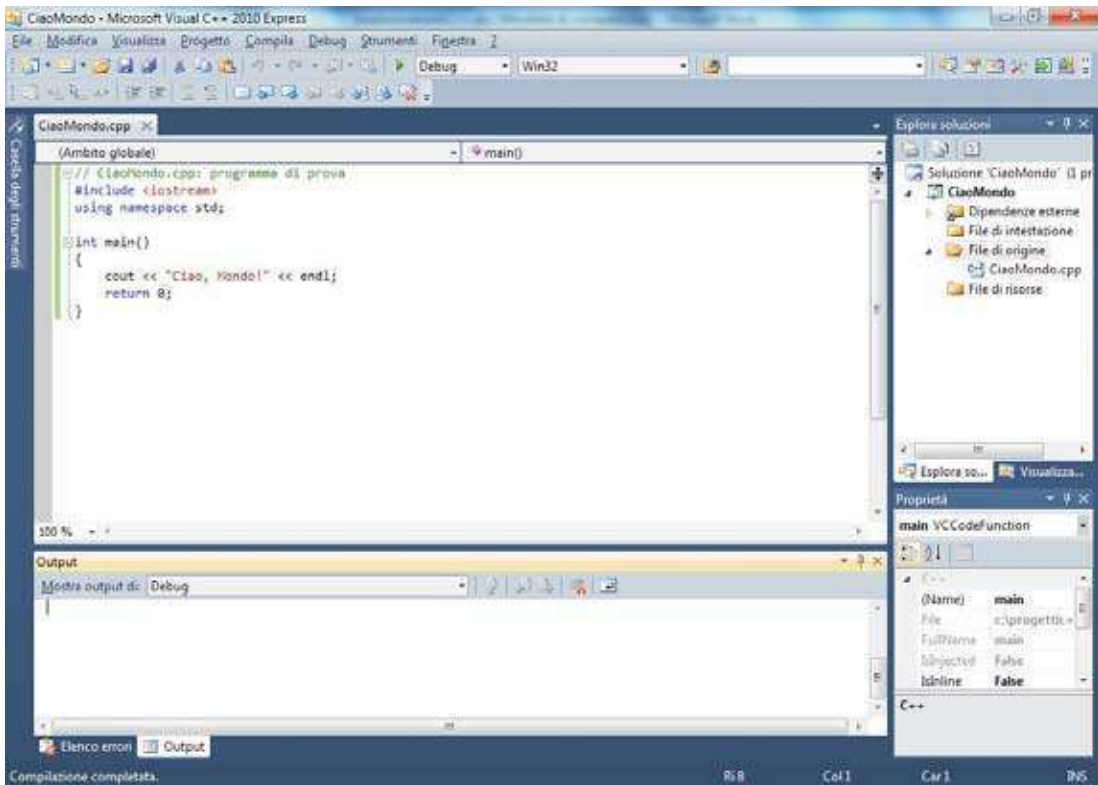
Nella successiva finestra della *Creazione guidata*, facciamo clic, a sinistra, su **Impostazioni applicazione** e, nel riquadro di destra, scegliamo **Applicazione console** come tipo di applicazione e mettiamo un segno di spunta sull'opzione **Progetto vuoto**. Facciamo infine clic sul pulsante **Fine**.



Viene creato il progetto e nella finestra **Esplora soluzioni** vengono create le cartelle *Dipendenze Esterne*, *File di intestazione* (file header con estensione **.h**), *File di origine* (file sorgente con estensione **.cpp**), *File di risorse*. Queste cartelle servono ad organizzare i file che compongono il progetto.

Per creare un nuovo testo sorgente in C++, occorre fare clic con il tasto destro del mouse sulla cartella *File di origine* e scegliere **Aggiungi, Nuovo elemento** (si può usare anche l'icona nella barra degli strumenti). Nella finestra che si apre, selezionare **File di C++ (.cpp)**, assegnare un nome al file e fare clic su **Aggiungi**.

Si apre la finestra dell'editor del programma dove si può scrivere il codice in C++.

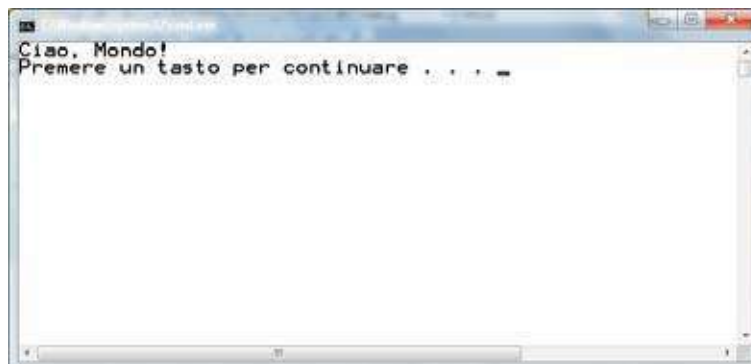


Si noti che il testo del codice viene colorato in modo automatico: in verde i commenti, in azzurro le parole chiave e in rosso i messaggi. Inoltre eventuali errori sono segnalati con una linea ondulata rossa: passando con il mouse sopra la parola sottolineata, viene visualizzato il messaggio di errore.

Salviamo il lavoro dal menu **File** con la scelta **Salva tutto**, oppure con un clic sull'icona della barra degli strumenti (la scorciatoia da tastiera è la combinazione di tasti **Ctrl + Maiuscolo + S**).

Il programma può essere compilato ed eseguito premendo il tasto **F5** oppure facendo clic sull'icona **Debug** della barra degli strumenti (freccia verde verso destra). I messaggi riguardanti la compilazione e l'esecuzione sono mostrati nella finestra **Output** nella parte inferiore del video. L'esecuzione provoca l'apertura di una finestra di console con la visualizzazione del messaggio e, subito dopo, la console viene chiusa.

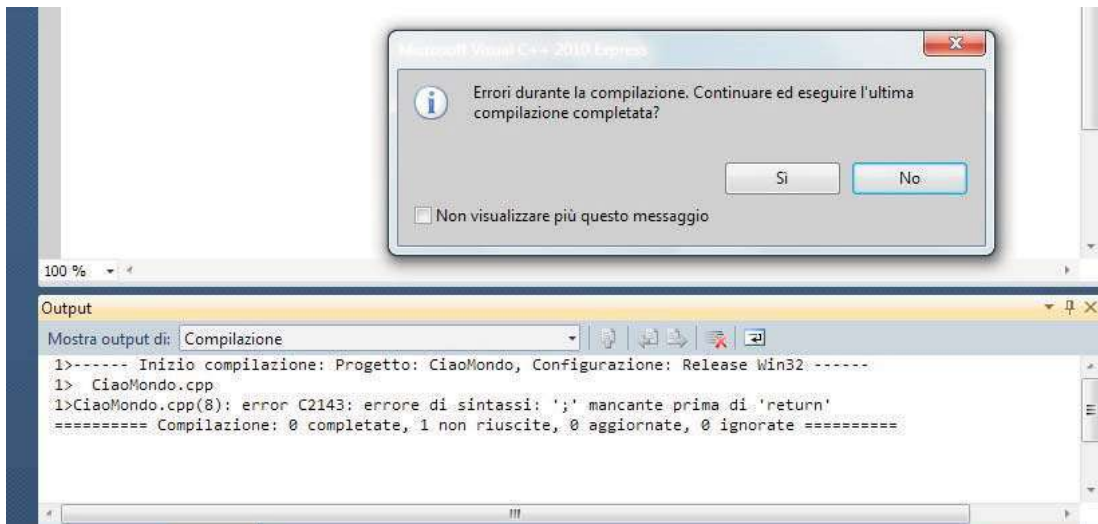
Per mantenere aperta la finestra di esecuzione, occorre scegliere, nel menu *Debug*, **Avvia senza eseguire debug** (la scorciatoia è **Ctrl + F5**).



La modalità di esecuzione in debug facilita l'individuazione e la risoluzione di eventuali errori di programmazione. Per effettuare la compilazione vera e propria si deve utilizzare il comando **Compila** dal menu *Compila* che crea un eseguibile (file con estensione **.EXE**), cioè un programma che può essere richiamato direttamente dal *Prompt dei comandi* del sistema operativo.

Se il testo sorgente del programma contiene errori, la compilazione non viene completata e le righe contenenti l'errore sono riportate nella finestra *Output*. Si provi, per esempio, a togliere il punto e virgola finale dell'istruzione *cout*: viene segnalato l'errore di sintassi

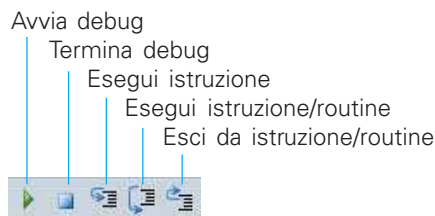
*',' mancante prima di 'return'*



Le attività più comuni che si possono attivare per il *debugging* dell'applicazione e in particolare per l'individuazione degli errori sono:

- impostare punti di interruzione (**breakpoint**) che fermano temporaneamente l'applicazione in corrispondenza di linee di codice prefissate;
- osservare il valore assunto dalle variabili o il risultato delle espressioni di calcolo;
- eseguire il programma passo dopo passo;
- impostare la prossima istruzione da eseguire.

Queste azioni possono essere attivate rapidamente utilizzando le apposite icone presenti nella barra degli strumenti **Debug**:



Per fissare punti di interruzione è possibile fare clic a lato della linea di codice; viene visualizzato un cerchio rosso ●. Facendo clic sul cerchio, il punto di interruzione viene eliminato.

Durante il debug (avviato con un clic sull'icona **Avvia debug**  oppure con il tasto **F5**) l'esecuzione dell'applicazione viene fermata in corrispondenza di ogni punto di interruzione. In quel momento è possibile visualizzare lo stato del programma, per esempio il valore assunto dalle variabili.

Per vedere il valore delle variabili e delle espressioni si deve passare con il mouse sopra la variabile, oppure aprire la finestra **Espressioni di controllo**, dal menu *Debug, Finestre*, e scrivere il nome della variabile.

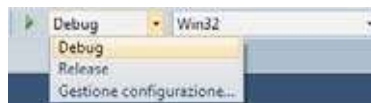
Per eseguire il programma passo passo:

- fare clic su **Esegui istruzione**  (oppure premere il tasto **F8**) per procedere linea per linea;
- fare clic su **Esegui istruzione/routine**  (oppure premere i tasti **Maiuscolo + F8**): in questo caso le subroutine vengono trattate come una singola linea di codice.

L'ambiente Visual C++ prevede due modalità di generazione dell'eseguibile:

- build di **Debug**, utilizzata dal programmatore durante lo sviluppo e il test del software,
- build di rilascio (**Release**), per la versione da distribuire agli utenti finali.

La scelta tra i due tipi di *build* può essere fatta nella casella della barra degli strumenti accanto all'icona **Debug**.



Con la seconda modalità, viene creata la sottocartella **Release** all'interno della cartella del progetto e al suo interno il file eseguibile (**.exe**), pronto per essere distribuito agli utenti finali.

Per chiudere il progetto aperto nell'ambiente di sviluppo, dopo aver salvato il lavoro, occorre scegliere **Chiudi soluzione** nel menu **File**.

# 6

**parte terza**

La programmazione

## **Programmazione ad oggetti**

### **OBIETTIVI DI APPRENDIMENTO**

In questo capitolo imparerai a definire le classi con attributi e metodi e a creare gli oggetti come istanze delle classi.

Conoscerai i principi della programmazione ad oggetti: incapsulamento dei dati, ereditarietà, polimorfismo.

Sarai quindi in grado di applicare in pratica questi principi utilizzando il linguaggio C++.

Classi, attributi, metodi  
Incapsulamento e information  
hiding  
Costruttore e distruttore  
Accesso pubblico e privato  
Messaggi e interfaccia  
Funzioni friend  
Ereditarietà  
Gerarchia delle classi  
Ereditarietà multipla  
Polimorfismo  
Overloading dei metodi  
Overriding dei metodi  
Overloading degli operatori  
Funzioni virtuali



## 1 L'astrazione

Il mondo in cui viviamo è caratterizzato da una notevole complessità. Osservando ciò che ci circonda ci accorgiamo, infatti, che ogni cosa è più o meno complessa. Si pensi, per esempio, al numero di nozioni contenute in questo libro e alla difficoltà richiesta per impararle, oppure ai componenti di un computer o alle diverse parti del motore del nostro mezzo di trasporto abituale. È improbabile che un individuo, riferendosi a questi oggetti, sia in grado di comprenderli e descriverli in ogni particolare. Ben pochi automobilisti, infatti, conoscono le funzioni di tutte le componenti del motore della loro automobile.

Ancora più ardua risulta la costruzione di sistemi complessi. Immaginiamo di dover costruire un modellino di aereo partendo dalle materie prime: plastica e ferro. Ovviamente, sia pur accurato, il modello non potrà mai incorporare tutte le caratteristiche o funzionalità di un aereo reale. Come per la costruzione di aeromodello si realizzano solo le funzionalità ritenute rilevanti, così nella descrizione della realtà si opera una semplificazione, limitandosi alla considerazione delle sole parti considerate rilevanti per la trattazione.

Se, per esempio, vogliamo descrivere le caratteristiche del motore della nostra automobile, possiamo soffermarci sul numero di cilindri (per esempio, 6) e sulla capacità di ciascun cilindro (per esempio, 500 cc), sul tipo di alimentazione (per esempio, benzina) e sul tipo di carburazione (per esempio, iniezione elettronica).

Tutte queste informazioni ci permettono di individuare un certo tipo di propulsore, ma non ci mettono sicuramente in grado di conoscerne tutte le caratteristiche. In altre parole, non potremmo mai realizzare un motore con tali caratteristiche senza informazioni aggiuntive (per esempio, come si costruisce un pistone o come si montano le fasce elastiche).

Questo processo di semplificazione della realtà complessa è detto **astrazione**.

Grazie all'astrazione prepariamo delle carte geografiche che ci permettono di raggiungere un determinato punto della città senza dover descrivere nei minimi dettagli l'aspetto e il colore delle abitazioni che si trovano in tale punto, oppure le informazioni anagrafiche sugli abitanti della zona. Una tale piantina non ci fornisce alcuna informazione storica sugli edifici o sui personaggi che vi hanno vissuto, ma questo non è lo scopo di una cartina stradale. Viceversa, consente di raggiungere i diversi punti della città in essa descritti, anche se ignoti.

Esistono diversi livelli di astrazione in base alle proprie esigenze: la descrizione del proprio scooter avrà dei livelli di astrazione diversi da quelli utilizzati dal meccanico per la riparazione dello scooter. Dovendo infatti impiegare il nostro ciclomotore, parleremo delle modalità di messa in moto, di rifornimento e di guida, mentre non tratteremo il diametro del foro della biella che compete invece al meccanico.

L'astrazione riveste un ruolo importante nella produzione del software. Un'applicazione software solitamente rappresenta un'astrazione della realtà. Si pensi a un programma di simulazione del volo. Nelle fasi che portano alla realizzazione del programma si crea un modello della realtà che si vuole riprodurre sullo schermo. Tale modello non è altro che la semplificazione delle componenti di un volo che rivestono un qualche interesse per i nostri scopi. Potremmo infatti essere interessati a riprodurre il rumore dell'aereo, oppure gli effetti provocati dal vento di coda, o, ancora, a riprodurre entrambe le caratteristiche.

Completata questa prima fase molto importante, detta di *astrazione* o **analisi del problema**, si passa alla sua implementazione attraverso la **programmazione** nel linguaggio prescelto.



## 2 Programmare con gli oggetti

Il computer esegue tutto ciò che viene richiesto ad esso, attraverso un linguaggio predefinito e compreso dalla macchina stessa. Il computer, infatti, svolge le funzioni richieste attraverso una sequenza di ordini (**istruzioni**), detta **programma**.

Nelle pagine precedenti abbiamo infatti visto come sia possibile richiedere al calcolatore l'elaborazione di un calcolo anche molto complesso, semplicemente elencando i singoli passi da percorrere per giungere alla soluzione. La descrizione di ciascun passo è realizzata con un linguaggio simbolico, cioè non naturale, ma costruito opportunamente per comunicare con il computer. Nella fattispecie, ci siamo occupati del linguaggio C++.

Nei progetti presentati in precedenza abbiamo usato un tipo di **programmazione procedurale**. In sostanza il programma, composto di dati e istruzioni, elenca le istruzioni che il computer deve eseguire secondo un ordine predefinito in grado di ottenere i risultati desiderati (*output*) partendo dai dati forniti in ingresso (*input*). I diversi percorsi possibili, a seconda dei casi che si possono verificare durante l'esecuzione, sono gestiti attraverso le **strutture di controllo** (sequenza, selezione e ripetizione).

Questo stile di scrittura era di semplice realizzazione e adatto all'implementazione di piccoli programmi. Attorno al 1960, quando i produttori di software furono chiamati a scrivere programmi via via sempre più lunghi e potenti, sia per ragioni commerciali sia per le maggiori capacità dei computer, questo modo di implementare il codice si è dimostrato inefficiente in quanto difficile da modificare e da riutilizzare in altri progetti.

Queste esigenze dei programmi commerciali hanno portato alla definizione di un nuovo stile di programmazione, basato sull'organizzazione e la suddivisione dei programmi in **moduli** funzionalmente indipendenti utilizzando le **funzioni**.

La costruzione di programmi ordinati, basati sull'uso delle strutture di controllo e sull'organizzazione modulare del codice, si chiama **programmazione strutturata**.

In un programma ben strutturato, la scrittura della funzione principale, **main()**, non richiede la conoscenza di tutte le istruzioni contenute nelle sottofunzioni. Una volta che la sottofunzione è stata implementata correttamente, non ci si preoccupa più della sua gestione. La scrittura del programma si limita al richiamo della funzione di cui dobbiamo conoscere il numero e tipo di parametri di input previsti e il tipo restituito in output. Inoltre nulla vieta di implementare nel tempo un modulo più efficiente che svolga le stesse funzionalità, senza dover modificare il resto del programma.

Vent'anni di programmazione strutturata hanno evidenziato che i benefici apportati da questa tecnica di programmazione non sono sufficienti a garantire un abbondante riutilizzo del software implementato. A partire dal 1980 si è quindi sviluppato un nuovo stile di scrittura del codice che ne consente un reimpiego sempre maggiore.

L'idea, di per sé molto semplice, rivoluziona il modo di implementare il codice inserendo un nuovo approccio, rappresentato dai dati del problema. Questo modo di programmare si chiama **programmazione orientata agli oggetti**.

La programmazione orientata agli oggetti non presuppone l'eliminazione delle tecniche precedenti, ma piuttosto le completa, aggiungendo loro una nuova dimensione.

La **programmazione orientata agli oggetti**, in breve **OOP** (*Object-Oriented Programming*) prende il nome dall'elemento su cui si basa: l'*oggetto*.

Il programma, realizzato con orientamento ad oggetti, si sviluppa attraverso le interazioni tra gli oggetti: durante l'esecuzione del programma, gli oggetti possono cambiare il loro stato e possono richiedere l'esecuzione di operazioni associate ad altri oggetti.

La programmazione orientata agli oggetti procura diversi vantaggi:

- facilità di **lettura** e di **comprensione** del codice, anche per persone diverse dall'autore;
- **manutenzione** del programma nel tempo per correzioni o miglioramenti;
- **robustezza** del programma in situazioni critiche o in operazioni che coinvolgono grandi quantità di dati;
- **riusabilità** di parti di codice o di moduli funzionali all'interno di altri programmi.

Gli **oggetti** rappresentano le entità del problema o della realtà che si vuole automatizzare con l'informatica. Un oggetto è in grado di memorizzare le informazioni che riguardano il suo stato; è anche possibile associare ad un oggetto un insieme di operazioni che esso può compiere.

In particolare, un oggetto può essere definito elencando sia le sue caratteristiche, sia il modo con cui interagisce con l'ambiente esterno, cioè i suoi comportamenti.

Gli **attributi** rappresentano gli elementi che caratterizzano l'oggetto, utili per descrivere le sue caratteristiche e definirne il suo stato.  
I **metodi** rappresentano i comportamenti e le funzionalità che l'oggetto mette a disposizione.

La struttura di un oggetto è completamente descritta quando vengono elencate le caratteristiche e i comportamenti dell'oggetto.

Per esempio, un libro è un *oggetto*: le caratteristiche del libro (titolo, peso, argomento trattato, numero di pagine, ecc.) corrispondono agli *attributi* dell'oggetto, mentre le azioni che si possono esercitare sull'oggetto (aprirlo a una certa pagina, leggerlo, sfogiarlo, sottolinearlo) corrispondono ai *metodi* di cui dispone l'oggetto considerato.

### 3 La classe

Gli oggetti sono definiti dalle classi.

La **classe** è la descrizione astratta degli oggetti attraverso gli attributi e i metodi. Per utilizzare un oggetto occorre crearlo come esemplare della classe, cioè come **istanza** di una classe.

Nella programmazione orientata agli oggetti, dopo aver identificato l'oggetto, solitamente rappresentato da un sostantivo, occorre individuarne gli attributi, ossia le caratteristiche rilevanti, solitamente rappresentati dagli aggettivi, ed infine i metodi, cioè le azioni da applicare agli oggetti, generalmente indicate dai verbi.

Dovendo, per esempio, realizzare un programma che gestisca la clientela di un'azienda, ossia memorizzi i dati anagrafici, consentendo l'inserimento di nuovi clienti, le eventuali modifiche e cancellazioni di quelli già inseriti, notiamo che i sostantivi sono l'azienda e i clienti.

Ciascun cliente prevede delle informazioni (o *attributi*), come il nome, l'indirizzo e il numero di telefono, e delle funzionalità, quali l'acquisizione delle informazioni (*inserimento*), la *modifica* di alcune di queste e la loro *cancellazione*.

Una classe viene rappresentata con uno schema grafico detto **diagramma della classe**, che ne evidenzia il nome, gli attributi e i metodi.

Per esempio, il diagramma della classe *Cliente* è così strutturato:

| Cliente                                  |
|------------------------------------------|
| nome<br>cognome<br>indirizzo<br>telefono |
| Inserisci<br>Modifica<br>Cancella        |

Lo schema precedente rappresenta la classe: la prima zona contiene il nome della classe, la seconda l'elenco degli *attributi* e la terza l'elenco dei *metodi*.

Gli elementi che formano la classe, attributi e metodi, si chiamano **membri** della classe. Il raggruppamento dei membri conferisce alla classe il significato di un'unità di programmazione, riutilizzabile in altri programmi: il controllo dell'esecuzione diventa più facile e la modifica degli oggetti nel tempo diventa più semplice. Questi aspetti descrivono il concetto di **incapsulamento**, che è uno dei concetti alla base della programmazione ad oggetti.

Il termine **incapsulamento** indica la qualità degli oggetti di poter incorporare al loro interno sia gli attributi sia i metodi, cioè le caratteristiche e i comportamenti dell'oggetto.

Tutto ciò che si riferisce ad un certo oggetto è racchiuso e contenuto all'interno dell'oggetto stesso. Si crea come una capsula, una barriera concettuale, che isola l'oggetto dalle cose esterne. Si dice che gli attributi e i metodi sono incapsulati nell'oggetto. In questo modo tutte le informazioni utili che riguardano un oggetto sono ben localizzate. Questa imposizione, che obbliga a raccogliere tutto quello che riguarda una singola entità all'interno di un oggetto, è uno dei vantaggi che viene offerto dalla programmazione orientata agli oggetti.

Creando tre istanze, *cliente1*, *cliente2* e *cliente3* della classe *Cliente*, si ottengono tre oggetti:

| cliente1                                                                                | cliente2                                                                                     | cliente3                                                                                 |
|-----------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| nome = Simone<br>cognome = Bianchi<br>indirizzo = via Europa 3<br>telefono = (02)345467 | nome = Antonio<br>cognome = Bruni<br>indirizzo = piazza Trieste 5<br>telefono = (06)89674536 | nome = Giacomo<br>cognome = Rossi<br>indirizzo = via Torino 10<br>telefono = (030)234561 |

Ogni classe possiede un particolare metodo predefinito, detto **costruttore**, che viene attivato quando si crea un oggetto.

Dopo aver creato l'oggetto, è possibile modificare le sue proprietà o invocare l'attivazione di un suo metodo.

Un'ulteriore fase, prima dell'implementazione del codice, è dedicata all'individuazione di eventuali generalizzazioni e, muovendosi dal generale al particolare, all'individuazione di alcune specificità dell'oggetto esaminato.

Si creano così delle classi molto generali, che possiamo chiamare **classi di base** (o **superclassi**), nelle quali vengono implementate le componenti comuni a un considerevole numero di oggetti, sfruttando le proprietà di **information hiding** (o *data hiding*).

Il termine **information hiding** indica il mascheramento delle modalità di implementazione di un'oggetto, rendendone disponibili all'esterno solo le funzionalità.

Chi utilizza un oggetto è interessato alle caratteristiche e alle funzioni che mette a disposizione, ma non sempre è interessato a conoscerne l'implementazione, cioè il modo con cui i metodi sono realizzati.

Molto spesso l'oggetto è stato costruito da un programmatore, diverso da colui che lo userà. A quest'ultimo interessa sapere invece come interagire con l'oggetto, quali sono i metodi che mette a disposizione e come poterli richiamare.

Questo modo di intendere gli oggetti induce a considerarli come delle scatole nere (*black box*). I dettagli sulle caratteristiche e la struttura dell'oggetto sono nascosti all'interno, garantendo l'*information hiding*.

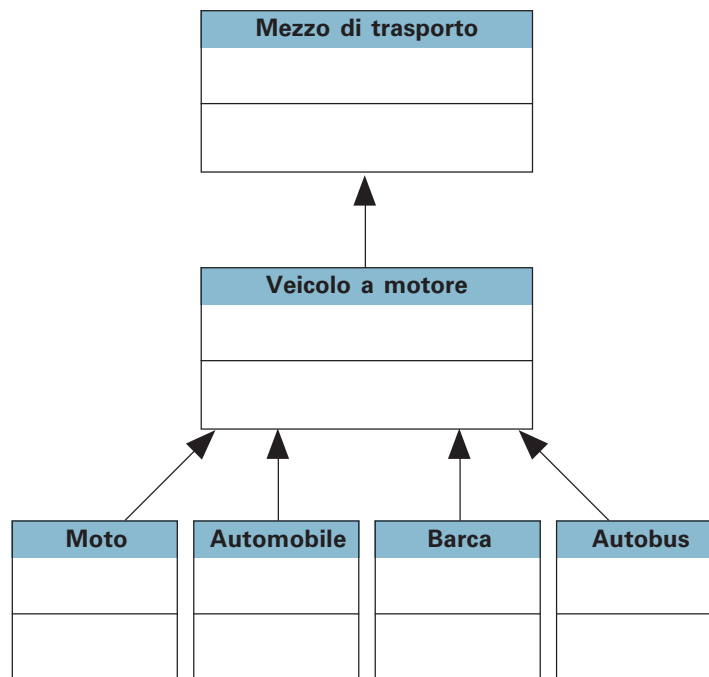
Partendo dai principi dell'*information hiding* si è sviluppato un nuovo modo di concepire il software come formato da scatole, che mettono a disposizione un insieme di funzionalità. È nata un'industria di componenti software. Il contenuto di questi componenti resta nascosto, mentre se ne conosce il modo con cui operano. Per creare nuovi programmi si uniscono diverse componenti, facendole interagire nel modo desiderato: si pensi per esempio alle componenti che costituiscono le interfacce grafiche nei moderni computer (finestre, menu, barre degli strumenti, bottoni).

Questo modo di pensare generalizzato permette da una parte di riferirsi ad una classe, senza entrare nel dettaglio degli attributi e dei metodi, e dall'altra di implementare una sola volta parti di codice che risultano comuni a progetti diversi.

Si consideri per esempio l'oggetto *Automobile*: possiamo effettuare una generalizzazione definendo l'oggetto *Veicoli a motore* come superclasse dell'oggetto *Automobile*.

Una generalizzazione ulteriore può essere ottenuta definendo l'oggetto *Mezzo di trasporto* come superclasse dell'oggetto *Veicolo a motore*.

Si ottiene in questo modo una **gerarchia di classi**, che può essere opportunamente rappresentata con un diagramma come nella figura seguente.



Le frecce partono dalla classe di livello inferiore verso la classe di livello superiore. La freccia si legge con la frase "è un" (in inglese **is a**): per esempio, l'autobus è un veicolo a motore. Ogni rettangolo, diviso in tre zone, descrive la classe con nome, attributi e metodi.

Per esempio, la classe *Veicolo a motore* è rappresentata in dettaglio dallo schema della figura a destra.

Nel descrivere, per esempio, un motore possiamo fare riferimento alle sue proprietà, quali la cilindrata, il numero di cilindri e il numero di giri. Sempre in relazione all'oggetto *Veicolo a motore*, possiamo pensare che sia possibile avviare il motore, spegnerlo, aumentarne i giri, ecc.

| Veicolo a motore                                       |
|--------------------------------------------------------|
| cilindrata<br>numero di cilindri<br>giri motore<br>... |
| Avvia<br>Spegni<br>Aumenta i giri<br>...               |

I simboli utilizzati negli schemi grafici precedenti rispettano gli standard del linguaggio **UML** (*Unified Modeling Language*), un insieme di regole riconosciute a livello internazionale per specificare, rappresentare, progettare e realizzare software, in particolare, nel contesto della programmazione, per descrivere le classi e gli oggetti.

Dopo aver implementato una classe di base, si può passare alla creazione di una classe più specifica (o **sottoclasse**), strettamente legata alla classe che si deve trattare nel proprio problema specifico. Tale classe *deriva* dalla precedente attraverso il meccanismo dell'**ereditarietà**, che è un altro concetto fondamentale della programmazione ad oggetti.

Con riferimento al diagramma precedente, l'oggetto *Automobile* eredita le caratteristiche (attributi) e i comportamenti (metodi) dell'oggetto *Veicolo a motore*.

Di conseguenza l'oggetto *Automobile* ha in dotazione, per effetto dell'ereditarietà, tutte le proprietà che l'oggetto *Veicolo a motore* ha messo a disposizione, per esempio *cilindrata* e *Avvia*.

Oltre all'ereditarietà, una classe derivata, per poter essere implementata in modo efficace, sfrutta le possibilità offerte dal **polimorfismo**, cioè la possibilità di definire più versioni di uno stesso metodo.

I concetti di ereditarietà, polimorfismo, incapsulamento e *information hiding*, caratteristici della programmazione orientata agli oggetti in generale, sono al centro delle specifiche del linguaggio C++: sono stati qui presentati in forma introduttiva e saranno poi spiegati in modo dettagliato, e dal punto di vista operativo, nei paragrafi successivi.



#### MATERIALI ONLINE

### 1. Strumenti per disegnare i diagrammi delle classi (UML)

## 4 Creazione di una classe

Nel linguaggio C++ le classi inglobano sotto un unico nome i *dati membro* (**attributi**) e le *funzioni membro* (**metodi**). Questa modalità di aggregazione realizza in pratica il concetto di **incapsulamento** della programmazione ad oggetti.

Il linguaggio C++ utilizza la dichiarazione **class** per definire la classe. La sintassi generale è:

```
class NomeClasse {
 // attributi
 tipo1 attributo1;
 tipo2 attributo2;
 ...
 // metodi
 tipo1 Funzione1() {...};
 tipo2 Funzione2() {...};
 ...
};
```

Quindi la classe non è composta solo da dati, ma anche da funzioni che rappresentano i metodi. I metodi sono definiti in modo identico alle normali funzioni di un programma, infatti i metodi di una classe si chiamano *funzioni membro* (**member function**).

L'utilizzo della classe all'interno del programma avviene attraverso la creazione delle sue **istanze**, cioè degli *oggetti*. La **dichiarazione di un oggetto** segue la normale sintassi della dichiarazione delle variabili:

```
NomeClasse NomeOggetto;
```

L'attributo di un oggetto è rappresentato dalla notazione:

```
NomeOggetto.NomeAttributo
```

La stessa notazione vale per l'uso delle funzioni che sono metodi dell'oggetto:

```
NomeOggetto.NomeFunzione()
```

La **dimensione di un'istanza** coincide con il numero di byte occupati dai suoi attributi. I metodi di una classe non occupano alcun spazio nell'istanza, in quanto sono comuni a tutte le istanze della classe.

## PROGETTO 1

### Definire una classe per rappresentare un rettangolo.

Il progetto mostra la modalità per la creazione di una classe contenente, come attributi, la base e l'altezza della figura e, come metodi, la funzione per assegnare i valori alle dimensioni del rettangolo e la funzione per calcolare l'area.

| Rettangolo      |
|-----------------|
| base<br>altezza |
| Assegna<br>Area |

Lo schema evidenzia che la classe degli oggetti *Rettangolo* è composta da due attributi, *base* e *altezza*, e da due metodi, *Assegna* e *Area*.

La codifica della classe *Rettangolo* nel linguaggio C++ è la seguente:

```
class Rettangolo {
 // attributi
 double base, altezza;
public:
 // metodi
 void Assegna(double b, double h) {
 base = b;
 altezza = h;
 }
 double Area() {
 return base*altezza;
 }
};
```

La classe *Rettangolo* così definita è pronta per essere utilizzata all'interno del programma, alla pari e con le stesse modalità viste in precedenza per le variabili e i tipi di variabile.

In particolare non si può usare la classe di oggetti, ma una sua istanza, cioè un **oggetto**, che viene creato con la dichiarazione:

```
Rettangolo tovaglia;
```

Questa riga di codice crea un oggetto *tovaglia* di tipo *Rettangolo* che occupa un numero di byte pari a  $\text{sizeof}(\text{base}) + \text{sizeof}(\text{altezza})$ , infatti la dimensione di ogni istanza della classe *Rettangolo* è:

$2 \times \text{sizeof}(\text{double})$  byte.

Il programma completo in linguaggio C++ per assegnare i valori alle dimensioni del rettangolo e per calcolarne l'area è il seguente:

```
// Rettangolo.cpp: area di un rettangolo
#include <iostream>
using namespace std;

class Rettangolo {
 // attributi
 double base, altezza;
public:
 // metodi
 void Assegna(double b, double h) {
 base = b;
 altezza = h;
 }
 double Area() {
 return base*altezza;
 }
};

int main()
{
 Rettangolo tovaglia;

 tovaglia.Assegna(2.0, 1.0);
 cout << "Area = " << tovaglia.Area() ;

 return 0;
}
```

Si osservi la modalità di chiamata dei metodi all'interno del programma con la notazione *nome oggetto* e *nome metodo* separati dal punto: *tovaglia.Assegna()* e *tovaglia.Area()*.

Da questo primo esempio elementare possiamo intuire l'utilità della definizione degli oggetti. Una volta creata la classe *Rettangolo*, essa può essere riutilizzata ogni volta se ne presenti la necessità. Inoltre si può disporre, non solo dei dati, ma anche delle funzioni ad essi associate.

Per rendere più efficiente il codice, soprattutto in presenza di metodi con molte istruzioni, è possibile inserire nella classe solamente il prototipo della funzione membro e riportare separatamente la sua implementazione.

Per indicare che non si tratta di una definizione di funzione del programma, ma della definizione di un metodo appartenente a una classe, si usa la notazione:

```
tipo NomeClasse::Funzione {
 ...
 return;
}
```

Il simbolo doppio due punti, ::, si chiama, in generale, **operatore di risoluzione della visibilità** (*scope resolution*) e, quando è riferito ai metodi delle classi, di **risoluzione della visibilità della classe** (*class scope*).

Seguendo questa modalità, il programma precedente, per il calcolo dell'area di un rettangolo, viene riscritto come indicato di seguito:

```
// Rettangolo.cpp: area di un rettangolo
#include <iostream>
using namespace std;

class Rettangolo {
 // attributi
 double base, altezza;
public:
 // metodi
 void Assegna(double, double);
 double Area();
};

void Rettangolo::Assegna(double b, double h)
{
 base = b;
 altezza = h;
}

double Rettangolo::Area()
{
 return base*altezza;
}

int main()
{
 Rettangolo tovaglia;

 tovaglia.Assegna(2.0, 1.0);
 cout << "Area = " << tovaglia.Area() ;

 return 0;
}
```

La scrittura dell'implementazione della funzione membro in modo separato, al di fuori della dichiarazione della classe, non solo rende più leggibile il codice, ma risponde anche a criteri di efficienza.

Infatti, nella prima versione dell'esempio precedente, con la funzione implementata all'interno della classe, il compilatore, compatibilmente con le proprie impostazioni, copia la funzione in ogni punto in cui è richiamata; invece con il metodo usato nella seconda versione, con la funzione implementata esternamente alla classe, ad ogni chiamata si inserisce un salto all'unica copia della memoria.



L'**assegnazione di due istanze** di una stessa classe causa la copia dei valori di tutti i dati membro (attributi) da un'istanza all'altra.

Per esempio, dichiariamo due istanze *r1*, *r2* della classe *Rettangolo*, definita nel programma precedente:

```
Rettangolo r1, r2;
```

L'istruzione

```
r1 = r2;
```

equivale alle due seguenti istruzioni di assegnazione:

```
r1.base=r2.base;
r1.altezza=r2.altezza;
```

Dopo l'assegnazione, l'istanza *r1* contiene gli stessi valori di *r2*.

In presenza di un'assegnazione tra istanze della stessa classe, infatti, viene predisposta la copia degli attributi da un'istanza all'altra. Non è quindi necessario definire un operatore di assegnazione specifico per ciascuna classe. Il compilatore crea automaticamente l'**operatore di assegnazione di default** (*default operator overloading*) dell'assegnazione, per ciascuna classe definita dall'utente.



#### AUTOVERIFICA

Domande da 1 a 3 pag. 365

Problemi da 1 a 4 pag. 368

## 5 Costruttore e distruttore

Con la dichiarazione di un'istanza, viene attivato in modo implicito un particolare metodo predefinito, detto **costruttore** (*constructor*) della classe, che consente di creare un oggetto.

Il programmatore può definire all'interno della classe una personalizzazione del costruttore inserendo un metodo avente lo stesso nome della classe: questo è indispensabile per impostare valori di inizializzazione al momento della creazione dell'oggetto. In una classe, infatti, non è consentito inizializzare gli attributi, come normalmente avviene con le variabili.

Per esempio, nella definizione della classe *Rettangolo* si può aggiungere il metodo costruttore *Rettangolo* per l'assegnazione dei valori iniziali posti uguali a zero per la base e per l'altezza del rettangolo.

```
class Rettangolo {
 // attributi
 double base, altezza;
public:
 // metodi
 void Assegna(double, double);
 double Area();
 // costruttore
 Rettangolo() {
 base = 0.0;
 altezza = 0.0;
 }
};
```

## PROGETTO 2

**Costruire una piccola calcolatrice che, dati due numeri, sia in grado di elaborare le quattro operazioni elementari: addizione, sottrazione, moltiplicazione e divisione.**

L'oggetto al centro del problema è la calcolatrice. Essa dispone di quattro metodi, rappresentati dalle quattro operazioni elementari, e prevede l'elaborazione di due dati numerici, che sono i suoi attributi. È anche presente il *costruttore* che consente di inizializzare gli attributi. Il diagramma della classe è quindi il seguente:

| Calcolatrice                                                                   |
|--------------------------------------------------------------------------------|
| a<br>b                                                                         |
| Addizione<br>Sottrazione<br>Moltiplicazione<br>Divisione<br><i>costruttore</i> |

Traducendo questo schema nel linguaggio C++, si ottiene la classe *Calcolatrice*:

```
class Calcolatrice {
public:
 // attributi
 double a, b;
 // metodi
 double Addizione();
 double Sottrazione();
 double Moltiplicazione();
 double Divisione();
 // costruttore
 Calcolatrice() {
 a = 0;
 b = 0;
 }
};
```

L'inserimento della clausola *public*, prima di ogni membro, ne garantisce la visibilità all'esterno della classe, come vedremo nel successivo Paragrafo 6.

I metodi della classe sono qui indicati come prototipi e verranno successivamente specificati nella loro implementazione.

La classe può essere quindi impiegata per creare l'istanza *calc*, alla quale vengono applicate le quattro operazioni aritmetiche:

```
Calcolatrice calc;
```

Un esempio di utilizzo di questa classe è dato dal seguente programma.

### Programma C++

```
// Calcola.cpp: calcolatrice con 4 operazioni
#include <iostream>
using namespace std;
```

```
class Calcolatrice {
public:
 // attributi
 double a, b;
 // metodi
 double Addizione();
 double Sottrazione();
 double Moltiplicazione();
 double Divisione();
 // costruttore
 Calcolatrice() {
 a = 0;
 b = 0;
 }
};

double Calcolatrice::Addizione()
{
 return a+b;
}
double Calcolatrice::Sottrazione()
{
 return a-b;
}
double Calcolatrice::Moltiplicazione()
{
 return a*b;
}
double Calcolatrice::Divisione()
{
 return a/b;
}

int main()
{
 Calcolatrice calc;

 cout << "Primo dato: ";
 cin >> calc.a;
 cout << "Secondo dato: ";
 cin >> calc.b;
 cout << "Risultati delle operazioni" << endl ;
 cout << "Addizione: " << calc.Addizione() << endl;
 cout << "Sottrazione: " << calc.Sottrazione() << endl;
 cout << "Moltiplicazione: " << calc.Moltiplicazione() << endl;
 cout << "Divisione: " << calc.Divisione() << endl;

 return 0;
}
```

Il *costruttore* viene richiamato automaticamente alla creazione di un'istanza. L'utilità dei costruttori è rilevante, in quanto essi permettono di preparare l'oggetto al suo utilizzo in modo chiaro e semplice. I costruttori non prevedono la restituzione di alcun valore e, di conseguenza, non deve essere specificato alcun tipo per questo metodo.

Nell'esempio del rettangolo abbiamo inserito un costruttore per inizializzare gli attributi *base* e *altezza* della classe:

```
class Rettangolo {
 ...
 ...
 // costruttore
 Rettangolo() {
 base = 0.0;
 altezza = 0.0;
 }
};
```

Quando poi si dichiara un'istanza per questa classe, i suoi attributi risultano automaticamente inizializzati:

```
int main()
{
 Rettangolo tovaglia;
 cout << tovaglia.base << endl;
 cout << tovaglia.altezza << endl;
 return 0;
}
```

La prima istruzione di output del programma precedente scrive il valore 0, così come la seconda.

Esiste un altro modo più compatto per personalizzare il costruttore, basato sulle caratteristiche di inizializzazione interna delle classi nel linguaggio C++:

```
Rettangolo() : base(0.0),altezza(0.0){}
```

Questa modalità e quella precedente sono equivalenti.

Come per una qualsiasi altra funzione, anche il costruttore può prevedere l'uso dei parametri. Nell'esempio che segue gli attributi sono inizializzati in base ai valori passati durante la creazione dell'istanza:

```
Rettangolo(double b, double h) : base(b),altezza(h) { }
```

In questo modo durante la creazione di un'istanza avviene anche l'inizializzazione degli attributi. Specificando che il costruttore attende due parametri, non è però possibile dichiarare un'istanza senza parametri. Con riferimento all'esempio precedente del rettangolo, la seguente dichiarazione non risulta corretta:

```
Rettangolo tovaglia;
```

Essa deve quindi essere sostituita dalla seguente:

```
Rettangolo tovaglia(2.0,1.0);
```

I valori tra parentesi vengono utilizzati per inizializzare gli attributi *base* (2.0) e *altezza* (1.0). In una classe esiste un altro metodo particolare che ha lo stesso nome della classe a cui appartiene, preceduto però dal carattere ~ (si legge *tilde* e si scrive con la tastiera nella combinazione Alt+126 sul tastierino numerico). Tale metodo prende il nome di **distruttore** (*destructor*) ed è richiamato automaticamente quando l'oggetto perde la visibilità e la memoria occupata dai suoi attributi viene liberata.

A differenza del costruttore, il distruttore non trova un largo impiego, tranne nelle classi che gestiscono la memoria dinamicamente: in questi casi il distruttore consente di rilasciare lo spazio di memoria allocato per l'oggetto.

Si osservi che il metodo distruttore non può avere argomenti né valori di ritorno.

Il distruttore, se non esplicitamente richiamato, viene eseguito alla chiusura del programma.

Si provi a modificare la precedente classe *Rettangolo* inserendo il prototipo del metodo distruttore *~Rettangolo* nella dichiarazione della classe e successivamente la definizione del metodo con la visualizzazione di un messaggio:

```
class Rettangolo {
 // attributi
 double base, altezza;
public:
 // metodi
 void Assegna(double, double);
 double Area(void);
 Rettangolo() : base(0.0), altezza(0.0){}
 ~Rettangolo();
};

Rettangolo::~Rettangolo() {
 cout << "oggetto eliminato" << endl;
}
```

Eseguendo il programma, alla chiusura dell'elaborazione, sul video compare il messaggio *oggetto eliminato*. Il metodo distruttore, come detto prima, non ha argomenti, né restituisce alcun valore; per questo motivo, prima del nome della funzione, non c'è l'indicazione del tipo restituito.



## COSTRUTTORE DI COPIA

Un particolare costruttore parametrizzato è il **costruttore di copia** (*copy constructor*), cioè un costruttore richiamato durante la creazione di un'istanza inizializzata da un'altra istanza della stessa classe. Utilizzando la classe *Rettangolo* del paragrafo precedente, si può, per esempio, dichiarare il seguente costruttore che raddoppia la base e incrementa di 5 l'altezza:

```
Rettangolo::Rettangolo(Rettangolo& r) {
 base = r.base * 2;
 altezza = r.altezza + 5;
}
```

Si possono poi creare le istanze nel seguente modo:

```
Rettangolo r1(10.0, 2.0);
Rettangolo r2(r1);
```

Il costruttore di copia viene richiamato durante la creazione della seconda istanza (*r2*) e assegna il valore 20 alla propria base e il valore 7 alla propria altezza.

Esiste un **costruttore di copia di default** che è fornito alle classi che non hanno definito un proprio costruttore di copia. È infatti possibile duplicare gli attributi di un'istanza in un'altra istanza della stessa classe, pur non avendo definito un costruttore apposito.

Anche in assenza del costruttore di copia si può infatti dichiarare:

```
Rettangolo r1(10.0, 2.0);
Rettangolo r2(r1);
```

In questo caso gli attributi dell'istanza *r2* assumono gli stessi valori degli attributi di *r1*. Il *costruttore di copia di default* infatti sarebbe simile a:

```
Rettangolo::Rettangolo(Rettangolo& r) : base(r.base),altezza(r.altezza) {}
```

Per verificare quanto detto si può scrivere il seguente codice che utilizza la classe presentata in precedenza e il *costruttore di copia di default*:

```
// Rettangolo.cpp: costruttore di copia
#include <iostream>
using namespace std;

class Rettangolo {
 // attributi
 double base, altezza;
public:
 // metodi
 void Assegna(double, double);
 double Area();
 Rettangolo(double b, double h) : base(b),altezza(h) { }
 ~Rettangolo();
};

void Rettangolo::Assegna(double b, double h)
{
 base = b;
 altezza = h;
}

double Rettangolo::Area()
{
 return base*altezza;
}

Rettangolo::~~Rettangolo() {
 cout << "oggetto eliminato" << endl;
}

int main()
{
 Rettangolo r1(10.0, 2.0);
 Rettangolo r2(r1);
 cout << r2.Area() << endl;

 return 0;
}
```

Il *costruttore di copia di default* è utile quando si devono replicare le istanze di un oggetto di partenza in diverse copie.

Con riferimento all'esempio della classe *Rettangolo*, se si crea l'istanza *r1* nel seguente modo:

```
Rettangolo r1(10.0, 2.0);
```

è possibile poi utilizzare le sue caratteristiche, *base* uguale a 10 e *altezza* uguale a 2, per definire altre istanze:

```
Rettangolo r2(r1), r3(r1);
```

Da quanto abbiamo appreso in precedenza, anche le istanze *r2* e *r3* saranno dotate di una base uguale a 10 e un'altezza uguale a 2.

Si osservi anche la buona leggibilità del codice realizzato secondo i principi della programmazione ad oggetti.



#### AUTOVERIFICA

Domande da 4 a 7 pag. 365-366

Problemi da 5 a 9 pag. 368

## 6 Visibilità dei membri

Negli esempi di classe presentati in precedenza è stata usata la clausola *public* prima della dichiarazione dei metodi:

```
class Rettangolo {
 // attributi
 double base, altezza;
public:
 // metodi
 void Assegna(double, double);
 double Area(void);
 // costruttore
 Rettangolo() {
 base = 0.0;
 altezza = 0.0;
 }
};
```

La clausola **public** definisce il tipo di accesso ai membri della classe: essa è necessaria quando si vuole che i membri specificati dopo di essa siano disponibili anche al di fuori della classe di appartenenza.

In contrapposizione a *public*, esiste la clausola **private** che viene utilizzata quando si vuole indicare al compilatore che tutti i membri che seguono, siano essi attributi o metodi, non possono essere visti al di fuori della classe.

Una clausola di accesso ha efficacia su tutti i membri che la seguono fino a una nuova clausola o alla fine della classe.

La clausola *private* è la clausola di *default*, quindi, in mancanza di dichiarazione sul tipo di accesso, un attributo o un metodo sono considerati di tipo *private*.

Nell'esempio della classe *Rettangolo*, gli attributi *base* e *altezza* sono quindi di tipo *private*, mentre tutti i metodi sono pubblici.

Si noti anche che il costruttore deve essere indicato tra i membri pubblici, altrimenti il compilatore non è in grado di creare le istanze dell'oggetto. Inoltre un metodo dichiarato di tipo *private* non può essere utilizzato al di fuori della classe di appartenenza.

L'uso delle clausole *public* e *private* risponde alle caratteristiche della programmazione ad oggetti, cioè alla possibilità di definire unità di programmazione che hanno al loro interno tutte le risorse descrittive per poter essere usate dalle istruzioni del programma.

I metodi, generalmente pubblici, sono gli strumenti attraverso i quali il programma può utilizzare gli attributi della classe, che hanno normalmente un accesso di tipo privato.

Questo spiega perché nell'esempio della classe *Rettangolo* è stato inserito il metodo *Assegna*: i valori acquisiti da tastiera vengono assegnati agli attributi della classe, non direttamente, ma tramite un metodo pubblico della classe.

Per verificare che effettivamente gli attributi dell'esempio sono di tipo privato, si provi ad inserire nella funzione *main*, l'istruzione per visualizzare il valore della base:

```
cout << tovaglia.base << endl;
```

Durante la compilazione viene segnalato un errore perché l'attributo non è accessibile.

L'uso dell'accesso *private* introduce un livello di protezione dei membri della classe.

Il progetto seguente illustra la differenza tra *public* e *private* nell'accesso agli attributi della classe.

### PROGETTO 3

#### Definire una classe per rappresentare una persona fisica.

Il progetto mostra la modalità per la creazione di una classe contenente i dati anagrafici di una persona, per esempio *codice*, *cognome*, *nome*, oltre ad un'informazione booleana *registrato* che indica se la persona è stata sottoposta ad un'approvazione, o registrazione in generale. Mentre i dati anagrafici sono accessibili in modo pubblico dalle istruzioni del programma, l'approvazione può essere effettuata solo attraverso il metodo *Registra* della classe.

Un altro metodo, *Mostra*, consente la visualizzazione dei dati della persona: se si vuole che la visualizzazione sia consentita solo per le persone effettivamente registrate, occorre usare un metodo della classe, perché il controllo sul valore dell'attributo *registrato* è possibile solo dall'interno della classe, essendo dichiarato con accesso *private*.

| Anagrafica |
|------------|
| codice     |
| cognome    |
| nome       |
| registrato |
| Registra   |
| Mostra     |

Dopo aver dichiarato la classe *Anagrafica*, l'istanza *persona* viene creata con l'istruzione:

```
Anagrafica persona; // oggetto persona
```

Un esempio completo di utilizzo di questa classe è dato dal seguente programma:

#### Programma C++

```
// Anag.cpp: persona fisica
#include <iostream>
#include <string>
using namespace std;
```



```

class Anagrafica {
public:
 // attributi
 int codice;
 string cognome;
 string nome;
 // metodi
 void Registra();
 void Mostra();
private:
 // attributo
 bool registrato;
};

// funzione membro per registrare un'anagrafica
void Anagrafica::Registra()
{
 registrato = true;
}

// funzione membro per visualizzare un'anagrafica
void Anagrafica::Mostra()
{
 if (registrato) {
 cout << codice << endl;
 cout << cognome << endl;
 cout << nome << endl;
 }
 else cout << "non registrato" << endl;
}

// funzione principale
int main()
{
 Anagrafica persona; // oggetto persona

 // inserimento dei dati
 cout << "Codice: ";
 cin >> persona.codice;
 cout << "Cognome: ";
 cin >> persona.cognome;
 cout << "Nome: ";
 cin >> persona.nome;

 // operazioni sull'oggetto
 persona.Registra();
 persona.Mostra();

 return 0;
}

```

Si provi, per esercizio, ad inserire nella funzione principale (*main*) precedente la seguente istruzione:

```
persona.registrato = true;
```

Si ottiene un errore di compilazione, in quanto l'attributo *registrato*, di tipo *private*, non è accessibile dall'esterno della classe.

Osservando il programma precedente, ci si può chiedere: “perché il programmatore dovrebbe imporsi di non disporre di alcune parti della classe che sta implementando, rendendole *private*?” Una prima motivazione consiste nell'utilità di imporre dei vincoli concettuali, in modo da implementare correttamente il codice, magari molto lungo e scritto da più mani. Inoltre, come vedremo in seguito parlando di ereditarietà, la creazione di una classe, nello sviluppo di software commerciale, potrebbe prevedere la possibilità di non fornire l'accesso a parti implementate ad hoc per la classe.

Nella programmazione ad oggetti l'interazione con gli oggetti prende il nome di **scambio di messaggi**.

Come abbiamo visto, l'invio di un **messaggio** è indicato con la sintassi:

oggetto.membro

dove *membro* può essere un metodo o un attributo. Quando il membro è un metodo, nel messaggio occorre indicare anche i parametri rispettandone il tipo.

L'invio di un messaggio a un oggetto può richiedere l'esecuzione di un metodo o il contenuto di un attributo.

L'insieme dei messaggi, che consentono l'interazione con l'oggetto, rappresenta la sua **interfaccia**.

L'interfaccia separa l'oggetto dal mondo esterno ed è costituita dall'elenco dei membri messi a disposizione dall'oggetto.

Tutti i membri *public* costituiscono l'*interfaccia* dell'oggetto con l'esterno.

Solo i componenti dell'interfaccia sono utilizzabili per scambiare *messaggi* che consentono l'interazione con l'oggetto.



## FUNZIONI FRIEND

Le funzioni non appartenenti ad una classe non possono accedere ai membri privati della classe. Esiste tuttavia la possibilità di definire una funzione che sia in grado di accedere ai membri privati di una classe, pur non appartenendovi. Questo diritto è concesso dalla classe stessa specificando il prototipo della funzione, preceduto dalla parola chiave **friend** (*amica*), ottenendo le cosiddette **friend function**, come mostrato nel progetto seguente.

## PROGETTO 4

**I prodotti venduti in un magazzino sono organizzati per reparto. Scrivere il programma per inserire i prodotti, assegnandoli ai reparti, oltre che per ottenere l'elenco completo dei prodotti e l'elenco dei prodotti appartenenti ad un reparto richiesto da tastiera.**

La classe *Prodotto* è descritta con il seguente diagramma:

| Prodotto                                                 |
|----------------------------------------------------------|
| descrizione<br>prezzo<br>numrep                          |
| Registra<br>Visualizza<br><i>friend</i> ElencaPerReparto |

La classe *Prodotto* possiede tre attributi, con accesso privato, che ne descrivono le proprietà, e i metodi pubblici per registrare e visualizzare le informazioni dei prodotti. Il diagramma precedente è tradotto in linguaggio C++ con la seguente dichiarazione:

```
class Prodotto {
 // attributi
 string descrizione;
 float prezzo;
 int numrep;
public:
 // metodi
 void Registra(string, float, int);
 void Visualizza();
 friend void ElencaPerReparto(int reparto, Prodotto prodotto[]);
};
```

I prodotti sono contenuti in un array di oggetti, che rappresenta un array di istanze della classe *Prodotto*; la dichiarazione dell'array è la seguente:

```
Prodotto prod[MAXPROD]; // array di oggetti Prodotto
```

Il valore *MAXPROD*, numero massimo dei prodotti, è dichiarato come costante all'inizio del programma per permettere una sua eventuale modifica successiva in modo semplice:

```
const unsigned int MAXPROD = 10;
```

Il programma, oltre alle funzioni che sono metodi della classe, utilizza due funzioni per la gestione dei prodotti:

- *InserisciProdotti*

Con una ripetizione enumerativa, per ogni prodotto, la funzione acquisisce da tastiera le informazioni del prodotto (descrizione, prezzo, numero del reparto) e le assegna agli attributi della classe *Prodotto* tramite il metodo *Registra*.

- *ListaProdotti*

Con una ripetizione enumerativa, per ogni prodotto, la funzione visualizza la descrizione e il prezzo del prodotto tramite il metodo *Visualizza*.

La funzione *ElencaPerReparto* per operare ha bisogno di due parametri: il numero di reparto e l'array dei reparti.

Di seguito è riportato il codice completo del programma.

### Programma C++

```
// Prodotti.cpp: prodotti e reparti
#include <iostream>
#include <string>
using namespace std;

const unsigned int MAXPROD = 10;
```

```
class Prodotto {
 // attributi
 string descrizione;
 float prezzo;
 int numrep;
public:
 Prodotto() : prezzo(0.0), numrep(0) {descrizione = "";}
 // metodi
 void Registra(string, float, int);
 void Visualizza();
 friend void ElencaPerReparto(int reparto, Prodotto prodotto[]);
};

void Prodotto::Registra(string s, float p, int n)
{
 descrizione = s;
 prezzo = p;
 numrep = n;
}

void Prodotto::Visualizza()
{
 cout << descrizione << ", ";
 cout << prezzo << endl;
}

// funzione friend
void ElencaPerReparto(int reparto, Prodotto prodotto[])
{
 for (int i=0; i<MAXPROD; i++)
 if (reparto == prodotto[i].numrep)
 prodotto[i].Visualizza();
}

// prototipi delle funzioni
void InserisciProdotti();
void ListaProdotti();

// dichiarazione delle istanze
Prodotto prod[MAXPROD]; // array di oggetti prodotto

// funzione principale
int main()
{
 int i;
 cout << "-----" << endl;
 InserisciProdotti();
 ListaProdotti();
 cout << "-----" << endl;
 cout << "Inserisci il numero di reparto: ";
 cin >> i;
 ElencaPerReparto(i, prod);

 return 0;
}
```

```

// inserimento prodotti
void InserisciProdotti()
{
 string desc;
 float prz;
 int np;
 char risp;

 for (int i=0; i<MAXPROD; i++) {
 cout << "Prodotto " << i+1 << endl;
 do {
 cout << "Descrizione prodotto: ";
 cin >> desc;
 cout << "Prezzo prodotto: ";
 cin >> prz;
 cout << "Numero reparto: ";
 cin >> np;
 prod[i].Registra(desc, prz, np);
 cout << "Confermi (s/n)? ";
 cin >> risp;
 } while (risp != 's');
 }
} //InserisciProdotti

// lista completa dei prodotti
void ListaProdotti()
{
 cout << "-----" << endl;
 cout << "Lista dei prodotti" << endl;
 cout << "-----" << endl;
 for (int i=0; i<MAXPROD; i++) {
 cout << "Prodotto " << i+1 << ": ";
 prod[i].Visualizza();
 }
} // ListaProdotti

```

Si può osservare che l'utente esterno vede la numerazione dei prodotti a partire da 1, mentre all'interno del programma la numerazione segue l'indice dell'array e quindi parte da 0. Si noti anche la differente sintassi tra i metodi e la funzione *friend*: i primi sono richiamati con la notazione

NomeOggetto.NomeMetodo()

mentre la seconda segue la normale sintassi delle funzioni:

NomeFunzione(argomenti).

La funzione *ElencaPerReparto* è dichiarata *friend function* della classe *Prodotto*, perché deve poter accedere all'attributo privato *numrep*. La scelta di vietare l'accesso a questo attributo con il contemporaneo consenso alla funzione *friend* sembra contraddittorio. Si potrebbero, infatti, prevedere soluzioni differenti, per esempio rendendo pubblico l'attributo o inserendo la funzione *friend* tra le funzioni membro della classe. Da questa osservazione si deduce che l'impiego di questo tipo di funzioni deve essere limitato a casi particolari, in quanto un uso eccessivo implica un errore concettuale di progettazione della classe. Si noti infine che la clausola **friend** può essere impiegata anche con nomi di classe, laddove si voglia concedere a un'altra classe l'accesso ai membri privati.



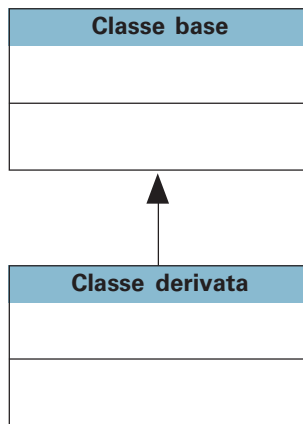
#### AUTOVERIFICA

Domande da 8 a 9 pag. 366  
Problemi da 10 a 11 pag. 368

## 7 Ereditarietà

L'**ereditarietà** è un concetto fondamentale della programmazione ad oggetti e rappresenta la possibilità di creare nuove classi a partire da una classe già esistente (**classe base** o *superclasse*). La nuova classe eredita tutti gli attributi e i metodi della classe base e può essere arricchita con nuovi attributi e nuovi metodi. La classe così ottenuta si chiama **classe derivata**.

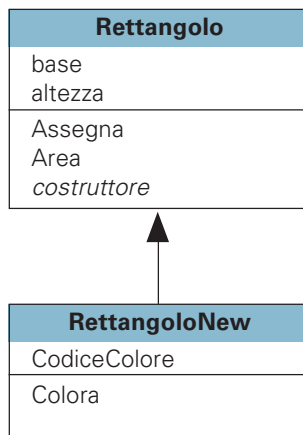
Come già detto nel Paragrafo 3, la derivazione di una classe si rappresenta con i diagrammi di classe attraverso i due rettangoli delle classi uniti da una freccia che parte dalla classe derivata verso la classe base:



Per esempio, volendo arricchire di altre funzionalità la classe *Rettangolo* vista in precedenza, possiamo *ereditare* da quest'ultima tutte le caratteristiche e poi introdurre le nuove componenti. Si tratta di un vero e proprio riciclo del software, poiché quanto già scritto e collaudato è incapsulato nella classe *Rettangolo* e viene inglobato nella nuova classe che si va a creare. Questa nuova classe dispone di tutto ciò che la classe *Rettangolo* metteva a disposizione ai propri utilizzatori, ma dà la possibilità di integrare nuovi attributi e nuovi metodi.

Supponiamo di voler aggiungere l'attributo *CodiceColore* per indicare con un numero il colore di riempimento della figura e il metodo *Colora* per assegnare un valore a questo attributo che ha un accesso privato.

La classe base *Rettangolo* e la classe derivata *RettangoloNew* sono rappresentate con il seguente diagramma:



Nel linguaggio C++ la classe derivata si dichiara con la seguente sintassi:

```
class ClasseDerivata : public ClasseBase {
 // attributi
 ...
 // metodi
 ...
};
```

I nomi della classe derivata e della classe base sono separati dai due punti (:). La clausola *public* è necessaria se si desidera disporre delle componenti pubbliche della classe di base anche nella classe derivata. Per esempio, la codifica della nuova classe *RettangoloNew* e del suo metodo *Colora* è la seguente:

```
class RettangoloNew : public Rettangolo {
 // attributi
 int CodiceColore;
public:
 // metodi
 void Colora(int);
};

void RettangoloNew::Colora(int c)
{
 CodiceColore = c;
}
```

In questo modo abbiamo definito una nuova classe che dispone sia delle potenzialità (proprietà e azioni) della classe *Rettangolo* sia delle nuove implementazioni.

Ovviamente il programma che utilizza la classe derivata deve contenere anche la dichiarazione della classe base e l'implementazione dei suoi metodi.

Definendo poi un nuovo oggetto *tovaglia* come istanza della classe *RettangoloNew*, il *main* del programma può utilizzare sia i metodi della classe base (*Assegna* e *Area*), sia i nuovi metodi definiti nella classe derivata (*Colora*).

```
int main()
{
 double b, h;
 int col;
 RettangoloNew tovaglia;

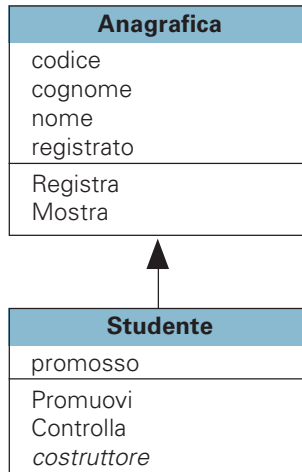
 // inserimento misure
 cout << "Misura della base: ";
 cin >> b;
 cout << "Misura dell'altezza: ";
 cin >> h;
 tovaglia.Assegna(b, h);
 // inserimento colore
 cout << "Numero colore: ";
 cin >> col;
 tovaglia.Colora(col);
 // calcolo area
 cout << "Area = " << tovaglia.Area() << endl;

 return 0;
}
```

## PROGETTO 5

### Costruire una classe per rappresentare uno studente, derivandola dalla classe base *Anagrafica*.

Con riferimento alla classe presentata nel Paragrafo 6, a partire dalla classe di base *Anagrafica*, si può derivare la classe *Studente*.



La nuova classe aggiunge alla classe base l'attributo *promosso*, con accesso privato. Questo dato, di tipo booleano, viene aggiornato tramite il metodo *Promuovi()*. Un secondo metodo, *Controlla()*, consente di verificare se l'alunno è stato promosso. Di seguito è riportato il programma completo in linguaggio C++ che mostra la codifica della classe derivata e dei suoi metodi, oltre ad alcuni esempi di utilizzo delle funzioni.

### Programma C++

```
// Studente.cpp: anagrafica studente
#include <iostream>
#include <string>
using namespace std;

class Anagrafica {
public:
 // attributi
 int codice;
 string cognome;
 string nome;
 // metodi
 void Registra();
 void Mostra();
private:
 // attributo
 bool registrato;
};

// funzione per registrare un'anagrafica
void Anagrafica::Registra()
{
 registrato = true;
}
```



```
// funzione per visualizzare un'anagrafica
void Anagrafica::Mostra()
{
 if (registrato) {
 cout << codice << endl;
 cout << cognome << endl;
 cout << nome << endl;
 }
 else cout << "non registrato" << endl;
}

class Studente : public Anagrafica {
 // attributi
 bool promosso;
public:
 // metodi
 void Promuovi();
 void Controlla();
 // costruttore
 Studente() {
 promosso = false;
 }
};

// funzione per registrare la promozione
void Studente::Promuovi()
{
 promosso = true;
}

// funzione per controllare la promozione
void Studente::Controlla()
{
 if (promosso) cout << "studente promosso" << endl;
 else cout << "studente non promosso" << endl;
}

// funzione principale
int main()
{
 Studente stud1; // istanza della classe
 char risp;

 // inserimento dei dati
 cout << "Codice: ";
 cin >> stud1.codice;
 cout << "Cognome: ";
 cin >> stud1.cognome;
 cout << "Nome: ";
 cin >> stud1.nome;

 // operazioni sull'oggetto
 stud1.Registra();
 stud1.Controlla();
 cout << "Promosso (s/n)? ";
 cin >> risp;
 if (risp == 's') stud1.Promuovi();
 stud1.Mostra();
 stud1.Controlla();

 return 0;
}
```

Come si può notare è possibile definire un costruttore anche nella classe derivata *Studente*. In questo caso inizializza l'attributo *promosso* assegnando ad esso il valore *false*. Il costruttore viene richiamato alla dichiarazione di un'istanza della classe derivata *Studente*.

Quando non viene esplicitamente definito il costruttore di una classe derivata, il compilatore crea un **costruttore di default** per richiamare il costruttore della classe di base al momento della creazione di un'istanza della classe derivata.

Più precisamente, i costruttori vengono richiamati dalla classe di base a quella derivata, risalendo l'ordine in cui le classi stesse sono richiamate. Se, durante la creazione dell'istanza, il compilatore incontra la dichiarazione di *classe2*, derivata da *classe1*, e quest'ultima risulta essere una classe primitiva, cioè non derivata, il compilatore richiama il suo costruttore. Completata l'esecuzione del costruttore di *classe1*, viene eseguito il costruttore della classe derivata *classe2*.

L'attributo *promosso* della classe *Studente* è un esempio significativo di incapsulamento e dell'utilità dell'interfaccia, tipico della programmazione orientata agli oggetti. Si può notare, infatti, che questo attributo assume inizialmente il valore *false* e può essere trasformato in *true* solamente richiamando il metodo *Promuovi*. In questo modo non è possibile ripristinare il valore *false* dopo che lo studente è stato promosso, rendendo l'attributo *promosso* più aderente alla realtà.

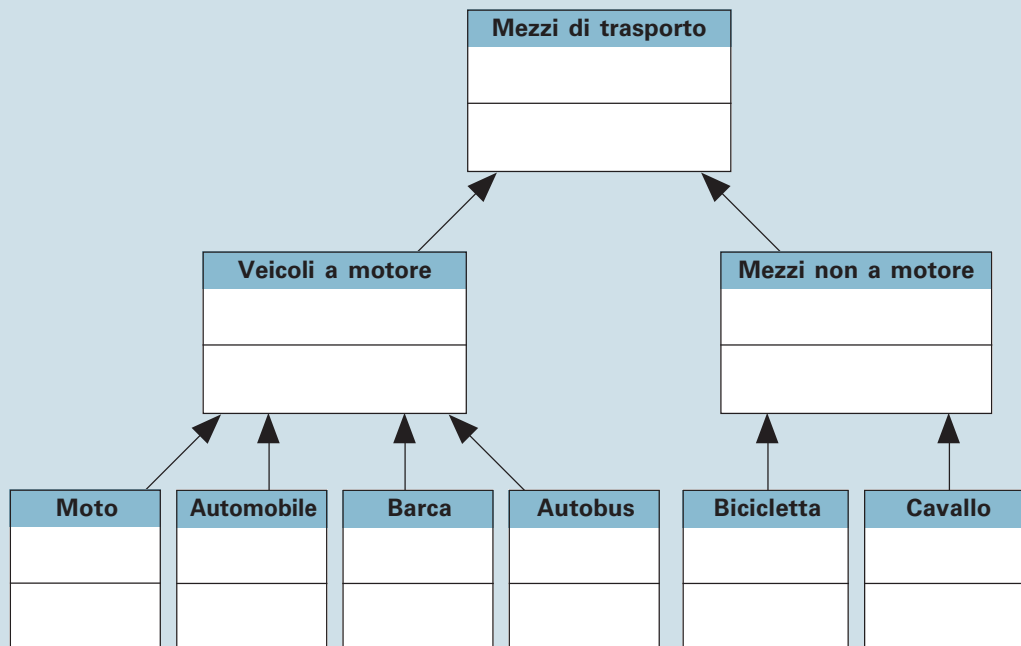


## GERARCHIA DELLE CLASSI ED EREDITARIETÀ MULTIPLA

Vediamo più in dettaglio alcune considerazioni teoriche già presentate nel Paragrafo 3.

La classe che è stata derivata da un'altra usando l'ereditarietà prende il nome di **sottoclasse**. La classe generatrice di una sottoclasse si chiama **superclasse** o **sopraclasse**.

Queste relazioni tra le classi individuano una **gerarchia di classi** che nasce da un processo di specializzazione. Infatti le classi che si trovano in cima alla gerarchia sono le più generali e man mano che si scende si trovano classi più specializzate.



La gerarchia delle classi si può descrivere con un modello grafico che usa i **diagrammi delle classi**: esso è costituito da un grafo orientato in cui i nodi sono rappresentati dalle classi e gli archi individuano la presenza di una relazione di ereditarietà. La direzione della freccia è dalla sottoclasse verso la sopraclasse.

Nell'esempio, la classe *Moto* è sottoclasse della classe *Veicoli a motore*, che a sua volta è sottoclasse di *Mezzi di trasporto*.

In alto ci sono le classi più generiche che diventano più specializzate man mano si scende lungo la gerarchia. Nel diagramma sono state indicate le classi solo attraverso il loro nome, tralasciando l'elenco degli attributi e dei metodi.

La sottoclasse eredita dalla sopraclasse tutti gli attributi e tutti i metodi, evitando di ripetere la descrizione degli elementi comuni nelle sottoclassi. L'ereditarietà consente di condividere le similarità tra le classi con la possibilità di inserire le differenze.

La nuova classe si differenzia dalla sopraclasse in due modi:

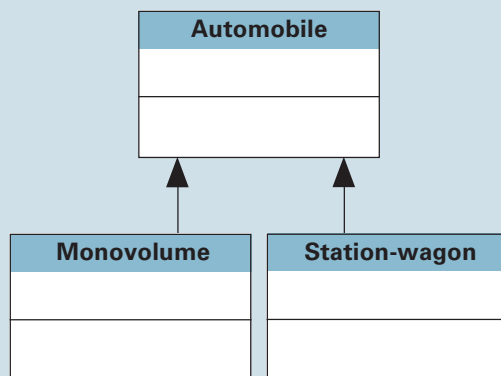
- **per estensione**, quando la sottoclasse aggiunge nuovi attributi e metodi che si sommano a quelli ereditati;
- **per ridefinizione**, quando la sottoclasse ridefinisce i metodi ereditati. In pratica viene data un'implementazione diversa di un metodo. Si crea un nuovo metodo che ha lo stesso nome del metodo ereditato da una sopraclasse, ma che ha una funzionalità diversa. Quando ad un oggetto della sottoclasse arriva un messaggio che richiama un metodo ridefinito, viene eseguito il nuovo codice e non quello che era stato ereditato.

Una sottoclasse ridefinisce un metodo perché vuole associare un comportamento diverso oppure perché vuole utilizzare un algoritmo più efficiente rispetto a quello della sopraclasse. In sostanza, l'ereditarietà serve a definire nuove classi, derivate da altre, che ereditano gli attributi e i metodi, con la possibilità di ridefinirli o di aggiungerne di nuovi.

Esistono due tipi di ereditarietà: *singola* e *multipa*.

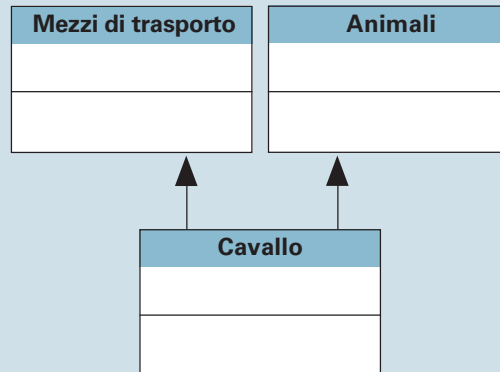
L'**ereditarietà singola** si verifica quando una sottoclasse deriva da un'unica sopraclasse. Usando i diagrammi delle classi, l'ereditarietà singola è indicata usando un'unica freccia uscente dalla sottoclasse e diretta verso la sopraclasse.

La sottoclasse ha una sola classe genitrice, ma questo non vieta alla sopraclasse di essere ereditata da più classi. Un esempio di ereditarietà semplice è rappresentata dalla sottoclasse *Monovolume*. Possiamo pensare di definire un'altra sottoclasse, per esempio *Station-wagon*, derivata sempre dalla sopraclasse *Automobile*.



L'**ereditarietà multipla**, invece, comporta che una classe derivi da due o più sopraclassi. In questo modo si possono fondere insieme gli attributi e i metodi provenienti da classi diverse per crearne una nuova. Usando i diagrammi delle classi, l'ereditarietà multipla è indicata con due o più frecce che escono dalla sottoclasse e sono dirette verso le sopraclassi.

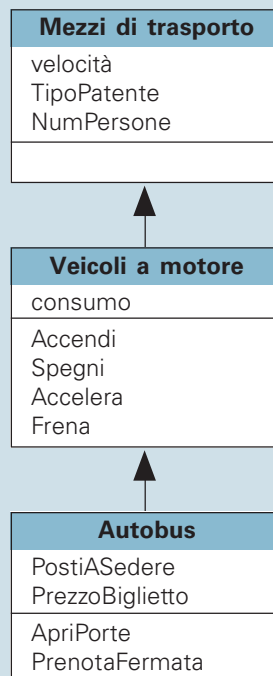
Per esempio si può considerare la classe *Cavallo*, derivante dalla classe *Mezzi di trasporto* e dalla classe *Animali*.



L'ereditarietà multipla è usata meno frequentemente rispetto all'ereditarietà singola: per questo motivo non tutti i linguaggi di programmazione la prevedono.

L'esempio seguente spiega l'ereditarietà sviluppando una gerarchia composta da tre classi, complete di attributi e metodi.

Le classi considerate sono: *Mezzi di trasporto*, *Veicoli a motore* e *Autobus*. Si ipotizza che tutti gli attributi e i metodi siano pubblici. In questo modo possono essere ereditati dalle sottoclassi.



La classe *Mezzi di trasporto* è caratterizzata dagli attributi *velocità*, *TipoPatente*, *NumPersone*.

La classe *Veicoli a motore* ha come attributi *consumo* e come metodi *Accendi*, *Spegni*, *Accelera*, *Frena*.

La classe *Autobus* ha gli attributi *PostiAsedere*, *PrezzoBiglietto* e i metodi *ApriPorte* e *PrenotaFermata*.

La classe *Veicoli a motore* è sottoclasse di *Mezzi di trasporto* ed è sopraclasse di *Autobus*.

Le sottoclassi sono definite per estensione delle sopraclassi.

L'**ereditarietà** è uno strumento che consente il riutilizzo programmabile del software creato, cioè non è necessario partire da zero nella costruzione di un nuovo programma, basta utilizzare un insieme di classi già definite che hanno comportamenti simili a quelli del nuovo programma ed estenderle. L'estensione consente di inserire gli elementi aggiuntivi usati dalla nuova applicazione. Tutte le classi che sono già state scritte non vengono ricodificate, con il vantaggio di utilizzare classi già collaudate e con un buon livello di affidabilità, cioè prive di errori, perché sono state usate e controllate in altri programmi. Il riutilizzo, favorito dall'ereditarietà, consente un notevole risparmio di tempo nella produzione del software. La programmazione orientata agli oggetti riserva molta importanza alle **librerie**. Tutti i linguaggi *Object-Oriented* (OO) forniscono librerie di classi molto vaste che possono essere sfruttate usando il meccanismo dell'ereditarietà. Quindi oltre ad imparare la sintassi di un linguaggio OO, è fondamentale conoscere la gerarchia e la struttura delle classi che compongono le sue librerie.



### AUTOVERIFICA

Problemi da 12 a 15 pag. 368-369

## 8 Polimorfismo e overloading

Abbiamo visto nel paragrafo precedente che con l'ereditarietà la classe derivata è definita come estensione della classe base, con l'aggiunta di nuovi attributi e nuovi metodi. In più, nella programmazione ad oggetti, è possibile ridefinire i metodi ereditati.

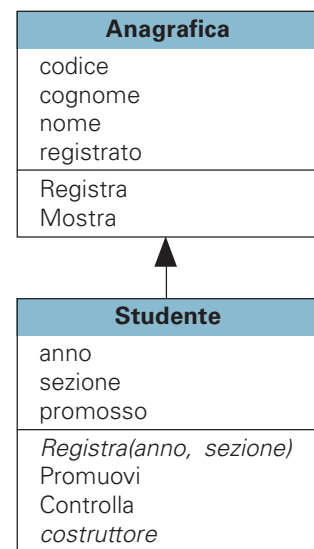
Il termine *polimorfismo*, in generale, indica la capacità di assumere più forme. Nella programmazione orientata agli oggetti è strettamente legato all'ereditarietà e alla ridefinizione dei metodi. Nell'ereditarietà, le classi derivate hanno la possibilità di ridefinire i metodi ereditati (mantenendo lo stesso nome) oppure lasciarli inalterati perché già soddisfacenti.

Il **polimorfismo** degli oggetti è la capacità di far assumere agli oggetti comportamenti diversi, determinati da una diversa **implementazione** dei suoi membri o da una diversa **firma**, cioè i metodi della classe ereditata differiscono dai metodi della classe base nel numero oppure nel tipo dei parametri, o in entrambi.

Nei linguaggi ad oggetti sono presenti due tipi di polimorfismo:

- l'**overloading** o *sovraccarico* dei metodi, che vedremo in questo paragrafo;
- l'**overriding** o *sovrapposizione* dei metodi, che sarà presentato nel paragrafo successivo.

Si consideri la classe definita nell'esempio del paragrafo precedente per rappresentare uno studente e si supponga di voler aggiungere ad essa anche gli attributi *anno* di corso e *sezione* di appartenenza dello studente, oltre a un metodo per registrare queste due informazioni che sono ad accesso privato. Questo metodo può avere lo stesso nome del metodo *Registra* definito nella classe base, ma opera in maniera differente: riceve due parametri e assegna il valore agli attributi *anno* e *sezione*.



La classe *Studiante* viene codificata nel seguente modo:

```
class Studiante : public Anagrafica {
 // attributi
 int anno;
 char sezione;
 bool promosso;
public:
 // metodi
 void Registra(int, char);
 void Promuovi();
 void Mostra();
 void Controlla();
 // costruttore
 Studiante() {
 promosso = false;
 anno = 0;
 }
};

// funzione per registrare anno e sezione
void Studiante::Registra(int a, char s)
{
 anno = a;
 sezione = s;
}
```

Una situazione come questa si chiama **overloading** (sovraccarico) del metodo (*function overloading*), cioè metodi con lo stesso nome hanno argomenti diversi (e solitamente comportamenti diversi). Si dice che i metodi differiscono per la *firma*, cioè nel numero oppure nel tipo di parametri, o in entrambi.

Occorre osservare che il metodo *Registra()* della classe di base *Anagrafica* continua ad essere disponibile e richiamabile: basta non specificare alcun argomento durante la chiamata. L'inserimento di un costruttore nella definizione di una classe non è un esempio di *overloading*, perché il costruttore viene associato alla classe in sostituzione del costruttore di *default*.



## MEMBRI PROTECTED

Nel Paragrafo 6, relativo alla visibilità dei membri, sono stati definiti gli accessi *public* e *private* ai membri di una classe. Il linguaggio C++ consente anche l'utilizzo della clausola *protected*.

La clausola **protected** indica la visibilità di un membro solamente alla classe stessa e alle classi derivate.

Si consideri, per esempio, la classe *Studiante*, presentata nel paragrafo precedente come classe derivata dalla classe base *Anagrafica* con l'overloading del metodo *Registra*. Questo metodo non può accedere all'attributo *registrato* della classe base perché questo è di tipo *private*.

Quindi se il metodo *Registra* della classe *Studente* venisse modificato con l'aggiunta dell'assegnazione del valore *true* all'attributo *registrato*, il compilatore segnalerebbe un errore.

```
// funzione per registrare anno e sezione
void Studente::Registra(int a, char s)
{
 anno = a;
 sezione = s;
 registrato = true;
}
```

Per rendere possibile questa operazione occorre dichiarare *protected* l'attributo *registrato* all'interno della classe base *Anagrafica*. L'attributo continua ad essere non visibile alle funzioni esterne alla classe, ma è disponibile per i metodi della classe derivata.

```
class Anagrafica {
public:
 // attributi
 int codice;
 string cognome;
 string nome;
 // metodi
 void Registra();
 void Mostra();
protected:
 // attributo
 bool registrato;
};
```

Si noti che è possibile sostituire l'istruzione

```
registrato = true;
```

con la chiamata al metodo della classe base

```
Anagrafica::Registra();
```



#### AUTOVERIFICA

Domande da 10 a 16 pag. 366-367

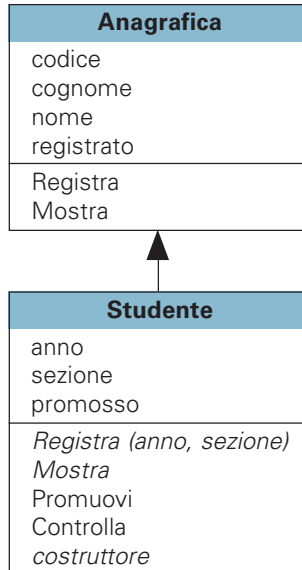
## 9 Riscrittura di metodi nelle classi derivate

Nella programmazione ad oggetti e nel linguaggio C++, oltre all'overloading, si può usare anche l'**overriding** (sovrapposizione) dei metodi, che consiste nel sovrascrivere, nella classe derivata, un metodo ereditato, cambiando le istruzioni che descrivono il comportamento, ma mantenendo uguale la *firma*, quindi il numero e il tipo dei parametri: al momento dell'esecuzione, il programma decide quale metodo utilizzare a seconda dell'oggetto, base o derivato, al quale viene applicato il metodo.

Per esempio, nella dichiarazione della classe *Studente* nel paragrafo precedente il costruttore assegna il valore di inizializzazione 0 all'attributo *anno*, oltre al valore *false* all'attributo *promosso*.

In questo modo possiamo controllare se uno studente è stato registrato e assegnato ad un anno di corso della scuola.

Il metodo *Mostra* deve essere perciò riscritto, rispetto alla classe base, utilizzando una diversa condizione nel controllo dello studente e aggiungendo la visualizzazione dei due nuovi dati, *anno* e *sezione*.



La rappresentazione della classe *Studente* in linguaggio C++ risulta così modificata:

```
class Studente : public Anagrafica {
 // attributi
 int anno;
 char sezione;
 bool promosso;
public:
 // metodi
 void Registra(int, char);
 void Promuovi();
 void Mostra();
 void Controlla();
 // costruttore
 Studente() {
 promosso = false;
 anno = 0;
 }
};
```

La codifica del metodo *Mostra*, a cui è stato applicato l'*overriding* nella classe derivata, è la seguente:

```
// funzione per visualizzare uno studente
void Studente::Mostra()
{
 Anagrafica::Mostra();
 if (anno != 0)
 cout << anno << sezione << endl;
}
```



Si noti che, in presenza di un *overriding*, per richiamare il metodo che è stato sovrapposto, è necessario specificare la classe di appartenenza davanti all'*operatore di risoluzione della visibilità (::)*. Per rafforzare i concetti presentati in questo paragrafo, vediamo un altro esempio simile alla classe *Studente*.

## PROGETTO 6

### Costruire una classe per rappresentare un docente, derivandola dalla classe base *Anagrafica*.

Con riferimento alla classe presentata nel Paragrafo 6, a partire dalla classe base *Anagrafica*, si deriva la classe *Docente*.

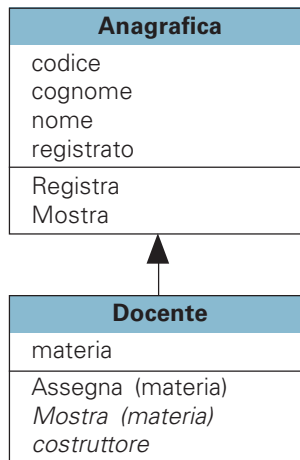
Nella classe derivata *Docente*, che eredita gli attributi *codice*, *cognome*, *nome* e *registrato* dalla classe *Anagrafica*, aggiungiamo l'attributo *materia* per rappresentare la materia di insegnamento.

Aggiungiamo anche due nuovi metodi:

- *Assegna*, con un parametro, per registrare la materia di insegnamento;
- *Mostra*, con un parametro, per visualizzare un docente della materia richiesta.

Il metodo *Mostra* della classe base viene sottoposto a *overloading*, perché modifica la sua *firma*.

La classe *Docente* contiene anche un costruttore che inizializza l'attributo *materia* con la stringa *\*\*\*\**. In questo modo è possibile controllare se al docente è stata assegnata la materia di insegnamento.



Il codice seguente rappresenta la classe derivata *Docente* e l'implementazione dei due metodi *Assegna* e *Mostra*.

```

class Docente : public Anagrafica {
 // attributi
 string materia;
public:
 // metodi
 void Assegna(string);
 void Mostra(string);
 // costruttore
 Docente() {
 materia = "****";
 }
};

```

```
// funzione per registrare la materia
void Docente::Assegna(string m)
{
 materia = m;
}

// funzione per visualizzare un docente della materia
void Docente::Mostra(string m)
{
 if (materia == m) {
 cout << codice << ", ";
 cout << cognome << ", ";
 cout << nome << endl;
 }
}
```

## PROGETTO 7

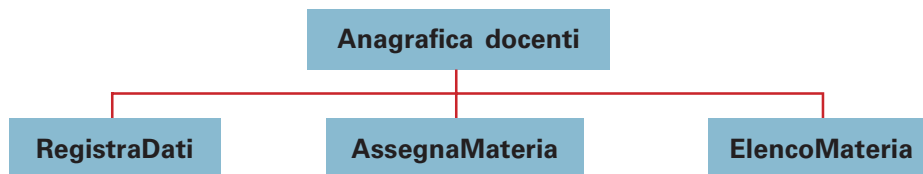
**Dato un elenco di 50 docenti di una scuola, dopo aver inserito i dati anagrafici di ciascuno (codice, cognome e nome), si deve assegnare ad essi la materia di insegnamento. Il programma deve anche consentire di ottenere l'elenco dei docenti che insegnano una materia richiesta da tastiera.**

La classe per rappresentare i docenti è quella presentata nell'esempio precedente. Qui si utilizza un array di 50 oggetti di tipo *Docente*, che viene dichiarato con l'istruzione:

```
Docente prof[50]; // array di oggetti
```

Il codice del docente coincide con la posizione nell'array aumentata di 1. In questo modo l'utente esterno considera la numerazione dei docenti da 1 a 50, mentre il programma segue la numerazione dell'array che parte da 0.

Le scelte del programma sono elencate in un menu che organizza le funzioni di gestione secondo lo schema seguente:



- *RegistraDati*  
La funzione richiede all'utente il codice del docente e controlla che sia un numero compreso tra 1 e 50. Acquisisce poi il cognome e nome del docente e registra i dati nella componente dell'array avente indice uguale al valore del codice diminuito di 1.
- *AssegnaMateria*  
La funzione richiede all'utente il codice del docente e controlla che sia un numero compreso tra 1 e 50. Acquisisce poi il nome della materia e assegna la materia al docente usando l'apposito metodo della classe.
- *ElencoMateria*  
La funzione acquisisce il nome della materia e con una ripetizione esamina tutte le componenti dell'array, visualizzando il codice, il cognome e il nome dei docenti della materia richiesta, se sono presenti nell'array.

**Programma C++**

```
// Docente.cpp: anagrafica docente
#include <iostream>
#include <string>
using namespace std;

class Anagrafica {
public:
 // attributi
 int codice;
 string cognome;
 string nome;
 // metodi
 void Registra();
 void Mostra();
private:
 // attributo
 bool registrato;
};

// funzione membro per registrare un'anagrafica
void Anagrafica::Registra()
{
 registrato = true;
}

// funzione membro per visualizzare un'anagrafica
void Anagrafica::Mostra()
{
 if (registrato) {
 cout << codice << endl;
 cout << cognome << endl;
 cout << nome << endl;
 }
 else cout << "non registrato" << endl;
}

class Docente : public Anagrafica {
 // attributi
 string materia;
public:
 // metodi
 void Assegna(string);
 void Mostra(string);
 // costruttore
 Docente() {
 materia = "***";
 }
};

// funzione per registrare la materia
void Docente::Assegna(string m)
{
 materia = m;
}
```

```

// funzione per visualizzare un docente della materia
void Docente::Mostra(string m)
{
 if (materia == m) {
 cout << codice << ", ";
 cout << cognome << ", ";
 cout << nome << endl;
 }
}

// prototipi delle funzioni di gestione
void RegistraDati();
void AssegnaMateria();
void ElencoMateria();

Docente prof[50]; // array di oggetti

// funzione principale
int main()
{
 int scelta;

 do {
 // menu delle scelte
 cout << "-----" << endl;
 cout << "Anagrafica docenti" << endl;
 cout << "-----" << endl;
 cout << "1. Registra dati anagrafici" << endl;
 cout << "2. Assegna materia di insegnamento" << endl;
 cout << "3. Elenco per materia" << endl;
 cout << "4. Fine lavoro " << endl;
 cout << "Scelta: ";
 cin >> scelta;
 switch(scelta) {
 case 1:
 RegistraDati();
 break;
 case 2:
 AssegnaMateria();
 break;
 case 3:
 ElencoMateria();
 break;
 }
 } while (scelta != 4);

 return 0;
}

// funzioni di gestione
void RegistraDati()
{
 int i;

```

```

do {
 cout << "Numero docente (1..50): ";
 cin >> i;
} while ((i<1) || (i>50));
i--;
prof[i].codice = i+1;
cout << "Cognome: ";
cin >> prof[i].cognome;
cout << "Nome: ";
cin >> prof[i].nome;
prof[i].Registra();
} // RegistraDati

void AssegnaMateria()
{
 int i;
 string mat;

 do {
 cout << "Numero docente (1..50): ";
 cin >> i;
 } while (i<1 || i>50);
 cout << "Materia: ";
 cin >> mat;
 i--;
 prof[i].Assegna(mat);
} // AssegnaMateria

void ElencoMateria()
{
 string mat;

 cout << "Quale materia? ";
 cin >> mat;
 for (int i=0; i<50; i++)
 prof[i].Mostra(mat);
} // ElencoMateria

```



### AUTOVERIFICA

Problemi da 16 a 22 pag. 369



## CLASSI E OVERLOADING DEGLI OPERATORI

L'**operator overloading** (sovraccarico dell'operatore) indica la possibilità di definire nuove funzionalità ai vari operatori del linguaggio (++ , -- , = , ecc.), in modo che il compilatore sia in grado di comprendere il loro impiego nei nuovi tipi di dati creati dal programmatore. L'*operator overloading* non è altro che un *function overloading* applicato a funzioni particolari chiamate **operator** seguite dall'operatore a cui si riferiscono. Di seguito vengono presentati due esempi di *overloading* degli operatori associati ad una classe. Nei progetti precedenti abbiamo usato il tipo *string* per dichiarare variabili non numeriche. Per poter utilizzare questo tipo, come già detto in precedenza, occorre inserire nel programma la direttiva `#include <string>`, che rende disponibile la libreria del linguaggio C++.

Nel linguaggio C e nelle versioni meno recenti di C++ questo tipo di dato non era disponibile e le stringhe venivano memorizzate in *array* di caratteri. Ancora oggi, per compatibilità con programmi e funzioni del passato, si ha a disposizione la libreria **<cstring>** con tutte le funzioni per la manipolazione di *array* di caratteri, quali **strcpy** per la copia di *array* di caratteri o **strcmp** per il loro confronto, presenti nella libreria *<string.h>* del linguaggio C.

Supponiamo quindi di non disporre del tipo *string* e di voler implementare la classe *Stringa* per la gestione degli *array* di caratteri. Per le diverse manipolazioni degli *array* si impiegano le funzioni di libreria, mentre con l'*operator overloading* si possono definire gli operatori per le usuali operazioni di assegnazione, confronto, ecc.

## PROGETTO 8

### Definire l'operatore = per l'assegnazione di array di caratteri.

Il progetto introduce una nuova classe *Stringa* e sovraccarica l'*operatore* = di assegnazione in modo che si possano assegnare array di caratteri anche ad oggetti di questo tipo. Il diagramma della classe *Stringa* è il seguente:

| Stringa               |
|-----------------------|
| s                     |
| operatore =<br>Mostra |

L'attributo *s* rappresenta un array di caratteri.

Il primo metodo rappresenta l'*operator overloading* per l'operazione di assegnazione di array di caratteri, il secondo consente di visualizzare un oggetto dichiarato come istanza della classe *Stringa*.

La codifica della classe *Stringa* e dei suoi metodi è riportata di seguito.

```
class Stringa {
 char s[30];
public:
 // metodi
 Stringa operator= (const char *sorgente);
 void Mostra();
};

Stringa Stringa::operator= (const char *sorgente)
{
 strcpy(s, sorgente);

 return *this;
}

void Stringa::Mostra()
{
 cout << s << endl;
}
```

Per poter utilizzare la funzione predefinita *strcpy* per la copia tra stringhe, occorre inserire nel programma la direttiva:

```
#include <cstring>
```

Per verificare le funzionalità della classe si può scrivere il seguente programma:

```
#include <iostream>
#include <cstring>
using namespace std;

// funzione principale
int main()
{
 Stringa s1, s2; // istanze della classe

 s1 = "Milano";
 s2 = "Roma";
 s1.Mostra();
 s2.Mostra();

 return 0;
}
```

La funzione *operator =* ha un valore di ritorno per consentire anche assegnazioni ripetute come:

```
s1 = s2 = "Napoli";
```

Il valore di ritorno è un oggetto di classe *Stringa*. La funzione *operator =* è una funzione membro della classe *Stringa*; per indicare che il valore di ritorno è la classe stessa si usa l'istruzione

```
return *this;
```

La parola riservata **this** del linguaggio C++ indica l'indirizzo della classe di appartenenza (quindi *this* è un puntatore implicito all'oggetto della classe). La parola *this* può essere tradotta in questo contesto come *oggetto corrente* per la funzione membro che si sta definendo.

Si deve osservare che l'*overloading* dell'operatore *=* proposto in questo esempio consente di assegnare valori costanti (racchiusi tra doppi apici) ad un oggetto di tipo *Stringa*. L'assegnazione tra due oggetti dello stesso tipo *Stringa*, come nella seguente istruzione:

```
s2 = s1;
```

è invece garantita dall'**operatore di assegnazione di default** per la classe *Stringa*, che è già stato presentato nel Paragrafo 4.

Un secondo esempio risolve il problema di rappresentare il confronto tra array di caratteri usando l'operatore *==*, normalmente non applicabile agli array.

## PROGETTO 9

### Definire l'operatore == per il confronto tra due array di caratteri.

Alla classe *Stringa* presentata nel progetto precedente, viene aggiunto un ulteriore *overloading* applicato all'operatore == per il confronto tra oggetti dichiarati come istanze della classe *Stringa*:

| Stringa                                          |
|--------------------------------------------------|
| s                                                |
| <i>operatore =</i><br><i>friend operatore ==</i> |

In questo caso, l'*overloading* sull'operatore deve essere necessariamente di tipo **friend**.

La codifica modificata della classe *Stringa* e dell'*overloading* dell'operatore == è riportata di seguito: l'operatore di confronto ha come parametri le due stringhe da confrontare e restituisce un valore booleano (*true/false*) come esito del confronto.

```
class Stringa {
 char s[30];
public:
 // metodi
 Stringa operator= (const char* sorgente);
 friend bool operator== (Stringa& a, Stringa& b);
 void Mostra();
};

Stringa Stringa::operator= (const char* sorgente)
{
 strcpy(s, sorgente);

 return *this;
}

void Stringa::Mostra()
{
 cout << s << endl;
}

bool operator== (Stringa &a, Stringa &b)
{
 bool c;

 if (strcmp(a.s, b.s) == 0) c = true;
 else c = false;

 return c;
}
```

Nell'*operator overloading* viene utilizzata la funzione *strcmp*, disponibile nella libreria *<cstring>*.



Si provi a verificare la funzionalità dell'operatore `==` eseguendo il programma con le seguenti istruzioni nel *main*:

```
#include <iostream>
#include <cstring>
using namespace std;

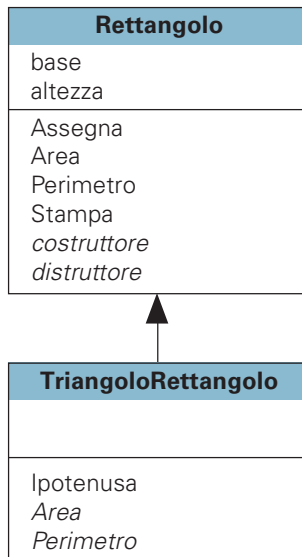
// funzione principale
int main(void) {
 Stringa s1, s2; // istanze della classe

 s1 = "Milano";
 s2 = "Parigi";
 if (s1 == s2) cout << "Stringhe uguali" << endl;
 else cout << "Stringhe diverse" << endl;

 return 0;
}
```

## 10 Funzioni virtuali

Si considerino le due seguenti classi: la classe base *Rettangolo* e la classe derivata *TriangoloRettangolo*.



La prima rappresenta una variante rispetto alla classe utilizzata nei paragrafi precedenti: sono presenti alcuni metodi aggiuntivi per il calcolo del perimetro e per la visualizzazione delle caratteristiche del rettangolo: *base*, *altezza*, misure del perimetro e dell'area.

La seconda oltre ad ereditare gli attributi e i metodi della classe base, ha un metodo aggiuntivo per il calcolo dell'ipotenusa e ridefinisce, mediante *overriding*, i due metodi per il calcolo di area e perimetro.

Come spiegato in precedenza, i membri privati non sono accessibili dalle classi derivate. Pertanto gli attributi *base* e *altezza* nella classe *Rettangolo* devono essere dichiarati **protected**: in questo modo gli attributi continuano ad essere non visibili alle funzioni esterne alla classe, ma sono disponibili per i metodi della classe derivata.

La codifica delle due classi con l'implementazione delle funzioni membro è la seguente:

```
class Rettangolo {
protected:
 // attributi
 double base, altezza;
public:
 // metodi
 void Assegna(double, double);
 double Area();
 double Perimetro();
 void Stampa();
 Rettangolo() : base(0.0),altezza(0.0) { }
 ~Rettangolo();
};

void Rettangolo::Assegna(double b, double h)
{
 base = b;
 altezza = h;
}

double Rettangolo::Area()
{
 return base*altezza;
}

double Rettangolo::Perimetro()
{
 return (base + altezza) * 2;
}

void Rettangolo::Stampa()
{
 cout << base << endl;
 cout << altezza << endl;
 cout << Area() << endl;
 cout << Perimetro() << endl;
}

Rettangolo::~~Rettangolo() {
 cout << "oggetto eliminato" <<endl;
}

class TriangoloRettangolo : public Rettangolo {
public:
 // metodi
 double Ipotenusa();
 double Area();
 double Perimetro();
};

double TriangoloRettangolo::Ipotenusa()
{
 return sqrt(base*base + altezza*altezza);
}
```

```
double Triangolo Rettangolo::Area()
{
 return (base*altezza) / 2;
}

double Triangolo Rettangolo::Perimetro()
{
 return base + altezza + Ipotenusa();
}
```

Poiché la funzione membro per il calcolo dell'ipotenusa del triangolo rettangolo richiede la funzione predefinita *sqrt*, per il calcolo della radice quadrata, occorre inserire nel programma la direttiva di inclusione della libreria `<cmath>`:

```
#include <cmath>
```

La funzione principale per un'esecuzione con dati di prova è la seguente:

```
int main()
{
 Rettangolo r1;
 Triangolo Rettangolo tr1;

 r1.Assegna(2.0, 4.0);
 tr1.Assegna(3.0, 4.0);

 r1.Stampa();
 tr1.Stampa();

 return 0;
}
```

L'esecuzione del programma produce risultati non corretti per l'area e il perimetro del triangolo rettangolo.

La classe *Triangolo Rettangolo* ha infatti ereditato anche il metodo *Stampa()* che contiene un riferimento alle funzioni membro *Area()* e *Perimetro()*. Sia pure in presenza di *overriding* nella classe derivata, il metodo *Stampa()*, nonostante venga applicato ad un'istanza della classe *Triangolo Rettangolo*, richiama il procedimento di calcolo del rettangolo e produce risultati errati. Per ovviare a questi inconvenienti, occorre premettere la clausola **virtual** alla dichiarazione dei metodi della classe base, che saranno sottoposti a polimorfismo.

La dichiarazione della classe base deve quindi essere modificata in questo modo:

```
class Rettangolo {
protected:
 // attributi
 double base, altezza;
public:
 // metodi
 void Assegna(double, double);
 virtual double Area();
 virtual double Perimetro();
 void Stampa();
 Rettangolo() : base(0.0), altezza(0.0) { }
 ~Rettangolo();
};
```

Essa contiene la specifica *virtual* prima della dichiarazione delle funzioni membro *Area()* e *Perimetro()*.

Con questa modifica il programma produce in output i risultati desiderati.



### OSSERVAZIONI SULLA PROGRAMMAZIONE ORIENTATA AGLI OGGETTI

Gli esempi presentati nei paragrafi precedenti ci consentono di fare alcune considerazioni sulla specificità della *programmazione orientata agli oggetti* (OOP).

La necessità di definire un oggetto nasce ogni qualvolta ci troviamo di fronte a un gruppo di dati che hanno qualcosa in comune e a cui si devono applicare le funzioni.

Si è già detto che la programmazione OOP è realizzata a strati, cioè sovrapponendo nuovo codice a quello preesistente.

Si supponga di avere una classe per la gestione di una rubrica telefonica. Con la programmazione tradizionale, dovendo per esempio applicare la gestione ad un'anagrafica di clienti, occorre avere a disposizione il codice sorgente per poter apportare le modifiche. Nella programmazione orientata agli oggetti, invece, la classe originale (*classe base*) è lasciata intatta, in quanto le variazioni al codice sono implementate in uno strato superiore (*classe derivata*).

Il vantaggio di questa metodologia di programmazione risulta ancora più evidente se si pensa al caso in cui la classe base sia stata acquistata da un produttore di software e non si disponga del codice sorgente (si pensi, per esempio, alle funzioni matematiche disponibili con il compilatore del linguaggio C++). Nella programmazione tradizionale saremmo costretti a riscrivere tutto quanto già implementato da altri nella classe base, per esempio la gestione dell'archivio dei nominativi, la loro ricerca, ecc. Il codice realizzato inoltre non sarebbe sicuro, cioè non testato, e potrebbe contenere errori e inefficienze, spesso presenti nelle prime versioni di un nuovo software. Nella programmazione OOP la classe base è invece priva di errori (*bug-free*), in quanto già testata. Le nuove implementazioni del codice non modificano la classe base che, quindi, continua a rimanere priva di errori. Il nuovo codice si sovrappone all'esistente richiedendo dei collaudi esclusivamente della nuova parte realizzata.

Con questo modo di programmare si ha quindi la possibilità di riutilizzare una grande quantità di codice, non dovendo necessariamente tenere in considerazione le caratteristiche del problema per il quale tale codice era stato creato originariamente. Classi generiche possono infatti ricoprire le esigenze di diverse problematiche, senza che si debba disporre del codice sorgente. Le implementazioni specifiche usufruiscono delle parti comuni già scritte e disponibili, riducendo i tempi e i costi di realizzazione di nuovo software.

### AUTOVERIFICA

Domande da 17 a 18 pag. 367-368

Problemi da 23 a 25 pag. 369



### MATERIALI ONLINE

**2. Programmi eseguibili e videate dei progetti del capitolo**

**3. Lezioni multimediali sulla programmazione ad oggetti in C++**

## DOMANDE

### Classi

- 1 Completa le frasi seguenti utilizzando una tra le parole elencate alla fine della domanda.
- a) Gli elementi base della OOP si chiamano .....
  - b) Gli ..... definiscono le caratteristiche di un oggetto.
  - c) I ..... definiscono le funzionalità dell'oggetto.
- metodi, pubblici, class, attributi, oggetti, istanza, privati, bit, byte, costruttori, function, operator**
- 2 Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
- a) È possibile accedere agli attributi di una classe attraverso i metodi V F
  - b) Con l'incapsulamento tutto ciò che si riferisce ad un oggetto è racchiuso all'interno dell'oggetto stesso V F
  - c) Una classe è un'unità di programmazione che può essere utilizzata in un solo programma V F
  - d) Le classi più generali si chiamano classi base o superclassi V F
- 3 Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
- a) Gli attributi non incidono sulla dimensione delle istanze V F
  - b) La dimensione di un'istanza coincide con il numero di byte occupati dai suoi metodi V F
  - c) Se nella classe sono indicati solo i prototipi dei metodi, per la loro implementazione si utilizza la sintassi: tipo nome\_classe::nome\_metodo V F

### Costruttore e distruttore

- 4 Quale delle seguenti frasi spiega in modo corretto il significato di costruttore?
- a) Metodo attivato quando si crea un'istanza.
  - b) Metodo attivato quando si crea una classe.
  - c) Metodo attivato quando si crea un metodo.
  - d) Metodo attivato quando si crea un attributo.
- 5 Volendo creare un costruttore per la classe *Lettere*, quale delle seguenti dichiarazioni è corretta:

a)

```
class Lettere {
... public:
Lettere() {}
...
};
```

c)

```
class Lettere {
... public:
constructor() {}
...
};
```

b)

```
class Lettere {
...
public:
constructor Lettere() {}
...
};
```

d)

```
class Lettere {
...
public:
~ Lettere() {}
...
};
```

- 6 Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
- a) Non implementando il costruttore e il distruttore, il compilatore li crea automaticamente (default)
  - b) Una classe può avere più di un costruttore
  - c) Se si implementa un costruttore, il compilatore include comunque quello di default
  - d) Il distruttore serve nella gestione dinamica della memoria per liberare spazio non più necessario
  - e) Se il programmatore ha definito un costruttore che prevede dei parametri, non è possibile dichiarare un'istanza senza argomenti

V F

V F

V F

V F

V F

- 7 Date le dichiarazioni

```
class Dato {
public:
 int a;
 double b;
 Dato(): a(3), b(5.0) {}
};
```

Qual è l'output del seguente programma?

```
int main() {
 Dato dato;
 cout << dato.a << ", ";
 cout << dato.b << endl;
 return 0;
}
```

- a) 3.0, 5.0
- b) 3, 5
- c) 8.0
- d) 3 5

### Visibilità dei membri

- 8 Completa le frasi seguenti utilizzando una tra le parole elencate alla fine della domanda.
- a) I costruttori delle classi devono essere definiti .....
  - b) Gli attributi e i metodi sono definiti ..... se nessun altro, eccetto la classe stessa, deve accedervi.
  - c) Gli attributi e i metodi sono definiti ..... se sono disponibili a tutti.
- metodi, pubblici, class, attributi, oggetti, istanze, privati, bit, byte, costruttori, distruttori**
- 9 Quando una funzione non appartenente a una classe può accedere ai membri privati della classe?
- a) In nessun caso.
  - b) In caso acceda ai metodi e non agli attributi.
  - c) Solo se dichiarata *friend* nella classe.
  - d) Sempre.
  - e) Solo se dichiarata *friend*.

### Ereditarietà, polimorfismo e overloading

- 10 Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
- a) Le classi derivate ereditano tutti i membri della classe di base
  - b) Le classi derivate non possono essere arricchite con funzionalità aggiuntive rispetto alla classe di base, è necessario integrare la classe di base
  - c) Una classe derivata dispone di tutti gli attributi e dei metodi della classe d'origine: questa caratteristica è nota con il termine incapsulamento

V F

V F

V F

- 11** Quali delle seguenti frasi definisce meglio il significato di ereditarietà multipla?
- a) La possibilità di derivare lo stesso metodo di una classe base in più classi derivate.
  - b) La possibilità di derivare gli stessi attributi di una classe base in più classi derivate.
  - c) La possibilità di derivare una classe da una o più classi.
  - d) La possibilità di derivare più classi da una classe base.
- 12** Quale delle seguenti definizioni meglio descrive il significato del termine *overloading*?
- a) L'aumento del numero di attributi di una classe.
  - b) L'arricchimento delle capacità di un metodo.
  - c) L'impostazione dell'accesso privato ai membri di una classe.
  - d) Un'impostazione assunta dal sistema se non specificata dall'utente.
- 13** Associa ad ogni tipo di funzione della colonna di sinistra il corrispondente significato scegliendo i termini nella colonna di destra.
- |                         |                                                                            |
|-------------------------|----------------------------------------------------------------------------|
| a) member function      | 1) assicura a una funzione non membro l'accesso ai membri privati          |
| b) friend function      | 2) ridefinisce una funzione già esistente per l'impiego di altri parametri |
| c) function overloading | 3) identifica un metodo di una classe                                      |
- 14** Avendo definito le classi A e B, a quali dati ha accesso la classe B?

```
class A {
 int a,b;
public:
 int c,d;
protected:
 int e,f;
};
```

```
class B: A {
 . . .
 . . .
};
```

- a) a tutti i dati
- b) solo a quelli protetti
- c) solo a quelli pubblici
- d) sia ai dati protetti sia a quelli pubblici
- e) a tutti gli attributi di A
- f) a nessun membro.

- 15** Associa ad ogni elemento della colonna di sinistra i corrispondenti privilegi, all'accesso dei membri di una classe di base, della colonna di destra:
- |                            |                                |
|----------------------------|--------------------------------|
| a) Funzioni di tipo friend | 1) Protetti, pubblici          |
| b) Classe derivata         | 2) Pubblici                    |
| c) Funzioni membro         | 3) Privati, protetti, pubblici |
| d) main()                  |                                |
- 16** Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
- a) Un attributo protetto è visto da una funzione non membro
  - b) Un attributo protetto è visto da una classe derivata
  - c) Un membro protetto è visto da una funzione *friend*
  - d) Un metodo protetto è visto da una funzione non membro
  - e) Un metodo privato è visto da una classe derivata

|   |   |
|---|---|
| V | F |
| V | F |
| V | F |
| V | F |
| V | F |

## Riscrittura di metodi

- 17** Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
- a) Una classe derivata può coprire un metodo della classe base scrivendo un nuovo metodo avente lo stesso nome e la stessa firma
  - b) Un metodo può essere ridefinito, consentendo il passaggio di parametri in numero e di tipo diversi
  - c) L'overriding dei metodi modifica un metodo all'interno della classe base
  - d) L'overriding dei metodi sovrascrive, nella classe derivata, un metodo ereditato, cambiando le istruzioni che ne descrivono il comportamento

|   |   |
|---|---|
| V | F |
| V | F |
| V | F |
| V | F |

- 18** Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
- a) La programmazione orientata agli oggetti può essere definita *bug-free* poiché riduce il rischio di commettere errori quando si utilizzano i codici già testati V F
  - b) La programmazione tradizionale permette di riutilizzare con maggior facilità i codici già testati rispetto alla programmazione orientata agli oggetti V F
  - c) Durante l'analisi di un oggetto è importante individuare eventuali membri comuni ad altri oggetti in modo da definire una classe di base condivisibile con altre applicazioni V F
  - d) Nella programmazione orientata agli oggetti, la classe derivata è lasciata intatta, in quanto le variazioni al codice sono implementate nella classe base V F

## PROBLEMI

### Creazione di classi

- 1** Creare una classe *Atleta* per rappresentare le informazioni di un giocatore.
- 2** Aggiungere alla classe precedente un metodo per assegnare all'atleta il nome della squadra dove gioca.
- 3** Creare una classe *Quadrato* con l'attributo *lato* e i metodi per il calcolo del perimetro e dell'area.
- 4** Creare una classe *CD* per le informazioni su un disco musicale e i metodi per aggiornare il prezzo.

### Costruttore e distruttore

- 5** Creare una classe *Conto* e un costruttore che assegni il valore iniziale 0 al suo saldo.
- 6** Creare la classe *Orologio* con le ore, minuti e secondi. I metodi della classe consentono di azzerare l'orario, correggere l'ora, leggere l'orario completo con ore, minuti e secondi.
- 7** Definire la classe per rappresentare un triangolo rettangolo: inserire i metodi per calcolare area e perimetro, oltre al costruttore per inizializzare gli attributi.
- 8** Il computer di bordo di un'automobile è in grado di calcolare il consumo medio conoscendo i km percorsi e i litri di carburante consumati. Definire la classe *Automobile* con il metodo per il calcolo oltre al costruttore per inizializzare gli attributi.
- 9** Definire la classe *Libro* con gli attributi: titolo, autore, anno, editore, città e soggetto. Implementare i metodi per inserire e visualizzare i dati dei libri, aggiungendo anche un costruttore e un distruttore.

### Visibilità

- 10** Dopo aver creato l'oggetto *Utente* composto dagli attributi privati *nome* e *parola d'ordine* (password), implementare la funzione per l'inserimento di tali attributi.
- 11** Gli studenti delle scuole superiori di una città sono iscritti a diversi Istituti. Scrivere il programma per inserire gli studenti, assegnandoli alle scuole, oltre che per ottenere l'elenco completo degli studenti e l'elenco degli studenti iscritti ad una determinata scuola. Le scuole sono identificate da un codice numerico che va da 1 al numero totale delle scuole. Utilizzare una funzione *friend* per ritrovare gli studenti della scuola richiesta da tastiera.

### Ereditarietà

- 12** Derivare dalla classe *Quadrato* del Problema 3 una nuova classe *Rettangolo* aggiungendo un secondo lato per l'altezza e riscrivendo i metodi per il calcolo del perimetro e dell'area.



- 13 Creare una classe Libro con titolo, editore, prezzo e derivare la classe Rivista aggiungendo l'attributo sulla periodicità (settimanale, mensile, trimestrale).
- 14 Creare la classe Veicolo e derivare le classi Automobile e Autobus: aggiungere gli attributi opportuni e il metodo che consente di visualizzare i valori degli attributi.
- 15 Derivare dalla classe Conto del Problema 5 una nuova classe ContoCorrente: aggiungere i metodi Preleva, che decrementa il saldo della cifra fornita come parametro, Versa, che incrementa il saldo, e Mostra, che visualizza il saldo attuale.

## Polimorfismo

- 16 Derivare, da una classe Anagrafica, la classe Dipendente aggiungendo gli attributi per la mansione e lo stipendio e modificando il metodo di registrazione della classe base con l'assegnazione della mansione che svolge nell'azienda.
- 17 Derivare, da una classe Anagrafica, la classe Cliente aggiungendo gli attributi per il telefono e la Partita IVA e modificando il metodo di registrazione della classe base con l'assegnazione della Partita IVA.
- 18 Costruire la classe base Dipendente e derivare le classi Operaio, Impiegato e Dirigente. Nella classe base l'attributo stipendio viene dichiarato protetto in modo che possa essere modificato anche con i metodi delle classi derivate.
- 19 Una classe Solido consente di rappresentare una figura solida e di calcolarne il volume; derivare da essa la classe Bilancia che consente di modificare il metodo per il calcolo, in modo da ottenere anche il peso del solido conoscendone il peso specifico.
- 20 Data una figura e il suo colore di sfondo, derivare una nuova figura caratterizzata anche dal colore del bordo. Modificare il metodo che assegna il colore di sfondo nella classe base, aggiungendo anche l'assegnazione del colore del bordo.
- 21 Creare la classe per rappresentare una retta avente equazione  $y = ax + b$  con il metodo per controllare se un generico punto appartiene alla retta. Derivare la classe per rappresentare la parabola  $y = ax^2 + bx + c$ , modificando il metodo per controllare se un punto appartiene alla parabola.
- 22 Durante un referendum una società specializzata nei sondaggi raccoglie i dati all'uscita di un seggio elettorale. In qualunque momento la società deve essere in grado di fornire una statistica delle opinioni raccolte: numero degli intervistati, numero dei sì, numero dei no e relative percentuali. Creare la classe per il sondaggio al referendum e derivare poi una nuova classe per il caso in cui il sondaggio riguardi le elezioni con scelta tra 4 liste.

## Overloading degli operatori e funzioni virtuali

- 23 Applicare l'overloading all'operatore +, in modo che si possano sommare due array di caratteri attraverso la loro concatenazione, utilizzando la funzione predefinita *strcat*. Verificare che l'array ottenibile dalla concatenazione non superi la capacità dell'array di destinazione.
- 24 Applicare l'overloading all'operatore ++, in modo che si possa costruire un nuovo array di caratteri concatenando un array di caratteri a se stesso. Verificare che la dimensione dell'array sia in grado di contenere il risultato della concatenazione.
- 25 La classe Cubo ha come attributo il lato e possiede come metodi le funzioni per il calcolo del volume e per la visualizzazione delle caratteristiche della figura solida (lato e volume). Derivare dalla classe Cubo la classe Sfera che deve ridefinire il metodo per il volume attraverso una formula di calcolo diversa. Utilizzare le funzioni virtuali in modo che il metodo di visualizzazione ereditato dalla classe base produca in output i risultati corretti.



## SCHEDA DI AUTOVALUTAZIONE

### CONOSCENZE

- ☐ Astrazione
- ☐ Programmazione procedurale e programmazione strutturata
- ☐ Classi, attributi, metodi
- ☐ Incapsulamento e information hiding
- ☐ Costruttore e distruttore
- ☐ Costruttore di copia
- ☐ Accesso pubblico e privato ai membri
- ☐ Messaggi e interfaccia
- ☐ Funzioni friend
- ☐ Ereditarietà
- ☐ Gerarchia delle classi
- ☐ Ereditarietà multipla
- ☐ Polimorfismo
- ☐ Membri protetti
- ☐ Overloading dei metodi
- ☐ Overriding dei metodi
- ☐ Overloading degli operatori nelle classi
- ☐ Funzioni virtuali
- ☐ Vantaggi della programmazione ad oggetti

### ABILITÀ

- ☐ Applicare i principi della programmazione ad oggetti utilizzando il linguaggio C++
- ☐ Creare una classe con attributi e metodi
- ☐ Dichiarare le istanze di una classe
- ☐ Definire un costruttore e un distruttore della classe
- ☐ Utilizzare i costruttori di copia
- ☐ Utilizzare le funzioni friend
- ☐ Creare una classe derivata
- ☐ Saper utilizzare le clausole public, private
- ☐ Rendere protetti i membri di una classe
- ☐ Applicare l'overloading ai metodi nelle classi derivate
- ☐ Saper riscrivere i metodi nelle classi derivate
- ☐ Applicare l'overloading agli operatori nelle classi
- ☐ Individuare le funzioni da rendere virtuali



**SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 663**

# 7

**parte terza**

La programmazione

## **Strutture di dati e algoritmi**

### **OBIETTIVI DI APPRENDIMENTO**

In questo capitolo imparerai ad organizzare i dati in strutture di tipo avanzato e a costruire algoritmi per rendere efficienti le operazioni di memorizzazione e di ricerca.

Conoscerai i concetti di template per le funzioni e per le classi.

Imparerai ad usare la libreria standard dei template (STL) con i contenitori, gli algoritmi e gli iteratori.

Sarai quindi in grado di utilizzare la metodologia di programmazione generica, basata sulla parametrizzazione delle componenti.

Ricerca in un array  
Ordinamento di un array  
Array paralleli  
Tabella  
Pila  
Coda  
Liste  
Template di funzioni  
Template di classi  
Libreria standard STL  
Contenitore per le stringhe  
Contenitore vector  
Gestione delle eccezioni  
Iteratori  
Contenitori list e map  
Algoritmi standard

## 1 Ricerca di un elemento in un array

In questo capitolo mettiamo in pratica tutto quello che abbiamo imparato sinora. Vedremo come implementare alcuni algoritmi di impiego comune per la soluzione di problemi generali e altri per una gestione efficace di grandi quantità di dati. Iniziamo con il problema della ricerca di un dato all'interno di un array.

La **ricerca sequenziale di un elemento in un array** consiste nell'esaminare le componenti dell'array in sequenza, vale a dire una dopo l'altra a partire dalla componente con il primo indice, secondo l'ordine che esse occupano nell'array, per controllare se tra le componenti dell'array ne esiste una avente valore uguale a quello cercato.

La ricerca si arresta quando è stato trovato un elemento uguale a quello di confronto o quando l'indice che controlla tutti gli elementi del vettore è arrivato all'ultimo elemento.

### PROGETTO 1

**Cercare un elemento in un array e fornire la sua posizione oppure un messaggio di non trovato.**

La struttura dati, le procedure di richiesta della dimensione dell'array e acquisizione degli elementi di un array sono identiche a quelle presentate nei programmi sugli array del Capitolo 5.

**Struttura dati utilizzata:**

MAX = 100  
vett = ARRAY[MAX] di tipo intero

**Dati di input**

dim : dimensione indicata dall'utente  
n : valore da cercare

**Dati di output**

posto: posizione dell'elemento cercato  
oppure  
messaggio indicante l'esito negativo della ricerca.

**Programma C++**

```
// Ricerca.cpp: ricerca in un array
#include <iostream>
using namespace std;

const int MAX = 100;
// variabili globali
int n; // numero da ricercare
int posto; // posizione del primo numero trovato

// prototipi delle funzioni
int ChiediDimensione();
void CaricaVettore(int v[], int d);
int Trova(int v[], int d);
```

```
// funzione principale
int main()
{
 int dim; // dimensione indicata dall'utente
 int vett[MAX]; // vettore in cui cercare il numero
 bool trovato; // indicatore se trovato

 dim = ChiediDimensione();
 cout << "Carica gli elementi del vettore" << endl;
 CaricaVettore(vett,dim);
 cout << "Numero da cercare ";
 cin >> n;
 trovato = Trova(vett, dim);
 if (trovato)
 cout << n << " si trova alla posizione " << posto << endl;
 else
 cout << n << " non e' presente nel vettore" << endl;

 return 0;
}

// dimensione dell'array
int ChiediDimensione()
{
 int d;
 do {
 cout << "Dimensione dell'array: ";
 cin >> d;
 } while (d<1 || d>MAX);

 return d;
} // ChiediDimensione

// caricamento delle componenti
void CaricaVettore(int v[], int d)
{
 for (int i=0; i<d; i++) {
 cout << "Elemento di posto " << i << ": ";
 cin >> v[i];
 }
} // CaricaVettore

// ricerca dell'elemento nell'array
int Trova (int v[], int d)
{
 bool s = false;

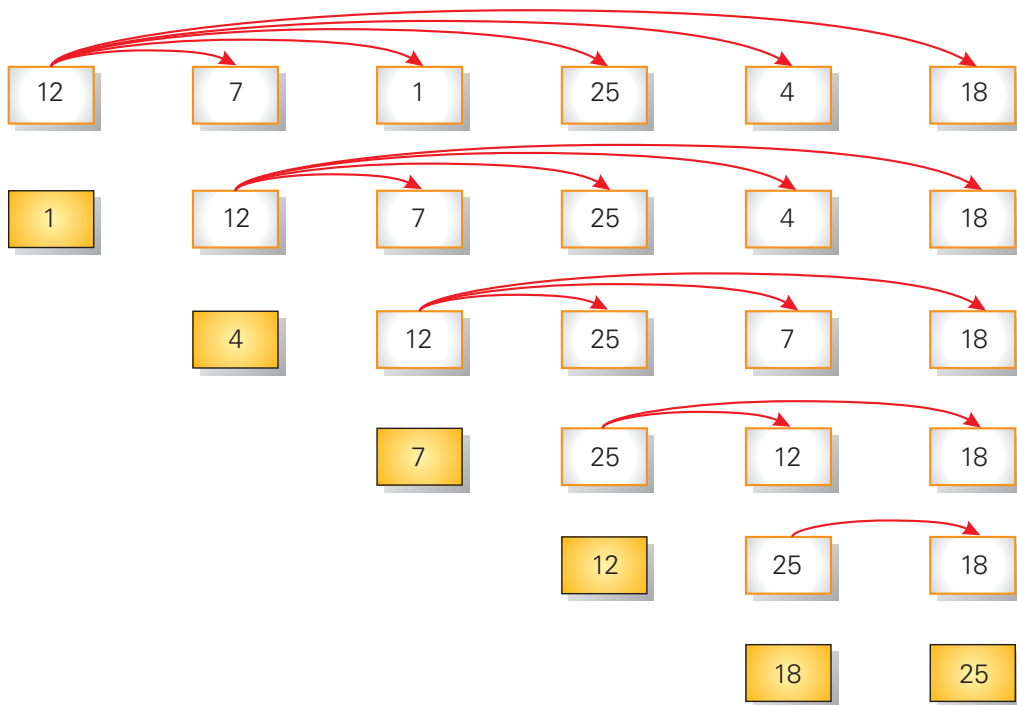
 posto = 0;
 do {
 if (v[posto]==n) s = true;
 else posto++;
 } while (posto<d && s==false);

 return s;
} // Trova
```

## 2 Ordinamento degli elementi di un array

**Ordinare gli elementi di un array** (in inglese **sort**) significa metterli in ordine alfabetico se sono stringhe o in ordine di grandezza se sono numeri (crescente o decrescente).

Esistono diversi metodi di sort. Qui viene presentato uno dei più semplici: il *metodo per sostituzione* che è efficace per un numero piccolo di elementi da ordinare.



Il metodo di ordinamento per sostituzione consiste nel confronto di ciascun elemento, a cominciare dal primo fino all'ultimo, con tutti gli altri elementi che sono disposti nelle posizioni successive a quella in esame.

Tutte le volte che tra gli elementi successivi all'elemento in esame si incontra un elemento più piccolo, si procede allo scambio tra i due in modo che, alla fine del confronto di ciascun elemento con i successivi, si ottenga nella posizione in esame il valore minimo rispetto ai successivi.

Alla fine gli elementi, dal più piccolo al più grande, compaiono ordinati all'interno dell'array a partire dal primo.

### PROGETTO 2

#### Ordinare gli elementi di un array di numeri interi in modo crescente.

Occorre organizzare due ripetizioni con contatore annidate l'una rispetto all'altra. Nell'iterazione più esterna viene percorso tutto il vettore fino al penultimo elemento, mentre la più interna prende in considerazione tutti gli elementi a destra di quello individuato dalla ripetizione esterna. Se il primo elemento risulta minore del secondo si procede allo scambio. Il problema in esame contiene, quindi, al suo interno il sottoproblema *Scambia* per lo scambio del contenuto di due variabili.

**Programma C++**

```
// Sort.cpp: ordinamento di un array
#include <iostream>
using namespace std;

 const int MAX = 100;

// prototipi delle funzioni
int ChiediDimensione();
void CaricaVettore(int v[], int d);
void Stampa(int v[], int d);
void Ordina(int v[], int d);
void Scambia(int& a, int& b);

// funzione principale
int main()
{
 int dim; // dimensione indicata dall'utente
 int vett[MAX]; // vettore da ordinare

 dim = ChiediDimensione();
 cout << "Carica gli elementi del vettore" << endl;
 CaricaVettore(vett,dim);
 cout << "Prima: " << endl;
 Stampa(vett,dim);
 Ordina(vett,dim);
 cout << "Dopo: " << endl;
 Stampa(vett,dim);

 return 0;
}

// dimensione dell'array
int ChiediDimensione()
{
 int d;
 do {
 cout << "Dimensione dell'array: ";
 cin >> d;
 } while (d<1 || d>MAX);

 return d;
} // ChiediDimensione

// caricamento delle componenti
void CaricaVettore(int v[], int d)
{
 for (int i=0; i<d; i++) {
 cout << "Elemento di posto " << i << ": ";
 cin >> v[i];
 }
} // CaricaVettore
```

```
// visualizza le componenti
void Stampa(int v[], int d)
{
 for (int i=0; i<d; i++) cout << v[i] << ' ';
 cout << endl;
} // Stampa

// ordinamento crescente dell'array
void Ordina(int v[], int d)
{
 for (int i=0; i<d-1; i++) {
 for (int j=i+1; j<d; j++)
 if (v[i]>v[j]) Scambia(v[i], v[j]);
 }
} // Ordina

// scambio di due variabili
void Scambia(int& a, int& b)
{
 int comodo;

 comodo = a;
 a = b;
 b = comodo;
} // Scambia
```

L'algoritmo descritto nel progetto precedente rappresenta l'ordinamento *crescente* degli elementi dell'array. Se si volesse l'ordinamento *decrescente*, basta cambiare il verso del confronto ed effettuare lo scambio quando un elemento è minore del successivo.



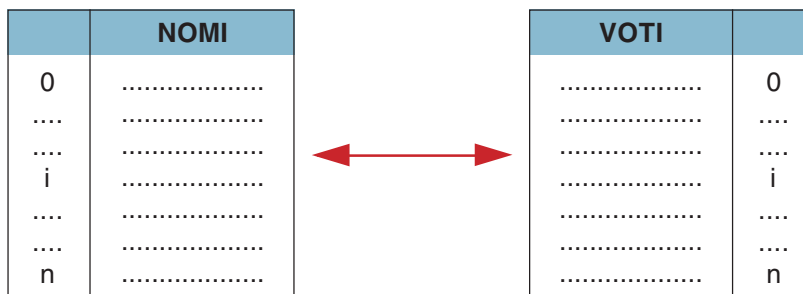
#### MATERIALI ONLINE

### 1. Ricerca di un elemento in un array ordinato (ricerca dicotomica)

## 3 Array paralleli

Supponiamo di dover organizzare i nomi degli studenti e le loro votazioni in una prova. Conviene costruire due array, uno per i nomi e uno per i voti: il legame tra nome e votazione è stabilito dall'indice che rappresenta la posizione degli elementi corrispondenti nei due array. Il nome e il voto aventi lo stesso indice  $i$  corrispondono allo stesso studente  $i$ -esimo.

Per questo motivo si dice che l'array dei nomi e l'array delle votazioni sono due **array paralleli**.



In questo modo le componenti di ciascun array sono tutte dello stesso tipo, ma i tipi nei due array possono essere diversi.



Per esempio le strutture di dati adatte per rappresentare i nomi di 30 studenti con i rispettivi voti conseguiti in una prova sono dichiarate nel seguente modo:

```
const int MAX = 30;

string nomi[MAX]; // array dei nomi
int voti[MAX]; // array dei voti
```

Un secondo esempio mostra la dichiarazione delle strutture di dati adatte per costruire un dizionario di 100 parole con le rispettive traduzioni in una lingua straniera:

```
const int MAX = 100;

string parole[MAX]; // vettore dei termini italiani
string traduz[MAX]; // vettore dei termini in lingua straniera
```

La funzione per caricare le parole italiane e le rispettive traduzioni in lingua straniera riceve i due vettori *parole* e *traduz*, oltre alla dimensione degli array, passati ai parametri *v1*, *v2* e *d*.

```
void CaricaVettori(string v1[], string v2[], int d)
{
 for (int i=0; i<d; i++) {
 cout << "Parola italiana di posto " << i << ": ";
 cin >> v1[i];
 cout << "Traduzione di posto " << i << ": ";
 cin >> v2[i];
 }
}
```

Dopo aver predisposto il dizionario, è possibile richiedere la traduzione in lingua straniera della parola italiana fornita in input.

La funzione di ricerca riceve i due vettori *parole* e *traduz*, oltre alla dimensione degli array, passati ai parametri *v1*, *v2* e *d*.

Inoltre un quarto parametro *traduci* passa alla funzione il termine da tradurre.

```
void Trova (string v1[], string v2[], int d, string traduci)
{
 bool s = false;
 int posto;
 string tradotto;

 posto = 0;
 do {
 if (v1[posto]==traduci) {
 tradotto = v2[posto];
 s = true;
 }
 else posto++;
 } while (posto<d && s==false);
 if (s) cout << tradotto << endl;
 else cout << "Termine non trovato" << endl;
}
```



## MATERIALE ONLINE

### 2. Passaggio di array alle funzioni

## 4 La tabella

Un array di strutture viene di solito chiamato **tabella**, perché si può schematizzare come una tabella con tante colonne quanti sono i campi della struttura e un numero di righe pari al numero delle componenti dell'array.

Per esempio, i dati e le loro strutture utili per la gestione di una biblioteca possono essere definiti nel modo seguente:

```
struct DataOdierna {
 int giorno;
 int mese;
 int anno;
};

struct Libro {
 string titolo;
 string autori[3];
 int AnnoEdiz;
 string LuogoEdiz;
 string CasaEd;
 DataOdierna DataAcquisto;
};

Libro biblioteca[1000];
```

Leggendo la definizione a cominciare dal basso si vede che l'intera biblioteca è definita come un array di 1000 componenti ciascuna delle quali ha la struttura di *Libro*. Quest'ultima è composta da sei campi, il secondo dei quali è composto da un array (gli autori di un libro possono essere più di uno) e l'ultimo (*DataAcquisto*) è un tipo definito in precedenza (*DataOdierna*).

Con queste definizioni di dati, nel programma, è consentito l'uso delle seguenti istruzioni:

```
biblioteca[5].DataAcquisto.mese = 7;
biblioteca[40].AnnoEdiz = 2004;
```

La struttura dati più adatta per rappresentare le informazioni di un dipendente con nome, indirizzo, livello, stipendi nei 12 mesi dell'anno più la tredicesima è la seguente:

```
struct Persona {
 string nome;
 string indirizzo;
 int livello;
 float stipendio[13];
};
```

All'interno della struttura il campo *stipendio* è un array.

Per rappresentare le informazioni relative a tutti i dipendenti di un'azienda si può utilizzare un array, le cui componenti sono di tipo *Persona*:

```
Persona dipendente[200];
```

## PROGETTO 3

### Classifica di una gara campestre.

Alla fine di una gara campestre vengono memorizzati, per ogni partecipante, il numero di pettorale, il nome e il tempo impiegato espresso in ore, minuti e secondi. Si vuole la stampa dei partecipanti ordinati rispetto al tempo impiegato.

Il problema si divide in tre parti: l'acquisizione dei dati, l'ordinamento e la stampa.

1. L'acquisizione dei dati prevede la memorizzazione di questi in una tabella, cioè in un *array di strutture*: ogni riga della tabella è composta da una struttura che contiene i dati di un partecipante, vale a dire il numero di pettorale, il nome e il tempo impiegato. A sua volta il tempo impiegato è definito da una struttura contenente i seguenti campi: ore (hh), minuti (mm), secondi (ss) e il totale secondi ottenuto dalla espressione  $hh*3600+mm*60+ss$ .
2. Per ordinare i partecipanti rispetto al tempo impiegato si utilizza l'algoritmo di *sort* visto nel Paragrafo 2.

Il campo della struttura che viene utilizzato per fare i confronti nell'algoritmo dell'ordinamento viene detto campo **chiave** dell'ordinamento.

La chiave per l'ordinamento in questo esempio è il campo *totale secondi*: questo campo è stato introdotto per facilitare l'operazione di confronto tra i tempi impiegati.

3. La stampa dei dati consiste nel trasferire in output i dati della tabella ordinata attraverso una ripetizione con contatore che permette di trattare tutte le strutture, dalla prima all'ultima, memorizzate in tabella.

### Programma C++

```
// Gara.cpp: classifica di una gara
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

const int MAX = 100;

// dichiarazione delle strutture
struct t {
 int hh, mm, ss;
 int TotSecondi;
};

struct partec {
 int pettorale;
 string nome;
 t tempo;
};
```

```

// prototipi delle funzioni
int ChiediDimensione();
void CaricaTabella(partec T[], int d);
void Scambia(partec& a, partec& b);
void Ordina(partec T[], int d);
void StampaTabella(partec T[], int d);

// funzione principale
int main()
{
 int dim; // dimensione indicata dall'utente
 partec tab[MAX]; // tabella dei partecipanti

 dim = ChiediDimensione();
 CaricaTabella(tab, dim);
 Ordina(tab, dim);
 StampaTabella(tab, dim);

 return 0;
}

// dimensione della tabella
int ChiediDimensione()
{
 int d;
 do {
 cerr << "Dimensione della tabella: ";
 cin >> d;
 } while (d<1 || d>MAX);

 return d;
} // ChiediDimensione

// caricamento delle componenti
void CaricaTabella(partec T[], int d)
{
 for (int i=0; i<d; i++) {
 cerr << "Numero di pettorale: ";
 cin >> T[i].pettorale;
 cerr << "Nome del concorrente: ";
 cin >> T[i].nome;
 cerr << "Ore impiegate: ";
 cin >> T[i].tempo.hh;
 cerr << "Minuti: ";
 cin >> T[i].tempo.mm;
 cerr << "Secondi: ";
 cin >> T[i].tempo.ss;
 T[i].tempo.TotSecondi = T[i].tempo.hh*3600 +
 T[i].tempo.mm*60 + T[i].tempo.ss;
 }
} // CaricaTabella

// scambio di due variabili
void Scambia(partec& a, partec& b)
{
 partec comodo;

```

```
comodo = a;
a = b;
b = comodo;
} // Scambia

// ordinamento crescente dell'array
void Ordina(partec T[], int d)
{
 for (int i=0; i<d-1; i++) {
 for (int j=i+1; j<d; j++)
 if (T[i].tempo.TotSecondi>T[j].tempo.TotSecondi)
 Scambia(T[i], T[j]);
 }
} // Ordina

// visualizza le componenti
void StampaTabella(partec T[], int d)
{
 // intestazione
 cout << setw(10) << "Pettorale"
 << setw(15) << "Nome"
 << setw(5) << "h"
 << setw(3) << "m"
 << setw(3) << "s"
 << setw(10) << "arrivo"
 << endl;
 // elenco partecipanti
 for (int i=0; i<d; i++)
 cout << setw(10) << T[i].pettorale
 << setw(15) << T[i].nome
 << setw(5) << T[i].tempo.hh
 << setw(3) << T[i].tempo.mm
 << setw(3) << T[i].tempo.ss
 << setw(10) << i+1
 << endl;
} // StampaTabella
```

Il programma precedente utilizza il flusso **cerr** (*standard error*) per i messaggi da inviare al video come supporto dell'input dell'utente, e il flusso **cout** (*standard output*) per l'output dei risultati dell'elaborazione.

In questo modo, in fase di testing del programma, o nel caso non fosse disponibile una stampante, è possibile effettuare una **ridirezione dell'output** dalla linea comandi che lancia l'esecuzione del programma:

```
C:\esercizi>Gara > elenco.txt
```

Volendo invece stampare l'output su carta, occorre fare una ridirezione dell'output verso la stampante **lpt1**, con la seguente linea comandi:

```
C:\esercizi>Gara > lpt1:
```



#### AUTOVERIFICA

Domande da 1 a 4 pag. 425

Problemi da 1 a 8 pag. 429

## 5 La pila

Con il termine **pila** (o *catasta*), in inglese **stack**, si indica un elenco di dati avente la caratteristica di permettere l'inserimento di nuovi elementi e l'estrazione degli elementi introdotti da un'unica estremità.

Possiamo rappresentare la struttura in astratto, definendo le procedure che possono operare su di essa e non permettendo altri tipi di intervento. La pila diventa così un elenco nel quale si può aggiungere un elemento con una funzione, detta **Push**, e dal quale può essere estratto un elemento con una funzione, detta **Pop**.

La regola di funzionamento della pila è: l'ultimo elemento inserito con un *Push* è il primo elemento ad essere estratto con un *Pop*.

Per evidenziare il funzionamento della struttura a pila, essa viene spesso chiamata struttura **LIFO**, acronimo di *Last In First Out*, ovvero l'ultimo ad entrare è il primo ad uscire.

Quando nell'esecuzione di un programma si incontra una chiamata a una funzione, occorre memorizzare il punto in cui tornare una volta che il modulo sia stato eseguito. Questo punto viene detto **indirizzo di rientro**. Se nella funzione si incontra un'altra chiamata, occorre memorizzare un secondo indirizzo di rientro e così via.

Quando il sottoprogramma termina, il rientro deve avvenire all'ultimo punto memorizzato per proseguire l'elaborazione nell'unità, programma o sottoprogramma, che aveva richiesto l'ultima chiamata alla funzione. Gli indirizzi di rientro seguono perciò la logica di funzionamento della pila e possono utilmente essere memorizzati in una struttura di dati di questo tipo (**stack delle chiamate**).

Durante l'esecuzione di un programma contenente funzioni ricorsive, può succedere che per un errore si verifichi una situazione di ricorsione senza fine.

Quando l'accumularsi di versioni dello stesso sottoprogramma causa il riempimento della memoria disponibile, viene segnalato il messaggio **stack overflow error**, cioè errore di traboccamento della pila.



### IMPLEMENTAZIONE DELLA PILA CON UN ARRAY

#### PROGETTO 4

##### Organizzare un algoritmo per la gestione di una pila di dati.

La gestione di una pila può essere realizzata appoggiandosi ad un array.

La pila è simulata con un array visto in "verticale" (come un bicchiere vuoto che viene riempito a partire dal fondo): l'inserimento e l'estrazione avvengono sempre dallo stesso lato.

L'uso della pila richiede una variabile *Testa* che indichi l'ultima posizione occupata, per sapere dove inserire o estrarre i dati.

Per organizzare i dati in una pila, è necessario realizzare una funzione per l'operazione di *Pop* e una per l'operazione di *Push*.

È, inoltre, utile avere a disposizione una funzione di svuotamento (o di inizializzazione) della pila e un'ulteriore funzione che mostri a video il contenuto della pila.

Il numero degli elementi dell'array è indicato dalla costante *MAX*.

Poiché gli indici dell'array sono compresi tra 0 e *MAX-1*, per inizializzare o svuotare la pila si è scelto di assegnare a *Testa* il valore *MAX*, coerentemente con il fatto che tale valore viene assunto da *Testa* se si esegue un'operazione di *Pop* su tutti gli elementi della pila. Inversamente la pila è piena quando *Testa* = 0.

Nell'operazione di *Pop* la variabile *Testa* si incrementa di 1, mentre nell'operazione di *Push* si decrementa di 1.

### Programma C++

```
// Pila.cpp: gestione di una pila con l'array
#include <iostream>
using namespace std;

const int MAX = 10;
int Pila[MAX]; // array per la pila
int Testa; // testa della pila

// prototipi delle funzioni
void SvuotaPila();
void Pop();
void Push();
void ScriviPila();
void PresentaMenu();

// funzione principale
int main()
{
 int scelta; // scelta nel menu
 SvuotaPila();
 do {
 do {
 PresentaMenu();
 cout << "Inserisci la tua scelta ";
 cin >> scelta;
 } while (scelta < 1 || scelta > 5);
 switch(scelta) {
 case 1:
 SvuotaPila();
 break;
 case 2:
 Pop();
 break;
 case 3:
 Push();
 break;
 case 4:
 ScriviPila();
 break;
 }
 } while (scelta != 5);

 return 0;
}
```

```
// inizializzazione della pila
void SvuotaPila()
{
 Testa = MAX;
 cout << "Pila vuota" << endl;
} // SvuotaPila

// estrazione di un elemento della pila
void Pop()
{
 if (Testa == MAX)
 cout << "Pila vuota. Nessun elemento estratto" << endl;
 else {
 cout << "Elemento estratto = " << Pila[Testa] << endl;
 Testa++; // sposta in basso la testa della pila
 }
} // Pop

// inserimento di un elemento nella pila
void Push()
{
 int dato; // elemento da inserire
 if (Testa == 0)
 cout << "Pila piena. Nessun elemento inserito" << endl;
 else {
 cout << "Dato da inserire = ";
 cin >> dato;
 Testa--; // sposta in alto la testa della pila
 Pila[Testa] = dato;
 }
} // Push

// visualizzazione della pila
void ScriviPila()
{
 cout << "Testa pila" << endl;
 for (int i=Testa; i<MAX; i++) cout << Pila[i] << endl;
 cout << "Base pila" << endl;
} // ScriviPila

// menu delle scelte
void PresentaMenu()
{
 cout << endl;
 cout << "1. Svuota pila" << endl;
 cout << "2. Estrai un dato" << endl;
 cout << "3. Aggiungi un dato" << endl;
 cout << "4. Scrivi pila" << endl;
 cout << "5. Fine" << endl;
} // PresentaMenu
```



## 6 La coda

Una struttura nella quale, in senso figurato, i dati entrano da una parte ed escono dalla parte opposta viene detta **coda**.

A differenza della pila, le funzioni che possono operare sulla struttura di coda funzionano in questo modo: si può aggiungere un elemento alla fine della struttura (in fondo alla coda) con la funzione **Push**, e si può estrarre un elemento alla testa della coda con la funzione **Pop**. La regola di funzionamento della coda è: il primo elemento inserito con un *Push* è il primo elemento ad essere estratto con un *Pop*.

Per evidenziare il funzionamento della struttura a coda, essa viene spesso chiamata struttura **FIFO**, acronimo di *First In First Out*, ovvero il primo ad entrare è il primo ad uscire.



### IMPLEMENTAZIONE DELLA CODA CON UN ARRAY

#### PROGETTO 5

##### Organizzare un algoritmo per la gestione di una coda di dati.

La gestione di una coda può essere realizzata appoggiandosi ad un array. Per sapere dove inserire nuovi dati occorre una variabile *Fine* che indichi l'ultima posizione occupata, mentre l'estrazione avviene sempre sul primo elemento della coda. Per l'inizializzazione o lo svuotamento della coda occorre assegnare -1 all'indice *Fine*. All'interno della funzione *Pop*, dopo aver estratto l'elemento di testa è necessario far scivolare a sinistra di un posto gli altri dati già caricati. La funzione *Push* inserisce un nuovo elemento in coda e deve aggiornare l'indice *Fine*. L'indice *Fine*, uguale alla dimensione dell'array - 1, segnala che la coda è piena.

##### Programma C++

```
// Coda.cpp: gestione di una coda con l'array
#include <iostream>
#include <iomanip>
using namespace std;

const int MAX = 10;
int Coda[MAX]; // array per la coda
int Fine; // fine della coda

// prototipi delle funzioni
void SvuotaCoda();
void Pop();
void Push();
void ScriviCoda();
void PresentaMenu();

// funzione principale
int main()
{
 int scelta; // scelta nel menu
 SvuotaCoda();
 do {
 do {
```

```
PresentaMenu();
cout << "Inserisci la tua scelta ";
cin >> scelta;
} while (scelta<1 || scelta>5);
switch(scelta) {
case 1:
 SvuotaCoda();
 break;
case 2:
 Pop();
 break;
case 3:
 Push();
 break;
case 4:
 ScriviCoda();
 break;
}
} while (scelta != 5);

return 0;
}

// inizializzazione della coda
void SvuotaCoda()
{
 Fine = -1;
 cout << "Coda vuota" << endl;
} // SvuotaCoda

// estrazione di un elemento della coda
void Pop()
{
 if (Fine == -1)
 cout << "Coda vuota. Nessun elemento estratto" << endl;
 else {
 cout << "Elemento estratto = " << Coda[0] << endl;
 // fa avanzare la coda di un posto
 for (int i=1; i<=Fine; i++) Coda[i-1] = Coda[i];
 Fine--; // sposta a sinistra la fine della coda
 }
} // Pop

// inserimento di un elemento nella coda
void Push()
{
 int dato; // elemento da inserire
 if (Fine == MAX-1)
 cout << "Coda piena. Nessun elemento inserito" << endl;
 else {
 cout << "Dato da inserire = ";
 cin >> dato;
 Fine++; // sposta a destra la fine della coda
 Coda[Fine] = dato;
 }
} // Push
```

```
// visualizzazione della coda
void ScriviCoda()
{
 cout << "Inizio coda - ";
 for (int i=0; i<=Fine; i++) cout << setw(5) << Coda[i];
 cout << " - Fine coda" << endl;
} // ScriviCoda

void PresentaMenu()
{
 cout << endl;
 cout << "1. Svuota coda" << endl;
 cout << "2. Estrai un dato" << endl;
 cout << "3. Aggiungi un dato" << endl;
 cout << "4. Scrivi coda" << endl;
 cout << "5. Fine" << endl;
} // PresentaMenu
```

## 7 Le liste

Nei paragrafi precedenti abbiamo visto come organizzare insiemi di dati omogenei attraverso gli array. In tutti gli algoritmi esaminati la quantità massima memorizzabile è stabilita dal programmatore a priori, ossia in modo prefissato e statico. Dopo aver compilato il programma, infatti, non è più possibile modificare le capacità degli array se non ricompilando il programma sorgente. Poco importa di quanta memoria centrale dispone il computer nel quale il programma viene eseguito. In questo paragrafo vediamo come ovviare a questa limitazione realizzando dei programmi che gestiscono la memoria occupata dai dati in modo dinamico.

Con **strutture dinamiche di dati** si intende, in generale, la possibilità di utilizzare strutture di dati, alle quali viene assegnato lo spazio necessario in memoria centrale quando serve e durante l'esecuzione del programma, incrementando e decrementando tale spazio in base alle esigenze dell'esecuzione in corso.

Le tecniche di programmazione per la gestione dinamica delle strutture di dati utilizzano il tipo **puntatore** in un'organizzazione a liste.

La **lista** è un insieme di elementi composti da un'informazione e da uno o più puntatori ad altri elementi dello stesso tipo.

Ogni elemento della lista si compone di due parti: l'informazione da memorizzare e il puntatore. Dovendo aggregare dati non omogenei, utilizziamo le strutture definite dall'utente, in cui tutti i campi sono destinati alle informazioni eccetto uno dedicato al puntatore.

```
struct Nodo {
 tipo1 dato1
 tipo2 dato2
 . . .
 Nodo* succ;
};
```



L'ultimo puntatore della lista ha valore **NULL**, che significa *nessun indirizzo*.

Nell'operazione di inserimento di un nuovo elemento nella lista, occorre creare un nuovo puntatore ad un'area di memoria che possa contenere la parte informazione e la parte puntatore: in informatica si dice **allocare** spazio in memoria.

Nelle operazioni di cancellazione di un elemento dalla lista, invece, la memoria occupata dall'elemento eliminato deve essere **liberata**, cioè resa disponibile al sistema operativo per altri usi.

Alla fine del programma, la lista deve essere comunque svuotata, eliminando i nodi uno ad uno, in modo da rilasciare la memoria inutilizzata.

In questo modo si realizza in pratica la **gestione dinamica delle strutture di dati**, in quanto lo spazio in memoria viene *allocato* (o liberato) quando serve durante l'esecuzione del programma.

Per gestire queste operazioni, nel linguaggio C++, sono disponibili due operatori:

- l'operatore **new** che alloca la memoria indirizzata da un nuovo puntatore, per esempio:

```
Nodo* nuovo;
nuovo = new Nodo;
```

Lo spazio riservato in memoria è pari alla dimensione della struttura *Nodo*.

- l'operatore **delete** che libera lo spazio di memoria puntata da un puntatore per possibili usi successivi, per esempio:

```
delete testa;
```

L'operatore ha come argomento il nome del puntatore.

La sintassi degli operatori *new* e *delete* cambia quando essi sono riferiti ad array. Per esempio:

```
int* p;
p = new int[20];
```

alloca un array di 20 puntatori ad interi;

```
delete[] p;
```

libera la memoria allocata per l'array: l'uso delle parentesi quadre aperta e chiusa, [], indica l'applicazione dell'operatore *delete* all'intero array.

## PROGETTO 6

**Inserire da tastiera un elenco di nomi e organizzarli in una lista ordinata alfabeticamente. L'ordine deve essere mantenuto per ogni nuovo inserimento e per ogni cancellazione di nomi.**

Si vuole costruire un programma che consenta di trattare una lista di stringhe con possibilità di inserire nuovi dati, togliere dati e visualizzare la lista.

Le funzioni del programma sono:

- inserimento, per inserire un nuovo elemento nella lista;
- eliminazione, per togliere un elemento dalla lista;
- visualizzazione, per scrivere sul video l'elenco completo degli elementi presenti nella lista;
- svuotamento della lista, per eliminare tutti i nodi della lista.

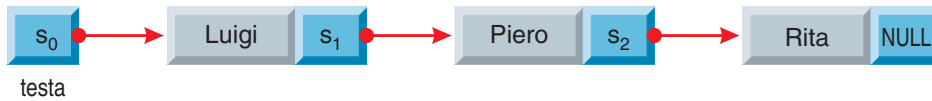
Ogni elemento della lista contiene un nome e il puntatore all'elemento successivo.

L'operazione di **inserimento** crea un nuovo nodo nella lista e poi ricerca la posizione dove collocare il nuovo nome per mantenere la lista ordinata.

Se la lista è vuota, il nodo viene collocato in prima posizione, altrimenti si scorre la lista finché si trova un nome maggiore o uguale rispetto al nuovo oppure finché si raggiunge la fine della lista. Trovata la posizione, occorre aggiornare i puntatori: il nome precedente punta al nuovo nome, il nuovo nome punta al nome che viene dopo nella lista ordinata.

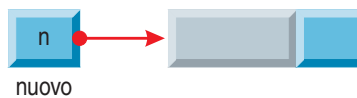
La successione delle operazioni è riassunta nello schema seguente.

Situazione iniziale:



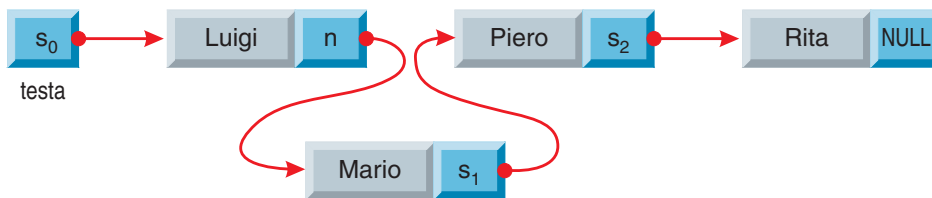
Viene inserito da tastiera un nuovo nome: Mario.

Con l'operatore **new** viene creata una nuova area contenente un nome e un puntatore, ossia un nuovo elemento della lista. L'indirizzo *n* di questa nuova area viene salvato nel puntatore *nuovo*.



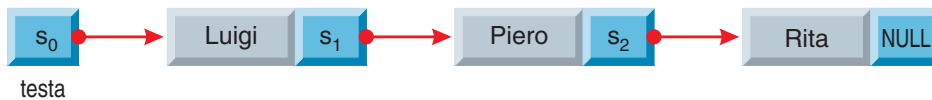
Nel nuovo elemento della lista, lo spazio per il nome assume il valore *Mario*, mentre lo spazio per il puntatore assume il valore *s1*.

Il puntatore dell'elemento che precede nell'ordine (*Luigi*) prende l'indirizzo del nuovo elemento (*n*).



Nell'operazione di **cancellazione** di un nome dall'elenco ordinato, dopo aver acquisito da tastiera il nome richiesto e dopo aver verificato che la lista non sia vuota, il procedimento risolutivo inizia la ricerca all'interno della lista utilizzando i legami tra gli elementi stabiliti dai puntatori. Se il nome richiesto viene trovato, l'elemento viene tolto dalla lista dopo aver aggiornato i puntatori: il nome precedente punta al nome successivo a quello da cancellare.

Situazione iniziale:



Volendo eliminare *Piero* dalla lista, *s1* diventa *s2* e la nuova lista risulta come la seguente:



Inoltre la memoria occupata dell'elemento eliminato deve essere liberata con l'operatore *delete*.

### Programma C++

```
// ListaNomi.cpp: lista ordinata di nomi
#include <iostream>
#include <string>
using namespace std;
```

```

// struttura per i nodi della lista
struct Nodo {
 string nome;
 Nodo* succ;
};

Nodo* testa; // puntatore alla testa della lista

// prototipi delle funzioni
void Inizializza();
bool ListaVuota();
void Inserisci();
void Elimina();
void Visualizza();
void SvuotaLista();
void PresentaMenu();

// funzione principale
int main()
{
 int scelta; // scelta nel menu

 Inizializza();
 do {
 do {
 PresentaMenu();
 cout << "Inserisci la tua scelta ";
 cin >> scelta;
 } while (scelta<1 || scelta>5);
 switch(scelta) {
 case 1:
 Inserisci();
 break;
 case 2:
 Elimina();
 break;
 case 3:
 Visualizza();
 break;
 case 4:
 SvuotaLista();
 break;
 }
 } while (scelta != 5);
 SvuotaLista();

 return 0;
}

// inizializzazione della lista
void Inizializza()
{
 testa = NULL;
} // Inizializza

// controllo se lista vuota
bool ListaVuota()
{
 // restituisce false se non vuota
 // true se vuota
 if (testa == NULL) return true;
}

```

```
 return false;
} // ListaVuota

// inserimento di un nodo nella lista
void Inserisci()
{
 string NomeInp; // nome da inserire
 Nodo* nuovo; // puntatore ad un nuovo nodo
 Nodo* temp; // puntatori per confronti
 Nodo* prec;

 cout << "Nome da inserire: ";
 cin >> NomeInp;
 nuovo = new Nodo; // alloca la memoria per il nuovo nodo
 nuovo->nome = NomeInp;
 nuovo->succ = NULL;
 if (ListaVuota()) testa = nuovo;
 else {
 if (testa->nome >= nuovo->nome) {
 temp = testa;
 testa = nuovo;
 testa->succ = temp;
 }
 else {
 temp = testa;
 while (temp!=NULL && nuovo->nome>=temp->nome) {
 prec = temp;
 temp = temp->succ;
 }
 nuovo->succ = temp;
 prec->succ = nuovo;
 }
 }
} // Inserisci

// estrazione di un nodo della lista
void Elimina()
{
 string NomeInp; // nome da eliminare
 Nodo* temp; // puntatori per confronti
 Nodo* prec;
 bool trovato = false; // indicatore se trovato

 cout << "Nome da eliminare: ";
 cin >> NomeInp;
 if (ListaVuota()) cout << "Lista vuota" << endl;
 else {
 if (testa->nome == NomeInp) {
 trovato = true;
 temp = testa->succ;
 cout << "Nome eliminato: " << testa->nome << endl;
 delete testa;
 testa = temp;
 }
 else {
 temp = testa->succ;
 prec = testa;
 do {
```

```

 if (temp->nome == NomeInp) {
 trovato = true;
 prec->succ = temp->succ;
 cout << "Nome eliminato: " << temp->nome << endl;
 delete temp;
 }
 else {
 prec = temp;
 temp = temp->succ;
 }
 } while (!trovato && temp != NULL);
}
if (!trovato)
 cout << "Il nome richiesto non esiste" << endl;
} // fine else
} // Elimina

// visualizzazione della lista
void Visualizza()
{
 Nodo* p;
 cout << "-- Elenco dei nomi: --" << endl;
 p = testa;
 while (p != NULL) {
 cout << p->nome << endl;
 p = p->succ;
 }
 cout << "-- Fine elenco --" << endl;
} // Visualizza

// toglie tutti i nodi della lista
void SvuotaLista()
{
 Nodo* p;
 Nodo* temp;
 while (testa != NULL) {
 cout << "Nome eliminato: " << testa->nome << endl;
 temp = testa;
 testa = testa->succ;
 delete temp;
 };
 cout << "Lista vuota" << endl;
} // SvuotaLista

// menu delle scelte
void PresentaMenu()
{
 cout << endl << "-----" << endl;
 cout << "1. Inserisci un nome" << endl;
 cout << "2. Elimina un nome" << endl;
 cout << "3. Visualizza la lista" << endl;
 cout << "4. Svuota la lista" << endl;
 cout << "5. Fine" << endl;
 cout << "-----" << endl;
} // PresentaMenu

```

**AUTOVERIFICA**

Domande da 5 a 10 pag. 425-426

Problemi da 9 a 12 pag. 429

**MANIPOLAZIONE**

### 3. Gestione di pila e coda con i puntatori

### 4. La struttura dati ad albero binario



## 8 Template di funzioni

In informatica il termine **template**, in generale, indica un modello utilizzabile per un insieme di problemi dello stesso tipo e avente parti o componenti che contengono parametri, in modo che possano essere adattate per specifiche esigenze.

La possibilità di riciclare il codice disponibile è una delle principali caratteristiche che ha determinato lo sviluppo del linguaggio C++.

Un primo approccio, utilizzato in passato in altri linguaggi, consiste nel **duplicare** il codice originario e nel modificare alcune parti manualmente. Questa metodologia è però poco efficiente in quanto può causare l'inserimento di nuovi errori nel programma.

Un secondo approccio, utilizzato da tutti i linguaggi di programmazione orientati agli oggetti, utilizza il codice ottenuto **ereditando** le caratteristiche di una classe già implementata e testata. In questo modo si riducono le possibilità di creare codice contenente errori. Nei progetti del capitolo precedente abbiamo utilizzato questo secondo approccio, ma nel linguaggio C++ esiste un terzo modo.

Il terzo approccio per riutilizzare il codice, realizzabile nel linguaggio C++ attraverso i **template**, ricopia il codice sorgente anziché ereditarlo: mediante i template, infatti, il compilatore sostituisce il codice specificato come parametro.

Il concetto di *template* può essere applicato alle funzioni.

I **template delle funzioni** consentono di inserire in un programma delle funzioni che contengono un parametro, che può poi essere sostituito con un tipo al momento della chiamata della funzione stessa.

Si pensi per esempio al problema molto comune dello scambio del contenuto di due variabili. Se lo scambio, nello stesso programma, deve essere effettuato su variabili di tipo numerico *int* e su variabili di tipo *string*, siamo costretti a scrivere due distinte funzioni:

```
void ScambiaInteri(int& a, int& b)
{
 int temp;

 temp = a;
 a = b;
 b = temp;
};

void ScambiaStringhe(string& a, string& b)
{
 string temp;

 temp = a;
 a = b;
 b = temp;
};
```

Si deve ricordare che, per poter rendere effettivo lo scambio, il passaggio dei parametri deve essere fatto per referenza.

Operando invece con i *template*, la funzione per lo scambio è scritta nel programma una sola volta e contiene al suo interno un parametro T:

```
template<typename T> void Scambia(T& a, T& b)
{
 T temp;
```

```
temp = a;
a = b;
b = temp;
};
```

L'intestazione del template di funzione inizia con la frase:

```
template<typename T>
```

La parola chiave **template** indica al compilatore che la definizione che segue sarà utilizzata da diversi tipi.

Nel *main* o, in generale, nella funzione chiamante la chiamata alla funzione determina automaticamente il tipo che deve essere sostituito al parametro *T* nel *template* sulla base del tipo dei parametri attuali:

```
int x, y;
string s1, s2;

Scambia(x, y);
Scambia(s1, s2);
```

In sostanza il *template* di funzione è un modello generico che si specializza in base alle specifiche esigenze di elaborazione.

Quando la funzione riguarda operazioni sugli oggetti si utilizza la parola chiave **class** al posto di **typename** nell'intestazione della funzione. Per esempio, il seguente *template* di funzione realizza lo scambio tra due oggetti che sono istanze di una classe:

```
template<class T> void Scambia(T& a, T& b)
{
 T temp;

 temp = a;
 a = b;
 b = temp;
};
```

Questo è anche il modo in cui si definiscono i *template di classe*, come spiegato nel prossimo paragrafo.

## 9 Template delle classi

Nella programmazione in C++, il *tipo parametrizzato*, o **class template**, è lo strumento che permette di definire i **modelli di classi**. Questi modelli consentono la creazione di istanze di classi diverse a seconda del parametro passato.

Lo schema generale della sintassi che definisce un *template* della classe è il seguente:

```
template<class T> class NomeClasse {
 // attributi
 T attributo1;
 tipo1 attributo2;
```

```
 . . .
 . . .
public:

 // metodi
 void Funzione1(T); T Funzione2();
 . . .
 . . .
};

 // implementazione delle funzioni membro
template <class T> void NomeClasse<T>::Funzione1(T nomeparametro)
{
 . . .
 . . .
}
template<class T> T NomeClasse<T>::Funzione2()
{
 . . .
 . . .
}
```

La dichiarazione della classe inizia con la frase:

template<class T>

*T* rappresenta il parametro che viene sostituito con il tipo fornito al momento dell'utilizzo del *template*. L'uso della lettera *T* è del tutto convenzionale: si può usare una qualsiasi altra lettera o parola.

L'implementazione delle funzioni membro, come si vede, contiene il riferimento al *template* attraverso la notazione:

NomeClasse<T>::Funzione()

Il parametro *T* è indicato nell'intestazione della classe, ma può anche essere utilizzato per indicare il tipo di un attributo, il valore di ritorno di un metodo o il tipo di un parametro passato al metodo stesso.

La dichiarazione delle istanze contiene la specificazione, tra parentesi angolari, < >, del tipo da sostituire al parametro *T*. Per esempio:

```
NomeClasse<int> istanza1;
NomeClasse<float> istanza2;
```

in generale

```
NomeClasse<tipo_utente> istanza;
```

Il tipo specificato tra parentesi angolari, < >, viene sostituito in tutte le occorrenze di *T* presenti nel *template*.

L'istanza di un *template* è quindi una classe, cioè la **specializzazione** della classe modello per un certo tipo di dato.

## PROGETTO 7

**Costruire un template di classe per un array generico che possa essere utilizzato per dati di tipo intero e di tipo stringa. La classe deve contenere anche i metodi per inserire i dati nell'array e per visualizzare il contenuto dell'array.**

Il *template* ha due attributi:

- *p* indica il puntatore al primo elemento dell'array
- *dim* rappresenta il numero delle componenti dell'array (dimensione).

Oltre alle funzioni membro per inserire i dati nell'array e per visualizzarne il contenuto, la classe contiene anche il costruttore e il distruttore.

| Vettore<T>                                            |
|-------------------------------------------------------|
| p<br>dim                                              |
| Inserisci<br>Visualizza<br>costruttore<br>distruttore |

La notazione con la lettera *T* tra parentesi angolari, *< >*, non è obbligatoria, serve solo a ricordare che il diagramma si riferisce a un *class template*.

La codifica in C++ è la seguente:

```
template<class T> class Vettore {
 T* p; // puntatore al primo elemento dell'array
 int dim; // dimensione dell'array
public:
 void Inserisci();
 void Visualizza();
 Vettore(int); // costruttore
 ~Vettore(); // distruttore
};
```

Il costruttore ha come argomento il numero delle componenti assegnate dall'utente all'array e alloca lo spazio di memoria necessario:

```
p = new T[dim];
```

Il distruttore libera la memoria allocata dall'array; l'istruzione *delete* contiene le parentesi quadre, [], per indicare che si tratta di un puntatore all'array:

```
delete [] p;
```

Di seguito è riportato il programma completo.

### Programma C++

```
// TVettore.cpp: template di array
#include <iostream>
#include <string>
using namespace std;
```

```

template<class T> class Vettore {
 T* p; // puntatore al primo elemento dell'array
 int dim; // dimensione dell'array
public:
 void Inserisci();
 void Visualizza();
 Vettore(int); // costruttore
 ~Vettore(); // distruttore
};

// costruttore
template<class T> Vettore<T>::Vettore(int d)
{
 dim = d;
 p = new T[dim];
}

// distruttore
template<class T> Vettore<T>::~~Vettore()
{
 delete [] p;
}

// inserimento componenti nell'array
template<class T> void Vettore<T>::Inserisci()
{
 for (int i=0; i<dim; i++) {
 cout << "Elemento di posto " << i << ": ";
 cin >> p[i];
 }
} // Inserisci

// visualizza componenti nell'array
template<class T> void Vettore<T>::Visualizza()
{
 for (int i=0; i<dim; i++) {
 cout << p[i] << endl;
 }
} // Visualizza

// funzione principale
int main()
{
 int dimensione;

 cout << "Dimensione del vettore: ";
 cin >> dimensione;

 cout << "Array di interi" << endl;
 Vettore<int> v1(dimensione);
 v1.Inserisci();
 cout << "-----" << endl;
 v1.Visualizza();
 cout << "-----" << endl;
}

```

```
cout << "Array di stringhe" << endl;
Vettore<string> v2(dimensione);
v2.Inserisci();
cout << "-----" << endl;
v2.Visualizza();
cout << "-----" << endl;

return 0;
}
```

Il programma mostra l'utilizzo dello stesso *template* con due tipi di array: array di interi e array di stringhe.

La stessa classe *Vettore* viene specializzata due volte, con il tipo *int* e con il tipo *string*, attraverso le due istruzioni:

```
Vettore<int> v1(dimensione);
Vettore<string> v2(dimensione);
```

Il tipo indicato tra parentesi angolari, *< >*, viene sostituito al parametro *T* nel *template* della classe. Queste istruzioni richiamano il costruttore della classe passando come parametro il numero delle componenti (*dimensione*).



## TEMPLATE DI CLASSE CON DUE PARAMETRI

I *template* delle classi possono essere definiti anche con più di un parametro. I parametri possono essere di tipo diverso. Lo schema generale di un *template* per una classe avente due parametri è il seguente:

```
template<class T1, class T2> class NomeClasse {
 // attributi
 T1 attributo1;
 T2 attributo2;
 // metodi
public:
 void Funzione();
};

template<class T1, class T2> void NomeClasse<T1, T2>::Funzione()
{
 : : :
 : : :
}
```

## PROGETTO 8

**Dato un elenco di studenti con matricola e nome, si vuole ottenere il nome di uno studente del quale viene fornita da tastiera la matricola, segnalando con un messaggio di errore il caso in cui la matricola richiesta non sia compresa nell'elenco.**

Il problema viene risolto con due array paralleli: il primo di tipo intero per le matricole, il secondo di tipo stringa per i nomi.

Il *template* della classe contiene i puntatori ai due array, oltre ai metodi per inserire i dati e per ricercare il nome corrispondente alla matricola richiesta.

| VettoriP<T1, T2>                                     |
|------------------------------------------------------|
| chiave<br>dato<br>dim                                |
| Inserisci<br>CercaDato<br>costruttore<br>distruttore |

La codifica del *template* è la seguente:

```
template<class T1, class T2> class VettoriP {
 T1* chiave; // puntatori al primo elemento degli array
 T2* dato;
 int dim; // dimensione degli array
public:
 void Inserisci();
 bool CercaDato(T1);
 VettoriP(int); // costruttore
 ~VettoriP(); // distruttore
};
```

All'interno del programma la classe specializzata per risolvere il problema specifico di questo esempio si chiama *scuola* ed è dichiarata con la seguente riga di codice:

```
VettoriP<int, string> scuola(dimensione);
```

Essa richiama il costruttore della classe fornendo come parametro il numero degli studenti (dimensione degli array paralleli), fornito dall'utente all'inizio del programma.

Il procedimento risolutivo ha carattere generale, nel senso che può essere utilizzato in tutti i problemi nei quali si ha un insieme di coppie di dati, dei quali il primo rappresenta il criterio o, come si dice in informatica la **chiave**, per ritrovare i dati, e il secondo il dato stesso. Inoltre il template può essere specializzato con qualsiasi tipo di dati: interi con stringhe, interi con interi, stringhe con stringhe, ecc.

### Programma C++

```
// TVettoriParalleli.cpp: template di array paralleli
#include <iostream>
#include <string>
using namespace std;

template<class T1, class T2> class VettoriP {
 T1* chiave; // puntatori al primo elemento degli array
 T2* dato;
 int dim; // dimensione degli array
public:
 void Inserisci();
 bool CercaDato(T1);
 VettoriP(int); // costruttore
 ~VettoriP(); // distruttore
};
```

```
// costruttore
template<class T1, class T2> VettoriP<T1, T2>::VettoriP(int d)
{
 dim = d;
 chiave = new T1[dim];
 dato = new T2[dim];
}

// distruttore
template<class T1, class T2> VettoriP<T1, T2>::~~VettoriP()
{
 delete [] chiave;
 delete [] dato;
}

// inserimento chiavi e dati
template<class T1, class T2> void VettoriP<T1, T2>::Inserisci()
{
 for (int i=0; i<dim; i++) {
 cout << "Codice: ";
 cin >> chiave[i];
 cout << "Dato: ";
 cin >> dato[i];
 }
} // Inserisci

// fornita la chiave, cerca il dato
template<class T1, class T2> bool VettoriP<T1, T2>::CercaDato(T1 k)
{
 bool trovato = false;
 int i = 0;

 while (i< dim && !trovato) {
 if (chiave[i] == k) {
 trovato = true;
 cout << dato[i] << endl;
 }
 i++;
 }

 return trovato;
} // CercaDato

// funzione principale
int main()
{
 int dimensione;
 int i;

 cout << "Numero degli studenti: ";
 cin >> dimensione;
 VettoriP<int, string> scuola(dimensione);
 cout << "Inserimento matricole e nomi" << endl;
 cout << "-----" << endl;
 scuola.Inserisci();
}
```



```

cout << "-----" << endl;
cout << "Quale matricola (0=fine)? ";
cin >> i;
while (i != 0) {
 if (!scuola.CercaDato(i))
 cout << "Matricola non trovata" << endl;
 cout << "Altra matricola (0=fine)? ";
 cin >> i;
}

return 0;
}

```

**AUTOVERIFICA**

Domande da 11 a 13 pag. 426-427  
 Problemi da 13 a 16 pag. 430

## 10 La libreria standard STL

La sigla **STL** (*Standard Template Library*) identifica, nel linguaggio C++, la **libreria dei template standard**. Si tratta di un insieme di algoritmi e di strutture di dati di uso comune nella programmazione.

La caratteristica di questa libreria consiste nel fatto che le classi e le funzioni sono parametrizzate, cioè le componenti della libreria sono implementate attraverso il meccanismo del *template* delle classi e del *template* delle funzioni, visto nei paragrafi precedenti.

La STL è stata creata da **Alexander Stepanov** e **Meng Lee**, ricercatori di HPL (*Hewlett-Packard Laboratories*) ed è diventata una parte dello standard ANSI/ISO del linguaggio C++.

L'attività di programmazione che utilizza i template e la libreria STL si chiama **programmazione generica**, perché può disporre di strutture di dati e di algoritmi che possono essere adattati e applicati in numerosi contesti di elaborazione.

Le componenti fondamentali della STL sono:

- contenitori
- algoritmi
- iteratori.

I **contenitori** sono strutture dati standard che sono utilizzate in numerosi ambiti applicativi, sia semplici sia complessi; per esempio: vettori, liste, code, pile, ecc. Come abbiamo visto nei paragrafi precedenti queste strutture dati sono di uso molto frequente.

Gli **algoritmi** sono funzioni che implementano procedimenti di uso molto comune per la gestione delle strutture di dati; per esempio: conteggi, ricerche, ordinamenti, oppure copie di dati da una struttura all'altra.

Gli **iteratori** sono puntatori che consentono una navigazione rapida ed efficiente all'interno delle strutture di dati.

Le principali componenti della libreria dei *template* standard saranno presentate nei paragrafi successivi.

La STL mette a disposizione dei programmatori un insieme di strutture di dati e di algoritmi che possono essere utilizzati senza bisogno di specificarne l'implementazione; inoltre gli stessi algoritmi possono essere applicati a strutture di dati diverse. Si tratta quindi di strumenti flessibili e già testati, pronti per l'uso in una grande varietà di applicazioni.

L'efficienza di queste componenti non riguarda soltanto la possibilità di essere specializzate sulla base di specifiche esigenze, come già visto in precedenza per i *template*, ma anche il fatto che, per essere *template standard*, esse devono portare con sé l'**overloading degli operatori** (+, ++, <<, >>, ecc.), proprio perché devono essere utilizzate con tipi diversi. Inoltre gli algoritmi devono operare in modo corretto con i normali **puntatori**, per poter accedere ai dati contenuti nelle strutture.

La produttività del lavoro del programmatore aumenta, perché da una parte egli può applicare i propri algoritmi alle strutture dati della libreria e, dall'altra, può utilizzare gli algoritmi standard della libreria con le proprie strutture di dati.

## 11 Il contenitore per le stringhe

Negli esempi dei capitoli precedenti abbiamo già ampiamente utilizzato il tipo **string** e la libreria **<string>**.

Essa fa parte della libreria dei template standard e consente di operare in modo efficiente su dati di tipo stringa.

In effetti *string* è una classe della libreria standard che rappresenta un caso particolare di **contenitore**, costituito da un array di caratteri.

La classe *string* possiede le funzioni membro per operare sui dati di tipo stringa e definisce al suo interno l'**overloading degli operatori**:

- =, per l'assegnazione
- ==, per il confronto tra due stringhe
- +, per la concatenazione di stringhe
- <<, per l'output su stream
- >>, per l'input da stream.

Questo è il motivo per il quale nei progetti dei capitoli precedenti abbiamo potuto usare in modo semplice ed efficiente il tipo *string* per i dati di tipo alfabetico e alfanumerico, con le stesse modalità utilizzate per i tipi semplici del linguaggio C++ (*int*, *char*, *float*, *double*, ecc.). Per poter utilizzare il tipo *string* il programma deve contenere la direttiva di inclusione:

```
#include <string>
```

Sulla base di queste osservazioni si può capire che l'assegnazione di un valore ad una variabile di tipo *string* può essere fatto con un'inizializzazione tramite la seguente istruzione:

```
string s1("Torino");
```

oppure con un'istruzione di assegnazione come la seguente:

```
s1 = "Torino";
```

Nel primo caso si utilizza il costruttore della classe *string*, nel secondo caso l'*overloading* dell'operatore =.

Le costanti di tipo stringa, come già visto in precedenza, sono delimitate da una coppia di virgolette, " "; e coincidono con le costanti degli array di caratteri.

Tra le **funzioni membro** della classe *string* presentiamo le seguenti, perché sono di frequente utilità nella programmazione.

- **string::substr(indice, numero)**

restituisce una stringa ottenuta come sottostringa della stringa alla quale il metodo è applicato: il primo argomento tra parentesi è un indice all'interno della stringa e indica la posizione dalla quale iniziare l'estrazione dei caratteri (l'indice della stringa inizia da 0); il secondo argomento indica il numero dei caratteri da estrarre, cioè la lunghezza della sottostringa restituita.

Per esempio:

```
string s1, s2;

s1 = "Torino";
s2 = s1.substr(0, 2);
cout << s2 << endl;
```

Questo segmento di programma produce in output la stringa "To".

- **string::size()**

restituisce il numero di caratteri della stringa (lunghezza della stringa).

Per esempio:

```
string s1;
s1 = "Torino";
cout << s1.size() << endl;
```

Questo segmento di programma produce in output il numero 6.

- **string::replace(indice, numero, stringa)**

sostituisce una sottostringa con un'altra stringa che può anche avere un numero di caratteri diverso dal numero dei caratteri da sostituire: il primo argomento tra parentesi è un indice all'interno della stringa e indica la posizione dalla quale iniziare l'estrazione dei caratteri (l'indice della stringa inizia da 0); il secondo argomento indica il numero dei caratteri da sostituire, il terzo argomento indica la stringa che viene rimpiazzata al posto della sottostringa.

Per esempio:

```
string s1("Torino");
string s2("esi");

s1.replace(5, 1, s2);
cout << s1 << endl;
```

Questo segmento di programma produce in output la stringa "Torinesi".

- **string::c\_str()**

converte un tipo *string* in un array di caratteri *char\** per rendere il codice compatibile con le librerie per la gestione delle stringhe del linguaggio C.

Per esempio:

```
string s1("Padova");
char s2[30];

strcpy(s2, s1.c_str());
cout << s2 << endl;
```

Come si vede, la funzione `c_str` consente di convertire la variabile di tipo `string` in una stringa alla quale si possa applicare la funzione `strcpy` del linguaggio C per la copia tra stringhe.

Per effettuare l'operazione inversa, invece, da `char*` a `string`, basta usare una normale operazione di assegnazione con l'operatore `=`.

Per esempio:

```
string s1 = "Bologna";
char s2[30] = "Ancona";

s1 = s2;
```



## MATERIALE ONLINE: 5. La funzione `getline()`

## 12 Il contenitore `vector`

Il componente **vector** della libreria STL è un esempio di contenitore standard che può essere utilizzato nei problemi che richiedono l'organizzazione dei dati in array di dati omogenei tra loro.

Per usare questo contenitore occorre inserire all'inizio del programma la dichiarazione di inclusione che richiama la libreria dei vettori di STL contenente il *template* **vector<T>**:

```
#include <vector>
```

La dichiarazione di un vettore deve poi specificare il tipo di dato delle componenti dell'array; per esempio, la seguente istruzione crea un array di interi vuoto:

```
vector<int> v;
```

Si noti l'analogia con la sintassi utilizzata nel Paragrafo 9 relativo ai *template* delle classi.

In fase di dichiarazione è possibile fissare anche una dimensione iniziale per il vettore; per esempio:

```
const int MAX = 10;
vector<int> v(MAX);
```

Si osservi l'uso delle parentesi tonde per racchiudere il valore della dimensione dell'array, perché si sta passando un argomento al costruttore della classe `vector`. Per gli array tradizionali, presentati nel Capitolo 5, abbiamo invece usato la sintassi con le parentesi quadre tipiche del linguaggio C:

```
const int MAX = 10;
int v[MAX];
```

La classe `vector` possiede alcune funzioni membro, cioè i metodi che consentono di gestire in modo facile ed efficiente una struttura di dati di tipo array.

In particolare:

- il metodo **push\_back(dato)** aggiunge il `dato` specificato come argomento in coda alle componenti già esistenti nel vettore;

- il metodo **pop\_back()** estrae l'ultima componente del vettore;
- il metodo **clear()** elimina tutte le componenti del vettore;
- il metodo **size()** restituisce il numero delle componenti attualmente presenti nel vettore (dimensione attuale);
- il metodo **empty()** restituisce un valore *true* se il vettore è vuoto, altrimenti il valore *false*.

Inoltre la classe *vector* ha il costruttore e il distruttore, oltre all'*overloading* degli operatori per poter gestire la classe.

L'uso del contenitore *vector* consente una gestione più efficiente, ma anche più sicura, delle strutture dati di tipo array essenzialmente per i seguenti motivi:

- è possibile inserire componenti nell'array senza preoccuparsi di superare le dimensioni della struttura;
- si possono cancellare elementi dall'array senza lasciare posizioni non occupate nella struttura;
- non serve il contatore degli elementi, perché l'array conosce la sua dimensione attraverso il metodo *size()*;
- la classe *vector* gestisce la memoria allocata, evitando la creazione di *memoria leak*; infatti il contenitore tiene traccia della memoria occupata dalle sue componenti: quando serve memoria la alloca, quando non serve la libera. Inoltre non si devono scrivere istruzioni esplicite di allocazione e deallocazione.

## PROGETTO 9

### Inserire da tastiera un elenco di numeri interi e visualizzarli.

Il programma acquisisce da tastiera un elenco di numeri finché l'utente inserisce il numero 0. I valori sono aggiunti al vettore con il metodo *push\_back()*. Nella seconda parte del programma le componenti del vettore vengono visualizzate con una ripetizione: il contatore parte da 0 e si arresta al valore corrispondente alla dimensione dell'array, ottenuta dal metodo *size()*.

#### Programma C++

```
// CVettore.cpp: contenitore vector
#include <iostream>
#include <vector>
using namespace std;

// funzione principale
int main()
{
 vector<int> v;
 int d;

 // inserimento componenti
 cout << "Numero (0=fine): ";
 cin >> d;
 while (d != 0) {
 v.push_back(d);
 cout << "Altro numero (0=fine): ";
 cin >> d;
 }
}
```

```
// visualizza componenti
for (int i=0; i<v.size(); i++)
 cout << v[i] << endl;

return 0;
}
```

Si può osservare che la singola componente del vettore è identificata con la notazione **v[i]**, come già visto per gli array nei capitoli precedenti. In realtà, in questo caso il programma usa l'*overloading* dell'**operatore []**, definito all'interno della classe *vector*.

In alternativa si può usare anche il metodo **at(indice)** della classe *vector* che restituisce la componente del vettore che si trova alla posizione indicata come argomento. Con questo metodo la ripetizione per visualizzare il contenuto del vettore può essere riscritta in questo modo:

```
// visualizza componenti con funzione at()
for (int i=0; i<v.size(); i++)
 cout << v.at(i) << endl;
```

L'uso della funzione *at()*, al posto dell'*operatore []* per ritrovare le componenti di un array, presenta il vantaggio di poter prevenire situazioni potenzialmente dannose alla gestione della memoria, come illustrato nel prossimo paragrafo.

## 13 Gestione delle eccezioni

Si consideri la situazione che si viene a creare quando, utilizzando il programma del paragrafo precedente, si inseriscono 5 numeri nell'array e il ciclo di lettura delle sue componenti chiede di leggere le componenti dall'indice 0 all'indice 9.

Utilizzando la seguente ripetizione con l'*operatore []*, il programma legge le componenti mancanti in posizioni imprevedibili della memoria, producendo output non corretti:

```
for (int i=0; i<10; i++)
 cout << v[i] << endl;
```

Vediamo cosa accade invece se si usa la seguente ripetizione:

```
for (int i=0; i<10; i++)
 cout << v.at(i) << endl;
```

Utilizzando il metodo *at()*, il programma, dopo aver letto la quinta componente, si interrompe con un errore di *run-time* che produce sul video un messaggio simile al seguente:

Abnormal program termination

Quindi il metodo *at()* gestisce implicitamente anche il controllo sul *range* del vettore, segnalando una situazione di errore quando il *range* viene superato.

Nel linguaggio dell'informatica si dice che il programma genera una **eccezione** (in inglese, *exception*).

Un'applicazione software ben costruita deve prevedere tutti i casi che si possono verificare durante l'esecuzione. Quindi il programmatore deve essere in grado di **gestire le eccezioni** (*error handling*), intercettando le situazioni nelle quali si crea l'eccezione, per evitare l'interruzione anomala del programma, e inviando, di conseguenza, opportuni messaggi o avvertimenti all'utente.

Il linguaggio C++ prevede la gestione strutturata delle eccezioni, racchiudendo gruppi di istruzioni che possono generare errori durante l'esecuzione e specificando le attività da svolgere quando l'eccezione viene rilevata e intercettata.

La gestione strutturata è rappresentata dall'istruzione **try ... catch**, secondo il seguente schema sintattico generale:

```
try {
 istruzioni1;
}
catch(errore) {
 istruzioni2;
}
```

La parola *try* indica l'inizio del blocco delle istruzioni, racchiuse tra due parentesi graffe, sottoposte al controllo dell'esecuzione con l'intercettazione dell'eventuale situazione di errore. Dopo *catch* viene inserito, tra una coppia di parentesi graffe, il codice che deve essere eseguito quando le istruzioni del blocco *try* provocano un'interruzione dell'esecuzione perché si è verificato un *errore* del tipo specificato come argomento di *catch*.

La gestione delle eccezioni può contenere anche più *catch* in corrispondenza di diversi tipi di errore.

Nel linguaggio C++ l'eccezione è rappresentata dalla classe **exception** e i nomi da indicare come argomento di *catch* sono istanze della classe *exception*. Tra gli oggetti predefiniti della classe *exception* c'è anche l'eccezione **out\_of\_range** che intercetta la situazione nella quale un'operazione fa riferimento alla componente di una struttura dati al di fuori della dimensione della struttura stessa.

Per esempio, la ripetizione precedente che legge le componenti di un vettore può essere riscritta inserendo anche la gestione dell'eccezione, come si vede nel codice seguente:

```
try {
 // visualizza componenti con funzione at()
 for (int i=0; i<10; i++)
 cout << v.at(i) << endl;
}
catch(out_of_range) {
 cerr << "Errore: lettura oltre la dimensione dell'array"
 << endl;
}
catch(...) {
 cerr << "Errore non previsto" << endl;
}
```

Quando si verifica la situazione di errore vengono eseguite le istruzioni scritte dopo *catch*, inviando un messaggio di avvertimento all'utente sullo *standard error*, **cerr**, cioè il video.

È possibile specificare diversi *catch* in base al tipo di errore restituito dall'eccezione.

Il *catch* con tre puntini tra le parentesi tonde indica qualsiasi tipo di errore.

## 14 Iteratori

La libreria STL mette a disposizione del programmatore gli iteratori per ritrovare in modo semplice e rapido dati organizzati nei contenitori.

Gli **iteratori** sono una generalizzazione dei puntatori e possono essere applicati con le stesse modalità ai diversi contenitori. Forniscono il modo per accedere ad una determinata posizione all'interno del contenitore.

Ci sono due particolari metodi che determinano l'inizio e la fine di un contenitore:

- **begin()** restituisce un iteratore che punta al primo elemento del contenitore;
- **end()** restituisce un iteratore che punta alla posizione successiva all'ultimo elemento del contenitore.



Un iteratore deve essere dichiarato con un'istruzione che lo associa al contenitore secondo la seguente sintassi generale:

```
contenitore<T>::iterator nome;
```

Per esempio, l'iteratore *i* per un vettore di *int* viene dichiarato in questo modo:

```
vector<int>::iterator i;
```

Con l'iteratore così definito è possibile accedere, attraverso l'algebra dei puntatori, a tutti gli elementi di un contenitore *vector*, utilizzando la ripetizione nell'intervallo *begin()* e *end()*.

Per esempio:

```
vector<int> v;
vector<int>::iterator i;

for (i=v.begin(); i!=v.end(); i++)
 cout << *i << endl;
```

Si può osservare che alla variabile *i* viene applicata l'aritmetica dei puntatori per passare dal valore iniziale *v.begin()* al valore finale *v.end()* con incremento unitario.

La visualizzazione dell'elemento del vettore viene indicata dalla notazione *\*i*, che indica, come per i puntatori, il valore della variabile puntata.

Si può effettuare una lettura parziale all'interno del contenitore indicando il numero di posizioni da leggere; per esempio la seguente ripetizione visualizza i primi tre elementi del vettore (supponendo che esistano) a partire dalla posizione iniziale:

```
vector<int>::iterator i;
for (i=v.begin(); i!=v.begin()+3 ; i++)
 cout << *i << endl;
```

Si deve fare attenzione ad usare gli iteratori solo dopo aver concluso le operazioni di inserimento dei dati nel contenitore, per non far puntare ad aree di memoria non ancora allocate o già deallocate.





## USO DI UN TEMPLATE DI FUNZIONE PER VISUALIZZARE UN *VECTOR*

Abbiamo già detto in precedenza che la caratteristica più importante della libreria STL consiste nel far assumere a contenitori e iteratori comportamenti indipendenti dal tipo di dato utilizzato.

Questa osservazione suggerisce la possibilità di costruire una funzione per visualizzare il contenuto di un contenitore *vector*, inserendo come parametro proprio il tipo *vector*.

Costruiamo cioè un **template di funzione** che possa essere usato per vettori di interi, vettori di stringhe o altro allo stesso modo.

```
template <class T> void Stampa(vector<T> v)
{
 typename vector<T>::iterator p;

 cout << "Visualizzazione componenti del vettore:" << endl;
 for (p = v.begin(); p != v.end(); p++)
 cout << *p << endl;
 cout << endl;
}
```

Si noti l'uso della parola chiave **typename** per specificare al compilatore che la parte che precede *p* è il nome di un tipo.

Questo *template* di funzione visualizza i valori delle componenti di un vettore con una ripetizione del tutto simile a quella vista nel paragrafo precedente. Infatti il vettore può essere costituito da componenti di qualsiasi tipo e il *template* di funzione *Stampa()* individua il tipo di dati in base al vettore passato durante la chiamata (tipo di parametro attuale).

## PROGETTO 10

**Inserire da tastiera un elenco di sigle di province, memorizzandole in un array (l'inserimento termina con un'asterisco). Visualizzare poi tutte le sigle inserite.**

Il vettore è di tipo *string*. L'inserimento è realizzato con il metodo *push\_back* e la visualizzazione attraverso un *template* di funzione al quale viene inviato il nome del vettore.

### Programma C++

```
// CVettoreStringhe.cpp: contenitore vector di stringhe
#include <iostream>
#include <vector>
#include <string>
using namespace std;

template <class T> void Stampa(vector<T> v)
{
 typename vector<T>::iterator p;

 cout << "Visualizzazione componenti del vettore:" << endl;
 for (p = v.begin(); p != v.end(); p++)
 cout << *p << endl;
 cout << endl;
}
```

```
// funzione principale
int main()
{
 vector<string> prov;
 string sigla;

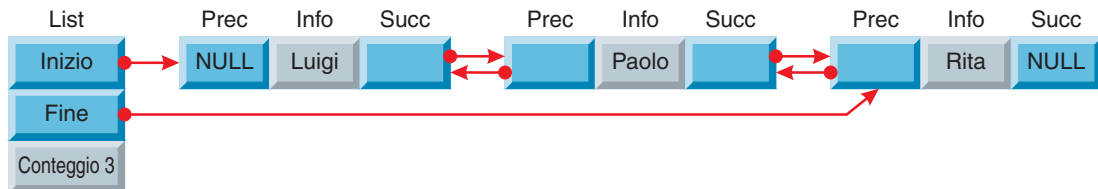
 // inserimento componenti
 cout << "Sigla provincia (*=fine): ";
 cin >> sigla;
 while (sigla != "") {
 prov.push_back(sigla);
 cout << "Altra provincia (*=fine): ";
 cin >> sigla;
 }

 Stampa(prov);

 return 0;
}
```

## 15 Il contenitore *list*

Il contenitore **list** della libreria STL implementa una struttura dati organizzata come una lista bidirezionale (*doubly-linked list*): in pratica ogni nodo della lista contiene, oltre alla parte dati, due puntatori, uno all'elemento successivo e uno all'elemento precedente.



Come il contenitore *vector*, la lista raggruppa dati omogenei tra loro.

Per usare questo contenitore occorre inserire all'inizio del programma la dichiarazione di inclusione che richiama la libreria delle liste di STL con il *template* **list<T>**:

```
#include <list>
```

La dichiarazione di un contenitore *list* deve poi specificare il tipo di dato delle componenti della lista; per esempio, la seguente istruzione crea una lista vuota di interi:

```
list<int> l;
```

Si noti l'analogia con la sintassi utilizzata per dichiarare un *vector*. Questo fornisce un'idea dell'unitarietà di impostazione della libreria STL.

Di seguito sono elencati alcuni tra i metodi associati alla classe *list*:

- **push\_back()**, per aggiungere un nuovo elemento alla fine della lista
- **push\_front()**, per aggiungere un nuovo elemento all'inizio della lista
- **size()**, per ottenere il numero degli elementi attualmente presenti nella lista

- **pop\_back()**, per eliminare l'ultimo elemento della lista
- **pop\_front()**, per eliminare il primo elemento della lista
- **clear()**, per eliminare tutti gli elementi della lista
- **empty()**, per controllare se la lista è vuota
- **sort()**, per ordinare la lista
- **reverse()**, per ristrutturare la lista in modo che gli elementi compaiano in ordine rovesciato, dall'ultimo al primo
- **merge()**, per effettuare la fusione di due liste in una sola (le due liste devono essere ordinate).

Anche a questo contenitore è possibile associare un **iteratore** e richiamare i metodi **begin()** ed **end()**, in modo del tutto analogo al contenitore *vector*.

## PROGETTO 11

**Memorizzare in una lista un insieme di nomi, acquisendoli da tastiera. Visualizzare poi l'elenco dei nomi prima in ordine alfabetico e poi in ordine inverso.**

Per inserire nella lista i nomi il programma usa il metodo *push\_back()*, che aggiunge un elemento di tipo *string* alla fine della lista, in coda agli elementi già esistenti. Con il metodo *sort()*, la lista viene poi ordinata e visualizzata. Per ottenere infine la lista in ordine rovesciato, si usa il metodo *reverse()*.

Si osservi che, per visualizzare il contenuto della lista, il programma esegue una ripetizione con contatore, attraverso i metodi *begin()* ed *end()* dell'iteratore associato:

```
list<string>::iterator i;
```

Il programa completo è riportato di seguito.

### Programma C++

```
// CLista.cpp: contenitore List
#include <iostream>
#include <list>
#include <string>
using namespace std;

// funzione principale
int main()
{
 list<string> elenco;
 string nome;

 // inserimento componenti
 cout << "Nome (*=fine): ";
 cin >> nome;
 while (nome != "") {
 elenco.push_back(nome);
 cout << "Altro nome (*=fine): ";
 cin >> nome;
 }
}
```

```
list<string>::iterator i;

elenco.sort();
for (i=enenco.begin(); i!=elenco.end(); i++)
 cout << *i << endl;

elenco.reverse();
for (i=enenco.begin(); i!=elenco.end(); i++)
 cout << *i << endl;

return 0;
}
```

Nel programma precedente è stato utilizzato il metodo *reverse()* per ottenere un elenco di dati in ordine rovesciato rispetto all'ordine originario della lista dal primo all'ultimo elemento.

Si deve però osservare che questo metodo rende permanente la memorizzazione degli elementi della lista in ordine rovesciato.

Volendo invece visualizzare solamente l'elenco in ordine inverso, mantenendo l'ordine di memorizzazione esistente all'interno del contenitore, si deve usare un **reverse\_iterator**, cioè un iteratore che percorre la lista in ordine inverso. In questo caso il metodo che restituisce l'iteratore al primo elemento si chiama **rbegin()** e il metodo per l'iteratore che punta alla fine della lista si chiama **rend()**.

La dichiarazione dell'iteratore inverso e la ripetizione per leggere il contenuto della lista dall'ultimo al primo elemento sono codificate in questo modo:

```
list<string>::reverse_iterator j;
for (j=enenco.rbegin(); j!=elenco.rend(); j++)
 cout << *j << endl;
```

Dopo aver visto i contenitori *vector* e *list*, si possono fare alcune considerazioni in termini di confronto e di criteri di scelta tra i due, a seconda del contesto applicativo.

Il vettore è organizzato in memoria centrale come un insieme di elementi contigui. Questo rende molto veloce l'accesso alle singole componenti del vettore, anche in modo diretto alla singola componente. Di contro la gestione della memoria diventa meno efficiente nel caso si debbano fare nuovi inserimenti o cancellazioni.

La lista bidirezionale, invece, è più efficiente nelle operazioni di inserimento e cancellazione, perché opera con i due puntatori, avanti e indietro, e non richiede memoria allocata in modo contiguo, anche se richiede complessivamente più memoria perché per ogni elemento occorre memorizzare, oltre ai dati, anche i due puntatori. Inoltre la lista bidirezionale non consente l'accesso diretto a un singolo elemento, perché non possiede l'*operatore [ ]* come il vettore.

Quindi è conveniente scegliere il contenitore *vector* quando si deve utilizzare una struttura dati che non richieda frequenti operazioni di inserimento o di eliminazione e nella quale si vuole accedere in modo veloce a un elemento di una specifica posizione. Il contenitore *list* invece è più efficiente quando l'accesso riguarda ugualmente tutti gli elementi, dal primo all'ultimo elemento, e la struttura deve essere frequentemente aggiornata con nuovi inserimenti o eliminazioni.

## 16 Il contenitore *map*

Il contenitore **map** della libreria STL implementa una struttura dati organizzata come un insieme di coppie di valori, di cui il primo rappresenta un valore **chiave** univoco all'interno della mappa, che consente di ritrovare facilmente il secondo valore associato nella coppia.

|                 |                 |
|-----------------|-----------------|
| chiave 1        | valore 1        |
| chiave 2        | valore 2        |
| ...             | ....            |
| chiave <i>n</i> | valore <i>n</i> |

Usando il gergo informatico, si dice che i valori sono *mappati*, cioè indicizzati, nel senso che a ciascun elemento è associata una chiave che consente di accedere velocemente ad esso. Per questo motivo la mappa è un esempio di **contenitore associativo** della libreria STL. Il primo e il secondo valore della coppia possono essere anche di tipo diverso.

Il contenitore *map* risolve problemi simili a quelli visti nel Paragrafo 3 a proposito di *array paralleli* (codice cliente e descrizione, matricola studente e nome, codice libro e titolo, ecc.). La differenza consiste nel fatto che nel caso del contenitore *map* si usa un'unica struttura di dati, anziché una coppia di array. Inoltre i valori che costituiscono il primo elemento della coppia sono tutti diversi tra loro e la struttura viene mantenuta automaticamente ordinata secondo l'ordine crescente dei valori delle chiavi.

Per usare questo contenitore occorre inserire all'inizio del programma la dichiarazione di inclusione che richiama la libreria delle liste di STL con il *template* **map<K,T>**:

```
#include <map>
```

La dichiarazione di un contenitore *map* deve poi specificare i due tipi di componenti della mappa; per esempio, la seguente istruzione crea una mappa di coppie, nelle quali il primo elemento è un intero e il secondo è una stringa:

```
map<int, string> m;
```

### PROGETTO 12

**I prodotti venduti da un'azienda commerciale sono identificati con una chiave, costituita da un codice numerico intero, e con una descrizione. Scrivere un programma per acquisire i dati dei prodotti e per visualizzarli in ordine crescente di codice.**

Per creare un contenitore adatto a contenere le informazioni sui prodotti si può usare la seguente dichiarazione:

```
map<int, string> prodotto;
```

L'accesso agli elementi della mappa avviene poi tramite la chiave; per esempio, per assegnare il valore *descr* al prodotto avente chiave *cod*, si scrive la seguente istruzione:

```
prodotto[cod] = descr;
```

Il valore della chiave è posto tra le parentesi quadre accanto al nome del contenitore.

### Programma C++

```
// CMap.cpp: contenitore map
#include <iostream>
#include <map>
#include <string>
using namespace std;

// funzione principale
int main()
{
 map<int, string> prodotto;
 int cod;
 string descr;

 // inserimento prodotti
 cout << "Codice prodotto (0=fine): ";
 cin >> cod;
 while (cod != 0) {
 cout << "Descrizione: ";
 cin >> descr;
 prodotto[cod] = descr;
 cout << "Codice prodotto (0=fine): ";
 cin >> cod;
 }

 // visualizza i prodotti
 map<int, string>::iterator i;
 for (i=prodotto.begin(); i!=prodotto.end(); i++)
 cout << i->first << ": " << i->second << endl;

 return 0;
}
```

Eseguendo il programma, si può osservare che i prodotti vengono elencati in ordine crescente di codice prodotto, indipendentemente dall'ordine con il quale sono stati inseriti, perché la mappa viene mantenuta ordinata secondo l'ordine della chiave.

Questo è il motivo per il quale il metodo *sort()* non è disponibile per il contenitore *map*.

Per visualizzare i prodotti si usa un iteratore, in modo del tutto analogo a quanto visto per i contenitori *vector* e *list*. L'unica differenza consiste nel fatto che ogni elemento della struttura è costituito da una coppia di valori e pertanto si deve usare l'**operatore ->** con l'iteratore corrente, per indicare il primo e il secondo dato della coppia, che si chiamano rispettivamente **first** e **second**:

```
map<int, string>::iterator i;
for (i=prodotto.begin(); i!=prodotto.end(); i++)
 cout << i->first << ": " << i->second << endl;
```

La modalità utilizzata è analoga a quella vista nel Capitolo 5 per i puntatori a dati di tipo *struct*.

In modo più preciso si dice che, nel contenitore *map*, l'iteratore punta a una coppia di dati, in inglese **pair**: i due campi della *pair* sono **first** e **second**, che indicano rispettivamente la **chiave** e il **valore**.

Il concetto di *pair* può essere utilizzato anche nelle operazioni di inserimento di nuovi elementi nella mappa, tramite il metodo **insert()**, in alternativa all'*operatore [ ]* visto nel programma precedente; per esempio, per assegnare la descrizione *prova* al prodotto avente codice *99*, si scrive la seguente istruzione:

```
prodotto.insert(pair<int, string>(99,"prova"));
```

Il *pair* è l'argomento passato alla funzione membro *insert()* del contenitore *map*.

Il metodo *insert()*, anche se non citato nei paragrafi precedenti, è disponibile anche per i contenitori *vector* e *list*.

La chiave è il mezzo per accedere agli elementi del contenitore. Per esempio, per eliminare il prodotto avente codice *99*, si deve usare il metodo **erase()**, passando come argomento il valore della chiave:

```
prodotto.erase(99);
```

Questo significa anche che la chiave non può essere modificata. Se si vuole modificare una chiave, per esempio il codice di un prodotto, occorre effettuare un'operazione di cancellazione, con il metodo *erase()*, della vecchia *pair*, e un'operazione di inserimento della nuova *pair* in cui è stata modificata la chiave.

Il contenitore *map* possiede anche i metodi **empty()** e **size()**, come i contenitori *vector* e *list*.

Uno dei vantaggi più importanti del contenitore *map* è rappresentato dalla possibilità di ritrovare facilmente i dati attraverso le chiavi. Per le operazioni di ricerca il contenitore *map* dispone del metodo **find()**, che restituisce un iteratore corrispondente alla coppia richiesta.

Il seguente esempio illustra la modalità di utilizzo della funzione membro *find()*.

### PROGETTO 13

#### Costruire un dizionario di parole che consenta la traduzione dei termini dall'italiano in una seconda lingua.

Il problema è già stato presentato come applicazione degli array paralleli nel Paragrafo 3. Qui il dizionario viene implementato usando il contenitore *map*. Si osservi che in questo caso ciascuna coppia è costituita da due dati *string* e quindi anche la chiave è un dato di tipo *string*.

```
map<string, string> dizionario;
```

Il programma è composto da due funzioni:

- la funzione *CaricaDizionario()* acquisisce i termini italiani e stranieri dalla tastiera e memorizza le coppie nel contenitore *map*;
- la funzione *Trova()* ricerca il termine italiano richiesto dall'utente e restituisce la relativa traduzione in lingua straniera, oppure un messaggio nel caso in cui il termine non sia presente nel dizionario.

Nella funzione *Trova()*, il valore restituito dal metodo *find()*, che riceve come argomento la chiave di ricerca, è di tipo *iterator*:

```
map<string, string>::iterator p;
```

Il vocabolo richiesto risulta presente nel dizionario se l'iteratore restituito è diverso dall'iteratore *end()*, cioè non è stata raggiunta la fine del contenitore:

```
p = dizionario.find(vocabolo);
if (p != dizionario.end()) { . . . }
```

### Programma C++

```
// CMapDizionario.cpp: traduzione di parole con contenitore map
#include <iostream>
#include <string>
#include <map>
using namespace std;

map<string, string> dizionario; // contenitore per le parole

// prototipi delle funzioni
void CaricaDizionario();
void Trova();

// funzione principale
int main()
{
 CaricaDizionario();
 Trova();

 return 0;
}

// caricamento del dizionario
void CaricaDizionario()
{
 string parola, traduz;

 cout << "-----" << endl;
 cout << "Carica il dizionario" << endl;
 cout << "Parola italiana (* = fine): ";
 cin >> parola;
 while (parola != "*") {
 cout << "Traduzione: ";
 cin >> traduz;
 dizionario[parola] = traduz;
 cout << "Parola italiana (* = fine): ";
 cin >> parola;
 }
} // CaricaDizionario

// cerca la parola nel dizionario
void Trova ()
{
 string vocabolo; // termine da ricercare
 string tradotto;
 map<string, string>::iterator p; // iteratore
```



```

cout << "-----" << endl;
cout << "Ricerca nel dizionario" << endl;
cout << "Parola da cercare (* = fine): ";
cin >> vocabolo;
while (vocabolo != "") {
 p = dizionario.find(vocabolo);
 if (p != dizionario.end())
 cout << "Traduzione: " << p->second << endl;
 else
 cout << "Termine non trovato" << endl;
 cout << "Altra parola da cercare (* = fine): ";
 cin >> vocabolo;
}
} // Trova

```

Il valore di ciascuna chiave è univoco all'interno di tutta la struttura *map*. Durante le operazioni di inserimento non viene effettuato alcun controllo nel caso di inserimento e, se la chiave è già presente, il nuovo valore associato sostituisce il precedente.

Per evitare questo, si può utilizzare la funzione membro *find()*, facendo controllare al programma la presenza nella mappa di un'eventuale chiave con lo stesso valore, prima di inserire un nuovo elemento nella struttura. Se la chiave è già presente, viene visualizzato un messaggio per l'utente.

Per esempio, la parte di codice, relativa all'inserimento dei prodotti del Progetto 12 di questo paragrafo, può essere riscritta aggiungendo il controllo sull'esistenza della chiave:

```

// inserimento prodotti
cout << "Codice prodotto (0=fine): ";
cin >> cod;
while (cod != 0) {
 i = prodotto.find(cod);
 if (i != prodotto.end())
 cout << "Chiave gia' esistente" << endl;
 else {
 cout << "Descrizione: ";
 cin >> descr;
 prodotto[cod] = descr;
 }
 cout << "Codice prodotto (0=fine): ";
 cin >> cod;
}

```

Si osservi che, per effettuare questa variazione al programma, occorre spostare la dichiarazione dell'iteratore

```
map<int, string>::iterator i;
```

all'inizio del *main()*.



**MATERIALE ONLINE**

## 6. Altri contenitori standard



### AUTOVERIFICA

Domande da 14 a 21 pag. 427-428  
Problemi da 17 a 27 pag. 430-431

## 17 Algoritmi generici

Come già detto nel Paragrafo 10, gli **algoritmi** della libreria STL sono funzioni che implementano procedimenti di uso comune per la gestione dei contenitori: conteggi, ricerche, ordinamenti, copie di dati da un contenitore all'altro. Il contenitore al quale l'algoritmo è applicato rappresenta un parametro per la funzione: pertanto gli algoritmi sono *template* di funzioni. Per poter usare gli algoritmi standard della libreria STL in un programma, occorre inserire all'inizio la direttiva di inclusione:

```
#include <algorithm>
```

Si faccia attenzione alla differenza tra le funzioni membro dei contenitori, che sono metodi e quindi vengono invocati secondo la notazione *oggetto.metodo*, e le funzioni non membro, come sono gli algoritmi generici della libreria STL, che usano la stessa sintassi delle normali funzioni di un programma.

Questo è importante perché alcuni procedimenti di uso comune possono essere metodi per un contenitore, mentre per altri contenitori occorre usare un *template* di funzione appartenente agli algoritmi della libreria STL.

Per esempio, l'algoritmo standard *find()* per la ricerca non si applica al contenitore *map*, per il quale esiste il metodo *find()*, come abbiamo già visto nel paragrafo precedente.

### • **find()**

L'algoritmo **find()** cerca un valore all'interno di un intervallo di un contenitore e restituisce un iteratore corrispondente alla posizione trovata; se non lo trova restituisce l'iteratore *end()* del contenitore:

```
vector<string> v1;
string cercato;

vector<string>::iterator p;
p = find(v1.begin(), v1.end(), cercato);
if (p != v1.end()) cout << *p << endl;
else cout << "Non trovato" << endl;
```

L'algoritmo *find()* richiede tre argomenti: il limite sinistro e il limite destro della ricerca oltre al valore da cercare.

### • **count()**

L'algoritmo generico **count()** conta gli elementi di un contenitore che hanno un valore uguale a un valore specificato. Anche questo algoritmo richiede tre parametri: i due limiti dell'intervallo di ricerca e il valore da confrontare.

## PROGETTO 14

**Dato un elenco di sigle di provincia, contare quante volte una provincia richiesta da tastiera compare nell'elenco**

Il programma usa un contenitore *vector* per memorizzare le stringhe delle sigle di provincia:

```
vector<string> prov;
```

Il risultato del conteggio è il valore restituito dalla funzione *count()*:

```
int conta = count(prov.begin(), prov.end(), cercata) ;
```

### Programma C++

```
// CVettoreConta.cpp: contenitore vector e algoritmo count
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
using namespace std;

// funzione principale
int main()
{
 vector<string> prov;
 string sigla;

 // inserimento componenti
 cout << "Sigla provincia (*=fine): ";
 cin >> sigla;
 while (sigla != "") {
 prov.push_back(sigla);
 cout << "Altra provincia (*=fine): ";
 cin >> sigla;
 }

 // conteggio della provincia richiesta
 string cercata;
 cout << "Sigla da contare: ";
 cin >> cercata;
 int conta = count(prov.begin(), prov.end(), cercata);
 cout << "Risultato conteggio = " << conta;

 return 0;
}
```

Anche l'algoritmo *count()* non si applica al contenitore *map*, per il quale occorre usare l'apposito metodo avente lo stesso nome.

#### • **for\_each()**

L'algoritmo **for\_each** (*per ciascuno*) consente di applicare il codice contenuto in una funzione, specificata come terzo argomento dell'algoritmo, agli elementi di un contenitore compresi tra i due limiti specificati come primo e secondo argomento dell'algoritmo.

Per esempio, la seguente istruzione esegue le operazioni della funzione *f* su tutti gli elementi di un vettore *v*:

```
for_each(v.begin(), v.end(), f);
```

Nell'istruzione compare solo il nome della funzione *f* senza parentesi tonde e senza specificazione di parametri.

L'implementazione della funzione *f* nel programma deve invece contenere, come parametro, una variabile dello stesso tipo degli elementi del contenitore.

## PROGETTO 15

In una lista vengono memorizzati i dati sui corsi attivati in un'università: la descrizione del corso, l'anno universitario del corso (1, 2, 3) e il cognome del docente. Si vogliono costruire i codici identificativi dei corsi, formati dai primi tre caratteri della descrizione, dalla cifra dell'anno e dai primi tre caratteri del cognome del docente. I codici così formati sono registrati in un vettore. Alla fine il programma deve visualizzare l'elenco dei codici creati.

Il programma utilizza una *struct* per le informazioni di ciascun corso:

```
struct corso {
 string descrizione;
 char anno;
 string docente;
};
```

I corsi, inseriti da tastiera, sono memorizzati in un contenitore *list*:

```
list<corso> daticorsi;
```

Il contenitore per i codici è un *vector*:

```
vector<string> codici;
```

La funzione che costruisce il codice di un corso è implementata in questo modo:

```
void CreaCodice(corso c)
{
 string cod;
 cod = c.descrizione.substr(0,3) + c.anno + c.docente.substr(0,3);
 codici.push_back(cod);
}
```

Come si vede la funzione *CreaCodice()* ha un argomento *c* dello stesso tipo *corso* degli elementi della lista. La funzione viene eseguita per ciascun elemento del contenitore *list* attraverso l'algoritmo *for\_each*:

```
for_each(daticorsi.begin(), daticorsi.end(), CreaCodice);
```

Il terzo argomento dell'algoritmo *for\_each* è il nome della funzione senza parentesi tonde e senza parametri.

Il programma completo utilizza anche una funzione per il caricamento dei dati nella lista e una funzione per la visualizzazione dei codici creati.

### Programma C++

```
// CorsiUniv.cpp: codici dei corsi universitari
#include <iostream>
#include <list>
#include <vector>
#include <algorithm>
#include <string>
using namespace std;
```

```

 struct corso {
 string descrizione;
 char anno;
 string docente;
 };

 list<corso> daticorsi;
 vector<string> codici;

 // prototipi delle funzioni
 void CaricaCorsi();
 void CreaCodice(corso);
 void VisualizzaCodici();

 // funzione principale
 int main()
 {
 CaricaCorsi();
 for_each(daticorsi.begin(), daticorsi.end(), CreaCodice);
 VisualizzaCodici();

 return 0;
 }

 // caricamento corsi
 void CaricaCorsi()
 {
 corso c;
 // inserimento dati
 cout << "Descrizione del corso (*=fine): ";
 cin >> c.descrizione;
 while (c.descrizione != "") {
 cout << "Anno del corso (1-3): ";
 cin >> c.anno;
 cout << "Cognome del docente: ";
 cin >> c.docente;
 daticorsi.push_back(c); // aggiunge alla lista
 cout << "Altra descrizione (*=fine): ";
 cin >> c.descrizione;
 }
 } // CaricaCorsi

 void CreaCodice(corso c)
 {
 string cod;
 cod = c.descrizione.substr(0,3) + c.anno + c.docente.substr(0,3);
 codici.push_back(cod);
 } // CreaCodice

 // visualizzazione codici creati
 void VisualizzaCodici()
 {
 vector<string>::iterator i;
 cout << "--- Elenco codici dei corsi ---" << endl;
 for (i=codici.begin(); i!=codici.end(); i++)
 cout << *i << endl;
 } // VisualizzaCodici

```

## 18 Algoritmi per l'ordinamento

L'ordinamento di un contenitore può essere eseguito con l'algoritmo generico **sort()**, che mette in ordine crescente gli elementi.

### PROGETTO 16

#### Visualizzare in ordine alfabetico le parole contenute in un vettore.

Il programma utilizza un contenitore *vector* per i termini da ordinare:

```
vector<string> termini;
```

L'algoritmo *sort()* viene applicato passando come parametri l'intervallo degli elementi da ordinare; per esempio, per ordinare tutto il contenitore, si deve scrivere la seguente istruzione:

```
sort(termini.begin(), termini.end());
```

#### Programma C++

```
// OrdinaVettore.cpp: uso di sort con contenitore vector per termini
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
using namespace std;

// funzione principale
int main()
{
 vector<string> termini;
 string t;

 // inserimento termini
 cout << "Termine (*=fine): ";
 cin >> t;
 while (t != "") {
 termini.push_back(t);
 cout << "Altro termine (*=fine): ";
 cin >> t;
 }

 // ordinamento alfabetico
 sort(termini.begin(), termini.end());

 // elenco ordinato
 vector<string>::iterator i;
 for (i=termini.begin(); i!=termini.end(); i++)
 cout << *i << endl;

 return 0;
}
```

L'algoritmo *sort()* restituisce l'ordine crescente degli elementi di un contenitore. Se si vuole l'ordine decrescente occorre usare l'algoritmo generico **reverse()**, di rovesciamento del contenitore, dopo aver riordinato il contenitore:

```
sort(termini.begin(), termini.end());
reverse(termini.begin(), termini.end());
```

Dopo l'esecuzione di questi due algoritmi, però, l'ordine degli elementi nel contenitore risulta rovesciato.

Se si vogliono ottenere gli elementi in ordine decrescente, mantenendo però la loro posizione nel contenitore, si deve prima ordinare gli elementi con l'algoritmo *sort()* e poi usare un **reverse\_iterator** con gli iteratori **rbegin()** e **rend()**, come già visto per le liste:

```
sort(termini.begin(), termini.end());
vector<string>::reverse_iterator j;
for (j=termini.rbegin(); j!=termini.rend(); j++)
 cout << *j << endl;
```

L'algoritmo *sort()* non si applica al contenitore *list*, che possiede un metodo con lo stesso nome. Ricordiamo inoltre che il contenitore *map* è implicitamente ordinato secondo i valori crescenti delle chiavi.

L'algoritmo **merge()** consente di fondere due contenitori in un unico contenitore ordinato. I contenitori di partenza devono anch'essi essere già ordinati. Per esempio, il seguente frammento di codice ordina e fonde due vettori *v1* e *v2* in un terzo vettore *v3* ordinato:

```
vector<string> v1, v2, v3;
...
sort(v1.begin(), v1.end());
sort(v2.begin(), v2.end());
merge(v1.begin(), v1.end(), v2.begin(), v2.end(), v3.begin());
```

Anche l'algoritmo *merge()* non si applica al contenitore *list* che possiede un metodo con lo stesso nome.

Gli algoritmi **max\_element()** e **min\_element()** restituiscono gli iteratori agli elementi con valore massimo e valore minimo del contenitore:

```
vector<int> v;
vector<int>::iterator i;

i = max_element(v.begin(), v.end());
cout << "Massimo: " << *i << endl;
i = min_element(v.begin(), v.end());
cout << "Minimo: " << *i << endl;
```

## ALTRI ALGORITMI STANDARD

Di seguito vengono presentati altri algoritmi standard della libreria STL di uso comune. Per mostrare l'uso di questi algoritmi faremo riferimento alle seguenti dichiarazioni di contenitori *vector*:

```
vector<int> v1;
vector<int> v2(3); // 3 componenti
```

Gli esempi riguardano il contenitore *vector*, ma gli algoritmi possono essere applicati anche ad altri contenitori standard.

Contenuto iniziale del vettore *v1*:

1    2    3    1    1    4    5    7

L'algoritmo **copy()** copia gli elementi di un contenitore, secondo un intervallo specificato, in un altro contenitore. Per esempio:

```
// copy: copia i primi tre elementi
copy(v.begin(), v.begin() + 3, v2.begin());
```

Contenuto del vettore *v2* dopo l'esecuzione di *copy()*:

1    2    3

L'algoritmo **replace()** sostituisce tutti gli elementi di un contenitore, compresi in un intervallo specificato e aventi un valore prefissato, con un nuovo valore fornito come parametro all'algoritmo:

```
// replace: cambia i valori 1 con valore 100
replace(v1.begin(), v1.end(), 1, 100);
```

Contenuto del vettore *v1* dopo l'esecuzione di *replace()*:

100   2    3    100   100   4    5    7

L'algoritmo **unique()** riduce a uno gli elementi adiacenti di uguale valore, anche se il contenitore non è ordinato.

```
vector<int>::iterator i;
// unique: riduce a uno gli elementi uguali adiacenti
i = unique(v1.begin(), v1.end());
v1.erase(i, v1.end());
```

Contenuto del vettore *v1* dopo l'esecuzione di *unique()*:

100   2    3    100   4    5    7

Riducendo a uno gli elementi di uguale valore, li sposta a sinistra, lasciando i vecchi valori memorizzati nel vettore. Occorre quindi anche eliminare, con il metodo **erase()**, gli elementi dalla posizione dell'ultimo elemento sistemato alla fine del vettore. L'algoritmo *unique()* non si applica al contenitore *list* che possiede un metodo con lo stesso nome.

### AUTOVERIFICA

Domande da 22 a 24 pag. 428  
Problemi da 28 a 33 pag. 431



### ■ MATERIALE ONLINE

**7. Programmi eseguibili e videate dei progetti presentati nel capitolo**  
**8. Lezioni multimediali sulla programmazione delle strutture di dati in C++**



## DOMANDE

### Array e tabelle

- 1 Le funzioni indicate sotto sono tratte dal Progetto 1 (*Ricerca di un elemento in un array*), ma hanno subito una modifica che non permette il caricamento corretto dei dati nel vettore. Quale modifica è stata fatta?

```
void ChiediDimensione()
{
 int d;
 do {
 cout << "Dimensione dell'array: ";
 cin >> d;
 } while (d<1 || d>MAX);
}

void CaricaVettore(int v[], int d)
{
 for (int i=0; i<d; i++) {
 cout << "Elemento di posto " << i << ": ";
 cin >> v[i];
 }
}
```

- 2 Supponendo che l'array *v* contenga i seguenti dati  
 $v = \{12, 53, 32, 20, 4\}$   
e che *i* e *j* valgano rispettivamente 1 e 4, come sono disposti i dati nell'array dopo l'esecuzione della seguente istruzione che richiama la funzione per lo scambio di due variabili (si ricordi che gli indici partono da 0) ?

```
if (v[i] > v[j]) Scambia(v[i],v[j]);
```

- 3 Quale tra le seguenti è la dichiarazione corretta di array paralleli per rappresentare autori, titoli e prezzi di 50 libri?
- a) Const int Max = 50; string[Max] autore; string[Max] titolo; float[Max] prezzo;
  - b) Const int Max = 50; float autore[Max]; float titolo[Max]; string prezzo[Max];
  - c) Const int Max = 50; string autore[Max]; string titolo[Max]; float prezzo[Max];
  - d) Const int Max = 50; string autore, titolo; float prezzo[Max];
- 4 Quale delle seguenti istruzioni di assegnazione è corretta per impostare il valore 2012 per l'anno di edizione del 9.o libro in un array di libri denominato *biblioteca*?
- a) biblioteca.AnnoEdiz[9] = 2012;
  - b) AnnoEdiz.biblioteca[9] = 2012;
  - c) biblioteca.AnnoEdiz[8] = 2012;
  - d) biblioteca[8].AnnoEdiz = 2012;

### Pila, coda e lista

- 5 Quando si verifica una situazione di *stack overflow* durante l'esecuzione di un programma?
- a) Quando, in un programma contenente funzioni ricorsive, l'accumularsi di versioni dello stesso sottoprogramma causa il riempimento della memoria.
  - b) Quando un programma contenente funzioni ricorsive termina.
  - c) Quando in una struttura a pila si esegue un'operazione di *Push* seguita da un'operazione di *Pop*.
  - d) Quando si usano array di dimensioni elevate.

- 6 Quale delle seguenti coppie di istruzioni si deve usare per l'operazione di *Push* in una pila rappresentata con un array (che parte da 0):
- Pila[Testa]=dato; Testa++;
  - Testa--; Pila[Testa]=dato;
  - Testa++; Pila[Testa]=dato;
  - Pila[Testa]=dato; Testa--;
- 7 Quale delle seguenti coppie di istruzioni si deve usare per l'operazione di *Pop* in una pila rappresentata con un array:
- Testa++; cout << Pila[testa] << endl;
  - Testa--; cout << Pila[testa] << endl;
  - cout << Pila[testa] << endl; Testa--;
  - cout << Pila[testa] << endl; Testa++;
- 8 Come viene chiamata la struttura coda?
- LIFO
  - FIFO
- 9 Quale delle seguenti coppie di istruzioni si deve usare per l'operazione di *Push* in una coda rappresentata con un array:
- Fine--; Coda [Fine] = dato;
  - Coda [Fine] = dato; Fine--;
  - Fine++; Coda [Fine] = dato;
  - Coda[Fine] = dato; Fine++;
- 10 Data la definizione

```
struct Prova {
 int c1;
 char c2[20];
};
Prova* p;
p = new Prova;
```

quale tra le seguenti è l'istruzione corretta per assegnare il valore 20 al campo *c1*?

- c1->p = 20;
- p->c1 = 20;
- p(c1) = 20;
- p.c1 = 20;

## Template

- 11 Quale tra le seguenti frasi definisce il significato di *template* di una funzione?
- Funzione con un parametro che può essere sostituito con un valore al momento della chiamata della funzione stessa.
  - Funzione con un parametro che può essere sostituito con un tipo o classe al momento della chiamata della funzione stessa.
  - Funzione con un parametro che può essere sostituito con un template di una classe al momento della chiamata della funzione stessa.
  - Funzione con un parametro che può essere sostituito con un array di valori al momento della chiamata della funzione stessa.

- 12** Quale tra le seguenti è la sintassi corretta per la dichiarazione di un *template* delle classi?
- template class<T> NomeClasse { ... }
  - template class NomeClasse<T> { ... }
  - template class NomeClasse <class T> { ... }
  - template<class T> class NomeClasse { ... }
- 13** Come si istanzia una classe *Prova* utilizzando il template *TProva* con il tipo *double*?
- TProva<double> Prova;
  - Prova<double> TProva;
  - TProva double Prova ;
  - <double> TProva Prova;

## Libreria STL

- 14** Quali tra i seguenti termini rappresentano le componenti fondamentali della libreria STL?
- puntatori
  - iteratori
  - attributi
  - metodi
  - contenitori
  - algoritmi

- 15** Che cosa scrive in output il seguente frammento di programma?

```
string s1, s2;

s1 = "Automobile";
s2 = s1.substr(0, 4);
cout << s2 << endl;
```

- 16** Che cosa scrive in output il seguente frammento di programma?

```
string s1("Automobile");
string s2("carro");

s1.replace(4, 6, s2);
cout << s1 << endl;
```

- 17** Associa ad ogni metodo della classe *vector* elencato a sinistra il corrispondente significato scegliendo tra le frasi elencate a destra.
- |                |                                            |
|----------------|--------------------------------------------|
| a) push_back() | 1) Elimina tutte le componenti del vettore |
| b) pop_back()  | 2) Restituisce il numero delle componenti  |
| c) clear()     | 3) Aggiunge una componente in coda         |
| d) size()      | 4) Controlla se il vettore è vuoto         |
| e) empty()     | 5) Elimina l'ultima componente del vettore |
- 18** Quale delle seguenti frasi spiega in modo corretto il significato dell'istruzione *try... catch*?
- Il trasferimento del controllo all'istruzione immediatamente successiva a quella nella quale è stato generato l'errore.
  - La gestione degli errori anche se non è stato generato alcun errore.
  - La gestione strutturata delle eccezioni che sia in grado di intercettare le situazioni di errore fornendo all'utente opportuni messaggi.
  - La visualizzazione di un messaggio di errore da parte del sistema e l'interruzione del programma.

- 19** Quale tra le seguenti è la sintassi corretta per dichiarare un iteratore per un vettore di interi?
- a) `vector<int>::iterator i;`
  - b) `vector<int> iterator i;`
  - c) `vector<int iterator> i;`
  - d) `iterator<int> i;`
- 20** Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
- a) Il vettore è organizzato in memoria centrale come un insieme di elementi contigui V F
  - b) Il vettore non consente l'accesso diretto ad una singola componente V F
  - c) La lista bidirezionale è più efficiente solo quando si devono fare poche operazioni di inserimento e cancellazione V F
  - d) Con la lista bidirezionale non si può usare l'operatore `[ ]` V F
- 21** Che cosa viene restituito dal metodo `find()` del contenitore `map`?
- a) chiave
  - b) iteratore
  - c) valore
  - d) coppia chiave + valore

### Algoritmi

- 22** Completa la tabella specificando con una lettera A o con una lettera M se le funzioni elencate sono algoritmi (A) o metodi (M) per i contenitori indicati sulle colonne.

|                         | vector | list | map |
|-------------------------|--------|------|-----|
| <code>find()</code>     |        |      |     |
| <code>count()</code>    |        |      |     |
| <code>for_each()</code> |        |      |     |

- 23** Quali delle seguenti istruzioni mette in ordine crescente gli elementi di un vettore `v`?
- a) `v.sort(v.begin(), v.end());`
  - b) `sort(v.begin(), v.end());`
  - c) `v.sort(begin(), end());`
  - d) `sort(begin(), end());`

- 24** Dato il seguente vettore:

```
vector<int> v1;
```

e supponendo che contenga i seguenti elementi:

3 4 4 4 5 6 6 7 7

scrivere nella colonna di destra il contenuto del vettore `v1` dopo l'esecuzione del codice indicato nella colonna a sinistra.

| Codice                                                                                                                               | Contenuto di <code>v1</code> |
|--------------------------------------------------------------------------------------------------------------------------------------|------------------------------|
| <code>replace(v1.begin(), v1.end(), 4, 10);</code>                                                                                   | .....                        |
| <code>vector&lt;int&gt;::iterator i;</code><br><code>i = unique(v1.begin(), v1.end());</code><br><code>v1.erase(i, v1.end());</code> | .....                        |

## PROBLEMI

### Array e tabelle

- 1 Dato l'elenco dei numeri estratti in una lotteria, dopo aver acquisito da tastiera il numero giocato, applicare la ricerca sequenziale per stabilire se il numero cercato è stato estratto.
- 2 Dato l'elenco dei treni in partenza da una stazione, dopo aver acquisito da tastiera il numero del treno da ricercare, stabilire se il treno cercato è compreso nell'elenco.
- 3 Dopo aver caricato in un array i voti degli studenti di una classe conseguiti in una prova, ordinarli in modo crescente e scrivere il valore dei primi tre voti dell'elenco ottenuto dopo l'ordinamento.
- 4 Dopo aver caricato in un array le temperature rilevate in una settimana, ordinarle in modo crescente e scrivere la temperatura massima.
- 5 Codificare la tabella che possa contenere i dati relativi ai DVD musicali.
- 6 Alla fine di un concorso vengono pubblicati i risultati delle prove dei concorrenti: vengono memorizzati, per ogni partecipante, il nome, il punteggio della prima prova e il punteggio della seconda prova. Si vuole ottenere la graduatoria finale con la visualizzazione dei concorrenti ordinati in modo decrescente rispetto al punteggio totale conseguito.
- 7 I modelli e i prezzi delle automobili vendute in un autosalone sono acquisiti da tastiera e organizzati in una tabella. Dopo aver ordinato le automobili secondo il prezzo crescente, visualizzare i modelli che hanno un prezzo inferiore ad una cifra fornita da tastiera e alla fine la percentuale di questi sul totale delle automobili esaminate.
- 8 Le trasmissioni in prima serata su un canale televisivo nei diversi giorni di un mese sono organizzati in una tabella con giorno, titolo, regista, durata. L'elenco viene successivamente ordinato. Fornendo da tastiera il giorno, l'utente può ottenere il titolo della trasmissione, il nome del regista e la durata.

### Pila, coda e lista

- 9 La percentuale di interessi attivi sul deposito di un conto corrente bancario cambia più volte nel corso dell'anno. Utilizzare una pila per memorizzare questi valori. Fornito poi il saldo attuale, calcolare gli interessi lordi maturati usando l'ultimo valore dell'interesse.
- 10 In una coda vengono registrate le prenotazioni per la partecipazione a un convegno secondo l'ordine di arrivo: inserire nel programma la codifica dell'operazione che consenta di registrare i nomi dei partecipanti man mano che arrivano e dell'operazione che consenta di visualizzare i nomi dei partecipanti, a cui si deve inviare una lettera di conferma, prendendo i nomi a partire dall'inizio della coda.
- 11 Scrivere il programma per visualizzare gli elementi, a partire dal quinto, all'interno di una lista di numeri formata da almeno 10 elementi, seguendo l'ordine stabilito dai puntatori.
- 12 Dopo aver dichiarato una lista per memorizzare un elenco ordinato di città, scrivere la funzione per inserire una nuova città fornita da tastiera in modo da mantenere ordinata la lista.

## Template

- 13** Costruire un template di funzione che possa essere utilizzata con dati numerici di tipo diverso (*int*, *double*, *float*) per determinare se, tra due numeri passati alla funzione, il primo numero è maggiore del secondo: la funzione deve restituire un valore booleano *true* o *false*.
- 14** Costruire un template di funzione per il calcolo dell'area di un triangolo, in modo che possa essere utilizzata sia quando le misure della base e dell'altezza sono valori *int*, sia quando sono *double*.
- 15** Costruire un template per rappresentare in forma analitica una retta: gli attributi sono i coefficienti *m* e *q* dell'equazione  $y = m x + q$ , i metodi consentono l'acquisizione dei coefficienti da tastiera e la visualizzazione dell'equazione della retta. Utilizzare poi il template per una retta avente i coefficienti interi e una seconda retta avente i coefficienti *double*.
- 16** Una matrice numerica può essere pensata come un array nel quale le componenti sono memorizzate riga per riga della matrice. Indicato con *rig* il numero delle righe e *col* il numero delle colonne, l'array ha dimensione *rig* \* *col* e l'elemento generico di una matrice di indici (*i*, *j*) corrisponde all'elemento dell'array che si trova nella posizione  $(i - 1) * col + j$ . Costruire una classe template per rappresentare una matrice numerica come un array e scrivere il programma che acquisisce e visualizza gli elementi di una matrice di numeri interi e di una matrice di numeri *float*.

## Libreria STL

- 17** Fornita da tastiera una stringa, visualizzare i caratteri della stringa uno per ogni riga del video.
- 18** Dato un elenco di codici formati da una sequenza di cifre e lettere, per ragioni di riservatezza si devono sostituire i primi 4 caratteri con "\*\*\*\*". Visualizzare i codici ottenuti con la sostituzione.
- 19** Dopo aver inserito da tastiera un elenco di numeri *float*, memorizzandoli in un vettore, visualizzare quelli che sono superiori ad una cifra fornita dall'utente.
- 20** Acquisire da tastiera un elenco di parole, memorizzandole in un vettore, finché l'utente segnala la fine dell'inserimento con un asterisco, \*. Visualizzare alla fine il numero delle parole memorizzate utilizzando il metodo opportuno del template *vector*. Successivamente, dopo aver eliminato tutte le parole dal vettore, controllare che il vettore sia vuoto.
- 21** Un vettore contiene le misurazioni (numeri *double*) acquisite da tastiera con un ciclo di inserimento. Successivamente l'utente fornisce una posizione nel vettore perché vuole ottenere il valore della misurazione corrispondente. Il programma visualizza il valore richiesto, gestendo l'eccezione che si verifica quando la posizione richiesta è al di fuori del range.
- 22** Dopo aver memorizzato in un vettore di interi 20 numeri, visualizzare gli ultimi 4 numeri inseriti usando un iteratore.
- 23** Utilizzando due array, uno numerico e uno stringa, si inseriscono da tastiera i nomi dei reparti di un centro commerciale e il numero di addetti di ciascun reparto. Visualizzare poi, con due distinti elenchi, tutti i nomi e tutti i numeri richiamando due volte un template di funzione.
- 24** Da tastiera si inserisce un elenco di numeri, memorizzandoli in una lista bidirezionale: se il numero è pari viene inserito all'inizio, altrimenti alla fine. Visualizzare poi i numeri inseriti prima in ordine di inserimento e poi in ordine crescente.

- 25** Organizzare con una mappa la rubrica con i nomi delle persone e i rispettivi numeri telefonici. Fornito poi il nome della persona, il programma visualizza il suo numero di telefono oppure un messaggio nel caso la persona non sia presente nella rubrica.
- 26** I nomi delle città e i corrispondenti Codici di Avviamento Postale (CAP) vengono inseriti da tastiera e memorizzati in una mappa, nella quale il CAP è la chiave. Fornito poi da tastiera il nome di una città, il programma visualizza il suo CAP oppure un messaggio nel caso la città non sia compresa nell'elenco, mentre fornendo il CAP restituisce il nome della città o un messaggio di errore.
- 27** Si devono organizzare in una struttura dati i valori degli occupati negli ultimi dieci anni. Utilizzare una mappa, assegnando il ruolo di chiave all'anno. Inserire i dati da tastiera e memorizzarli nel contenitore, controllando anche che non venga inserito due volte lo stesso anno. Calcolare poi il valore medio dei dieci anni e visualizzare il risultato.

### Algoritmi

- 28** Dato un vettore di numeri, controllare se il numero richiesto dall'utente è presente nel vettore.
- 29** Dato un vettore contenente l'elenco dei comuni di provenienza degli studenti di una classe, contare quante volte compare il nome di un comune richiesto dall'utente.
- 30** Dato un vettore di numeri, memorizzare per ciascun numero in un altro vettore il doppio del numero se è minore di 100, il triplo se è maggiore o uguale.
- 31** Dato l'elenco dei cognomi degli studenti di 1A e quelli degli studenti di 1B, memorizzare in un unico vettore l'elenco alfabetico degli studenti delle due classi.
- 32** Dopo aver memorizzato in un vettore i tempi conseguiti da 10 concorrenti in una prova di velocità, copiare in un altro vettore i primi tre tempi migliori.
- 33** I voti assegnati a 30 studenti di una classe in una prova di italiano sono memorizzati in un vettore. Dopo aver ordinato i voti in ordine crescente, visualizzare quali voti diversi tra loro sono stati assegnati, riducendo a uno i voti uguali.



## SCHEDA DI AUTOVALUTAZIONE

### CONOSCENZE

- ☐ algoritmi di ricerca e ordinamento di array
- ☐ array paralleli
- ☐ tabelle
- ☐ pila
- ☐ coda
- ☐ liste dinamiche
- ☐ template di funzioni
- ☐ template delle classi
- ☐ libreria standard STL
- ☐ contenitori
- ☐ iteratori
- ☐ gestione delle eccezioni
- ☐ algoritmi generici e algoritmi standard

### ABILITÀ

- ☐ scrivere il programma per la ricerca sequenziale in un array
- ☐ scrivere il programma per l'ordinamento di un array
- ☐ caricare ed elaborare dati di una tabella
- ☐ utilizzare gli array per rappresentare le strutture di pila e di coda
- ☐ implementare le operazioni per la gestione dinamica delle liste
- ☐ definire un template di funzione
- ☐ definire un template di classe
- ☐ definire un template di classe con due parametri
- ☐ utilizzare le funzioni membro del contenitore string
- ☐ utilizzare il contenitore vector
- ☐ utilizzare un template di funzione per visualizzare un contenitore vector
- ☐ inserire nel codice l'uso degli iteratori
- ☐ utilizzare il contenitore list
- ☐ utilizzare il contenitore map
- ☐ applicare gli algoritmi generici
- ☐ applicare gli algoritmi per l'ordinamento
- ☐ applicare gli algoritmi standard



**SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 663**



# 8

**parte terza**

La programmazione

## Flussi di dati

### **OBIETTIVI DI APPRENDIMENTO**

In questo capitolo imparerai la gerarchia delle classi del linguaggio C++ per la gestione degli stream.

Sarai in grado di costruire applicazioni che organizzano i dati in archivi e che utilizzano gli stream con operazioni di input e output.

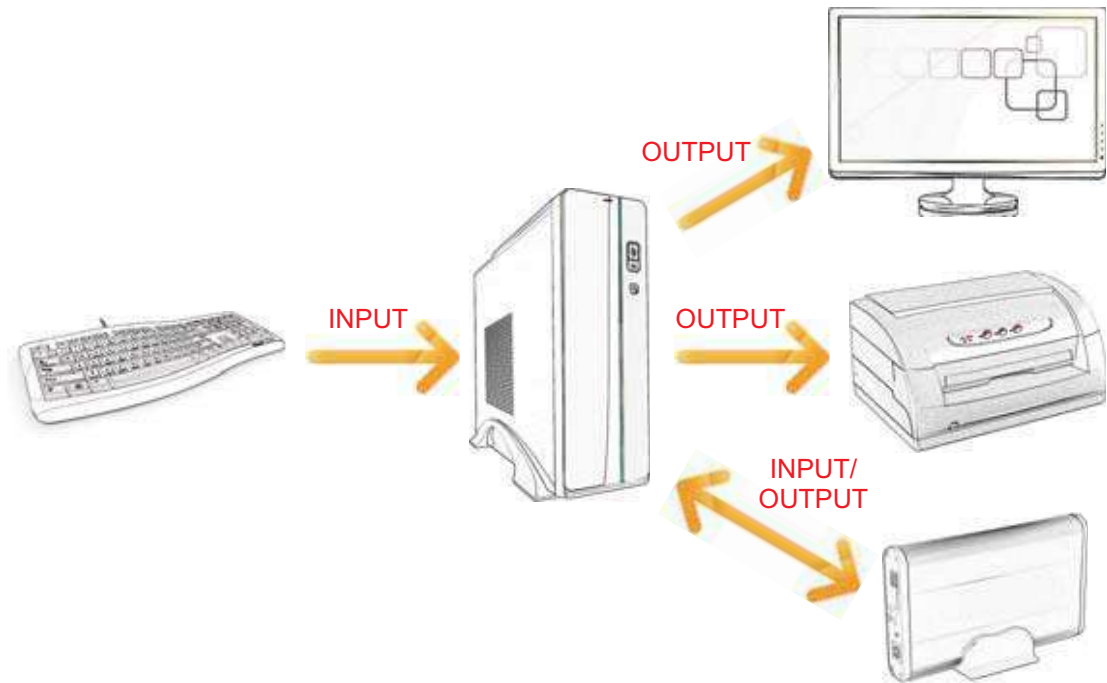
Imparerai a distinguere i diversi tipi di file e le diverse modalità per l'accesso agli archivi di dati, utilizzando le istruzioni di lettura e scrittura e gli iteratori per gli stream.

- Classi di input/output
- Input e output standard
- Overloading dell'operatore <<
- Archivi di dati
- File di testo e file binari
- Operazioni e accesso agli archivi
- Lettura e scrittura in un file di testo
- Apertura esplicita dei file
- Errori nelle operazioni sui file
- Gestione strutturata delle eccezioni
- Controllo della fine del file
- Stream su stampante
- Archivi con record a lunghezza costante
- Iteratori per gli stream

## 1 Le classi di input/output

Il linguaggio C++ non dispone di proprie istruzioni per la gestione dell'interazione con l'utente. Si appoggia infatti su appositi comandi scritti per l'architettura della macchina e facenti parte delle librerie standard del linguaggio.

Le operazioni di ingresso e uscita dei dati sono gestite attraverso le risorse delle **librerie standard di I/O** (*Input/Output*): esse forniscono anche gli strumenti per leggere e scrivere negli **archivi di dati** che si trovano sui supporti di memoria di massa (dischi fissi, dischi esterni, nastri, memorie USB).



I flussi di dati dal programma alla periferica (*output*), sia essa lo schermo, la stampante o un disco, e i flussi di dati dalla periferica, la tastiera o il disco, al programma (*input*) sono gestiti attraverso gli *stream*.

Il termine *stream* si traduce in italiano con **flusso**.

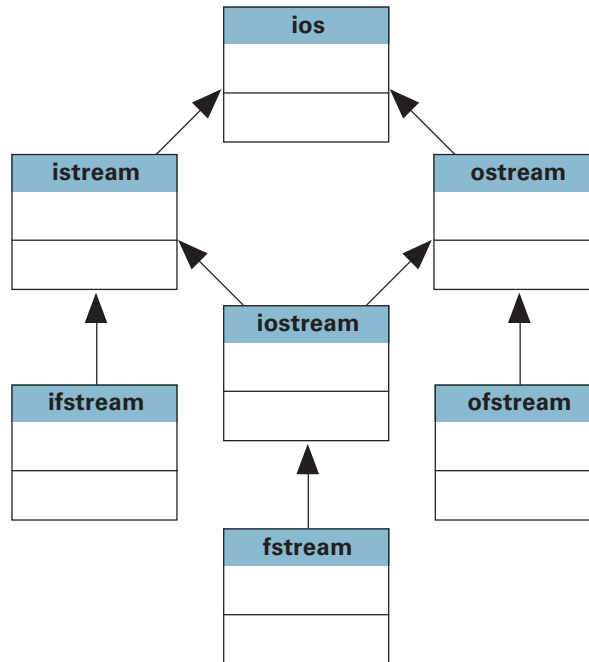
Per semplicità possiamo definire lo **stream** come il nome interno al programma utilizzato per identificare la periferica.

Dal punto di vista della programmazione ad oggetti lo *stream* è l'**oggetto periferica**, ossia l'istanza della classe definita per la gestione delle periferiche. È infatti attraverso l'uso dei metodi (funzioni membro) di uno *stream* o, più in generale, di una classe, che si gestiscono i flussi di dati da e verso una periferica.

Ogni volta che si utilizza uno *stream*, per esso viene riservata un'area di memoria centrale, detta **buffer di I/O**, utilizzata per le operazioni di lettura e scrittura sul file.

Le classi di input/output, o classi **iostream**, consentono al programma di trasformare oggetti di un certo tipo in caratteri di testo visualizzabili sullo schermo oppure di convertire i dati provenienti dalla tastiera oppure, ancora, di scrivere e leggere dati in formato binario sulle memorie di massa.

Le classi sono organizzate in una struttura gerarchica, avente come radice la **classe ios**, secondo il seguente schema:



Le dichiarazioni delle classi di input/output sono contenute nei due principali file di libreria:

- **iostream**, che contiene la definizione delle classi *ios*, *istream*, *ostream*, *iostream* per la gestione dei flussi standard di input/output;
- **fstream**, che contiene le definizioni delle classi *fstream*, *ifstream*, *ofstream* per la gestione dei flussi collegati ai file registrati su memoria di massa o, in generale, ai file identificati con un nome logico.

Quindi i programmi che utilizzano i flussi devono contenere le direttive di inclusione:

```
#include <iostream>
#include <fstream>
```

Il primo tipo di inclusione è già stato utilizzato in tutti i programmi dei capitoli precedenti.

## 2 Input e output standard

Come abbiamo già visto a livello introduttivo nel Capitolo 5, i programmi scritti in linguaggio C++ utilizzano quattro flussi o **stream predefiniti**, che vengono aperti al momento dell'esecuzione:

- **cin** è un oggetto della classe *istream* e indica il flusso associato all'unità standard di input (*standard input*), cioè la tastiera;
- **cout** è un oggetto della classe *ostream* e indica il flusso associato all'unità standard di output (*standard output*), cioè il video;
- **cerr** è un oggetto della classe *ostream* e indica il flusso associato all'unità standard per la visualizzazione dei messaggi di errore (*standard error*), cioè il video;
- **clog** è un oggetto della classe *ostream* e indica il flusso per la registrazione delle operazioni svolte dal sistema durante una sessione di lavoro (*standard log*).

Le operazioni di output su *stream* sono eseguite attraverso l'**operatore di scrittura <<**



Le operazioni di input su *stream* sono eseguite attraverso l'**operatore di lettura >>**



Per leggere da standard input singoli caratteri o array di caratteri si può usare anche la funzione (metodo) **get()**.

Per esempio, per leggere un nome avente al massimo 19 caratteri, si possono usare le istruzioni:

```
char nome[20];
cin.get(nome, 20);
```

Occorre precisare che l'operatore **>>** trascura gli spazi, le tabulazioni e i caratteri di fine riga (**'\n'**). Per un corretto funzionamento delle operazioni di input che riguardano stringhe di caratteri, che possono contenere anche spazi bianchi (per esempio, cognome e nome di una persona acquisiti sulla stessa riga), è più appropriato utilizzare il metodo **getline()** dell'oggetto *cin*.

La lettura con *get()* o *getline()* termina con **'\n'**, che viene convertito in **'\0'**, oppure con la lettura del numero di caratteri specificati meno 1, aggiungendo in coda il carattere **'\0'** di fine stringa. Solitamente anche una lettura con *cin* termina con il carattere **'\n'** (*newline*). L'operatore **>>** non inserisce tale carattere nella stringa di input e lo lascia nel *buffer di I/O*, cioè nella memoria temporanea, associato allo stream *cin*.

Questo significa che una successiva operazione di input con *get()* o *getline()*, trovando il carattere di terminazione, lo scarica dal buffer di I/O, lo converte in **'\0'** e non esegue l'acquisizione dei dati. Occorre quindi eliminare il carattere di terminazione con il metodo **ignore**:

```
cin.ignore(80, '\n');
```

che rimuove dal buffer di input tutti i caratteri, fino a quando incontra il primo carattere **'\n'** (*newline*). Il numero 80 indica il numero massimo di caratteri di una riga.

Nell'esempio seguente, i dati acquisiti da tastiera vengono subito dopo visualizzati. Il programma mostra un esempio di uso del metodo **getline()**. Si osservi alla fine del programma anche l'uso del metodo **get()** per gestire l'attesa dell'inserimento di un carattere da parte dell'utente.

```
// Tastiera.cpp: acquisizione di caratteri

#include <iostream>
using namespace std;

int main()
{
 char codice[5], nome[40], telefono[24];
```

```
// acquisizione dei dati
cout << "Inserimento nuovo cliente " << endl;
cout << "----- " << endl;
cout << "Codice cliente: ";
cin >> codice;
cin.ignore(80, '\n');
cout << "Nome Societa': ";
cin.getline(nome, 40);
cout << "Telefono: ";
cin >> telefono;
cin.ignore(80, '\n');
// visualizza
cout << codice << endl;
cout << nome << endl;
cout << telefono << endl;
cout << endl;
// attesa
cout << "premi Invio per continuare" << endl;
cin.get();
}
```

Gli operatori `>>`, `<<` e le funzioni `get()` e `getline()`, oltre che per gli stream standard `cin` e `cout`, sono impiegati anche con gli *stream* di file, come vedremo nei prossimi paragrafi.



## OVERLOADING DEGLI OPERATORI <<

L'operatore `<<` può essere sottoposto a *overloading* per poter effettuare output di dati che sono istanze di classi.

Per mostrare un'applicazione dell'**overloading dell'operatore** `<<`, il progetto seguente fa riferimento alla classe *Stringa* presentata nei Progetti 8 e 9 del Capitolo 6, dove erano stati mostrati due tipi di overloading, per gli operatori di assegnazione `=` e di confronto `==`. Aggiungiamo ora ai metodi della classe *Stringa* l'overloading dell'operatore `<<` per visualizzare le istanze della classe *Stringa*.

### PROGETTO 1

**Definire l'operatore << per l'output di una stringa organizzata con un array di caratteri.**

Nella classe *Stringa* definita nei progetti del Capitolo 6, viene aggiunto l'overloading dell'operatore `<<` per le operazioni di output di un oggetto dichiarato come istanza della classe *Stringa*. Esso sostituisce il metodo *Mostra()* presente nei progetti del Capitolo 6, rendendo il programma più ordinato e omogeneo con le usuali operazioni di output realizzate con l'istruzione `cout`.

| Stringa                                                   |
|-----------------------------------------------------------|
| s                                                         |
| operatore =<br>friend operatore ==<br>friend operatore << |

Si osservi che, nel linguaggio C++, l'overloading sull'operatore << è di tipo **friend**, perché questo operatore, per poter funzionare correttamente, non può essere membro di una classe, ma una funzione a sé stante. La codifica modificata della classe *Stringa* e dell'overloading dell'operatore << è riportata di seguito: l'operatore di output ha due parametri, rappresentati dal flusso di output che è un oggetto di classe *ostream* e dall'oggetto di classe *Stringa* da visualizzare.

```
class Stringa {
 char s[30];
public:
 // metodi
 Stringa operator= (const char* sorgente);
 friend bool operator== (Stringa&, Stringa&);
 friend ostream& operator<< (ostream&, Stringa&);
};

ostream& operator<< (ostream& os, Stringa& a)
{
 os << a.s;

 return os;
}
```

Per l'implementazione degli altri overloading di operatori si rivedano i progetti del Capitolo 6. Il tipo restituito dalla funzione `operator <<` è una referenza allo stream *os* di tipo *ostream* per consentire anche operazioni di output ripetute, come la seguente:

```
cout << s1 << s2 << s3;
```

Per verificare il funzionamento dell'operatore così definito, si provi ad eseguire il programma con il seguente *main()*, che sfrutta entrambi gli operatori sottoposti ad overloading: l'operatore `=`, definito nel Progetto 8 del Capitolo 6 per l'assegnazione, e l'operatore `<<`, definito ora per la visualizzazione.

```
// funzione principale
int main()
{
 Stringa s1, s2; // istanze della classe

 s1 = "Milano";
 s2 = "Roma";
 cout << s1;
 cout << endl;
 cout << s2;
 cout << endl;
 cout << s1 << ' ' << s2 << endl;

 return 0;
}
```

Il programma produce correttamente l'output anche nel caso di visualizzazione di più stringhe sulla stessa riga.

L'esempio precedente suggerisce la possibilità di aggiungere, per analogia, anche l'**overloading dell'operatore >>** per poter effettuare l'input di oggetti *Stringa* dal flusso *cin* della tastiera. In questo caso dobbiamo usare come flusso un oggetto di classe **istream** al posto di *ostream*.

Seguendo la traccia precedente si può scrivere la codifica del metodo per la classe *Stringa*. Anche questo overloading deve essere di tipo **friend**.

```
istream& operator>> (istream& is, Stringa& a)
{
 is >> a.s;

 return is;
}
```

Dopo aver definito l'overloading dell'operatore >> si possono usare istruzioni come le seguenti per acquisire da tastiera i valori per le istanze della classe *Stringa*, anche con l'immissione di diverse variabili in una stessa istruzione:

```
cin >> s1;
cin >> s3 >> s4;
```



#### AUTOVERIFICA

Domande da 1 a 3 pag. 466

Problemi da 1 a 2 pag. 468



#### MATERIALI ONLINE

1. Applicazione dell'overloading dell'operatore <<
2. Manipolatori e formattazione dell'output

## 3 Archivi di dati

Comunemente per indicare un archivio di dati si usa il termine inglese **file**.

In generale il termine *file* sta ad indicare tutto ciò che può essere registrato su un supporto di memoria di massa (testo, lettera, relazione, programma, archivio di dati).

Il file può essere anche considerato come un insieme di record legati tra loro da un nesso logico, cioè un insieme di informazioni strutturate e organizzate in un archivio.

Il file è quindi una struttura di dati che realizza, nella memoria di massa, la struttura in memoria centrale, vista nel Capitolo 7 con il nome di tabella. Ricordiamo infatti che una tabella è un array di record.

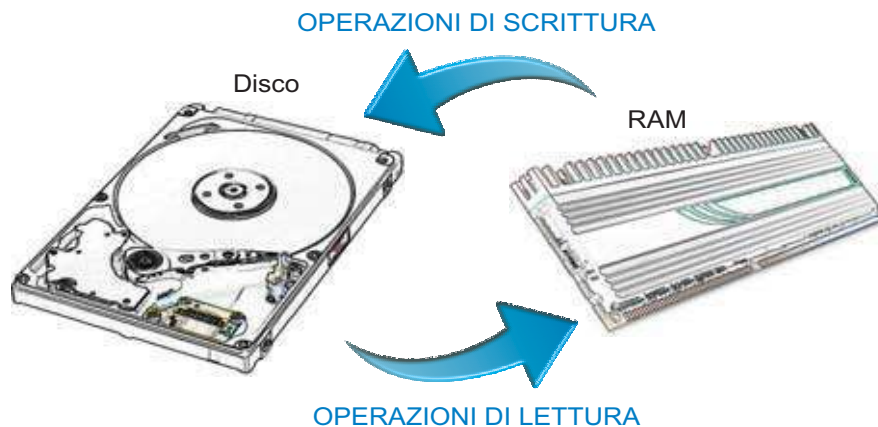
Questa struttura viene organizzata su un supporto di memoria esterna. I file vengono identificati attraverso un nome di file: di solito i nomi sono significativi, cioè ricordano il tipo di dati che contengono.

Le caratteristiche dei nomi di file (lunghezza, set di caratteri utilizzabili) dipendono dal *file system* del sistema operativo in uso.

Le operazioni fondamentali riguardanti il trattamento di un file sono:

- **apertura** del file: si stabilisce un collegamento tra la memoria centrale e l'unità di memoria di massa che contiene il file sul quale si vogliono fare operazioni di lettura e scrittura;
- **chiusura** del file: si chiude il collegamento tra la memoria centrale e l'unità di memoria di massa che contiene il file. Dopo la chiusura non è più possibile effettuare alcuna operazione su quel file a meno che non si effettui una nuova apertura.
- **lettura** dal file: le operazioni che trasferiscono dati dal file alla memoria centrale;
- **scrittura** sul file: le operazioni che trasferiscono dati dalla memoria centrale al file.

La lettura si indica anche con *input* e la scrittura con *output*, e le operazioni che trasferiscono informazioni dalla memoria centrale al file e viceversa si chiamano, in generale, **operazioni di I/O** (*Input/Output*) sul file.



Con l'operazione di apertura del file viene anche riservata un'area di memoria centrale, detta **buffer di I/O**, utilizzata per velocizzare le operazioni di lettura e scrittura sul file. L'area viene rilasciata con l'operazione di chiusura o con specifici comandi di svuotamento del buffer.

## 4 File di testo e file binari

I file trattati dal *file system* di un sistema operativo si possono classificare secondo due tipologie, in base all'organizzazione dei dati in essi contenuti:

- file di testo o file ASCII
- file binari.

La sostanziale differenza tra questi due tipi di file riguarda la loro leggibilità: i **file di testo** o **file ASCII** contengono i caratteri di fine riga (**EOL**, *End of Line*) per fornire un'impaginazione delle informazioni contenute e ciascun byte rappresenta un carattere della codifica ASCII, mentre i **file binari** sono considerati come un flusso continuo di bit, anch'essi raggruppati in byte, ma non necessariamente distinti l'uno dall'altro.

Storicamente la classificazione in file ASCII e in file binari era molto rilevante in quanto le informazioni riguardanti l'utente, per esempio le lettere, le bolle, le fatture, oltre ai programmi sorgenti realizzati dal programmatore, o, in una sola parola, i **dati**, venivano memorizzati in file prontamente leggibili da parte dell'utente, senza bisogno di un apposito programma di lettura.

Viceversa le informazioni riguardanti il computer, per esempio i programmi applicativi, i comandi di sistema, gli interpreti dei comandi (*shell*) o, in generale, i **programmi eseguibili**, venivano memorizzati in file prontamente comprensibili dal processore della macchina, dato che le informazioni erano memorizzate in **linguaggio macchina**.

Un tempo quindi i dati a disposizione dell'utente erano organizzati in file di testo. Con l'evolversi e lo specializzarsi dei programmi per l'automazione dell'ufficio (*office automation*), i file utilizzati da tali pacchetti sono stati personalizzati in base al programma che li ha creati. Per questa ragione, i file creati da un programma di elaborazione di testi (per esempio *Word*) sono leggibili e modificabili esclusivamente da quel programma. La stessa cosa accade con un foglio elettronico o un database: infatti, nell'ambiente *Windows*, per visualizzare in modo leggibile un file creato con *Word* o con *Access* non si utilizza il programma *Blocco Note*, che consente di visualizzare correttamente solo file di testo.



Oggi, quindi, i file di testo vengono impiegati solamente per i linguaggi di programmazione e nei casi in cui il trasferimento dei dati da un applicativo all'altro sia incompatibile. I programmi applicativi professionali, invece, per una maggiore efficienza conservano le informazioni, da essi trattate, in file binari. In questa evoluzione i programmi eseguibili hanno mantenuto le caratteristiche proprie dei file binari.

Per la loro caratteristica i **file di testo** risultano particolarmente adatti per memorizzare dati di lunghezza diversa. Inoltre occupano più spazio su disco, però sono più facili da gestire e possiedono una maggiore portabilità, nel senso che possono essere letti su computer aventi sistemi operativi diversi.

I **file binari** sono invece adatti per archivi di record a lunghezza fissa, occupano meno spazio su disco e sono più veloci nelle operazioni di lettura e scrittura.

Nei paragrafi seguenti useremo i file binari per gli archivi aventi **record a lunghezza costante** e i file di testo per gli archivi con **record a lunghezza variabile**.

## 5 Operazioni e accesso agli archivi

Un programma in linguaggio C++, che deve utilizzare i dati contenuti in un file, prevede le seguenti fasi di lavoro:

- 1) definizione di una variabile di tipo *fstream*, *ifstream*, *ofstream* da associare al file
- 2) apertura del file specificandone il nome e il tipo di accesso
- 3) operazioni di manipolazione o di interrogazione del file
- 4) chiusura del file.

Le operazioni di **manipolazione** riguardano la **lettura** di dati dall'archivio alla memoria centrale e, viceversa, la **scrittura** di dati dalla memoria centrale all'archivio.

Ci possono essere anche operazioni di **riscrittura** quando i dati già esistenti nell'archivio vengono aggiornati sostituendo quelli esistenti con dati nuovi.

Le **interrogazioni** consentono il reperimento all'interno dell'archivio delle informazioni necessarie per l'elaborazione desiderata. Il risultato dell'interrogazione può essere rappresentato mediante una **visualizzazione** sul monitor del computer oppure con una **stampa** su carta. Gli output su stampante sono correntemente indicati con il termine **report**.

La tabella seguente riassume i metodi delle classi di libreria I/O del linguaggio C++ per eseguire le principali operazioni sui file; il loro significato e le modalità di utilizzo verranno spiegate negli esempi seguenti.

| Operazioni | Metodi / Operatori | File di testo | File binario |
|------------|--------------------|---------------|--------------|
| Apertura   | open()             | ✓             | ✓            |
| Scrittura  | <<                 | ✓             |              |
| Scrittura  | write()            |               | ✓            |
| Lettura    | >>                 | ✓             |              |
| Lettura    | read()             |               | ✓            |
| Chiusura   | close()            | ✓             | ✓            |

Indipendentemente dai dati che esso contiene, un file è comunque composto da un insieme di byte registrati su disco.

Le modalità con le quali il programma utilizza i byte contenuti nei blocchi fisici del disco, rendendoli disponibili in memoria centrale per l'elaborazione, rappresentano il tipo di accesso al file:

- accesso **sequenziale**: in fase di lettura i dati possono essere ritrovati scorrendo tutti i record del file a partire dal primo fino all'ultimo, nello stesso ordine con il quale sono stati registrati.
- accesso **diretto**: potendo utilizzare record di lunghezza fissa e aventi tutti la stessa struttura, ci si può riferire alla specifica componente, identificando la sua posizione all'interno del file.

L'accesso diretto è anche detto **random** (casuale), mettendo in evidenza la contrapposizione con l'organizzazione sequenziale, in quanto l'utente può accedere alla componente di suo interesse senza leggere tutte quelle che la precedono.

## 6 Lettura e scrittura in un file di testo

Un file può essere utilizzato per scrivere nuove registrazioni, per leggere dati o per aggiungere dati in coda a quelli già registrati.

Un file può essere aperto per le seguenti operazioni:

- scrivere caratteri in un file (*output*)
- leggere caratteri da un file (*input*)
- aggiungere caratteri in un file in coda a quelli già esistenti (*append*).

La scrittura a partire dall'inizio, eseguita su un file già esistente, provoca la cancellazione di eventuali dati presenti nell'archivio; è per questa ragione che dobbiamo distinguere l'operazione di scrittura (*output*) dall'operazione di aggiunta in coda (*append*).

Nelle operazioni di lettura in un archivio con *accesso sequenziale*, occorre introdurre un opportuno controllo per segnalare il raggiungimento della *fine del file*.

Il progetto seguente descrive il **programma di scrittura** di un file di testo, in generale un file avente record di lunghezza variabile. Poiché la scrittura avviene dall'inizio, di fatto il programma esegue la *creazione* di un nuovo archivio, anche nei casi in cui l'archivio già esiste.

### PROGETTO 2

#### Creare un archivio con i nomi degli amici.

Si vuole creare un archivio su memoria di massa che contenga i nomi dei propri amici. Viene inserito in input l'elenco dei nomi degli amici: i nomi devono essere uno a uno registrati in un file di testo, ogni riga del file contiene un nome di amico. L'archivio è aperto in scrittura e ciò comporta la cancellazione di un eventuale archivio preesistente con lo stesso nome e la scrittura dei dati inseriti a partire dall'inizio del file.

All'inizio del programma viene definito l'oggetto *fout* della classe **ofstream**, associato al nome fisico del file, *amici.dat*, cioè al nome con cui il file è registrato su disco. La lettera iniziale della classe (lettera 'o', di *ofstream*) indica che *fout* è un oggetto di output.

Il costruttore contiene l'**apertura implicita del file**: vedremo in seguito gli esempi con l'apertura esplicita, tramite il metodo *open()*.

La scrittura sul file viene eseguita con l'**operatore <<** applicato all'oggetto *fout*. Al termine del programma il file viene chiuso con il metodo **close()**.

Si noti la direttiva di inclusione della libreria **fstream** per consentire al programma di utilizzare le classi di I/O:

```
#include <fstream>
```

### Programma C++

```
// CreaAmici.cpp: creazione archivio
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
 string nome;
 ofstream fout("amici.dat"); // apre il file in scrittura
 cout << "inserisci i nomi" << endl;
 cout << "uno per riga (Ctrl+Z per terminare)" << endl;
 while (cin >> nome) {
 fout << nome << endl; // scrive sul file
 }
 fout.close(); // chiude il file

 return 0;
}
```

La fine dell'inserimento dei nomi dalla tastiera è segnalata dall'utente con la combinazione dei tasti Ctrl + Z (nei sistemi Linux, Ctrl + D), che indica la fine dell'input da linea comandi.

Per controllare il contenuto di un file di testo già esistente si può utilizzare un **programma di lettura**.

Poiché i dati registrati vengono letti dal file uno di seguito all'altro, a partire dall'inizio, nello stesso ordine con il quale erano stati scritti, cioè con *accesso sequenziale*, occorre inserire all'interno del programma di lettura un controllo sul raggiungimento della **fine del file**.

## PROGETTO 3

### Visualizzare il contenuto dell'archivio degli amici precedentemente creato.

Per la visualizzazione del contenuto dell'archivio degli amici si procede utilizzando una ripetizione che comprende la lettura sequenziale di ogni riga del file e la sua visualizzazione. La ripetizione si arresta dopo che è stata letta l'ultima registrazione del file.

All'inizio del programma viene definito un oggetto *fin* della classe **ifstream**, associato al nome fisico del file, *amici.dat*. La lettera iniziale della classe (lettera 'i', di *ifstream*) indica che *fin* è un oggetto di input.

La lettura sul file viene eseguita con l'**operatore >>** applicato all'oggetto *fin*.

Per usare la classe *ifstream* occorre inserire la direttiva di inclusione della libreria **fstream**.

### Programma C++

```
// LeggiAmici.cpp: Lettura archivio
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
```

```
int main()
{
 string nome;
 ifstream fin("amici.dat"); // apre il file in lettura
 while (fin >> nome) { // legge dal file fino alla fine
 cout << nome << endl;
 }
 fin.close(); // chiude il file

 return 0;
}
```

Si possono aggiungere altri dati ad un archivio creato in precedenza specificando la modalità **append** nell'apertura del file.

Per esempio, se si vogliono aggiungere altri nomi di amici all'archivio, si può usare un programma del tutto simile a quello usato per l'inserimento dei nomi.

L'unica modifica richiesta riguarda l'aggiunta della costante **ios::app** nella dichiarazione del file, per specificare la modalità di apertura in *append*. La costante appartiene all'insieme delle **costanti enumerative** per l'apertura del file: esse sono definite nella classe **ios**, contenuta nella libreria *iostream*. Le altre costanti, presentate in seguito, sono **ios::in**, per la modalità di lettura, e **ios::out**, per la modalità di scrittura.

### Programma C++

```
// AggiungiAmici.cpp: aggiunta di nuovi nomi
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
 string nome;
 // apre il file in append
 ofstream fout("amici.dat", ios::app);
 cout << "inserisci nuovi nomi" << endl;
 cout << "uno per riga (Ctrl+Z per terminare)" << endl;
 while (cin >> nome) {
 fout << nome << endl; // scrive sul file
 }
 fout.close(); // chiude il file

 return 0;
}
```

Dopo aver costruito il programma per l'aggiunta di nuovi nomi, si può verificare che i nomi già esistenti nell'archivio siano stati mantenuti, eseguendo di nuovo il programma di lettura del file.

## 7 Apertura esplicita dei file

L'operazione di apertura di un file può essere ottenuta anche tramite il metodo **open()** applicato all'oggetto *stream*. In questo caso l'apertura si compone di due passi:

- creazione di una variabile *stream* con costruttore senza argomenti, per esempio:

```
ofstream fout;
```

- connessione dello *stream* all'archivio di dati tramite il metodo *open()*, specificando il nome del file e la modalità di apertura; per esempio, nel programma di aggiunta di nuovi dati, si usa la seguente istruzione:

```
fout.open("amici.dat", ios::app);
```

Di seguito, con riferimento all'archivio di nomi dei progetti precedenti, sono descritte le variazioni da apportare alle prime istruzioni dei programmi per la creazione del file, la lettura sequenziale e l'aggiunta di nuovi dati in coda a quelli già esistenti, utilizzando il metodo *open()*:

```
ofstream fout;
fout.open("amici.dat", ios::out);

ifstream fin;
fin.open("amici.dat", ios::in);

ofstream fout;
fout.open("amici.dat", ios::app);
```

Le costanti *ios::out* e *ios::in* possono essere anche tralasciate perché sono i valori di *default* rispettivamente per gli oggetti di tipo *ofstream* e *ifstream*.

A titolo di esempio viene riportato il programma completo di creazione dell'archivio, modificato con l'introduzione dell'apertura esplicita dell'archivio, tramite il metodo *open()*.

### Programma C++

```
// CreaAmici2.cpp: apertura esplicita
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
 string nome;
 ofstream fout; // dichiara lo stream
 fout.open("amici.dat"); // connette lo stream all'archivio
 cout << "inserisci i nomi" << endl;
 cout << "uno per riga (Ctrl+Z per terminare)" << endl;
 while (cin >> nome) {
 fout << nome << endl; // scrive sul file
 }
 fout.close(); // chiude il file

 return 0;
}
```

## 8 Errori nelle operazioni sui file

Nei programmi che utilizzano i file è opportuno aggiungere un controllo di eventuali errori o anomalie che si possono verificare in fase di apertura, per esempio *file non esistente* oppure *errore in apertura* per un accesso non corretto alla memoria di massa.

### • Prima modalità

Una prima modalità di test è costituita dalla struttura:

```
if (!nomestream) {
 cout << "Errore nell'apertura del file" << endl;
}
```

Il programma di lettura dell'archivio utilizzato nel paragrafo precedente può essere riscritto in questo modo:

```
// LeggiAmici2.cpp: Lettura archivio
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
 string nome;
 ifstream fin("amici.dat"); // apre il file in lettura
 if (!fin) {
 cout << "Errore nell'apertura del file" << endl;
 }
 else {
 while (fin >> nome) { // legge dal file
 cout << nome << endl;
 }
 fin.close(); // chiude il file
 }

 return 0;
}
```

### • Seconda modalità

Con la seconda modalità, il controllo sull'apertura del file può essere ottenuto con un test del valore restituito dal metodo **fail()**: esso restituisce il valore *true* quando un'operazione sul file produce un errore, oppure quando viene richiesta un'operazione non consentita per il tipo di file.

```
if (nomestream.fail()) {
 cout << "Errore nell'apertura del file" << endl;
}
```

Il programma di lettura può essere riscritto nel modo seguente:

```
// LeggiAmici3.cpp: Lettura archivio
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
```

```
int main()
{
 string nome;
 ifstream fin("amici.dat"); // apre il file in lettura
 if (fin.fail()) {
 cout << "Errore nell'apertura del file" << endl;
 }
 else {
 while (fin >> nome) { // legge dal file
 cout << nome << endl;
 }
 fin.close(); // chiude il file
 }

 return 0;
}
```

Altri tre metodi per gli *stream* forniscono informazioni sull'utilizzo del file durante l'esecuzione:

- **good()** indica se l'operazione sullo *stream* è andata a buon fine
- **bad()** indica se l'operazione sullo *stream* ha prodotto un errore irrecuperabile
- **is\_open()** controlla che lo *stream* sia aperto.



## GESTIONE STRUTTURATA DELLE ECCEZIONI

Abbiamo già visto nel Capitolo 7 l'uso della struttura **try ... catch** per la gestione delle **eccezioni** in un programma C++. La gestione può essere ulteriormente personalizzata dal programmatore rispetto alle specifiche esigenze di controllo sulle situazioni che possono provocare interruzioni indesiderate del programma.

All'interno di una funzione è possibile richiamare il gestore delle eccezioni attraverso l'istruzione **throw** (imperativo del verbo *lanciare*) con la quale il controllo dell'esecuzione ritorna alla funzione chiamante e salta alle istruzioni scritte nella zona *catch* della struttura di gestione dell'eccezione.

L'istruzione *throw* è quindi un'uscita dalla funzione, alternativa all'uscita regolare tramite l'istruzione *return*.

Il progetto seguente mostra l'utilizzo del gestore delle eccezioni con la struttura *try ... catch* e dell'istruzione *throw*.

## PROGETTO 4

### Calcolare il valore medio dei numeri registrati in un file di testo.

Il programma contiene una funzione per la lettura del file *numeri.txt* e il calcolo della media (somma dei valori diviso il numero dei valori letti). Si devono controllare le situazioni anomale che si possono verificare nel caso si riscontri un errore nell'apertura del file (per esempio, il file non esiste), oppure nel caso in cui il file esista, ma sia vuoto. In entrambi i casi accade un'eccezione perché il divisore nel calcolo della media risulterebbe uguale a 0 provocando un errore di *run-time*.

Nella situazione regolare la funzione riceve dal *main*, come parametro, il nome del file che contiene i numeri e, leggendo il file, somma i numeri e li conta. Alla fine calcola la media e la restituisce alla funzione chiamante.

Il tipo di errore è rappresentato tramite una struttura, che contiene due campi: un codice errore numerico e una descrizione; quest'ultima è il messaggio che viene visualizzato per l'utente quando si verifica l'eccezione.

```
struct TipoErrore {
 int num;
 string msg;
};
```

Quando si verifica l'eccezione, viene lanciata la gestione con l'istruzione *throw* che fa uscire dalla funzione restituendo il tipo di errore riscontrato. Il controllo passa alla sezione *catch* del gestore delle eccezioni che, a sua volta, esegue le istruzioni previste per quel tipo di errore (normalmente, messaggi di avvertimento per l'utente).

```
try {
 istruzioni;
}
catch(TipoErrore e) {
 cerr << "Errore n. " << e.num
 << " (" << e.msg << ")" << endl;
}
catch(...) {
 cerr << "Errore indefinito" << endl;
}
```

Si noti la presenza di una seconda sezione *catch*, avente come argomento tre puntini, . . ., per indicare una situazione di errore generico, non previsto in modo specifico dal programmatore.

Il programma completo è riportato di seguito.

### Programma C++

```
// Eccezioni.cpp: uso di try...catch e di throw
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

 struct TipoErrore {
 int num;
 string msg;
 };

// prototipo della funzione
double MediaDaFile(string nomefile);

// funzione principale
int main()
{
 try {
 double media;
 media = MediaDaFile("numeri.txt");
 cout << "Operazione andata a buon fine: "
 << "Media dei numeri = "
 << media << endl;
 }
}
```



```
 catch (TipoErrore e) {
 cerr << "Errore n. " << e.num
 << " (" << e.msg << ")" << endl;
 }
 catch (...) {
 cerr << "Errore indefinito" << endl;
 } // fine try

 cout << "Fine esecuzione" << endl;

 return 0;
}

double MediaDaFile(string nomefile)
{
 TipoErrore errore;
 ifstream fin(nomefile.c_str());
 if (!fin) {
 errore.num = 1;
 errore.msg = "Errore nell'apertura del file: " + nomefile;
 throw errore;
 }

 int conta = 0;
 double dato, somma=0.0;
 while (fin >> dato) {
 somma+= dato;
 conta++;
 }

 if (!conta) {
 errore.num = 2;
 errore.msg = "File " + nomefile + " vuoto";
 throw errore;
 }

 fin.close();

 double m = somma / conta;

 return m;
}
```

I possibili output del programma sono:

- quando il file non esiste:  
Errore n. 1 (Errore nell'apertura del file: numeri.txt)  
Fine esecuzione
- quando il file esiste, ma è vuoto (la condizione *!conta* significa che *conta* non è diverso da zero):  
Errore n. 2 (File numeri.txt vuoto)  
Fine esecuzione
- quando il file esiste e contiene i numeri:  
Operazione andata a buon fine: Media dei numeri = *valore calcolato*.  
Fine esecuzione

Si osservi nel programma precedente l'uso del metodo **c\_str()**, nella funzione *MediaDaFile*, per convertire il tipo *string* in un array di caratteri per il nome del file associato al flusso *fin*. Nella gestione delle eccezioni per le operazioni di *Input/Output* sui file, è possibile utilizzare la funzione membro **exceptions()** della classe *ios*. Essa restituisce o imposta i bit della **maschera sullo stato del file** (*flag di stato*). Quando il valore di uno di questi bit viene impostato, il programma lancia un'eccezione standard.

I possibili valori simbolici della maschera sono:

- **badbit**, errore irrecuperabile sullo stream
- **eofbit**, fine dello stream raggiunto durante l'operazione di input
- **failbit**, operazione non consentita sullo stream
- **goodbit**, situazione regolare con assenza di errore.

Il valore di *default* della maschera è *goodbit*.

Il seguente programma mostra l'utilizzo di questo gestore delle eccezioni durante le operazioni di lettura di un file di testo.

L'istruzione **catch** cattura l'eccezione con una variabile *e* di tipo **ios::failure** che è un oggetto appartenente alla classe **exception** della libreria standard del linguaggio C++ e riferito alle operazioni sugli stream. Il tipo di eccezione è il valore restituito dal metodo **what()** applicato all'oggetto di tipo *ios::failure*.

```
// Eccezioni2.cpp: uso di exceptions
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main ()
{
 string dato;
 ifstream archivio;
 // eccezioni gestite dal programma
 archivio.exceptions (ifstream::eofbit | ifstream::failbit | ifstream::badbit);
 try {
 archivio.open ("righe.txt");
 while (archivio >> dato)
 cout << dato << endl;
 archivio.close();
 }
 catch (ios::failure e) {
 cout << "Tipo di eccezione: " << e.what() << endl;
 }

 return 0;
}
```

## 9 Controllo della fine del file

Come si è già visto in precedenza, nel programma di lettura sequenziale di un file, occorre inserire un controllo sul raggiungimento della fine del file. Il modo più semplice è stato presentato negli esempi precedenti con l'uso di *while (fin >> variabile)*. In alternativa si può utilizzare il metodo **eof()**, *end of file*, che assume il valore *true* quando viene raggiunta la fine del file.

La ripetizione per la lettura sequenziale del file viene realizzata con la seguente struttura *while*:

```
nomestream >> variabile;
while (!nomestream.eof()) {
 cout << variabile << endl;
 nomestream >> variabile;
}
```

Si notino le due operazioni di lettura dal file, una esterna prima di *while* e una interna alla ripetizione.

Di seguito viene presentata un'ulteriore versione del programma di lettura sequenziale del file, che utilizza il metodo *eof()* per il controllo di fine file.

```
// LeggiAmici4.cpp: Lettura archivio
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
 ifstream fin("amici.dat"); // apre il file in lettura
 string nome;
 fin >> nome; // legge dal file
 while (!fin.eof()) {
 cout << nome << endl;
 fin >> nome;
 }
 fin.close(); // chiude il file

 return 0;
}
```

## 10 Stream su stampante

La **stampante**, dal punto di vista del sistema operativo, è a tutti gli effetti un file di testo sul quale si possono fare solo operazioni di scrittura.

Quindi l'utilizzo della stampante in un programma per produrre l'output sulla carta richiede due operazioni:

- la dichiarazione di uno stream di tipo *ofstream*:

```
ofstream stampante;
```

- l'apertura dello *stream* con il metodo **open()**, indicando come argomento il nome fisico della stampante, per esempio **LPT1** per la stampante parallela nel sistema operativo Windows:

```
stampante.open("LPT1");
```

Disponendo di una stampante USB, occorre ridirigere il nome LPT1 sulla stampante USB, dall'amministrazione del sistema operativo.

## PROGETTO 5

### Stampare un elenco di nomi contenuto in archivio.

Il programma utilizza due stream: uno stream in input per l'archivio dei dati e uno stream in output per la stampante. Una ripetizione legge tutti i nomi in modo sequenziale e stampa su carta i dati.

#### Programma C++

```
// Stampa.cpp: stampa archivio su carta
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
 ifstream fin; // stream per il file
 ofstream stampante; // stream per la stampante
 fin.open("amici.dat"); // apre il file in lettura
 stampante.open("LPT1"); // apre lo stream della stampante
 string nome;
 while (fin >> nome) { // Legge dal file
 stampante << nome << endl;
 }
 fin.close(); // chiude il file

 return 0;
}
```

## 11 Uso degli stream in un'applicazione gestionale

A questo punto siamo in grado di costruire una semplice applicazione software per la gestione di una rubrica con nomi e numeri di telefono, inserendo in un unico programma le funzionalità tipiche di una procedura gestionale:

- registrazione dati
- aggiunta di nuovi dati
- lettura sequenziale dell'archivio e visualizzazione dei dati
- stampa su carta
- ricerca di un dato nell'archivio.

## PROGETTO 6

### Gestione di una rubrica telefonica.

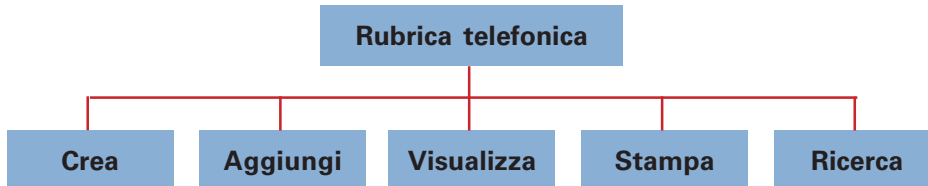
I dati delle persone, nome e numero telefonico, sono registrati in un file di testo. Il numero telefonico è di tipo stringa, perché viene considerato come una sequenza di caratteri rappresentati dalle cifre.

In fase di registrazione il nome e il numero telefonico di una persona sono registrati sul file uno di seguito all'altro, separati da uno spazio bianco e con un carattere di fine linea *endl*, utilizzando l'operatore << in un output ripetuto, come abbiamo fatto nei progetti dei capitoli precedenti per la visualizzazione dei dati sul video:

```
archivio << nome << " " << telefono << endl;
```

Durante la lettura vengono poi acquisite in successione due stringhe, assegnando alla prima il significato di nome e alla seconda quello di numero telefonico. In sostanza per il programma vengono letti e scritti dati come sequenze di byte: è compito del programmatore interpretare i dati rendendoli disponibili all'utente esterno secondo il significato che hanno nel problema.

Le scelte del programma sono elencate in un menu che organizza le funzioni di gestione secondo lo schema seguente:



Le funzioni *Crea()*, *Aggiungi()*, *Visualizza()*, *Stampa()* hanno la stessa struttura dei programmi presentati nei paragrafi precedenti per inserire, aggiungere, visualizzare e stampare i dati. Nella funzione *Ricerca()* l'utente fornisce da tastiera il nome di una persona. Il programma legge sequenzialmente l'archivio della rubrica e cerca il nome. Se lo trova, visualizza il numero telefonico, altrimenti scrive un messaggio per l'utente segnalando che il nome non è presente nella rubrica.

Si noti che l'apertura e la chiusura del file deve essere fatta all'interno di ciascuna funzione, perché esse operano sullo stesso file, ma con modalità diverse per il metodo *open()*: **ios::out** per la prima, **ios::app** per la seconda e **ios::in** per le altre tre.

Occorre anche osservare che in questo progetto lo *stream* è dichiarato con un oggetto di tipo *fstream*, quindi si deve sempre specificare la modalità di apertura.

In particolare, per l'operazione di aggiunta occorre usare entrambe le costanti:

```
ios::out | ios::app
```

separate dal segno | (operatore Or).

Il nome fisico del file è dichiarato come costante all'inizio del programma, per permetterne una facile modifica nel caso il programma debba essere utilizzato con un diverso archivio di dati.

Si osservi l'uso del metodo **c\_str()** nelle operazioni di *open* sull'archivio: esso converte il tipo *string* in un array di caratteri per il nome fisico del file associato al flusso.

### Programma C++

```
// Rubrica.cpp: gestione di una rubrica telefonica
// inserimento, aggiunta, visualizzazione,
// stampa e ricerca
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

const string NOMEARCH = "rubrica.dat";
fstream archivio; // dichiara stream
string nome, telefono;
```

```
// prototipi delle funzioni
int Menu();
void Crea();
void Aggiungi();
void Visualizza();
void Stampa();
void Ricerca();

// funzione principale
int main ()
{
 int scelta;

 do {
 scelta = Menu();
 switch (scelta) {
 case 1:
 Crea();
 break;
 case 2:
 Aggiungi();
 break;
 case 3:
 Visualizza();
 break;
 case 4:
 Stampa();
 break;
 case 5:
 Ricerca();
 break;
 case 6:
 cout << "fine lavoro" << endl;
 break;
 }
 } while (scelta!=6);

 return 0;
}

// menu delle scelte
int Menu()
{
 int s;

 cout << "-----" << endl;
 cout << "Gestione rubrica telefonica" << endl;
 cout << "1. Creazione rubrica" << endl;
 cout << "2. Aggiunta nuovi nomi" << endl;
 cout << "3. Visualizzazione rubrica" << endl;
 cout << "4. Stampa rubrica" << endl;
 cout << "5. Ricerca di nomi" << endl;
 cout << "6. Fine lavoro" << endl;
 // scelta dell'utente
 cout << "Scegli: ";
 cin >> s;
 cout << "-----" << endl;

 return s;
} // Menu
```

```
// creazione archivio
void Crea()
{
 archivio.open(NOMEARCH.c_str(), ios::out); // apre il file per la scrittura
 cout << "Inserisci un nome (* = fine): ";
 cin >> nome;
 while (nome != "") {
 cout << "Telefono: ";
 cin >> telefono;
 archivio << nome << " " << telefono << endl; // scrive sul file
 cout << "Inserisci un altro nome (* = fine): ";
 cin >> nome;
 }
 archivio.close(); // chiude il file
} // Crea

// aggiunta di dati all'archivio
void Aggiungi()
{
 archivio.open(NOMEARCH.c_str(), ios::out | ios::app); // apre il file in append
 if (!archivio) {
 cout << "Errore nell'apertura del file" << endl;
 }
 else {
 cout << "Nome: ";
 cin >> nome;
 cout << "Telefono: ";
 cin >> telefono;
 archivio << nome << " " << telefono << endl; // scrive sul file
 archivio.close(); // chiude il file
 }
} // Aggiungi

// Lettura dei dati dall'archivio
void Visualizza()
{
 archivio.open(NOMEARCH.c_str(), ios::in); // apre il file in lettura
 if (!archivio) {
 cout << "Errore nell'apertura del file" << endl;
 }
 else {
 archivio >> nome; // Legge dal file
 while (!archivio.eof()) {
 archivio >> telefono;
 cout << nome << "; " << telefono << endl;
 archivio >> nome;
 }
 archivio.close(); // chiude il file
 }
} // Visualizza

void Stampa()
{
 ofstream stampante; // dichiara uno stream
```

```
stampante.open("LPT1"); // apre lo stream della stampante
archivio.open(NOMEARCH.c_str(), ios::in); // apre il file in lettura
if (!archivio) {
 cout << "Errore nell'apertura del file" << endl;
}
else {
 cout << "Stampa in corso" << endl;
 archivio >> nome; // legge dal file
 while (!archivio.eof()) {
 archivio >> telefono;
 stampante << nome << "; " << telefono << endl;
 archivio >> nome;
 }
 archivio.close(); // chiude il file
 cout << "Stampa terminata" << endl;
}
stampante.close(); // chiude lo stream della stampante
} // Stampa

void Ricerca()
{
 string cercato;
 bool trovato = false;

 cout << "Inserisci il nome da cercare: ";
 cin >> cercato;
 archivio.open(NOMEARCH.c_str(), ios::in); // apre il file in lettura
 if (!archivio) {
 cout << "Errore nell'apertura del file" << endl;
 }
 else {
 archivio >> nome; // legge dal file
 while (!archivio.eof() && !trovato) {
 archivio >> telefono;
 if (nome == cercato) {
 cout << "Telefono: " << telefono << endl;
 trovato = true;
 }
 archivio >> nome;
 }
 archivio.close(); // chiude il file
 if (!trovato)
 cout << "nome inesistente" << endl;
 }
} // Ricerca
```

#### AUTOVERIFICA

Domande da 4 a 8 pag. 467  
Problemi da 3 a 14 pag. 468-469



**MATERIALE ONLINE**

### 3. Programmi eseguibili e videate dei progetti sui file di testo



## 12 Accesso sequenziale per gli archivi con record a lunghezza costante

Nelle applicazioni gestionali l'uso di archivi su memoria di massa riguarda comunemente file formati da record aventi tutti la stessa struttura e quindi la stessa lunghezza: si pensi agli archivi anagrafici di clienti e fornitori di un'azienda oppure agli archivi di docenti e studenti in una scuola, agli archivi dei movimenti di contabilità oppure dei movimenti di magazzino, ecc.

In tutti questi casi il programma di gestione degli archivi contiene la dichiarazione delle strutture di dati come la seguente, riferita ad un archivio di dipendenti di un'azienda:

```
struct Persona {
 int ID; // matricola
 char nome[50]; // cognome e nome
 double stipendio; // stipendio
};
```

La dichiarazione utilizza il tipo di dato *struct*, presentato nel Capitolo 5.

Per la gestione di file con record di lunghezza costante, come già detto, è opportuno utilizzare file di tipo binario (**binary file**), specificando nell'istruzione di apertura la costante **ios::binary**, dopo le costanti che specificano la modalità di apertura:

```
fstream nomestream;
nomestream.open(nomefile, ios::in | ios::out | ios::binary);
```

per l'apertura in lettura e scrittura dall'inizio, oppure

```
nomestream.open(nomefile, ios::in | ios::app | ios::binary);
```

per l'apertura in lettura e scrittura in coda al file.

Il tipo **fstream** per l'archivio può essere sostituito con **ifstream** per gli archivi sottoposti ad operazioni di lettura e con **ofstream** per gli archivi sui quali si devono fare operazioni di scrittura.

L'accesso a un file binario può essere **sequenziale**, quando si scrivono i record di seguito a quelli già registrati e si leggono a partire dal primo, oppure **diretto** (o **random**), quando il programma è in grado di posizionarsi su uno specifico record, di cui si fornisce il numero, per operazioni di lettura o scrittura.

L'**operazione di scrittura** su un file binario viene realizzata dal metodo **write()**, specificando la variabile di tipo *struct* e la sua lunghezza; per esempio:

```
nomestream.write(variable, sizeof(variable));
```

L'operatore **sizeof** fornisce la lunghezza della variabile in byte.

L'**operazione di lettura** su un file binario viene realizzata dal metodo **read()**, specificando la variabile di tipo *struct* e la sua lunghezza; per esempio:

```
nomestream.read(variable, sizeof(variable));
```

## PROGETTO 7

### Gestione dell'archivio anagrafico dei dipendenti di un'azienda.

L'applicazione riguarda la gestione delle anagrafiche dei dipendenti di un'azienda. Ogni dipendente è identificato attraverso un numero di matricola che coincide con il numero del record del dipendente nell'archivio (campo *ID* del record *Persona*).

L'applicazione deve comprendere le procedure per:

- l'inserimento di un nuovo record
- la visualizzazione di tutti i record registrati.

L'archivio può successivamente essere usato per altre esigenze gestionali, per esempio:

- la ricerca di un record corrispondente al numero di record fornito da tastiera
- la modifica di alcuni campi di uno specifico record (per esempio, la variazione dello stipendio di un dipendente del quale viene fornita la matricola).

Prima di iniziare le operazioni sul file è opportuno **creare il file** con un semplice programma che apre il file e lo chiude subito dopo. In questo modo viene creato lo spazio su disco per il file e le successive operazioni di scrittura sono considerate comunque registrazioni in *append*.

### PROGRAMMA DI CREAZIONE DELL'ARCHIVIO

#### Programma C++

```
// CreaDipendenti.cpp: creazione archivio dipendenti

#include <iostream>
#include <fstream>
#include <string>
using namespace std;

const string NOMEARCH = "anagrafe.dat";

int main()
{
 ofstream archivio;
 archivio.open(NOMEARCH.c_str(), ios::out | ios::binary);
 archivio.close();
 cout << "Nuovo archivio dipendenti creato" << endl;

 return 0;
}
```

Il seguente programma mostra un esempio di **accesso sequenziale al file binario** per aggiungere nuove anagrafiche all'archivio dei dipendenti. Alla fine visualizza anche l'elenco completo dei dipendenti registrati, attraverso una lettura sequenziale del file binario.

### PROGRAMMA PER L'INSERIMENTO DI NUOVI DIPENDENTI E LA VISUALIZZAZIONE DEI DIPENDENTI REGISTRATI

#### Programma C++

```
// InserisciDipendenti.cpp: inserimento dipendenti

#include <iostream>
#include <fstream>
using namespace std;
```

```
struct Persona {
 int ID; // matricola
 char nome[50]; // cognome e nome
 double stipendio; // stipendio
};
Persona dipendente;

int main()
{
 ofstream fout;

 // aggiunge altri dati
 fout.open("anagrafe.dat", ios::app | ios::binary);
 if (!fout) {
 cout << "Errore nell'apertura dell'archivio" << endl;
 }
 else {
 cout << "Matricola del dipendente (0 = fine): ";
 cin >> dipendente.ID;
 while (dipendente.ID != 0) {
 cin.ignore(80, '\n');
 cout << "Cognome e nome del dipendente: ";
 cin.getline(dipendente.nome, 50);
 cout << "Stipendio (euro): ";
 cin >> dipendente.stipendio;
 fout.write((char *) &dipendente, sizeof(dipendente));
 cout << "Inserisci un'altra matricola (0 = fine): ";
 cin >> dipendente.ID;
 }
 fout.close();
 }

 // visualizza i dipendenti registrati
 ifstream fin;
 fin.open("anagrafe.dat", ios::in | ios::binary);
 if (!fin) {
 cout << "Errore nell'apertura dell'archivio" << endl;
 }
 else {
 cout << "Elenco dipendenti registrati:" << endl;
 while (fin.read((char *) &dipendente, sizeof(dipendente))) {
 cout << dipendente.ID << ":"
 << dipendente.nome << '\t'
 << dipendente.stipendio << "euro"
 << endl;
 }
 fin.close();
 }

 return 0;
}
```

Il buffer che viene scritto sulla memoria di massa è una sequenza di *char*, pertanto nella dichiarazione della struttura si utilizza un **array di char** per il cognome e nome; l'uso del tipo *string* richiederebbe la conversione tramite la funzione membro **c\_str()** della classe *string*, come visto nel Paragrafo 11 del Capitolo 7.

Inoltre in fase di scrittura e lettura di un record viene realizzata un'operazione di *casting* sul record del dipendente, perché la funzione deve ricevere un puntatore a carattere.

```
fout.write((char*) &dipendente, sizeof(dipendente));
```

L'input da tastiera del cognome e nome del dipendente è realizzato con il metodo **getline()** applicato al flusso *cin*, perché la stringa può contenere anche spazi bianchi; si deve usare anche il metodo **ignore()**, secondo quanto spiegato nel precedente Paragrafo 2.

#### AUTOVERIFICA

Problemi da 15 a 19 pag. 469



#### MATERIALI ONLINE

### 4. Accesso diretto per gli archivi con record a lunghezza costante 5. La stringa come stream

## 13 Iteratori per gli stream

Nella libreria dei template standard (STL), che abbiamo visto nel Capitolo 7, sono disponibili anche gli **iteratori** per accedere agli stream (**stream iterator**) con operazioni di lettura e scrittura.

Gli iteratori sono:

- **istream\_iterator**, per l'input da stream
- **ostream\_iterator**, per l'output su stream.

Per utilizzare questi iteratori occorre inserire nel programma la direttiva di inclusione:

```
#include <iterator>
```

La sintassi generale della dichiarazione di un **ostream\_iterator** è:

```
ostream_iterator<T> nomeiteratore(nomestream, carattere);
```

dove *T* indica il tipo degli oggetti contenuti nello *stream*. Il secondo argomento *carattere* indica il delimitatore dei valori scritti nello stream.

Per esempio, le seguenti istruzioni dichiarano un iteratore *oo* per lo stream di output *fout* che contiene stringhe ed è associato al file *prova.dat* che si trova su disco, usando come delimitatore il carattere *new line* (`'\n'`):

```
ofstream fout("prova.dat");
ostream_iterator<string> oo(fout, "\n");
```

## PROGETTO 8

### Acquisire da tastiera un elenco di nomi di colori e scriverli in un file di testo su disco.

Il nome fisico del file su disco è *tavolozza.txt*, il nome interno dello stream è *fout* e il nome dell'iteratore è *oo*. Il valore acquisito da tastiera viene assegnato allo stream con l'operatore *\**. Per accodare un nuovo valore sullo stream l'iteratore dello stream viene poi incrementato.

#### Programma C++

```
// Colori.cpp: uso di stream iterator (scrittura)
#include <iostream>
#include <fstream>
#include <string>
#include <iterator>
using namespace std;

int main()
{
 string nc;

 // dichiarazione dello stream
 ofstream fout("tavolozza.txt");
 // iteratore per lo stream di output
 ostream_iterator<string> oo(fout, "\n") ;

 cout << "Nome del colore (* = fine): ";
 cin >> nc;
 while (nc != "") {
 *oo = nc;
 oo++;
 cout << "Altro colore (* = fine): ";
 cin >> nc;
 }

 return 0;
}
```

La sintassi generale della dichiarazione di un **istream\_iterator** è:

```
istream_iterator<T> nomeiteratore(nomestream);
```

dove *T* indica il tipo di oggetti contenuti nello stream.

Per esempio, le seguenti istruzioni dichiarano un iteratore *ii* per lo stream di input *fin* formato da stringhe e associato al file *prova.dat* che si trova su disco:

```
ifstream fin("prova.dat");
istream_iterator<string> ii(fin) ;
```

Nelle operazioni di input su stream, quando viene raggiunta la fine dello stream, l'iteratore di input assume uno speciale valore di *fine stream*, denominato **eos** (*end of stream*), che è l'iteratore alla posizione successiva alla fine del file. Questo particolare iteratore predefinito è dichiarato con la seguente istruzione:

```
istream_iterator<T> eos;
```

## PROGETTO 9

**Leggere dal file su disco, creato con il progetto precedente, i nomi dei colori e visualizzarli sul video.**

Il programma scorre tutto il file dall'inizio alla fine con una ripetizione che incrementa l'iteratore di input e che termina quando l'iteratore raggiunge il valore *eos* di fine stream:

```
while (ii != eos) { ... }
```

### Programma C++

```
// LeggiColori.cpp: uso di stream iterator (lettura)
#include <iostream>
#include <fstream>
#include <string>
#include <iterator>
using namespace std;

int main()
{
 // dichiarazione dello stream
 ifstream fin("tavolozza.txt");
 // iteratore per lo stream di input
 istream_iterator<string> ii(fin);
 // iteratore per il controllo di fine file
 istream_iterator<string> eos;

 while (ii != eos) {
 cout << *ii << endl;
 ++ii;
 }

 return 0;
}
```

Il progetto seguente mostra l'utilizzo di entrambi gli iteratori sugli stream, di input e di output, nello stesso programma.

## PROGETTO 10

**Caricare in un vettore i nomi dei colori leggendoli dal file sul disco. Dopo aver messo in ordine alfabetico i nomi dei colori, registrare l'elenco ordinato su un nuovo file.**

I nomi dei colori sono contenuti nel file utilizzato nei progetti precedenti, *tavolozza.txt*. L'elenco ordinato viene registrato su un nuovo file, *ordinati.txt*.

Gli stream per questi due file sono *fin* e *fout*. Gli iteratori di input e di output si chiamano rispettivamente *ii* e *oo*.

Per memorizzare i dati letti dal file, il programma utilizza un contenitore standard **vector**, visto nel Capitolo 7.

Con questo contenitore l'operazione di caricamento è rappresentata in forma molto compatta con un'unica istruzione:

```
vector<string> c(ii, eos);
```

Essa dichiara un contenitore *vector* di oggetti *string* di nome *c* e carica in esso tutti i dati contenuti nello stream di input scorrendo tutte le sue componenti con l'iteratore di input *ii*, finché l'iteratore raggiunge la fine dello stream (valore *eos* per l'iteratore).

I dati, ora disponibili nel vettore, possono essere ordinati con l'algoritmo standard **sort()**, già presentato nel capitolo precedente. Gli elementi del vettore sono esaminati dal primo all'ultimo, utilizzando i metodi *begin()* ed *end()* che restituiscono gli iteratori standard del contenitore *vector*, per l'inizio e la fine del vettore.

```
sort(c.begin(), c.end());
```

Infine, la scrittura sullo stream dei dati ordinati è realizzata con un altro algoritmo standard, **copy()**, già visto nel capitolo precedente, indicando come argomenti i due iteratori sul vettore e il nome dello stream:

```
copy(c.begin(), c.end(), oo);
```

Per utilizzare il contenitore e i due algoritmi standard occorre inserire nel programma le seguenti dichiarazioni di inclusione:

```
#include <vector>
#include <algorithm>
```

### Programma C++

```
// Colori2.cpp: Legge i colori, ordina e scrive su file ordinato
// usa iteratori per gli stream, vector e algoritmi standard
#include <iostream>
#include <fstream>
#include <string>
#include <iterator>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
 // dichiarazione degli stream
 ifstream fin("tavolozza.txt"); // input
 ofstream fout("ordinati.txt"); // output

 // iteratori per gli stream
 istream_iterator<string> ii(fin); // input
 ostream_iterator<string> oo(fout, "\n"); // output

 // iteratore per il controllo di fine file
 istream_iterator<string> eos;

 // vettore per i colori
 vector<string> c(ii, eos); // legge il file
 sort(c.begin(), c.end()); // ordina
 copy(c.begin(), c.end(), oo); // copia su stream di output

 return 0;
}
```

Un ultimo progetto mostra l'utilizzo dell'iteratore sullo stream e del contenitore *map*.

## PROGETTO 11

**In un file sul disco sono registrati i voti assegnati in una prova scolastica (da 0 a 10). Leggendo il file si vuole contare il numero di prove per ciascun voto assegnato.**

I voti assegnati nella prova sono registrati nel file *esiti.dat* (un voto per ogni riga). Lo stream per questo file si chiama *fin* e l'iteratore di input è *ii*.

Per memorizzare i dati letti dal file, il programma utilizza un contenitore standard **map**, visto nel Capitolo 7. In esso la *chiave* è rappresentata dal voto, mentre il valore associato è il numero delle prove alle quali è stato assegnato quel voto.

```
map<int, int> voti;
```

Per la lettura dallo stream e il conteggio delle prove, il programma utilizza l'algoritmo standard **for\_each**, visto nel capitolo precedente, con il quale la funzione *Conta* viene applicata a ciascun elemento, di tipo *int*, letto dallo stream:

```
for_each(ii, eos, Conta);
```

La funzione *Conta* incrementa di 1 il valore dell'elemento della mappa corrispondente alla chiave rappresentata dal voto acquisito come parametro:

```
void Conta(const int v)
{
 voti[v]++;
}
```

Da ultimo, i valori memorizzati nel contenitore *map* vengono visualizzati con una ripetizione che utilizza un iteratore sulla mappa, dall'inizio alla fine. Si ricordi che il primo elemento della mappa (voto) è indicato con *first* e il secondo elemento (numero di prove) con *second*. I risultati ottenuti sono naturalmente ordinati per voto crescente, perché la mappa viene mantenuta ordinata secondo la chiave.

Per utilizzare il contenitore *map* e l'algoritmo standard *for\_each* occorre inserire nel programma le seguenti dichiarazioni di inclusione:

```
#include <map>
#include <algorithm>
```

Il programma completo che risolve il problema è riportato di seguito.

### Programma C++

```
// Voti.cpp: conta i voti raggruppati
#include <iostream>
#include <fstream>
#include <iterator>
#include <map>
#include <algorithm>
using namespace std;

map<int, int> voti; // contenitore per il conteggio
```



```
// prototipo della funzione
void Conta(const int v);

// funzione principale
int main()
{
 // dichiarazione dello stream
 ifstream fin("esiti.dat");
 // iteratore per lo stream
 istream_iterator<int> ii(fin);
 // iteratore per il controllo di fine file
 istream_iterator<int> eos;

 // legge il file
 for_each(ii, eos, Conta);

 // risultati
 map<int, int>::iterator i;
 for (i=voti.begin(); i!=voti.end(); i++)
 cout << "Voto " << i->first << ": " << i->second << endl;

 return 0;
}

// funzione per il conteggio
void Conta(const int v)
{
 voti[v]++;
} // Conta
```

#### **AUTOVERIFICA**

Domande da 9 a 10 pag. 468  
Problemi da 20 a 23 pag. 469



#### **MATERIALI ONLINE**

- 6. Programmi eseguibili e videate dei progetti precedenti**
- 7. Lezioni multimediali sulla gestione dei file in C++**

## DOMANDE

### Input/output

- 1 Quale tra i seguenti termini indica la radice della gerarchia delle classi di input/output?
  - a) stream
  - b) ios
  - c) buffer
  - d) istream
  
- 2 Completa la seguente tabella scrivendo accanto ad ogni frase il nome dello stream predefinito che svolge la funzione indicata

| Funzione                                                                                       | Stream predefinito |
|------------------------------------------------------------------------------------------------|--------------------|
| flusso associato all'unità standard di input                                                   |                    |
| flusso associato all'unità standard di output                                                  |                    |
| flusso associato all'unità standard per la visualizzazione dei messaggi di errore              |                    |
| flusso per la registrazione delle operazioni svolte dal sistema durante una sessione di lavoro |                    |

- 3 Quale delle seguenti implementazioni di una classe usa in modo corretto l'overloading dell'operatore <<?

a) 

```
class dato {
...
public:
// metodi
...
ostream &operator<< (ostream &os, dato &a);
};
```

b) 

```
class dato {
...
public:
// metodi
...
friend ostream &operator<< (ostream &os, dato &a);
};
```

c) 

```
class dato {
...
public:
// metodi
...
ostream operator<< (ostream &os, dato &a);
};
```

d) 

```
class dato {
...
public:
// metodi
...
dato operator<< (ostream &os, dato &a);
};
```

## File di testo

- 4 Quale tra le seguenti frasi rappresenta la definizione di *buffer* di I/O?
- Una parte di un file riservata per le operazioni di I/O.
  - Una parte di un record riservata per le operazioni di I/O.
  - Una parte della memoria centrale riservata per le operazioni di I/O.
  - Una parte del disco riservata per le operazioni di I/O.
- 5 Date le funzioni e gli operatori di gestione dei file, indica quali sono specifiche dei file di testo e quali dei file binari, mettendo un segno di spunta nella colonna adatta.

|         | File di testo | File binario |
|---------|---------------|--------------|
| open()  |               |              |
| <<      |               |              |
| write() |               |              |
| >>      |               |              |
| read()  |               |              |
| close() |               |              |

- 6 Quali di queste affermazioni sono vere (V) e quali false (F)?
- Le operazioni di scrittura trasferiscono i dati dalla memoria di massa alla memoria centrale V F
  - In un programma di scrittura occorre usare un oggetto della classe *ifstream* V F
  - La modalità *append* permette di eseguire operazioni di lettura e di aggiunta di dati in un file V F
  - L'operatore << permette di eseguire operazioni di scrittura su file di testo V F
- 7 Quale delle seguenti strutture di ripetizione si può utilizzare per leggere sequenzialmente il contenuto di uno stream denominato *archivio*?
- while (archivio.eof()) {...}
  - while (!archivio.eof()) {...}
  - if (eof(archivio)) {...}
  - if (!eof(archivio)) {...}
- 8 Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
- L'istruzione *nomestream.open(nomefile, ios::out);* permette solo operazioni di scrittura V F
  - Le operazioni di lettura trasferiscono i dati dalla memoria di massa alla memoria centrale V F
  - L'istruzione *nomestream.open(nomefile, ios::in);* cancella il file preesistente V F
  - L'istruzione *nomestream.open(nomefile, ios::app);* permette di eseguire operazioni di lettura e di modifica su un file V F

## Archivi e iteratori

9 Data una struttura e uno stream in input come nella seguente definizione

```
struct rec {
 char campo1[20];
 char campo2[15];
};
struct rec dato;
ifstream fin;
```

qual è l'istruzione corretta per la lettura dal file?

- a) `fin.read((char*) &rec, sizeof(rec))`
- b) `fin.read((char*) &dato, sizeof(dato))`
- c) `fin.read(sizeof(rec), (char*) &rec)`
- d) `fin.read(sizeof(dato), (char*) &dato)`

10 Qual è la sintassi corretta per un iteratore di stream in input?

- a) `istream_iterator<T> nomestream(nomeiteratore);`
- b) `istream_iterator<nomestream> nomeiteratore(T);`
- c) `istream_iterator<T> nomeiteratore(nomestream);`
- d) `istream_iterator<nomeiteratore> nomestream(T);`

## PROBLEMI

### Input/output

- 1 Scrivere il programma per acquisire da tastiera un elenco di nominativi formati da cognome e nome utilizzando la funzione che consente di leggere anche stringhe contenenti spazi bianchi.
- 2 Scrivere il programma per acquisire da tastiera le stringhe con i dati anagrafici di una persona (Cognome, Nome, Città, Nazione) e per visualizzarle dopo l'inserimento.

### File di testo

- 3 Creare un file di testo per memorizzare i nomi delle squadre di calcio di serie A. Aggiungere poi con un secondo programma le squadre di serie B.
- 4 Visualizzare il contenuto del file creato nel problema precedente.
- 5 Scrivere un programma per registrare in un file di testo i nomi delle province di una regione italiana.
- 6 Leggere il contenuto del file creato nel problema precedente caricando i dati in un vettore di stringhe. Visualizzare poi l'elenco ordinato delle città.
- 7 Registrare in un file di testo un insieme di numeri. Leggendo poi il file si vuole visualizzare accanto a ciascun numero l'ammontare del 20% del valore.
- 8 Contare quante righe contiene un file di testo, dopo aver controllato che il file sia effettivamente presente nella memoria di massa.
- 9 Un file contiene l'elenco delle distanze per ciascuna tappa di una gara ciclistica. Si deve calcolare attraverso una funzione la lunghezza media delle tappe, controllando anche le eccezioni che si verificano quando il file non esiste o quando il file è vuoto.

- 10** Registrare in un file di testo un insieme di numeri. Leggendo poi il file dall'inizio fino alla fine si vuole contare quante volte compare un numero prefissato e fornito da tastiera dall'utente.
- 11** Un file di testo contiene i nomi degli studenti che sono nuovi iscritti in una scuola. Leggendo il file dei nomi, viene fornita da tastiera la classe assegnata a ciascuno studente, stampando via via sulla carta il nome di ciascuno studente con la classe.
- 12** Un file di testo contiene un elenco di nazioni. Leggendo sequenzialmente il file, il programma visualizza il nome della nazione e chiede all'utente il nome della capitale. I nomi delle nazioni e delle capitali vengono stampati su carta.
- 13** Registrare in un file di testo i nomi degli insegnanti di una scuola con le materie di insegnamento. Fornito poi da tastiera il nome di un insegnante, il programma visualizza la materia oppure un messaggio nel caso il nome richiesto non sia compreso nell'archivio.
- 14** Registrare in un file di testo le targhe di automobili e i rispettivi modelli. Il programma deve consentire di aggiungere nuove automobili, stampare su carta l'elenco completo dei dati dell'archivio e di ricercare il modello corrispondente alla targa fornita dall'utente.

### **Archivi con record a lunghezza costante**

- 15** Organizzare in un archivio ad accesso sequenziale le informazioni sui CD musicali preferiti. Scrivere poi la funzione che permetta di visualizzare i dati registrati.
- 16** Le informazioni relative ai partecipanti alle gare studentesche di atletica vengono registrate in un archivio ad accesso sequenziale. Scrivere la funzione che consenta di visualizzare il contenuto dell'archivio.
- 17** Un elenco di nomi di studenti con la lingua straniera studiata viene registrato in un file con un primo programma di scrittura. Fornendo poi da tastiera la lingua straniera richiesta, un secondo programma legge sequenzialmente il file degli studenti e visualizza solo i nomi degli studenti relativi a quella lingua straniera.
- 18** Dato l'archivio con i dati relativi ai libri di una biblioteca, costruire un programma che consenta di visualizzare i libri di un autore richiesto da tastiera.
- 19** Dato un archivio sequenziale contenente l'anagrafica dei dipendenti con cognome, nome e funzione, stampare l'elenco completo dei dipendenti.

### **Iteratori di stream**

- 20** Registrare in un file su disco i nomi delle regioni italiane, utilizzando un iteratore di output sullo stream.
- 21** Visualizzare l'elenco delle regioni contenuto nel file creato con il problema precedente, utilizzando un iteratore di input sullo stream.
- 22** Leggere i nomi contenuti nel file delle regioni e visualizzarne l'elenco in ordine alfabetico, utilizzando gli iteratori di input e output sullo stream, un vettore e gli algoritmi della libreria standard.
- 23** Un file su disco contiene un elenco di stringhe che rappresentano i codici dei clienti intestatari delle fatture emesse. Il programma, dopo aver letto l'archivio, visualizza il numero di fatture per ciascun codice cliente.



## SCHEDA DI AUTOVALUTAZIONE

### CONOSCENZE

- ☐ Classi di input/output
- ☐ Input e output sulle unità standard
- ☐ Overloading dell'operatore << e dell'operatore >>
- ☐ Archivi di dati
- ☐ Differenza tra file di testo e file binari
- ☐ Operazioni sugli archivi
- ☐ Tipi di accesso
- ☐ Lettura e scrittura in un file di testo
- ☐ Apertura esplicita dei file
- ☐ Controllo degli errori nelle operazioni sui file
- ☐ Gestione strutturata delle eccezioni
- ☐ Controllo della fine del file
- ☐ Stream su stampante
- ☐ Archivi con record a lunghezza costante
- ☐ Iteratori per gli stream

### ABILITÀ

- ☐ Classificare gli oggetti delle classi di input/output
- ☐ Applicare l'overloading degli operatori << e >>
- ☐ Individuare le applicazioni per l'uso di file di testo e di file binari
- ☐ Utilizzare le operazioni di apertura e chiusura
- ☐ Effettuare operazioni di lettura e scrittura su file di testo
- ☐ Controllare gli errori nelle operazioni sui file
- ☐ Gestire le eccezioni in modo strutturato
- ☐ Inserire il controllo della fine del file durante la lettura
- ☐ Produrre output sulla stampante
- ☐ Costruire un'applicazione gestionale completa per la gestione di archivi
- ☐ Effettuare l'accesso sequenziale agli archivi di record con lunghezza costante
- ☐ Utilizzare gli iteratori per gli stream
- ☐ Leggere dati da uno stream a un contenitore



**SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 663**

# Programmazione guidata dagli eventi e interfaccia utente

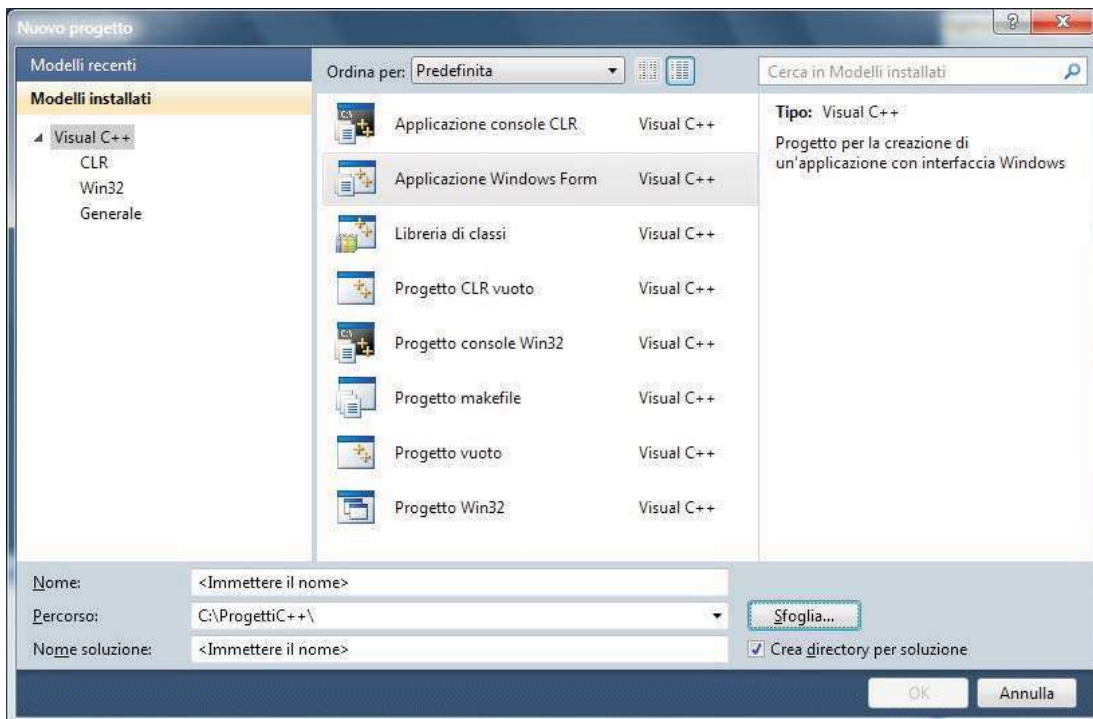
## 1 L'ambiente di programmazione

Con **programmazione visuale** (*visual programming*) si intende l'attività di sviluppo del software basato sull'uso di un'interfaccia grafica (finestre, pulsanti, icone). Il lavoro del programmatore è supportato da un ambiente di sviluppo grafico tale da mostrare immediatamente l'aspetto dell'applicazione come apparirà all'utente finale.

Molti ambienti per la realizzazione del software utilizzano l'interfaccia visuale per il programmatore e creano applicazioni per l'utente finale che presentano gli oggetti tipici dell'**interfaccia grafica**: finestre, pulsanti di comando, caselle di testo, caselle combinate. Inoltre il software gestisce gli **eventi** che accadono durante l'esecuzione del programma provocati dal computer o dalle attività dell'utente (per esempio, l'apertura di una finestra oppure un clic con il mouse sopra un pulsante o sopra una scelta di menu).

Nelle pagine seguenti sono presentate le modalità operative per costruire programmi guidati dagli eventi e interfacce grafiche per l'utente attraverso l'ambiente di sviluppo **Microsoft Visual C++ Express**, già presentato in precedenza (nell'inserito dopo il Capitolo 5) per la realizzazione di *Applicazioni console*.

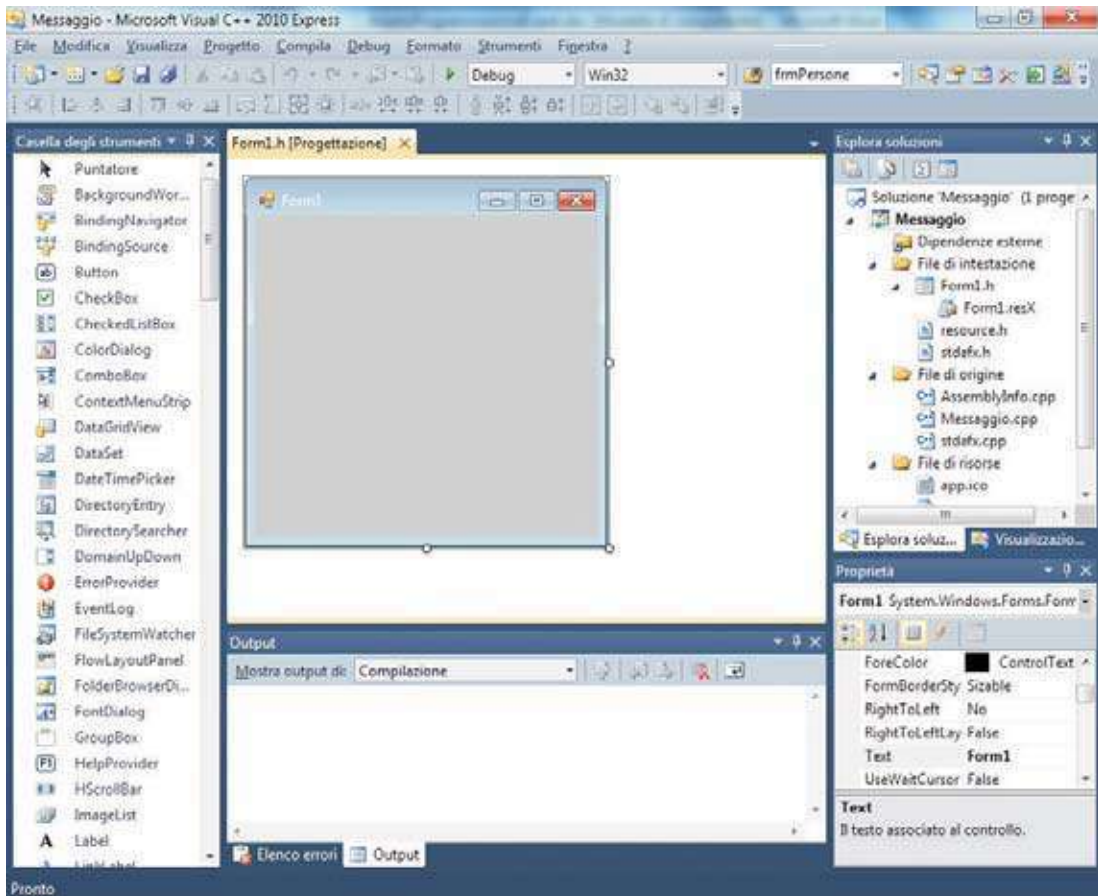
Scegliendo, nella finestra **Nuovo progetto, Applicazione Windows Form** si possono creare programmi con interfaccia grafica per il sistema operativo Windows.



Ciascuna finestra utilizzata nel programma corrisponde a un *form*.

Un **form** è una finestra sulla quale il programmatore inserisce gli elementi che l'utente finale vede. Gli oggetti dell'interfaccia grafica che vengono inseriti in un *form* vengono chiamati **controlli** (dall'inglese *control*).

Le istruzioni che vengono eseguite dal programma sono formate dalle parole chiave, cioè dal **codice** del linguaggio Visual C++ e sono associate al *form*. L'organizzazione dei vari elementi dell'applicazione viene gestita automaticamente dall'ambiente di programmazione e rappresentata nel riquadro di destra (*Esplora soluzioni*).



La finestra di lavoro principale mostra, nella parte centrale, l'anteprima del *form* del progetto appena creato.

Nella parte superiore si trova la **Barra dei menu**, contenente i comandi classici delle applicazioni Windows, come *File*, *Modifica*, *Visualizza*, ecc., più altri menu specifici dell'applicazione.

Al di sotto della *Barra dei menu* si trovano le **barre degli strumenti**. Le barre degli strumenti, come in tutte le applicazioni dell'ambiente Windows, contengono un insieme di icone che rendono più veloce l'accesso alle scelte, di uso più frequente, contenute nei diversi menu di Visual C++. Come impostazione predefinita viene visualizzata la **Barra degli strumenti standard** che contiene le icone dei comandi più comuni (*Nuovo progetto*, *Apri*, *Salva*, *Avvia debug*, *Esplora soluzioni*, ecc.).

È possibile modificare l'insieme delle barre visualizzate utilizzando il menu **Visualizza, Barre degli strumenti** e selezionando le barre preferite.



Sotto le *barre degli strumenti* si trova, solitamente nella parte sinistra della finestra, la **Casella degli strumenti** che permette di inserire tutti i controlli e i componenti standard dell'interfaccia utente (come caselle di testo, pulsanti, etichette, ecc.) semplicemente trascinandoli sul *form*.

Come impostazione predefinita vengono mostrati i *Controlli Comuni*, cioè quelli più utilizzati nei progetti standard. Facendo clic su *Tutti i Windows Form* si visualizza l'elenco completo oppure, a seconda delle esigenze, si può scegliere le altre categorie come *Contenitori*, *Menu e barre degli strumenti*, *Dati*, *Componenti*, ecc.

Nella zona a destra è presente il riquadro **Esplora soluzioni** che mostra l'elenco dei componenti del progetto. Nelle impostazioni predefinite vi è una linguetta che permette di visualizzare la finestra **Visualizzazione classi** contenente variabili e funzioni globali e le classi utilizzate nel progetto o definite dal programmatore.

Nella zona inferiore sono visualizzati il riquadro **Elenco errori** (contenente i riferimenti agli errori di programmazione) e il riquadro **Output** che mostra l'output del programma.

Nella zona centrale viene mostrato sia il codice che l'anteprima del *form*. Si possono aprire più file contemporaneamente e navigare tra di essi utilizzando le linguette. Per ogni *form* (per esempio *Form1*) è presente *Form1.h*, che mostra il codice, e *Form1.h[Progettazione]* che mostra l'anteprima.

Si osservi che il file che definisce il *form* è un *File di intestazione* (avente l'estensione **.h**), mentre il programma sorgente in linguaggio C++, avente lo stesso nome del progetto ed estensione **.cpp**, viene creato nella cartella *File di origine* della soluzione. Esso contiene essenzialmente la direttiva di inclusione per il *form*

```
#include "Form1.h"
```

e la funzione **main()** che crea la finestra principale e la manda in esecuzione.

Per comodità tutti questi riquadri appena descritti possono essere spostati all'interno della finestra principale oppure separati in una finestra a parte. Ogni riquadro possiede cinque modalità:

- *Finestra mobile*, l'elemento viene visualizzato in una finestra separata rispetto alla finestra principale.
- *Finestra ancorata*, la finestra viene sempre visualizzata; una finestra ancorata è riconoscibile dall'icona .
- *Documento a schede*, l'elemento viene visualizzato come scheda nella zona centrale.
- *Nascondi automaticamente*, l'elemento viene nascosto quando non utilizzato; l'elemento viene mostrato automaticamente quando il mouse passa su di esso. Una finestra in questo stato è riconoscibile dall'icona .
- *Nascondi*, l'elemento non viene mostrato.

L'elemento viene chiuso facendo clic sul pulsante  che si trova sempre a destra nella parte alta.

Per passare dallo stato *Finestra ancorata* allo stato *Nascondi automaticamente*, e viceversa, si deve fare clic sulle icone  e .

Per aprire un elemento in precedenza chiuso, nel menu *Visualizza*, occorre scegliere l'elemento desiderato, oppure si può fare clic sull'icona relativa nella *Barra degli strumenti*.

## 2 Gli oggetti per le interfacce grafiche

Gli **oggetti** per la creazione delle interfacce grafiche sono i form e i controlli. Il termine *oggetto* è utilizzato propriamente, perché sia i form che i controlli sono *oggetti* creati utilizzando *classi* che sono rese disponibili dall'ambiente di programmazione.

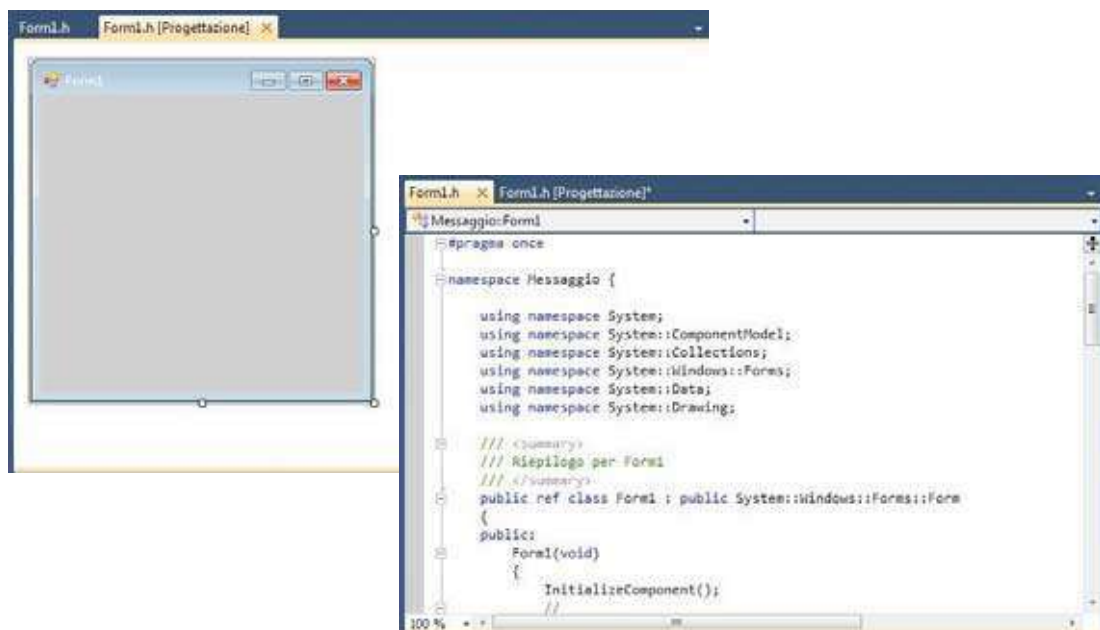
Il **form** è una finestra di interfaccia per l'utente; ciascun form visualizza controlli, immagini o altri form. Il form contiene al suo interno il codice associato ai controlli per gestire le risposte agli eventi causati dalle azioni dell'utente, per esempio il clic su un bottone o l'inserimento di testo in una casella.

I form possono essere schermate introduttive, finestre di dialogo, oppure documenti dell'applicazione con presentazione di dati e risultati.

I **controlli** sono strumenti come caselle di testo, bottoni, icone, etichette, liste di scelta, disegnate dal programmatore direttamente su un form, per ricevere input o visualizzare output: sono gli oggetti tipici che vediamo in tutte le applicazioni dell'ambiente Windows. L'ambiente di sviluppo Visual C++ fornisce una grande quantità di oggetti che il programmatore può utilizzare modificandone le proprietà e i comportamenti e senza preoccuparsi di come l'oggetto sia stato definito. Per disegnare i controlli su un form il programmatore deve prendere un controllo dalla *Casella degli strumenti* e trascinarlo sul form.

Come si è già accennato in precedenza, i form e i controlli sono oggetti creati automaticamente a partire da una classe predefinita e quindi possiedono proprietà che ne specificano i valori iniziali (per esempio la posizione all'interno del form o il colore) e le caratteristiche generali. Aniché inserire nel form un numero elevato di bottoni e icone per le scelte dell'utente, spesso è opportuno raggruppare i comandi dell'applicazione in una **Barra dei menu** lungo il lato superiore del form, secondo gli standard dell'interfaccia grafica di Windows: per fare questo il programmatore dispone di un apposito componente nella *Casella degli strumenti*.

Dopo aver creato gli oggetti, il programmatore deve scrivere il **codice** associato al form e ai controlli per gestire le risposte dell'applicazione alle azioni dell'utente o agli eventi del sistema. Il codice viene scritto nella finestra del codice associata al form disegnato sul video. Il codice che definisce il form viene creato automaticamente ed è visibile nella finestra del codice.



Come impostazione predefinita viene visualizzata la **finestra di progettazione** (nell'immagine *Form1.h [Progettazione]*). Per passare alla **finestra del codice** si deve fare clic sulla linguetta con la scritta *Form1.h* oppure utilizzare la combinazione di tasti **Ctrl + Alt + 0**.

Per ritornare alla finestra di progettazione si deve fare clic sulla linguetta con la scritta *[Progettazione]*, oppure utilizzare la combinazione di tasti **Maiuscolo + F7**.

### 3 Creazione di applicazioni in Visual C++

Il progetto software è l'insieme dei form, del codice e dei file di risorse che costituiscono un'applicazione. Nella finestra *Esplora soluzioni* viene visualizzato l'elenco di tutti i file di un'applicazione.

In generale, per realizzare un'applicazione dotata di interfaccia grafica, si seguono le seguenti fasi:

1. progettare l'**applicazione**
2. progettare le **finestre** che compongono l'interfaccia per l'utente
3. disegnare l'interfaccia, inserendo tutti i **controlli** (pulsanti, menu, ecc.)
4. definire le **proprietà** degli oggetti (posizione, dimensione, ecc.)
5. associare ai controlli il codice di **gestione degli eventi** (per esempio: quando si fa clic sul pulsante X viene eseguita la funzione Y)
6. eseguire il **test** dell'applicazione.

#### PROGETTO 1

##### Creare un'applicazione con interfaccia grafica per visualizzare la data e l'ora corrente.

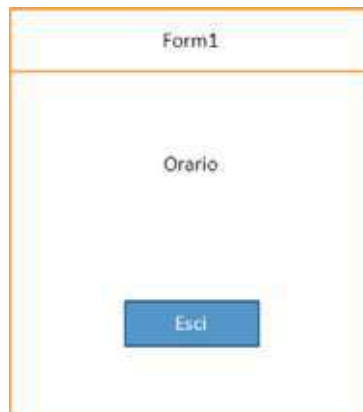
Per creare l'applicazione seguiremo i punti visti in precedenza:

##### 1. Progettare l'applicazione

L'applicazione deve mostrare semplicemente il giorno e l'orario in cui il programma è stato avviato senza svolgere nessun'altra operazione. Deve essere anche possibile chiudere il programma.

##### 2. Progettare le finestre

L'applicazione è composta da un solo form, di cui viene riportato lo schema.



Facendo clic sul pulsante *Esci*, l'applicazione si chiude.

### 3. Disegnare l'interfaccia

- Aprire il programma Visual C++ e creare un nuovo progetto, come abbiamo visto in precedenza, attraverso la scelta **Nuovo Progetto** nel menu **File** oppure con la combinazione di tasti **CTRL + Maiuscolo + N**. Selezionare l'opzione **Applicazione Windows Form**.
- Assegnare il nome *Messaggio* al nuovo progetto.
- A questo punto viene visualizzato un form (chiamato *Form1*) vuoto.

Nella finestra *Esplora soluzioni* viene visualizzato il progetto *Messaggio* che contiene il file di intestazione *Form1.h* e il file sorgente *Messaggio.cpp* con la funzione *main()*.

Nella finestra di progettazione del form, aggiungiamo i controlli. Ne servono due: una **Label** (etichetta), che contiene il messaggio e un **Button** (pulsante di comando) che permette di terminare il programma. Per aggiungere questi controlli, basta selezionarli nella *Casella degli strumenti* (solitamente posta a sinistra) e trascinarli sul form con il mouse.

L'icona della *Label*, all'interno della *Casella degli strumenti*, è . Dopo averla trascinata sul form, viene mostrata in questo modo . L'icona del *Button*, all'interno della *Casella degli strumenti*, è . Dopo averla trascinata sul form, viene mostrata in questo modo .

### 4. Definire le proprietà dei controlli

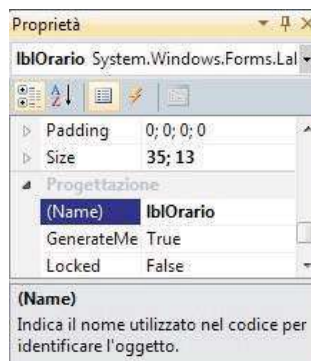
Molte proprietà, per esempio la posizione all'interno del form o le dimensioni del controllo, possono essere direttamente modificate agendo sull'oggetto con il mouse.

Le altre proprietà vengono visualizzate nella finestra **Proprietà** (dal menu **Visualizza, Altre finestre**, scegliere **Finestra proprietà**, oppure fare clic sull'icona della *Barra degli strumenti standard*, oppure premere la combinazione di tasti **Alt + Invio** dopo aver selezionato l'oggetto).

Le proprietà degli oggetti saranno descritte nei paragrafi successivi.

Modifichiamo il testo visualizzato nel *Button1*: selezioniamo il controllo e, nella finestra *Proprietà*, per la proprietà **Text** impostiamo il valore *Esci*.

È buona norma assegnare i nomi degli oggetti in modo da riconoscerli agevolmente all'interno del codice. Solitamente vengono assegnati nomi con un prefisso che permette di individuare il tipo di oggetto (per esempio **lbl** per le *Label* e **btn** per i *Button*) e una o più parole di senso compiuto che indicano la funzione (per esempio *lblOrario* e *btnEsci*). Impostiamo quindi la proprietà **Name** dei due oggetti assegnando i nomi scelti.



### 5. Gestire gli eventi

Si vuole che, all'apertura del form, la proprietà *Text* della casella di testo riporti la data e l'ora. Per fare questo occorre fare doppio clic in un punto qualsiasi del form: si apre la finestra del codice e viene creata automaticamente una funzione *Form1\_Load* che viene eseguita quando il form viene caricato, cioè quando accade l'evento **Load** (caricamento) sull'oggetto *Form1*.

L'intestazione e i parametri vengono scritti automaticamente.

Il programmatore deve scrivere, tra le parentesi graffe, le istruzioni per mostrare la data e l'orario nell'etichetta:

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e) {
 DateTime t = System::DateTime::Now;
 lblOrario->Text = System::Convert::ToString(t);
}
```

La variabile *t* di tipo **DateTime** acquisisce l'ora del sistema. L'istruzione successiva significa: alla proprietà **Text** dell'oggetto puntato da *lblOrario* viene assegnato il valore dell'ora corrente dopo averla convertita in stringa (**ToString**).

Aggiungiamo poi la possibilità di chiudere il programma con un clic sul pulsante di comando *btnEsci*.

Facendo doppio clic sul *Button*, viene mostrato il codice del form e viene creata una nuova funzione *btnEsci\_Click*, che viene eseguita ogni volta che si fa clic sul pulsante (evento **Click** sul pulsante): il codice è costituito dalla chiamata del metodo **Exit** di **Application**, con il quale si chiude il programma.

```
private: System::Void btnEsci_Click(System::Object^ sender, System::EventArgs^ e) {
 Application::Exit();
}
```

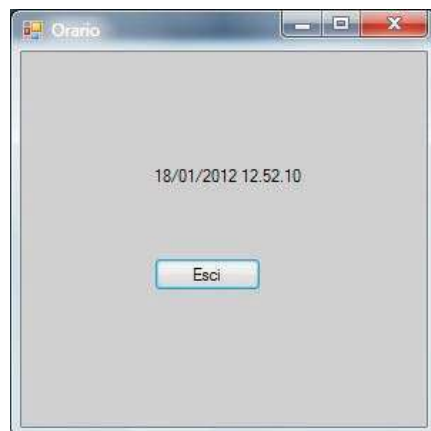
Si noti che per entrambe le funzioni sono stati aggiunti automaticamente due parametri: **sender** che indica l'oggetto che ha provocato l'evento ed **e** che è un array contenente gli argomenti passati insieme all'oggetto.

A questo punto occorre salvare il progetto: nel menu **File**, scegliere **Salva tutto**  oppure fare clic sull'icona **Salva tutto** nella *Barra degli strumenti*.

### 6. Effettuare il test dell'applicazione

Premere l'icona per avviare il **debug** dell'applicazione. Se il programma non contiene errori, si apre il form come una normale finestra di Windows. Facendo clic sul pulsante con la scritta *Esci*, il programma termina.

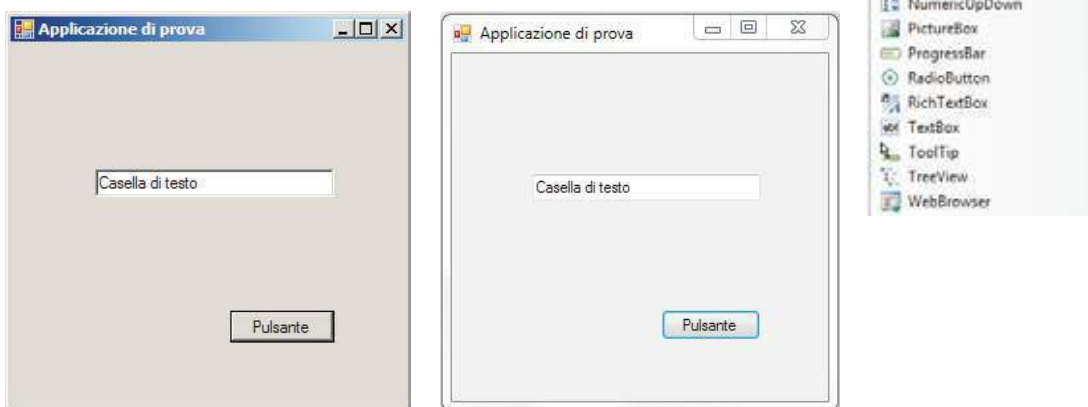
Per il debug e l'esecuzione del programma in ambiente *Microsoft Visual C++ Express*, vale quanto già visto in precedenza (nell'inserto dopo il Capitolo 5) per la realizzazione di *Applicazioni console*.



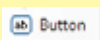
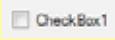
## 4 Form e controlli

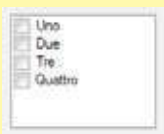
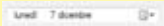
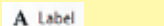
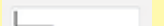
Nel progetto del paragrafo precedente sono stati utilizzati tre oggetti grafici (*Form*, *Button* e *Label*) e alcune proprietà, come *Name* e *Text*. L'elenco dei controlli grafici più utilizzati viene mostrato nella *Casella degli strumenti* facendo clic su **Controlli comuni**.

L'aspetto dei controlli può essere variato modificandone gli attributi. Tuttavia è importante sottolineare che l'aspetto dipende anche dalla versione di Windows con la quale viene eseguito il programma. Per esempio lo stesso *form* con un pulsante si presenta in maniera differente su un computer avente *Windows XP*, rispetto allo stesso programma eseguito in *Windows Vista* o *Windows 7*.

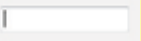


La seguente tabella fornisce una descrizione dei controlli principali.

|                                                                                     |                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------------------------------------------------------------------------|-------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <b>Puntatore</b>                                | È il primo oggetto della casella degli strumenti. Consente di selezionare, spostare e ridimensionare i form e i controlli. Il puntatore consente di gestire i controlli esistenti sul form.                                                                                                                                                                                                                                   |
|  | <b>Button</b><br>(Bottone, Pulsante di comando) | È il classico pulsante utilizzato per iniziare o terminare un'elaborazione. Un <i>Button</i> esegue un'azione quando viene scelto dall'utente. L'utente sceglie in generale un <i>Button</i> facendo un clic su di esso oppure premendo la barra spaziatrice dopo averlo selezionato. I pulsanti <i>OK</i> o <i>Annulla</i> delle applicazioni per Windows sono esempi di pulsanti di comando.                                |
|  | <b>CheckBox</b><br>(Casella di controllo)       | È un quadratino (con un'eventuale crocetta) per gestire le scelte dell'utente del tipo <i>Vero/Falso</i> o <i>Sì/No</i> . La casella di controllo permette di attivare/disattivare un'opzione aggiungendo/togliendo il segno di spunta. Le caselle di controllo servono quindi per specificare le opzioni da parte dell'utente tra quelle offerte dall'applicazione. L'utente può selezionare più opzioni contemporaneamente. |

|                                                                                     |                                                       |                                                                                                                                                                                                                                                                                                          |
|-------------------------------------------------------------------------------------|-------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    | <b>CheckedListBox</b><br>(Lista di controllo)         | Viene visualizzato un elenco di elementi che può essere selezionato/deselezionato aggiungendo/togliendo un segno di spunta. A differenza delle <i>checkbox</i> (dove ad ogni controllo corrisponde un'opzione) una <i>checked listbox</i> può contenere numerose opzioni.                                |
|    | <b>ComboBox</b><br>(Casella combinata)                | Permette di effettuare una scelta singola tra due o più opzioni selezionandola direttamente con il mouse. La casella combinata, come opzione predefinita, mostra solo un'opzione. L'elenco delle opzioni disponibili viene mostrato solamente facendo clic sul triangolino posto a destra del controllo. |
|    | <b>DateTimePicker</b><br>(Data e Ora da calendario)   | Permette all'utente di selezionare una data e un'ora, inserendola direttamente o scegliendola da un calendario. Come opzione predefinita un <i>DateTimePicker</i> mostra la data odierna.                                                                                                                |
|    | <b>Label</b> (Etichetta)                              | È un'etichetta contenente un testo, un messaggio o un'intestazione che l'utente non può modificare.                                                                                                                                                                                                      |
|    | <b>LinkLabel</b> (Etichetta ipertestuale)             | È un'etichetta simile a una <i>Label</i> ma contenente un collegamento ipertestuale.                                                                                                                                                                                                                     |
|   | <b>ListBox</b><br>(Casella di riepilogo)              | Presenta una lista all'interno della quale l'utente può scegliere un'opzione. La casella di riepilogo consente di creare un elenco che l'utente può scorrere per selezionare da esso una o più voci, per esempio un elenco di nomi, di codici o di città.                                                |
|  | <b>ListView</b><br>(Vista di una lista)               | Viene mostrata una lista di elementi che può essere visualizzata in layout differenti. Per esempio è possibile mostrare i dati in tabella o ordinare rispetto una colonna.                                                                                                                               |
|  | <b>MaskedTextBox</b><br>(Casella di testo formattata) | È una casella di testo nella quale può essere inserito solamente un tipo di dato secondo una maschera di inserimento predefinita (per esempio una data, un numero, un testo di lunghezza prefissata).                                                                                                    |
|  | <b>MonthCalendar</b><br>(Calendario mensile)          | Mostra un calendario del mese corrente. La data odierna viene indicata con un colore differente. L'utente, attraverso due piccole frecce, ha la possibilità di navigare attraverso i mesi.                                                                                                               |
|  | <b>NumericUpDown</b><br>(Controllo di scorrimento)    | È una casella di selezione che permette di visualizzare valori numerici. È possibile inserire direttamente il valore oppure incrementarlo o decrementarlo facendo clic sulle due piccole frecce poste a destra del controllo.                                                                            |
|  | <b>PictureBox</b><br>(Casella di immagine)            | Permette di visualizzare un'immagine o un oggetto grafico in generale.                                                                                                                                                                                                                                   |
|  | <b>ProgressBar</b> (Barra di avanzamento)             | È un controllo di indicatore di stato: viene utilizzata per mostrare una percentuale o lo stato di avanzamento di un'operazione.                                                                                                                                                                         |



|                                                                                     |                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------------------------------------------------------------------------------|-----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    | <b>RadioButton</b><br>(Pulsante di opzione)         | <p>È un singolo elemento, che fa parte di un gruppo di bottoni per le scelte multiple, a forma di piccolo cerchio che può essere selezionato per indicare l'opzione prescelta. Il pulsante di opzione permette di selezionare un'opzione, scegliendo tra un gruppo di pulsanti di azione. Quando si seleziona un pulsante di opzione, esso viene riempito con un pallino nero. A differenza di quello che accade per le caselle di controllo, in un gruppo di pulsanti di opzione è consentita la selezione di un unico pulsante, nel senso che una scelta esclude le altre.</p> |
|    | <b>RichTextBox</b><br>(Casella di testo arricchito) | <p>È una casella che può contenere del testo <i>RTF</i> (Rich Text Format). Il testo può essere inserito direttamente da tastiera oppure importato da un file di testo di tipo <i>RTF</i>.</p>                                                                                                                                                                                                                                                                                                                                                                                   |
|    | <b>TextBox</b><br>(Casella di testo)                | <p>Casella per inserire o modificare una stringa di caratteri. La casella di testo è un'area nella quale il testo può essere immesso dall'utente o visualizzato dall'applicazione.</p>                                                                                                                                                                                                                                                                                                                                                                                           |
|    | <b>ToolTip</b><br>(Suggerimento)                    | <p>È una piccola finestra rettangolare, di solito con sfondo giallo chiaro, in cui vengono visualizzate informazioni o suggerimenti su un controllo. Le informazioni vengono visualizzate quando il mouse passa sopra il controllo.</p>                                                                                                                                                                                                                                                                                                                                          |
|   | <b>TreeView</b><br>(Vista ad albero)                | <p>Visualizza una struttura gerarchica di elementi. È possibile espandere o raggruppare i <i>nodi</i> presenti in modo tale da mostrare o nascondere i <i>nodi figli</i>.</p>                                                                                                                                                                                                                                                                                                                                                                                                    |
|  | <b>WebBrowser</b>                                   | <p>Permette di visualizzare una pagina Web con le stesse funzionalità di un browser.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

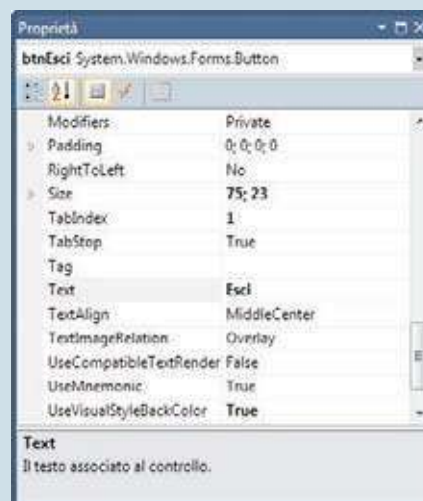


## LE PROPRIETÀ DEGLI OGGETTI

Gli attributi (proprietà) dei controlli vengono impostati nella finestra **Proprietà**, che viene mostrata solitamente a destra. Se la finestra non è aperta, si può scegliere dal menu **Visualizza Altre finestre** e poi **Finestra Proprietà**, oppure premere la combinazione di tasti **Alt + Invio**.

La finestra *Proprietà* contiene (dall'alto verso il basso):

- La **casella di riepilogo degli oggetti**, sotto la barra del titolo, che mostra il *form* o il controllo selezionato. Per visualizzare un elenco che contiene tutti gli elementi selezionabili, basta fare clic sulla piccola freccia che si trova a destra della casella: si può scegliere dall'elenco il *form* corrente oppure un controllo di cui si desidera modificare le proprietà.





- **L'elenco delle proprietà**, che consiste in una lista a due colonne in cui vengono visualizzate (in ordine alfabetico o per categoria) tutte le proprietà che possono essere modificate in fase di progettazione e le impostazioni correnti di ciascuna proprietà. Per visualizzare l'elenco delle impostazioni predefinite si deve fare clic sulla freccia verso il basso a destra della casella oppure scorrere l'elenco delle impostazioni disponibili facendo doppio clic sul nome della proprietà nella colonna a sinistra.
- Una breve spiegazione del significato della proprietà selezionata; come già detto, per avere maggiori informazioni è sempre opportuno utilizzare la *Guida in Linea*.

Se si desidera impostare le stesse proprietà per più controlli contigui, si deve selezionare il gruppo di controlli trascinando il mouse con il tasto sinistro premuto per delimitare l'area che contiene i controlli e quindi fornire i valori per le proprietà con un'unica impostazione. In alternativa, per selezionare più controlli non contigui, occorre fare clic su di essi tenendo premuto il tasto **Ctrl**.

Le principali **proprietà** degli oggetti di uso più frequente sono:

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>        | Nome utilizzato per identificare un <i>form</i> o un controllo.                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>BackColor</b>   | Imposta il colore di sfondo dell'oggetto. I colori utilizzabili sono attributi della classe predefinita <i>System::Drawing::Color</i> .                                                                                                                                                                                                                                                                                                                                                                                |
| <b>ForeColor</b>   | Imposta il colore di primo piano utilizzato per il testo e la grafica di un oggetto.                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>BorderStyle</b> | Imposta lo stile del bordo di un oggetto.                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Enabled</b>     | Determina se il controllo è attivato, cioè se è in grado di rispondere agli eventi generati dall'utente (per esempio un clic del mouse): esso può assumere i valori <i>True</i> e <i>False</i> . I controlli non abilitati sono visibili ma assumono un aspetto più chiaro rispetto agli altri.                                                                                                                                                                                                                        |
| <b>Font</b>        | Specifica le caratteristiche del font utilizzato per i caratteri: <ul style="list-style-type: none"> <li>• <b>Name</b>, il nome del font (per esempio <i>Arial</i>, <i>Times New Roman</i>, ecc.),</li> <li>• <b>Size</b>, la misura dei caratteri</li> <li>• <b>Bold</b>, il grassetto, impostabile come <i>True</i> o <i>False</i></li> <li>• <b>Italic</b>, corsivo, impostabile come <i>True</i> o <i>False</i></li> <li>• <b>Underline</b>, sottolineato, impostabile come <i>True</i> o <i>False</i>.</li> </ul> |
| <b>Size</b>        | Le dimensioni dell'oggetto definite dalle sottoproprietà <i>Width</i> e <i>Height</i> (larghezza e altezza).                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>TabIndex</b>    | Imposta l'ordine di tabulazione di un oggetto all'interno del <i>form</i> , cioè l'ordine con il quale si può passare da un controllo all'altro premendo il tasto di tabulazione della tastiera.                                                                                                                                                                                                                                                                                                                       |
| <b>Text</b>        | Per un <i>form</i> è il testo visualizzato nella barra del titolo. Per un controllo è il testo visualizzato sopra oppure vicino ad esso.                                                                                                                                                                                                                                                                                                                                                                               |
| <b>TextAlign</b>   | Determina l'allineamento del testo di un controllo.                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Visible</b>     | Determina la visibilità ( <i>True</i> o <i>False</i> ) del controllo. Se impostato su <i>False</i> il controllo diventa invisibile.                                                                                                                                                                                                                                                                                                                                                                                    |

Ricordando che tutti i controlli sono oggetti creati a partire dalle *classi* fornite dal sistema, la modifica delle proprietà utilizza la sintassi dei puntatori alle strutture:

```
nomeOggetto->proprietà = valore
```

## 5 Form

L'elemento principale di un'applicazione visuale è il **form** che permette di visualizzare controlli, immagini o altri form. Nell'ambiente Visual C++ il programmatore deve creare, in modo visuale, il form, definirne le proprietà e scrivere il codice associato per gestire gli eventi. Ogni form è un oggetto creato a partire dalla classe *Form* compresa nella **libreria delle classi** predefinite del linguaggio Visual C++.

Un nuovo form viene creato come classe derivata dalla classe base *System::Windows::Forms::Form*.

In generale, tuttavia, per creare un nuovo form, è sufficiente utilizzare il comando **Aggiungi nuovo elemento** all'interno del menu **Progetto** (oppure fare clic sull'icona **Aggiungi nuovo elemento** nella *Barra degli strumenti*) e scegliere **Windows Form**.

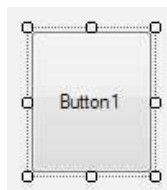
La tabella seguente illustra i principali attributi, caratteristici della classe, che possono essere modificati per adattare il form alle esigenze dell'applicazione da realizzare.

|                                                        |                                                                                                                                                                                                                        |
|--------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>FormBorderStyle</b>                                 | Determina l'aspetto e il comportamento del bordo e della barra del titolo del <i>form</i> . È possibile impostare se visualizzare il bordo oppure se dare all'utente la possibilità di ridimensionare il <i>form</i> . |
| <b>Autoscroll, AutoscrollMargin, AutoscrollMinSize</b> | Permettono di gestire le barre di scorrimento che vengono eventualmente visualizzate se il contenuto del <i>form</i> non rientra nell'area visibile.                                                                   |
| <b>Autosize, AutosizeMode</b>                          | Permettono di gestire il ridimensionamento automatico del <i>form</i> .                                                                                                                                                |
| <b>MinimumSize, MaximumSize</b>                        | Determinano le dimensioni minime e massime del <i>form</i> .                                                                                                                                                           |
| <b>StartPosition</b>                                   | Determina la posizione iniziale del <i>form</i> .                                                                                                                                                                      |
| <b>WindowState</b>                                     | Determina la modalità di apertura del <i>form</i> .                                                                                                                                                                    |

## 6 Pulsanti di comando, caselle di testo e finestre di messaggio

I **bottoni** o **tasti di scelta** o **pulsanti di comando** vengono creati utilizzando la classe **Button** (*System::Windows::Forms::Button*) e rappresentano i bottoni dell'interfaccia grafica dei programmi per Windows. Per aggiungere un pulsante in un *form*, basta trascinare l'icona

 **Button** dalla *Casella degli strumenti* al *form*. Selezionando con un clic il pulsante, vengono mostrati, sui bordi dell'oggetto, i quadratini bianchi per il ridimensionamento del controllo con il mouse.



È possibile inoltre trascinare l'oggetto per modificarne la posizione. Queste caratteristiche (dimensione e posizione) sono attributi dell'oggetto: esse sono indicate con **Size** e **Location**, rispettivamente, e possono essere modificate anche impostandone i valori nella finestra *Proprietà*. Tuttavia, è molto più semplice per il programmatore poter modificare questi attributi, direttamente, utilizzando il mouse.

Facendo doppio clic sull'oggetto, viene creata automaticamente una funzione (chiamata *NomeOggetto\_Click*) che sarà eseguita ogni volta che si fa clic sul pulsante con il mouse.

I principali attributi della classe *Button* sono:

|                       |                                                                                                          |
|-----------------------|----------------------------------------------------------------------------------------------------------|
| <b>FlatStyle</b>      | Determina lo stile del bottone ( <i>piatto o popup</i> ).                                                |
| <b>FlatAppearance</b> | Se <i>FlatStyle</i> è impostato come <b>Flat</b> , imposta l'aspetto del bordo e i colori del controllo. |

La casella di testo **TextBox** è uno dei controlli più utilizzati nei programmi visuali in quanto permette all'utente di inserire dati. Per aggiungere una casella di testo si deve trascinare l'icona  dalla *Casella degli strumenti* nel *form*.

Selezionando con un clic la casella di testo, vengono mostrati i quadratini bianchi che permettono di modificarne la lunghezza, ma non l'altezza, in quanto, come impostazione predefinita, l'oggetto può contenere solamente una riga di testo.



Per modificare questa opzione, occorre fare clic sul piccolo triangolo posto in alto a destra della casella (*Smart Tag* dell'oggetto) e selezionare l'opzione **Multiline**.

Accanto a una casella di testo risulta opportuno inserire un'etichetta che ne descriva il contenuto. Per fare questo si utilizza il controllo **Label**.



Il testo da visualizzare può essere modificato direttamente nella finestra *Proprietà*, impostando l'attributo **Text** del controllo *Label*.

## PROGETTO 2

### Date la base e l'altezza di un rettangolo, calcolarne l'area e il perimetro.

Si vuole creare un programma con interfaccia grafica che, dato un rettangolo, permetta all'utente di inserire, utilizzando le caselle di testo, i valori della base e dell'altezza. Facendo clic su un pulsante di comando, il programma calcola e visualizza i valori di area e perimetro.

#### Dati di input:

base e altezza del rettangolo.

#### Dati di output:

area e perimetro del rettangolo.

#### Nome del progetto

*Rettangolo* di tipo *Applicazione Windows Form*.

#### Disegno dell'interfaccia grafica

Inseriamo all'interno del *form* quattro caselle di testo, per contenere base, altezza, perimetro e area, e un pulsante per eseguire il calcolo. È opportuno assegnare nomi significativi agli oggetti inseriti.

| Classe  | Proprietà dell'oggetto |                                        |
|---------|------------------------|----------------------------------------|
| Form    | Text                   | <i>Area e Perimetro del rettangolo</i> |
| TextBox | Name                   | txtBase                                |
| TextBox | Name                   | txtAltezza                             |
| TextBox | Name                   | txtArea                                |
| TextBox | Name                   | txtPerimetro                           |
| Button  | Name                   | btnCalcola                             |
|         | Text                   | <i>Calcola</i>                         |
| Button  | Name                   | btnEsci                                |
|         | Text                   | <i>Esci</i>                            |



txtBase  
txtAltezza  
btnCalcola  
txtArea  
txtPerimetro  
btnEsci

Per semplicità non sono state indicate le *Label* di descrizione delle caselle: *lblBase*, *lblAltezza*, *lblPerimetro*, *lblArea*. Le descrizioni sono impostate tramite le proprietà *Text* delle *Label*.

### Gestione degli eventi

Facciamo doppio clic sul pulsante *Calcola* per creare la funzione che calcola e visualizza i valori. Innanzitutto si dichiarano le variabili:

```
double base;
double altezza;
double perimetro;
double area;
```

Il secondo passo è prendere i valori contenuti nelle caselle di testo *txtBase* e *txtAltezza* e memorizzarli nelle variabili *base* e *altezza*. Per fare ciò si utilizzano gli attributi *Text* delle caselle di testo.

Per effettuare i calcoli, il testo deve essere convertito in un valore numerico di tipo *Double*:

```
base = System::Convert::ToDouble(txtBase->Text);
altezza = System::Convert::ToDouble(txtAltezza->Text);
```

Tenuto conto delle dichiarazioni create automaticamente dall'ambiente Visual C++ per il form (e visibili nella finestra del codice), le due istruzioni possono essere scritte anche in modo semplificato:

```
base = Convert::ToDouble(txtBase->Text);
altezza = Convert::ToDouble(txtAltezza->Text);
```

Tuttavia la notazione precedente è più chiara e documenta meglio il codice.

Quindi si effettuano i calcoli:

```
perimetro = (base + altezza) * 2;
area = base * altezza;
```

e si inseriscono i valori nelle *TextBox* (sempre utilizzando l'attributo *Text*) dopo aver operato la conversione in stringhe:

```
txtPerimetro->Text = System::Convert::ToString(perimetro);
txtArea->Text = System::Convert::ToString(area);
```

Il codice Visual C++ completo, per la gestione degli eventi *Click* sui due pulsanti, è il seguente:

```
private: System::Void btnEsci_Click(System::Object^ sender, System::EventArgs^ e) {
 Application::Exit();
}
private: System::Void btnCalcola_Click(System::Object^ sender, System::EventArgs^ e) {
 // variabili di input
 double base;
 double altezza;
 // variabili di output
 double perimetro;
 double area;

 // esegue i calcoli
 base = System::Convert::ToDouble(txtBase->Text);
 altezza = System::Convert::ToDouble(txtAltezza->Text);
 perimetro = (base + altezza) * 2;
 area = base * altezza;
 txtPerimetro->Text = System::Convert::ToString(perimetro);
 txtArea->Text = System::Convert::ToString(area);
}
```

Nelle caselle di testo si inseriscono i valori per la base e per l'altezza. Facendo clic sul pulsante *Calcola*, vengono visualizzati i valori calcolati per area e perimetro del rettangolo.



Si deve osservare che le proprietà degli oggetti possono essere utilizzate direttamente, senza dover dichiarare variabili opportune. Quindi i calcoli della precedente funzione potrebbero essere scritti utilizzando solamente due istruzioni:

```
txtPerimetro->Text =
System::Convert::ToString((System::Convert::ToDouble(txtBase->Text) +
System::Convert::ToDouble(txtAltezza->Text)) * 2);

txtArea->Text = System::Convert::ToString(System::Convert::ToDouble
(txtBase->Text) * System::Convert::ToDouble(txtAltezza->Text));
```

Tuttavia la versione precedente è più leggibile e documenta meglio il codice, rendendolo comprensibile anche a distanza di tempo.

**PROGETTO 3**

**Modificare il progetto precedente per il calcolo dell'area e del perimetro del rettangolo, aggiungendo un controllo che avvisa l'utente quando mancano alcuni dati.**

Se l'utente fa clic sul pulsante *Calcola* prima di aver inserito i valori della base e dell'altezza, viene visualizzata una finestra di dialogo con il messaggio "Inserire i dati per il calcolo". Il programma utilizza la struttura di selezione *if . . . else . . .*

**Gestione degli eventi**

```
private: System::Void btnCalcola_Click(System::Object^ sender, System::EventArgs^ e) {
 // variabili di input
 double base;
 double altezza;
 // variabili di output
 double perimetro;
 double area;

 // controlla se una o entrambe le caselle sono vuote
 if (txtBase->Text == "" || txtAltezza->Text == "") {
 MessageBox::Show("Attenzione", "Inserire i dati per il calcolo");
 }
 else {
 // esegue i calcoli
 base = System::Convert::ToDouble(txtBase->Text);
 altezza = System::Convert::ToDouble(txtAltezza->Text);
 perimetro = (base + altezza) * 2;
 area = base * altezza;
 txtPerimetro->Text = System::Convert::ToString(perimetro);
 txtArea->Text = System::Convert::ToString(area);
 }
}
```

**Prova di esecuzione**

Se non è stato inserito un valore nella casella *txtBase* o nella casella *txtAltezza* (cioè il testo è uguale alla stringa vuota), allora appare una finestra di dialogo che comunica il messaggio: "Inserire i dati per il calcolo". In caso contrario si procede normalmente.



Per verificare che la casella di testo sia vuota, si può anche controllare il valore della proprietà **TextLength** (lunghezza, cioè numero dei caratteri, del testo contenuto nella casella):

```
if (txtBase->TextLength == 0 || txtAltezza->TextLength == 0) {
 MessageBox::Show("Inserire i dati per il calcolo", "Attenzione");
}
```

La **finestra di dialogo** con il messaggio è realizzata con l'oggetto **MessageBox** e con il metodo **Show**. La sintassi completa è:

```
MessageBox::Show(Testo, Titolo, Bottoni);
```

dove *Testo* è il messaggio da visualizzare nella finestra; *Titolo* è il testo da visualizzare nella barra del titolo della finestra di messaggio; *Bottoni* è uno dei valori dell'enumerazione **MessageBoxButtons** che specifica i pulsanti da visualizzare nella finestra di messaggio.

I valori di *MessageBoxButtons* sono i seguenti:

|                         |                                                                            |
|-------------------------|----------------------------------------------------------------------------|
| <b>OK</b>               | La finestra di messaggio contiene un pulsante OK.                          |
| <b>OKCancel</b>         | La finestra di messaggio contiene i pulsanti OK e Annulla.                 |
| <b>AbortRetryIgnore</b> | La finestra di messaggio contiene i pulsanti Interrompi, Riprova e Ignora. |
| <b>YesNoCancel</b>      | La finestra di messaggio contiene i pulsanti Sì, No e Annulla.             |
| <b>YesNo</b>            | La finestra di messaggio contiene i pulsanti Sì e No.                      |
| <b>RetryCancel</b>      | La finestra di messaggio contiene i pulsanti Riprova e Annulla.            |

Per esempio:

```
MessageBox::Show("Confermi il dato? ", "Richiesta", MessageBoxButtons::YesNo)
```

La risposta dell'utente può essere controllata con una struttura *if . . . else . . .* come la seguente:

```
if ((MessageBox::Show("Confermi il dato?", "Richiesta", MessageBoxButtons::YesNo)
== DialogResult::Yes)) {
 . . .
}
else {
 . . .
}
```

Il pulsante premuto dall'utente è rappresentato da **DialogResult** che è un'enumerazione del linguaggio Visual C++ con i seguenti valori: *None* (nessuna scelta effettuata dall'utente), *OK*, *Cancel*, *Abort*, *Retry*, *Ignore*, *Yes*, *No*.



#### AUTOVERIFICA

Problemi da 1 a 11 pag. 507



#### MATERIALE ONLINE

### 1. Progetti con interfaccia grafica per l'utente

## 7 Caselle di controllo e pulsanti di opzione

La **casella di controllo** (*CheckBox*)  è costituita da un piccolo quadrato nel quale si può aggiungere o togliere un segno di spunta con un clic del mouse. Questo controllo offre all'utente la possibilità di esprimere un'opzione del tipo Vero/Falso o Sì/No.

Di solito si usano più controlli *CheckBox* contemporaneamente, in modo da permettere all'utente la selezione delle opzioni desiderate, come accade nelle finestre di dialogo per la stampa di testi o per la ricerca di parole all'interno di documenti. In un gruppo di caselle di controllo si possono selezionare uno o più controlli contemporaneamente oppure non selezionarne alcuno.

Accanto a una casella di controllo è posizionata la descrizione del controllo, che viene impostata con la proprietà **Text**.

Lo *stato* del controllo (cioè selezionato o deselezionato) viene determinato dalla proprietà **Checked** dell'oggetto. Questa proprietà è un valore booleano che può essere *True* (controllo selezionato) o *False* (controllo deselezionato).

Per raggruppare due o più *CheckBox* si utilizza il controllo **GroupBox** inserendo i controlli all'interno di esso. Il controllo *GroupBox* non è nell'elenco *Controlli comuni* della *Casella degli strumenti*, ma viene visualizzato in **Contentori** oppure in **Tutti i controlli**.

### PROGETTO 4

#### Evidenziatore di testo.

Si tratta di creare un'applicazione che permetta di evidenziare e colorare il testo contenuto in una casella di testo.

**Dati di input:** un testo.

**Dati di output:** il testo evidenziato e colorato.

#### Nome del progetto:

*Evidenziatore di tipo Applicazione Windows Form.*

#### Disegno dell'interfaccia grafica

| Classe   | Proprietà dell'oggetto |                      |
|----------|------------------------|----------------------|
| Form     | Text                   | <i>Evidenziatore</i> |
|          | Size.Width             | 500                  |
|          | Size.Height            | 200                  |
| TextBox  | Name                   | txtEvidenz           |
|          | Size.Width             | 460                  |
| GroupBox | Name                   | grpEvidenz           |
|          | Text                   | <i>Controlli</i>     |
|          | Size.Width             | 460                  |
| CheckBox | Name                   | chkSfondo            |
|          | Text                   | <i>Sfondo giallo</i> |
| CheckBox | Name                   | chkColora            |
|          | Text                   | <i>Colora</i>        |





### Gestione degli eventi

Attraverso il controllo *chkSfondo* lo sfondo della casella di testo viene evidenziato con il colore giallo, mentre con il controllo *chkColora* il testo viene colorato di rosso. I due controlli possono essere selezionati contemporaneamente.

All'inizio dell'esecuzione del programma i due controlli sono deselezionati. Dopo aver inserito il testo nella casella, la selezione o la deselezionazione di una *CheckBox* avvia la funzione che formatta il testo a seconda delle caselle selezionate.

### Codice Visual C++

```
private: System::Void chkSfondo_CheckedChanged(System::Object^ sender,
System::EventArgs^ e) {
 if (chkSfondo->Checked)
 txtEvidenz->BackColor = Color::Yellow;
 else
 txtEvidenz->BackColor = Color::White;
}

private: System::Void chkColora_CheckedChanged(System::Object^ sender,
System::EventArgs^ e) {
 if (chkColora->Checked)
 txtEvidenz->ForeColor = Color::Red;
 else
 txtEvidenz->ForeColor = Color::Black;
}
```

Come si vede dal codice, il cambiamento dell'evidenziazione è realizzato dalla funzione di gestione degli eventi **CheckedChanged** sugli oggetti di tipo *CheckBox*, che sono provocati dal clic del mouse sulla casella di controllo.

I **pulsanti di opzione** (*RadioButton*)  *RadioButton* sono piccoli cerchi che possono essere selezionati con il mouse per effettuare una scelta tra quelle elencate. Come le *caselle di controllo*, anche i *RadioButton* vengono raggruppati utilizzando il controllo *GroupBox*. Tuttavia, a differenza delle *CheckBox*, è possibile selezionare solamente un'opzione per volta. Selezionando un controllo, gli altri appartenenti allo stesso gruppo vengono automaticamente deselezionati, quindi ogni opzione risulta alternativa alle altre.

Accanto ad ogni pulsante di opzione si può scrivere una descrizione impostandola come proprietà **Text** del controllo. Lo stato del controllo, selezionato o deselezionato, viene determinato dalla proprietà **Checked**, che assume il valore booleano *True* o *False*.

## PROGETTO 5

**Creare un'interfaccia che permetta di acquisire i dati anagrafici (nome, cognome, città e sesso) di una persona.**

### Dati di input:

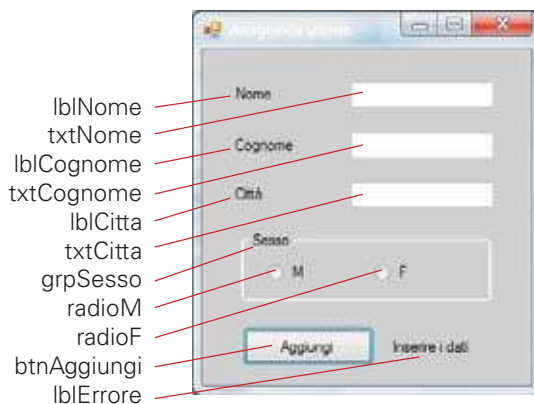
nome, cognome, città, sesso di una persona.

### Nome del progetto:

Anagrafica di tipo *Applicazione Windows Form*.

## Disegno dell'interfaccia grafica

| Classe      | Proprietà dell'oggetto |                          |
|-------------|------------------------|--------------------------|
| Form        | Text                   | <i>Anagrafica utente</i> |
| Label       | Name                   | lblNome                  |
|             | Text                   | <i>Nome:</i>             |
| Label       | Name                   | lblCognome               |
|             | Text                   | <i>Cognome:</i>          |
| Label       | Name                   | lblCitta                 |
|             | Text                   | <i>Città:</i>            |
| TextBox     | Name                   | txtNome                  |
| TextBox     | Name                   | txtCognome               |
| TextBox     | Name                   | txtCitta                 |
| GroupBox    | Name                   | grpSesso                 |
|             | Text                   | <i>Sesso</i>             |
| RadioButton | Name                   | radioM                   |
|             | Text                   | <i>M</i>                 |
| RadioButton | Name                   | radioF                   |
|             | Text                   | <i>F</i>                 |
| Button      | Name                   | btnAggiungi              |
|             | Text                   | <i>Aggiungi</i>          |
| Label       | Name                   | lblErrore                |
|             | Text                   | <i>Inserire i dati</i>   |



## Gestione degli eventi

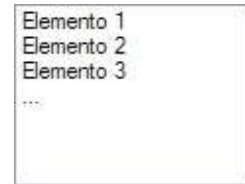
All'inizio dell'esecuzione le caselle di testo sono vuote e il sesso non selezionato. L'utente inserisce i dati della persona. Quando fa clic sul pulsante *Aggiungi*, il programma controlla che le tre caselle di testo non siano vuote. Se non sono vuote (cioè l'utente ha inserito sia il nome che il cognome e la città) verifica anche che almeno uno dei due *RadioButton* sia selezionato. L'etichetta *lblErrore* contiene i messaggi di segnalazione all'utente.

## Codice Visual C++

```
private: System::Void btnAggiungi_Click(System::Object^ sender,
System::EventArgs^ e) {
 if (txtNome->TextLength == 0 || txtCognome->TextLength == 0 ||
txtCitta->TextLength == 0)
 lblErrore->Text = "Inserire i dati anagrafici";
 else
 if (!radioF->Checked && !radioM->Checked)
 lblErrore->Text = "Selezionare il sesso";
 else
 lblErrore->Text = "Dati inseriti correttamente";
}
```

## 8 Casella di riepilogo e casella combinata

La **casella di riepilogo** (*ListBox*) presenta all'utente una lista all'interno della quale può effettuare delle scelte. Questo controllo consente di creare un elenco che l'utente può scorrere per selezionare da esso una o più voci, per esempio nomi, codici, città, ecc.



Se il numero di elementi dell'elenco è maggiore di quelli visualizzabili, viene automaticamente aggiunta al controllo una barra di scorrimento verticale.

Gli elementi possono essere selezionati o deselezionati; la proprietà **SelectedIndex** indica la posizione dell'elemento selezionato all'interno della lista. Se non viene selezionato alcun elemento, il valore della proprietà *SelectedIndex* è uguale a -1. Il primo elemento dell'elenco corrisponde al valore *SelectedIndex* = 0.

La proprietà **SelectedItem** contiene invece il valore dell'elemento selezionato corrispondente alla posizione *SelectedIndex*. In generale il valore contenuto in una *ListBox* è di tipo testo. L'insieme degli elementi presenti nella lista sono rappresentati dalla proprietà **Items** (oggetto di tipo **Collection**). La proprietà **Items->Count** indica il numero degli elementi presenti nella lista. Poiché la numerazione utilizzata parte da 0, il massimo valore di *SelectedIndex* è pari a *Items->Count* - 1.

Per esempio se il valore di *Count* è uguale a 9, cioè nella lista ci sono 9 elementi, *SelectedIndex* può variare da 0 a 8.

Per aggiungere o rimuovere elementi in una *ListBox* si usano i metodi **Add**, **Remove**, **RemoveAt** e **Clear**.

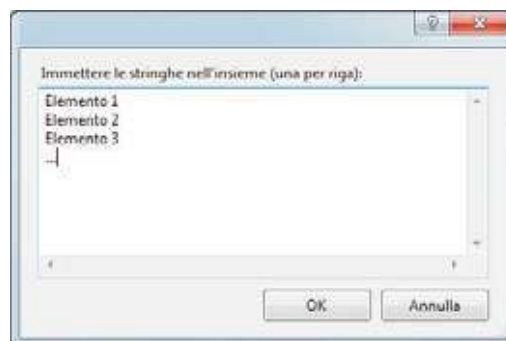
Per esempio:

```
listBox1->Items->Add(Nome);
listBox1->Items->RemoveAt(0);
listBox1->Items->Remove("Roma");
listBox1->Items->Clear();
```

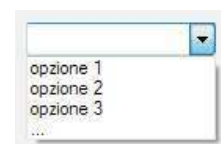
aggiunge alla lista la stringa *Nome*  
rimuove l'elemento di indice 0 (il primo elemento)  
rimuove l'elemento "Roma" dalla lista  
rimuove tutti gli elementi della lista (svuota)

Per **ordinare gli elementi** si utilizza la proprietà **Sorted**, che può assumere un valore booleano (*True* o *False*). Come impostazione predefinita gli elementi non sono ordinati (vengono visualizzati secondo l'ordine di inserimento).

Per inserire i dati in fase di progettazione si deve fare clic sul piccolo triangolo (*Smart Tag*) posto in alto a destra nella *ListBox*, scegliendo l'opzione **Modifica elementi** e inserendo gli elementi uno per riga.



La **casella combinata** (controllo *ComboBox*) consente all'utente di operare una scelta scrivendola in una casella di testo oppure selezionando una scelta in un elenco.



La casella combinata si chiama così perché è composta da una casella di testo e da una casella di riepilogo. L'utente è in grado di digitare il testo nella casella di testo o di selezionare una voce nella casella di riepilogo. Le caselle combinate offrono il vantaggio di presentare all'utente lunghi elenchi di dati senza occupare spazio eccessivo all'interno del form.

Come per le caselle di riepilogo si possono utilizzare le proprietà: **Items**, **Count**, **SelectedIndex** e i metodi **Add**, **Remove** e **Clear**. In fase di progettazione si possono inserire alcuni dati facendo clic sul piccolo triangolo  posto in alto a destra nella *ComboBox*, scegliendo l'opzione **Modifica elementi** e inserendo gli elementi uno per riga.

## PROGETTO 6

### Ricerca il Codice di Avviamento Postale di una città (CAP), oppure il nome della città corrispondente a un CAP.

Il programma consente di gestire i dati relativi al Codice di Avviamento Postale (CAP) di un elenco di città. Dopo aver inserito i dati (il nome e il CAP di ciascuna città), facendo clic sul nome di una città, viene visualizzato il CAP e viceversa.

#### Dati di input

per ciascuna città:

nome

CAP

#### Dati di output

CAP della città selezionata

nome della città avente il CAP selezionato.

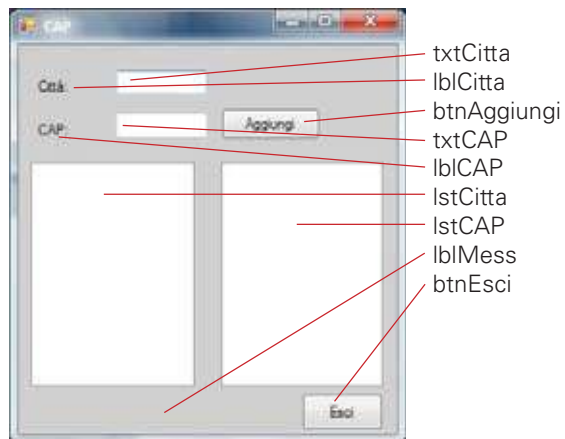
#### Nome del progetto

*Cap di tipo Applicazione Windows Form.*

#### Disegno dell'interfaccia grafica

Il form contiene due caselle di testo per inserire e visualizzare il nome e il CAP di una città. Ci sono poi due *ListBox*: una per i nomi e l'altra per i CAP, e due pulsanti di comando: il primo per aggiungere i dati nelle caselle di riepilogo e il secondo per chiudere il programma.

| Classe  | Proprietà dell'oggetto |             |
|---------|------------------------|-------------|
| Form    | Text                   | CAP         |
| Label   | Name                   | lblCitta    |
|         | Text                   | Città:      |
| Label   | Name                   | lblCAP      |
|         | Text                   | CAP:        |
| TextBox | Name                   | txtCitta    |
| TextBox | Name                   | txtCAP      |
| Button  | Name                   | btnAggiungi |
|         | Text                   | Aggiungi    |
| ListBox | Name                   | lstCitta    |
| ListBox | Name                   | lstCAP      |
| Label   | Name                   | lblMess     |
| Button  | Name                   | btnEsci     |
|         | Text                   | Esci        |



### Gestione degli eventi

Il programma comprende due tipi di attività: l'inserimento dei dati e il recupero dei dati nell'elenco dei dati inseriti.

Quando l'utente preme sul pulsante *Aggiungi*, dopo aver inserito il nome della città e il CAP, i dati vengono registrati in due distinte *ListBox*. In fase di ricerca, quando l'utente fa clic sul nome all'interno della lista delle città, oppure sul CAP all'interno della lista dei codici, nelle caselle di testo vengono visualizzati i dati relativi della città.

### Codice Visual C++

```
private: System::Void btnAggiungi_Click(System::Object^ sender,
System::EventArgs^ e) {
 if (txtCAP->TextLength == 0 || txtCitta->TextLength == 0)
 MessageBox::Show("Inserire tutti i dati", "Attenzione",
 MessageBoxButtons::OKCancel);
 else
 {
 lstCitta->Items->Add(txtCitta->Text);
 lstCAP->Items->Add(txtCAP->Text);
 txtCitta->Clear();
 txtCAP->Clear();
 txtCitta->Focus();
 }
}

private: System::Void lstCitta_SelectedIndexChanged(System::Object^ sender,
System::EventArgs^ e) {
 txtCitta->Text=System::Convert::ToString(lstCitta->SelectedItem);
 txtCAP->Text=System::Convert::ToString(lstCAP->Items[lstCitta->SelectedIndex]);
}

private: System::Void lstCAP_SelectedIndexChanged(System::Object^ sender,
System::EventArgs^ e) {
 txtCAP->Text=System::Convert::ToString(lstCAP->SelectedItem);
 txtCitta->Text=System::Convert::ToString(lstCitta->Items[lstCAP->SelectedIndex]);
}

private: System::Void btnEsci_Click(System::Object^ sender, System::EventArgs^ e) {
 Application::Exit();
}
```

La funzione che gestisce l'evento *Click* sul pulsante *Aggiungi*, dopo aver inserito gli elementi nelle liste, svuota le caselle di testo con il metodo **Clear**:

```
txtCitta->Clear();
txtCAP->Clear();
```

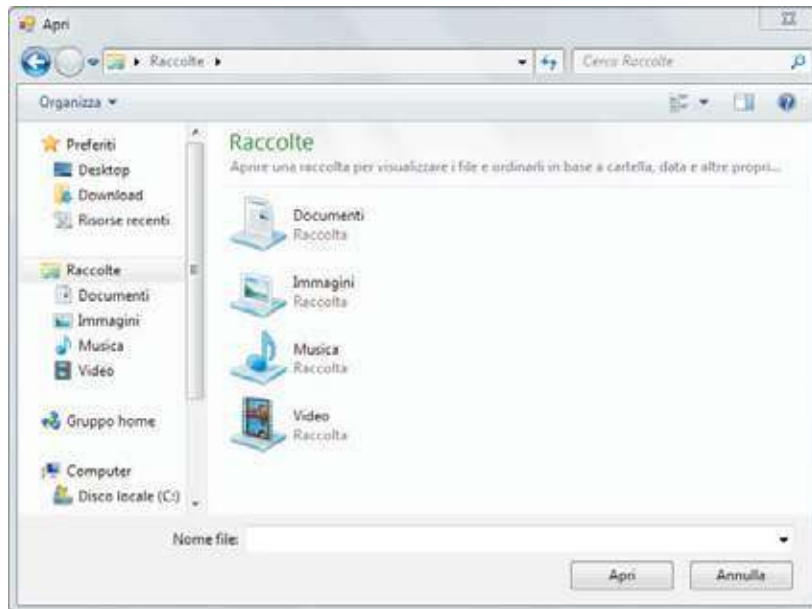
Riposiziona inoltre il cursore del mouse nella casella della città con il metodo **Focus**, in modo che l'utente sia pronto per inserire un nuovo valore. Si dice che la casella di testo acquisisce il **focus** (evidenziazione di un oggetto dell'interfaccia grafica):

```
txtCitta->Focus();
```

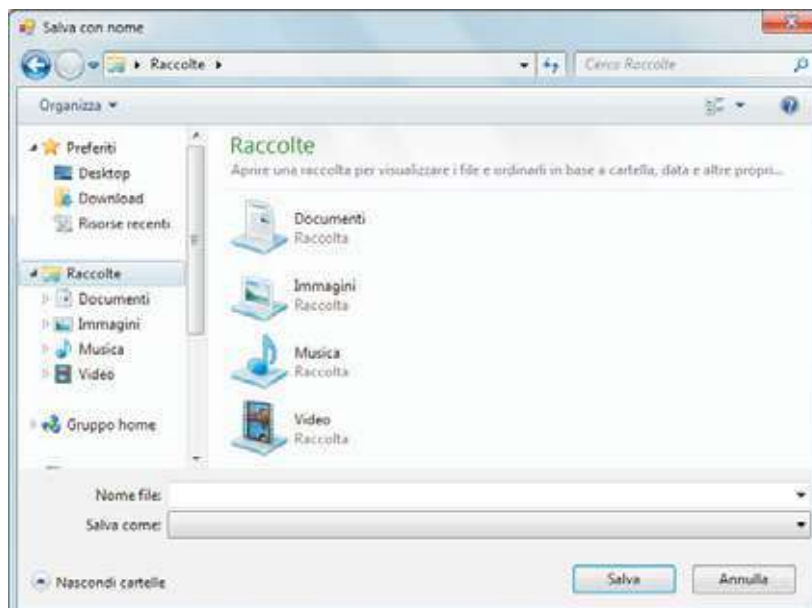
## 9 Finestre di dialogo comuni

Oltre alle finestre di messaggio, si possono utilizzare le **finestre di dialogo comune** che permettono di gestire in modo semplice alcune operazioni comuni, come l'apertura o il salvataggio di un file, la scelta di un colore, la stampa o l'anteprima di stampa. Le *finestre di dialogo comune* sono create utilizzando classi derivate della classe **CommonDialog**, per rappresentare i controlli grafici contenuti nell'elenco **Finestre di dialogo** della *Casella degli strumenti*.

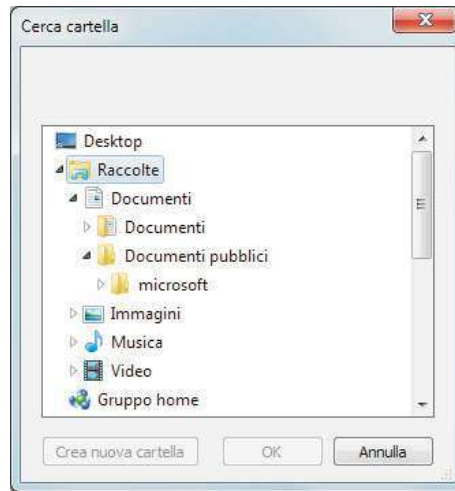
- La **finestra di dialogo Apri** (*OpenFileDialog*) viene utilizzata per recuperare il nome e il percorso di un file da aprire.



- La **finestra di dialogo Salva con nome** (*SaveFileDialog*) permette di salvare un file indicando il nome e il percorso.



- La **finestra di dialogo Sfoglia** (*FolderBrowserDialog*) visualizza la struttura delle cartelle presenti sul disco; all'interno di essa l'utente può scegliere un percorso.



- La **finestra di dialogo Colore** (*ColorDialog*) mostra i colori disponibili e i controlli che permettono all'utente di definire un colore personalizzato.



Per esempio, si può utilizzare la finestra di dialogo *Colore* per modificare il colore dello sfondo di una casella di testo.

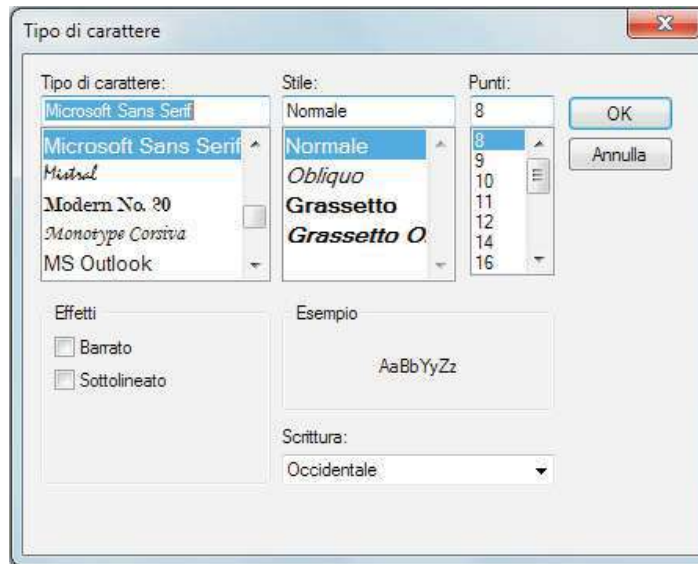
Aggiungiamo in un form una casella di testo *txtNome*, un pulsante di comando chiamato *btnSfondo* e una finestra di dialogo *Colore* chiamata *colorDialog1*. Quando si trascina una finestra di dialogo dalla *Casella degli strumenti* sopra il form, essa viene posizionata sotto il form. Scriviamo poi il seguente codice associato al pulsante di comando:

```
private: System::Void btnSfondo_Click(System::Object^ sender,
System::EventArgs^ e) {
 colorDialog1->ShowDialog();
 txtNome->BackColor = colorDialog1->Color;
}
```

Il codice mostra la finestra di dialogo per la scelta del colore.

Il colore selezionato (*colorDialog1->Color*) viene utilizzato come colore di sfondo della casella di testo (*txtNome->BackColor*).

- La **finestra di dialogo Font** (*FontDialog*) permette all'utente di selezionare un tipo di carattere tra quelli installati sul computer locale.



Per esempio si può usare la finestra di dialogo *Font* per modificare il font di una casella di testo. Aggiungiamo in un form una casella di testo *txtNome*, un pulsante *btnFont* e una finestra di dialogo *Font* chiamata *fontDialog1*. Scriviamo poi il seguente codice associato al pulsante di comando:

```
private: System::Void btnFont_Click(System::Object^ sender, System::EventArgs^ e) {
 fontDialog1->ShowDialog();
 txtNome->Font = fontDialog1->Font;
}
```

Il codice mostra la finestra di dialogo per la scelta del carattere. Le informazioni scelte dall'utente (*fontDialog1->Font*) vengono impostate sulla casella di testo (*txtNome->Font*). L'attributo **Font** dell'oggetto *FontDialog* comprende numerose informazioni: per esempio stile del font e dimensioni.

Vi sono poi altre tre finestre di dialogo che vengono utilizzate per gestire la stampa:

- La **finestra di dialogo Imposta pagina** (*PageSetupDialog*) per modificare le impostazioni di stampa relative alla pagina, come i margini e l'orientamento della pagina;
- La **finestra di dialogo Anteprima di stampa** (*PrintPreviewDialog*), per visualizzare un'anteprima di stampa;
- La **finestra di dialogo Stampa** (*PrintDialog*) per scegliere la stampante con la quale stampare il documento e modificarne le impostazioni.

Come abbiamo detto, quando uno di questi controlli viene inserito nel progetto, non viene visualizzato nel form, ma resta posizionato nella parte inferiore della finestra di progettazione. Per attivare una finestra di dialogo, occorre utilizzare il metodo **ShowDialog** applicato alla finestra.

Per gestire le scelte, fatte dall'utente all'interno della finestra, si usa il tipo enumerativo **DialogResult**. I valori dell'enumerazione sono: *Abort*, *Cancel*, *Ignore*, *No*, *Ok*, *Retry*, *Yes*.



Le finestre di dialogo per l'apertura e il salvataggio di file, viste in precedenza, hanno proprietà comuni che permettono di stabilire il tipo di file da aprire o la cartella iniziale:

| Proprietà               | Significato                                                                                                                        |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <b>DefaultExt</b>       | Estensione del file predefinita.                                                                                                   |
| <b>FileName</b>         | Stringa che contiene il nome e il percorso del file selezionato nella finestra di dialogo.                                         |
| <b>Filter</b>           | Stringa di filtro per determinare il tipo di file.                                                                                 |
| <b>FilterIndex</b>      | Valore numerico che permette di selezionare uno tra più filtri.                                                                    |
| <b>InitialDirectory</b> | Cartella di partenza della finestra di dialogo.                                                                                    |
| <b>RestoreDirectory</b> | Valore booleano che stabilisce se la cartella corrente viene memorizzata come predefinita alla chiusura della finestra di dialogo. |
| <b>Title</b>            | Titolo della finestra di dialogo.                                                                                                  |

## PROGETTO 7

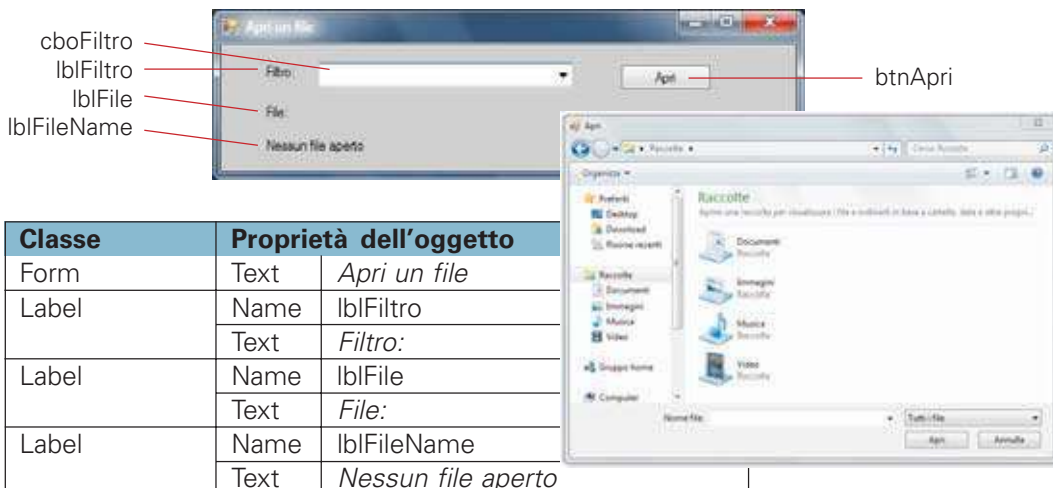
### Aprire un file navigando all'interno del file system.

Si vuole definire anche un filtro in modo da visualizzare solamente alcuni tipi di file (per esempio solo immagini o solo documenti).

#### Nome del progetto

*GestioneFile* di tipo *Applicazione Windows Form*.

#### Disegno dell'interfaccia grafica



| Classe   | Proprietà dell'oggetto |                                           |
|----------|------------------------|-------------------------------------------|
| Form     | Text                   | <i>Apri un file</i>                       |
| Label    | Name                   | lblFiltro                                 |
|          | Text                   | <i>Filtro:</i>                            |
| Label    | Name                   | lblFile                                   |
|          | Text                   | <i>File:</i>                              |
| Label    | Name                   | lblFileName                               |
|          | Text                   | <i>Nessun file aperto</i>                 |
| ComboBox | Name                   | cboFiltro                                 |
|          | Items                  | <i>Tutti i file  *. *</i>                 |
|          |                        | <i>Documenti Word  *.doc;*.docx</i>       |
|          |                        | <i>Immagini  *.jpg;*.jpeg;*.bmp;*.gif</i> |
|          |                        | <i>File di testo  *.txt;*.rtf</i>         |
| Button   | Text                   | <i>Tutti i file  *. *</i>                 |
|          | Name                   | btnApri                                   |
| Button   | Text                   | <i>Apri</i>                               |
|          | Name                   | ApriFile                                  |

**Gestione degli eventi**

Per decidere i tipi di file da visualizzare nella finestra di dialogo si deve impostare la proprietà **Filter** dell'oggetto *ApriFile*. La stringa del filtro deve contenere una descrizione del filtro seguita dalla barra verticale | e dal modello del filtro.

*Descrizione | Modello*

Per esempio, per impostare l'apertura di documenti Word (con estensione *.doc* o *.docx*) si deve usare la seguente stringa:

Documenti Word | \*.doc; \*.docx

L'utente deve scegliere il tipo di file da aprire utilizzando la *ComboBox* che permette di selezionare uno tra i seguenti filtri:

Documenti Word | \*.doc;\*.docx  
 Immagini | \*.jpg;\*.jpeg;\*.png;\*.gif  
 File di testo | \*.txt;\*.rtf  
 Tutti i file | \*.\*

Dopo aver scelto il tipo di documento, si deve fare clic sul pulsante *Apri* e selezionare il file dalla finestra di dialogo. Il nome del file viene visualizzato nella label *lblFileName*.

**Codice Visual C++**

```
private: System::Void btnApri_Click(System::Object^ sender, System::EventArgs^ e) {
 ApriFile->Filter = System::Convert::ToString(cmbFiltro->SelectedItem);
 ApriFile->ShowDialog();
 lblFileName->Text = ApriFile->FileName;
}
```

**AUTOVERIFICA**

Problemi da 12 a 16 pag. 507

**MATERIALE ONLINE****2. Altri controlli per l'interfaccia utente****10 I menu**

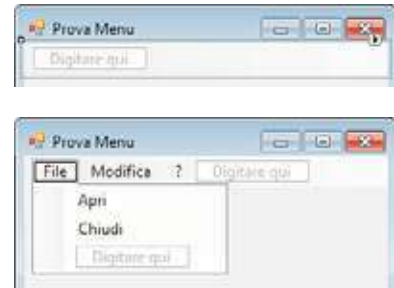
L'interfaccia delle applicazioni in ambiente Windows contiene solitamente, nella parte alta della finestra, la **Barra dei menu**, un insieme di voci sulle quali si può fare clic per aprire un elenco di scelte possibili, chiamato **menu a tendina**.

Le voci sono raggruppate per tipo di funzionalità. Le più comuni, presenti nella maggioranza dei programmi sono:

- **File**, contenente i comandi per aprire, chiudere, salvare o stampare un file
- **Modifica**, con i comandi *Annulla*, *Ripristina*, *Taglia*, *Copia*, *Incolla*, *Elimina*, *Seleziona tutto*, ecc.
- **Cerca**, con i comandi per trovare o sostituire una parola
- **Visualizza**, per modificare la visualizzazione del file aperto
- **?**, per aprire la Guida in Linea o per avere informazioni sul programma.

Per creare un menu in Visual C++ si utilizza l'oggetto **MenuStrip**. Trascinando l'oggetto dalla *Casella degli strumenti* al form, viene visualizzata l'icona nella zona inferiore della finestra di progettazione e un menu vuoto nella zona superiore del *form*.

Le voci e le sottovoci vengono aggiunte scrivendole nel menu. Per ogni voce viene creato automaticamente un nuovo oggetto di tipo **ToolStripMenuItem** del quale si possono modificare gli attributi.



All'interno di ciascun menu si può utilizzare una linea di separazione per suddividere le voci in gruppi (scegliere **Separator** nella casella **Digitare qui**).

Dopo aver impostato il menu si passa alla scrittura del codice per le funzioni associate alle voci, facendo doppio clic su ciascuna voce del menu.

## PROGETTO 8

**Creare un programma con un menu che permetta di modificare in modo interattivo lo stile (font, allineamento, colore dello sfondo, colore del testo) di una casella di testo.**

Lo schema seguente descrive la struttura del menu:

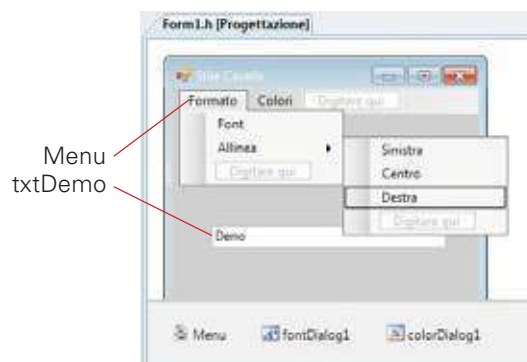
- Formato
  - Font
  - Allinea
    - Sinistra
    - Centro
    - Destra
- Colori
  - Colore sfondo
  - Colore testo

### Nome del progetto

*StileCasella* di tipo *Windows Application Form*.

### Disegno dell'interfaccia grafica

| Classe      | Proprietà dell'oggetto |                      |
|-------------|------------------------|----------------------|
| Form        | Text                   | <i>Stile Casella</i> |
| TextBox     | Name                   | txtDemo              |
|             | Text                   | <i>Demo</i>          |
| MenuStrip   | Name                   | Menu                 |
| FontDialog  | Name                   | fontDialog1          |
| ColorDialog | Name                   | colorDialog1         |



Per creare il menu occorre trascinare l'oggetto *MenuStrip* nel form e aggiungere, tramite l'interfaccia grafica, le voci desiderate. Occorre anche inserire le finestre di dialogo per la scelta del font e del colore.

## Gestione degli eventi

Tutti gli eventi sono legati al clic sulle voci del menu:

### • Formato

**Font:** facendo clic su questa voce si apre la finestra di dialogo per l'impostazione del font che viene poi applicato ai caratteri della casella di testo.

**Allinea:** facendo clic su questa voce si apre un sottomenu per la scelta tra i tre tipi di allineamento del testo (*Sinistra, Centro, Destra*).

### • Colori

**Colore sfondo:** facendo clic su questa voce viene aperta una finestra di dialogo per la scelta del colore. Il colore selezionato viene impostato come colore di sfondo.

**Colore testo:** facendo clic su questa voce viene aperta una finestra di dialogo per la scelta del colore. Il colore selezionato viene impostato come colore del testo.

## Codice Visual C++

```
private: System::Void coloreSfondoToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^ e) {
 colorDialog1->ShowDialog();
 txtDemo->BackColor = colorDialog1->Color;
}

private: System::Void coloreTestoToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^ e) {
 colorDialog1->ShowDialog();
 txtDemo->ForeColor = colorDialog1->Color;
}

private: System::Void fontToolStripMenuItem_Click(System::Object^ sender,
System::EventArgs^ e) {
 fontDialog1->ShowDialog();
 txtDemo->Font = fontDialog1->Font;
}

private: System::Void sinistraToolStripMenuItem_Click(System::Object^ sender,
System::EventArgs^ e) {
 txtDemo->TextAlign = HorizontalAlignment::Left;
}

private: System::Void centroToolStripMenuItem_Click(System::Object^ sender,
System::EventArgs^ e) {
 txtDemo->TextAlign = HorizontalAlignment::Center;
}

private: System::Void destraToolStripMenuItem_Click(System::Object^ sender,
System::EventArgs^ e) {
 txtDemo->TextAlign = HorizontalAlignment::Right;
}
```

L'impostazione dell'allineamento per il testo della casella è rappresentata dall'enumerazione **HorizontalAlignment** che può assumere i valori: **Left**, **Center** e **Right**.

## AUTOVERIFICA

Problemi da 17 a 20 pag. 507-508



## MATERIALI ONLINE

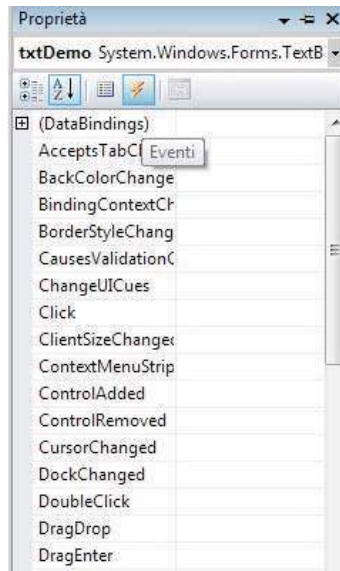
### 3. Barra degli strumenti

### 4. Menu contestuali

## 11 Metodi e gestione degli eventi

L'interfaccia utente di un'applicazione di Visual C++ è composta da oggetti, ovvero i form e i controlli utilizzati per consentire agli utenti di immettere e visualizzare informazioni. Ciascun oggetto riconosce specifiche azioni provocate dall'utente o dal sistema operativo, per esempio un clic su un pulsante, l'apertura di un form, l'inserimento di dati in una casella di testo oppure la modifica di un dato già presente. Come già visto in precedenza, tutte queste azioni si chiamano **eventi**.

Per visualizzare l'elenco degli eventi associati ad un controllo grafico (bottone, casella, lista, ecc.) occorre fare clic sul controllo per selezionarlo e poi aprire la finestra **Proprietà** (dalla *Barra degli strumenti* o con la combinazione di tasti *Alt + Invio*): nella parte alta della finestra fare clic sull'icona **Eventi** (icona con la saetta gialla).



Il programmatore può inserire il codice associato a un particolare evento, cioè il codice che viene eseguito quando accade l'evento, facendo doppio clic sulla casella accanto al nome dell'evento: si apre la finestra del codice con l'intestazione preimpostata della funzione di **gestione degli eventi** (nome della funzione e parametri). Per esempio il codice per l'evento **TextChanged**, che accade quando l'utente modifica il contenuto della casella di testo *txtDemo*, deve essere scritto tra le parentesi graffe della seguente funzione:

```
private: System::Void txtDemo_TextChanged(System::Object^ sender,
System::EventArgs^ e) {

}
```

Gli elementi essenziali della procedura di gestione dell'evento sono:

| Termine                    | Descrizione                                                                                                                            |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <b>txtDemo</b>             | è il nome del controllo (casella di testo).                                                                                            |
| <b>txtDemo_TextChanged</b> | è il nome predefinito della funzione che viene avviata ogni volta che si modifica il contenuto della casella di testo <i>txtDemo</i> . |
| <b>TextChanged</b>         | è il tipo di evento.                                                                                                                   |

I form e i controlli di Visual C++ rispondono a un insieme predefinito di eventi. Per esempio, con un pulsante di comando, si possono verificare i seguenti **eventi**, che sono comuni anche ad altri oggetti dell'interfaccia grafica.

| Nome              | Descrizione                                                                                                                              |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Click</b>      | L'oggetto viene cliccato.                                                                                                                |
| <b>DragOver</b>   | L'oggetto viene trascinato.                                                                                                              |
| <b>DragDrop</b>   | L'oggetto viene rilasciato dopo essere stato trascinato.                                                                                 |
| <b>KeyDown</b>    | Viene premuto un tasto (mentre l'oggetto è attivo).                                                                                      |
| <b>KeyPress</b>   | Viene premuto un tasto (mentre l'oggetto è attivo) e viene restituito il valore del codice <i>ASCII</i> corrispondente al tasto premuto. |
| <b>KeyUp</b>      | Viene rilasciato un tasto (mentre l'oggetto è attivo).                                                                                   |
| <b>MouseDown</b>  | Il pulsante del mouse viene premuto sull'oggetto.                                                                                        |
| <b>MouseMove</b>  | Il puntatore del mouse viene spostato sull'oggetto.                                                                                      |
| <b>MouseUp</b>    | Il pulsante del mouse viene rilasciato sull'oggetto.                                                                                     |
| <b>MouseLeave</b> | Il puntatore del mouse viene spostato dall'oggetto.                                                                                      |
| <b>MouseWheel</b> | La rotella del mouse viene ruotata (mentre l'oggetto è attivo).                                                                          |

Un altro particolare tipo di eventi riguarda il **focus** o evidenziazione degli oggetti: nell'ambiente Windows, i clic del mouse e l'input della tastiera possono essere ricevuti da una sola finestra, un solo form, in generale un solo controllo per volta. Si può dire che l'oggetto su cui si trova l'evidenziazione possiede il *focus* e ha in genere la didascalia o la barra del titolo evidenziate.

Gli eventi collegati al focus sono:

| Nome             | Descrizione                                                         |
|------------------|---------------------------------------------------------------------|
| <b>GotFocus</b>  | L'oggetto riceve il <i>focus</i> , cioè viene messo in primo piano. |
| <b>LostFocus</b> | L'oggetto perde il <i>focus</i> .                                   |

Il focus viene assegnato o tolto con un clic del mouse o con il tasto di tabulazione.

Altri tipi di eventi sono invece specifici dell'oggetto *Form*:

| Nome              | Descrizione                                                        |
|-------------------|--------------------------------------------------------------------|
| <b>Load</b>       | Evento generato quando il form viene caricato in memoria centrale. |
| <b>Activate</b>   | Il form diventa finestra attiva, cioè riceve il <i>focus</i> .     |
| <b>Deactivate</b> | Il form va in secondo piano, cioè perde il <i>focus</i> .          |

La routine di eventi viene eseguita solo quando si verifica l'evento specifico; se non accade alcun evento, il programma rimane inattivo. Il programmatore deve prevedere le azioni dell'utente e le risposte ottenute dal programma. Quindi occorre scrivere il codice per ciascun evento.

## PROGETTO 9

### Contachilometri del mouse: programma per determinare lo spazio percorso (in pixel) dal mouse all'interno di una determinata area.

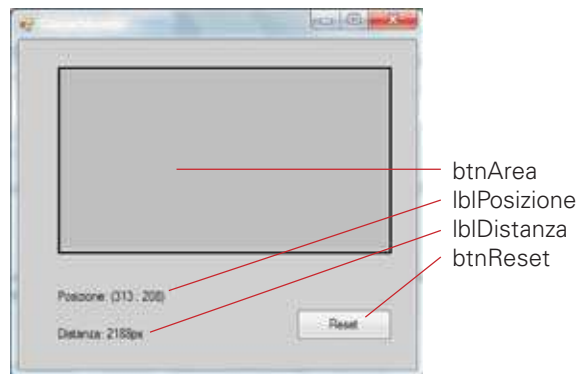
Il programma, ogni volta che il mouse si muove all'interno dell'area prestabilita, memorizza la posizione del mouse, misura la differenza tra la posizione attuale e quella precedente e incrementa la distanza totale, visualizzandone mano a mano il valore.

#### Nome del progetto

*ContaChilometriMouse* di tipo *Applicazione Windows Form*.

#### Disegno dell'interfaccia grafica

| Classe | Proprietà dell'oggetto |                        |
|--------|------------------------|------------------------|
| Form   | Text                   | <i>Contachilometri</i> |
|        | Size                   | 300; 300               |
| Button | Name                   | btnArea                |
|        | FlatStyle              | Flat                   |
|        | Size                   | 260; 160               |
|        | Text                   |                        |
| Button | Name                   | btnReset               |
|        | Text                   | <i>Reset</i>           |
| Label  | Name                   | lblPosizione           |
|        | Text                   | <i>Posizione</i>       |
| Label  | Name                   | lblDistanza            |
|        | Text                   | <i>Distanza: 0px</i>   |



#### Gestione degli eventi

Il programma deve prevedere la gestione di tre eventi:

- evento **MouseMove** sul controllo **btnArea**: il cursore si muove sopra il controllo.

1. viene incrementata la distanza orizzontale e verticale
2. viene calcolata la distanza assoluta
3. viene memorizzata la nuova posizione
4. vengono aggiornate le *Label* con i nuovi valori di posizione e distanza percorsa.

- evento **MouseLeave** sul controllo **btnArea**: il cursore esce dal controllo.

Viene visualizzato l'avviso "Non sei all'interno dell'area"

- evento **Click** sul pulsante **btnReset** ed evento **Load** sul form (caricamento iniziale del form).

1. vengono azzerate le distanze (orizzontale, verticale e assoluta)
2. viene aggiornata la *Label* contenente la distanza.

Per calcolare la distanza percorsa ad ogni movimento del mouse (evento **MouseMove**) viene eseguita la seguente sequenza di calcoli:

1.  $dist_x = dist_x + |posizioneMouseAttuale_x - posizioneMouseVecchia_x|$
2.  $dist_y = dist_y + |posizioneMouseAttuale_y - posizioneMouseVecchia_y|$
3.  $dist = \sqrt{dist_x^2 + dist_y^2}$
4.  $posizioneMouseVecchia_x = posizioneMouseAttuale_x$
5.  $posizioneMouseVecchia_y = posizioneMouseAttuale_y$

**Codice Visual C++**

```
// Dichiarazione delle variabili
int distanzaX, distanzaY;
int posizioneX, posizioneY;
double distanza;

// Il cursore esce dall'area
private: System::Void btnArea_MouseLeave(System::Object^ sender,
System::EventArgs^ e) {
 lblPosizione->Text = "Non sei all'interno dell'area";
}

// Il cursore si muove sopra l'area
private: System::Void btnArea_MouseMove(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e) {
 // alle distanze orizzontale e verticale viene aggiunta
 // la differenza di posizione in valore assoluto
 Point p = Control::MousePosition;
 distanzaX += Math::Abs(p.X - posizioneX);
 distanzaY += Math::Abs(p.Y - posizioneY);
 // calcolo della distanza assoluta
 distanza = Math::Truncate(Math::Sqrt(Math::Pow(distanzaX, 2.0) +
Math::Pow(distanzaY, 2.0)));
 // aggiorna la posizione
 posizioneX = p.X;
 posizioneY = p.Y;
 // aggiorna la label
 lblPosizione->Text = "Posizione: (" + Convert::ToString(posizioneX) + " ; "
+ System::Convert::ToString(posizioneY) + ")";
 lblDistanza->Text = "Distanza: " + Convert::ToString(distanza) + "px";
}

// Click sul pulsante Reset
private: System::Void btnReset_Click(System::Object^ sender,
System::EventArgs^ e) {
 distanzaX = 0;
 distanzaY = 0;
 distanza = 0;
 lblDistanza->Text = "Distanza: " + Convert::ToString(distanza) + "px";
}

// Inizializzazione delle distanze
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e) {
 distanzaX = 0;
 distanzaY = 0;
 distanza = 0;
 lblDistanza->Text = "Distanza: " + Convert::ToString(distanza) + "px";
}
```

La posizione del mouse è ottenuta con l'istruzione

```
Point p = Control::MousePosition;
```

che assegna a un oggetto di tipo **Point** la posizione del mouse (proprietà **MousePosition**). Le coordinate sono le proprietà **X** e **Y** dell'oggetto: *p.X* e *p.Y*.



Per i calcoli, il programma utilizza alcuni metodi predefiniti della classe **Math**:

**Math::Abs()** per il calcolo del valore assoluto

**Math::Sqrt()** per il calcolo della radice quadrata

**Math::Pow()** per l'elevamento a potenza

**Math::Truncate()** per la parte intera di un numero.

## PROGETTO 10

### Creare un semplice browser per visualizzare le pagine Web.

Il browser permette di inserire l'indirizzo del sito in cui navigare oppure di ricaricare una pagina.

#### Nome del progetto

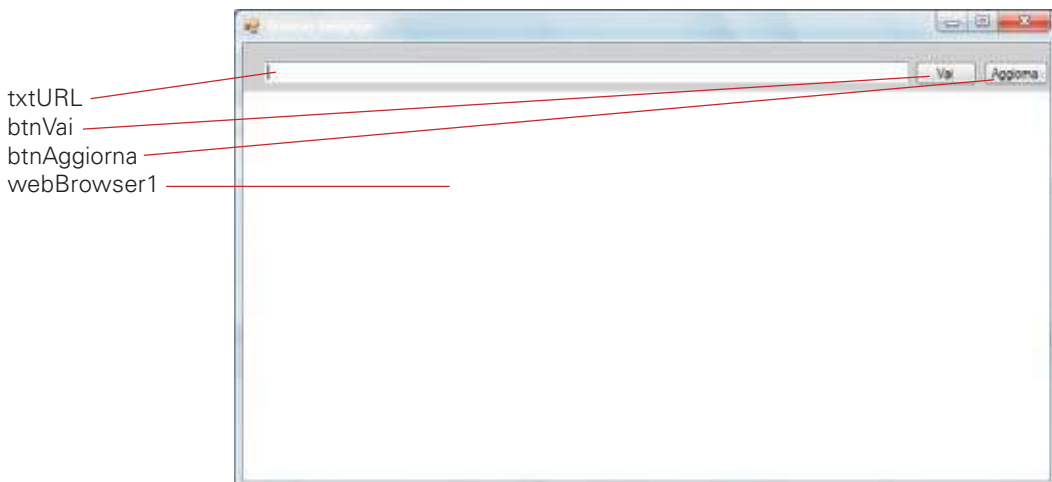
*Browser di tipo Applicazione Windows Form.*

#### Disegno dell'interfaccia grafica

Il form contiene un controllo per il browser, una casella di testo per inserire l'indirizzo del sito Web, un pulsante per avviare il caricamento della pagina Web e un pulsante per aggiornare la pagina.

| Classe     | Proprietà dell'oggetto |                         |
|------------|------------------------|-------------------------|
| Form       | Text                   | <i>Browser Semplice</i> |
|            | WindowState            | Maximized               |
| TextBox    | Name                   | txtURL                  |
| Button     | Name                   | btnVai                  |
|            | Text                   | <i>Vai</i>              |
| Button     | Name                   | btnAggiorna             |
|            | Text                   | <i>Aggiorna</i>         |
| WebBrowser | Name                   | WebBrowser1             |

La proprietà **WindowState** del form è impostata al valore **Maximized** per rendere disponibile l'intero schermo per il browser.



### Gestione degli eventi

Gli eventi da gestire sono due: il clic sul pulsante *Vai* per avviare la navigazione nel sito richiesto e il clic sul pulsante *Aggiorna* per ricaricare la pagina.

Per il primo evento il programma utilizza il metodo **Navigate** dell'oggetto *WebBrowser*, che ha come argomento l'URL fornito dall'utente tramite la casella dell'indirizzo *txtURL*. Per il secondo evento si usa il metodo **Refresh**.

Sono disponibili per l'oggetto *WebBrowser* anche i metodi **GoBack**, **GoForward**, **GoHome**, che corrispondono ai pulsanti standard *Indietro*, *Avanti* e *Pagina iniziale* di un browser.

### Codice Visual C++

```
private: System::Void btnVai_Click(System::Object^ sender, System::EventArgs^ e) {
 if (txtURL->TextLength == 0)
 MessageBox::Show("Inserire l'indirizzo", "Attenzione");
 else
 webBrowser1->Navigate(txtURL->Text);
}

private: System::Void btnAggiorna_Click(System::Object^ sender,
System::EventArgs^ e) {
 webBrowser1->Refresh();
}
```

### AUTOVERIFICA

Problemi da 21 a 24 pag. 508



### MATERIALI ONLINE

## 5. Programmi eseguibili dei progetti

## PROBLEMI

### Form e controlli

- 1 Creare un programma *Termina* con un solo controllo che permette di chiudere il programma.
- 2 Disegnare un form con tre caselle di testo, con le relative etichette, e due pulsanti di comando.
- 3 Modificare il colore di sfondo di un pulsante.
- 4 Cambiare lo stile e il font di un pulsante.
- 5 Cambiare le dimensioni di un form modificando i valori nella finestra *Proprietà*.
- 6 Cambiare le dimensioni di un pulsante di comando agendo sui valori visualizzati nella finestra *Proprietà*.
- 7 Creare un programma che, dato un valore inserito in una casella di testo, ne calcoli il doppio.
- 8 Creare un programma che, dati due valori in caselle di testo, ne calcoli la somma e il prodotto.
- 9 Creare un programma che calcoli l'area di un triangolo, date la base e l'altezza.
- 10 Creare un programma con tre caselle di testo contenenti valori numerici. Facendo clic sul pulsante di comando *Aggiungi*, tutti i valori delle caselle devono essere incrementati del 20%.
- 11 Dato il lato, calcolare il perimetro e l'area di un quadrato.

### Controlli per l'interfaccia grafica

- 12 Creare un form che calcoli il valore dell'imposta IVA su un prodotto. La percentuale da applicare viene selezionata da un elenco di valori in una casella combinata.
- 13 Creare una semplice calcolatrice che esegua la somma e il prodotto di due numeri. I numeri inseriti e il risultato devono essere scritti in una ListBox.
- 14 Creare un programma che permetta di inserire un elenco di nomi in una ListBox. Il programma, in ogni istante, indica in una casella di testo quanti nomi sono stati inseriti. Aggiungere un pulsante per dare la possibilità all'utente di rimuovere i nomi dalla ListBox.
- 15 La domanda di un quiz ha tre risposte disponibili (una sola corretta). Il programma controlla la risposta e segnala all'utente se la risposta è corretta.
- 16 Creare un programma che permetta di inserire dati relativi a un prodotto: nome, prezzo e disponibilità (Sì/No). Il programma controlla che siano stati inseriti tutti i dati in modo corretto.

### Menu

- 17 Creare un menu che contenga tutte le voci del programma "Blocco Note" di *Windows*.
- 18 Costruire un programma che permetta di modificare il titolo della finestra attraverso un comando presente in un menu.

- 19 Creare un menu che permetta di selezionare il nome del mese e il nome del giorno della settimana.
- 20 Creare una semplice calcolatrice che permetta di selezionare le operazioni da compiere tramite menu.

### Gestione di eventi

- 21 Creare una semplice calcolatrice che esegua la somma e il prodotto di due numeri. Le operazioni non sono svolte facendo clic sui tasti, ma semplicemente passando con il mouse sopra due *Label* con le scritte "Somma" e "Prodotto".
- 22 Creare un programma che apre una finestra di messaggio ogni volta che il mouse si sposta sopra una *TextBox*.
- 23 Creare un programma con una *TextBox* e un *Button*. Una *Label* descrive le operazioni svolte dall'utente sui due controlli (per esempio "Sta scrivendo", "Ha cliccato il pulsante", ecc).
- 24 Aggiungere, nell'ultimo progetto di questo capitolo, relativo alla creazione di un semplice Web Browser, due pulsanti che permettano tornare alla pagina visitata precedentemente o alla successiva.

# 9

**parte quarta**

Reti e Web

## Reti e Internet

### **OBIETTIVI DI APPRENDIMENTO**

In questo capitolo imparerai i concetti e i termini riguardanti le reti e Internet.

Sarai inoltre in grado di utilizzare le applicazioni di Internet quali Web, posta elettronica, motori di ricerca e le diverse forme di comunicazione e interazione in rete. Conoscerai le implicazioni collegate alle regole di sicurezza sulle reti e le norme di legge che si riferiscono ai contenuti del Web.

Le reti

Classificazione delle reti

Mezzi trasmissivi

Dispositivi di rete

Le reti pubbliche

Intranet e Extranet

I protocolli

Internet e Web

Il sistema dei nomi di dominio

Le applicazioni di Internet e del Web

La sicurezza delle reti

## 1 Le reti

Con il termine **rete** si fa riferimento, in generale, ai servizi che si ottengono dall'integrazione tra le tecnologie delle telecomunicazioni e le tecnologie dell'informatica.

Una **rete** è un insieme di sistemi per l'elaborazione messi in comunicazione tra loro.

L'interconnessione tra sistemi deve tener conto di problematiche di vari tipi:

- **elettroniche**, per stabilire il collegamento fisico tra i diversi sistemi;
- **informatiche**, per disporre di un sistema operativo e di software in grado di gestire le connessioni;
- **telematiche**, per utilizzare un'infrastruttura per la trasmissione di dati.

Occorre sottolineare come sia stato importante sfruttare l'infrastruttura rappresentata dalle reti telefoniche, originariamente previste per la sola comunicazione verbale, in quanto, nonostante non fossero efficienti per la comunicazione via computer, avevano una caratteristica molto rilevante: la vasta diffusione. La rete telefonica è una rete capillare nelle case e negli uffici e quindi è importante puntare al miglioramento della sua efficienza con progetti tecnologici dedicati soprattutto all'utilizzo delle reti di computer e alla trasmissione di informazioni multimediali per un vasto numero di utenti.

I principi fondamentali delle reti sono:

- deve essere possibile connettere tra loro due computer di modelli diversi e con sistemi operativi diversi;
- nella comunicazione tra sistemi di elaborazione la **distanza** deve diventare un elemento influente, cioè le modalità di connessione e le applicazioni delle reti devono essere le stesse sia quando i computer stanno nello stesso edificio di un'azienda, sia quando essi si trovano a migliaia di chilometri di distanza tra loro.

Il compito principale delle moderne reti di computer è la condivisione di risorse.

Le **risorse** possono essere rappresentate da:

- apparecchiature hardware
- file di dati
- programmi software
- servizi di rete o servizi Web.

## 2 Classificazione delle reti

In generale, quando si parla di condivisione, all'interno delle reti ci sono computer che mettono a disposizione risorse e computer che le utilizzano. Da questo punto di vista, le reti si possono suddividere in due importanti tipi di architetture: reti di tipo *client/server* e reti di tipo *peer to peer*.

Nell'architettura **client/server** i sistemi connessi alla rete possono essere:

- computer **server** (*servente*) che mettono in condivisione delle risorse (per esempio dischi fissi, lettori ottici o la stampante) o offrono servizi per gli altri computer connessi alla rete.
- computer **client** (*cliente* o *richiedente*), cioè i sistemi che utilizzano i servizi e le risorse messe in condivisione da altri computer.

In alcune reti la distinzione tra client e server è netta: in questo caso il server viene detto **server dedicato**. Nella maggioranza delle reti locali, tuttavia, i ruoli di server e di client non sono predefiniti, perché un computer può mettere a disposizione un archivio di dati, diventando server di dati, e nello stesso tempo può utilizzare la stampante di un altro computer, del quale diventa client.

Di solito, nelle reti di grandi dimensioni, per i sistemi serventi si usa anche il termine **host** (ospitante) per indicare sistemi di elaborazione di grandi prestazioni, destinati ad essere centro di distribuzione di informazioni per gli utenti della rete.

Quando un server e un client stabiliscono una comunicazione e inizia l'erogazione del servizio richiesto, per esempio l'esecuzione di un programma, si hanno quindi due possibilità:

- esecuzione **lato client** (o in *locale*): il programma viene trasmesso dal server, caricato nella memoria RAM del client ed eseguito sul client stesso;
- esecuzione **lato server** (o in *remoto*): il programma viene eseguito sul server che trasmette poi i risultati al client.

Nelle reti **peer to peer** (spesso indicate con la sigla **p2p**) ogni computer funziona sia come client che come server, mettendo a disposizione dell'intera rete alcune risorse condivise, tipicamente memorie di massa (dischi fissi e dischi ottici).

Tutti i sistemi operativi moderni, per esempio *Windows*, permettono di condividere cartelle o stampanti in rete, creando automaticamente delle reti di tipo *peer to peer*.

In generale questo modello viene utilizzato solamente in casi particolari o per reti di piccole dimensioni, in quanto presenta molti problemi dal punto di vista della sicurezza: non è possibile, infatti, avere un unico amministratore di rete, né controllare gli accessi degli utenti tramite identificazione. Inoltre questo tipo di rete è meno efficiente rispetto alle reti *client/server*, anche se è di più semplice gestione.

Nella rete Internet sono molto diffusi alcuni software in grado di creare reti *peer to peer* di grandi dimensioni a cui accedono contemporaneamente milioni di utenti sparsi in tutto il mondo e che vengono utilizzate, a volte anche illegalmente, per la condivisione di ogni tipo di file, soprattutto file musicali e film.

Un aspetto particolarmente importante nello studio delle reti è la loro dimensione. Dal punto di vista dell'estensione le reti sono classificate in:

• **Reti Locali LAN** (*Local Area Network*)

Sono le reti di piccole dimensioni corrispondente a un ufficio, a un laboratorio scolastico, ma anche a una struttura aziendale composta da più edifici contigui. Sono utilizzate soprattutto per consentire a un gruppo di utenti di condividere i dati o i programmi software utili per le attività di un'azienda o di un Ente. Essendo di ridotte dimensioni, consentono una notevole velocità nella trasmissione, oltre che una semplificazione delle procedure per garantire la sicurezza e il controllo degli accessi.

• **Reti geografiche WAN** (*Wide Area Network*)

Le reti WAN hanno dimensioni geograficamente estese e possono arrivare a coprire l'intero pianeta. Normalmente, per realizzare questo tipo di rete, vengono utilizzate le infrastrutture già esistenti, per esempio le linee telefoniche oppure linee dati ad alta velocità.

Per garantire la protezione, le banche, le grandi aziende, i grandi Enti della Pubblica Amministrazione, la rete dei Centri di ricerca, utilizzano reti geografiche dedicate.

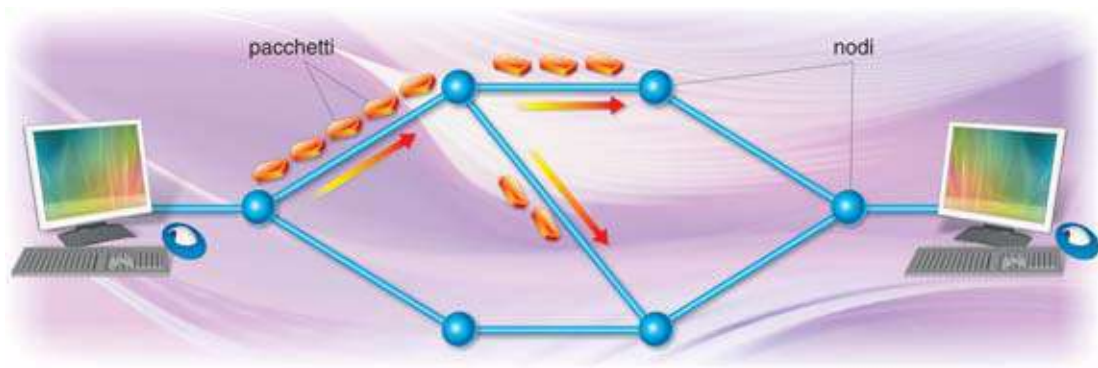
Ovviamente è possibile collegare tra loro reti di tipo diverso. Per esempio è possibile collegare due reti LAN, oppure reti LAN e reti WAN. Questo è il principio che sta alla base di **Internet**, che significa appunto interconnessione tra reti diverse (*internetworking*).

### 3 Mezzi trasmissivi e dispositivi di rete

La comunicazione tra sistemi consiste essenzialmente in un **messaggio** a un server da parte di un client, che richiede l'esecuzione di un lavoro. Il server esegue il lavoro e restituisce la risposta.

Un **messaggio** è un insieme di caratteri e di dati che devono essere trasferiti da un sistema ad un altro; è quindi un insieme di informazioni organizzate in modo da costituire un'entità completa che può essere trasmessa tra due sistemi di rete.

Per aumentare la velocità di trasmissione dei dati, i messaggi da trasmettere sono suddivisi in segmenti chiamati **pacchetti**, secondo l'idea che un messaggio di piccole dimensioni ha più possibilità di trovare una linea di trasmissione libera e, inoltre, può lasciare spazio ad altri messaggi. I pacchetti sono corredati dalle informazioni riguardanti il destinatario e il numero d'ordine in modo che possano essere ricomposti, una volta giunti a destinazione, per riformare il messaggio originale.



La trasmissione di pacchetti si chiama **commutazione di pacchetto**: essa utilizza la tecnica dell'instradamento, in modo che i pacchetti raggiungano le loro destinazioni attraverso i nodi della rete seguendo i percorsi più idonei a seconda del traffico sulla rete stessa.

Nelle reti l'unità di misura della velocità di trasmissione è il **bit per secondo** (indicato con **bps** o con **bit/s**) e i suoi multipli (**Kbps** per migliaia, **Mbps** per milioni, **Gbps** per miliardi di bit per secondo).

La caratteristica più importante di un segnale è la **frequenza** che si misura in Hertz (**Hz**), con i suoi multipli (**MHz** e **GHz**). Un insieme finito di frequenze adiacenti è denominato **banda**. L'intervallo di valori tra i quali varia la frequenza di un segnale si chiama **larghezza di banda**. Per esempio, le comunicazioni vocali telefoniche sono trasmesse con frequenze tra 300 e 3400 Hz, quindi si può dire che la larghezza di banda della comunicazione telefonica è uguale a 3100.

Nella pratica per le reti si utilizza spesso la velocità di trasmissione per determinare la larghezza di banda e quindi si usano come unità di misura i **bps**, **Kbps** e **Mbps**.

Se la larghezza di banda ha valori elevati si dice che la trasmissione avviene in **banda larga**. Nel linguaggio corrente poi la **banda larga** indica la possibilità di far viaggiare grandi quantità di dati e di informazioni multimediali ad alte velocità sullo stesso cavo, o in generale sullo stesso mezzo trasmissivo.

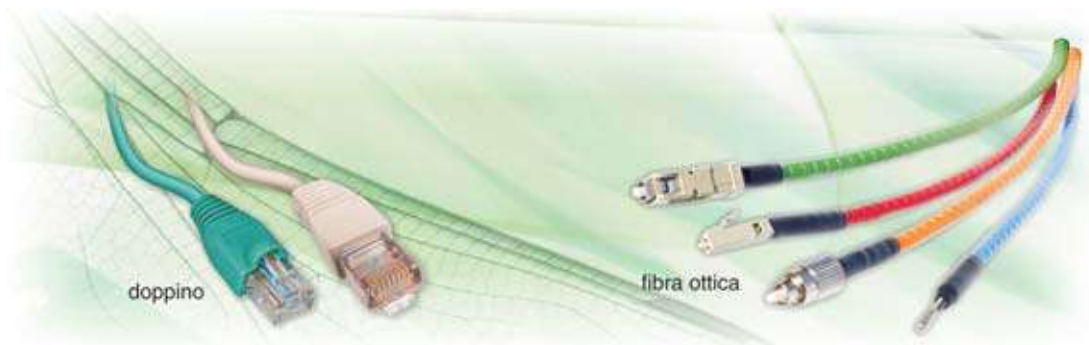
Gli elementi di connessione tra i sistemi di elaborazione sono i connettori, i cavi, i dispositivi hardware necessari per il **cablaggio** delle reti oppure antenne e parabole nel caso di installazioni **wireless**.



I mezzi trasmissivi più utilizzati sono:

#### • Doppino telefonico

Questo canale fisico, detto cavo **UTP** (*Unshielded Twisted Pair*, coppia intrecciata non schermata), è stato impiegato in modo esteso dalle compagnie fornitrici del servizio telefonico: tale mezzo consiste in un doppio filo di rame (da cui appunto deriva il nome) intrecciato per neutralizzare l'influenza dei campi elettromagnetici presenti nell'ambiente e che interferiscono sul segnale trasmesso. Questo mezzo, utilizzato da decenni per la telefonia, ha avuto negli anni recenti un netto miglioramento nella qualità di produzione con un aumento della velocità e dell'efficienza della trasmissione, permettendone l'impiego anche nei cablaggi delle reti locali.



#### • Fibra ottica

Una fibra ottica si presenta come un sottile filo di vetro: i segnali elettrici vengono convertiti in impulsi luminosi con un modulatore e, instradati nella fibra ottica, vengono trasmessi alla velocità della luce all'altro capo della fibra, dove si ottiene la riconversione in segnali elettrici, consentendo di raggiungere alte velocità, dell'ordine dei Gbps. La fibra ottica è soprattutto adottata per la realizzazione di **dorsali**, cioè interconnessione di reti, e come linea di trasmissione dati nelle principali città. Tuttavia le fibre ottiche stanno sostituendo i doppini nelle reti telefoniche, ma anche nelle reti LAN, offrendo maggiore efficienza e velocità, oltre alla potenzialità della banda larga.

#### • Wireless

Le trasmissioni **wireless**, senza fili, usano l'etere come mezzo trasmissivo e consentono di creare facilmente reti senza installare alcun tipo di cablaggio. Sono definite dallo standard denominato **IEEE 802.11** e dalle sue versioni successive. Si possono avere velocità di trasmissione teoriche comparabili alla velocità delle normali reti cablate. Nelle reti wireless si utilizza un apparecchio, detto **access point**, collegato fisicamente alla rete, che comunica con gli utenti attraverso segnali radio. I computer degli utenti devono essere dotati di una scheda per il collegamento wireless, detta **client wireless** (o *terminal adapter wi-fi*).



Di seguito vengono illustrati i principali dispositivi hardware necessari per il cablaggio delle reti.

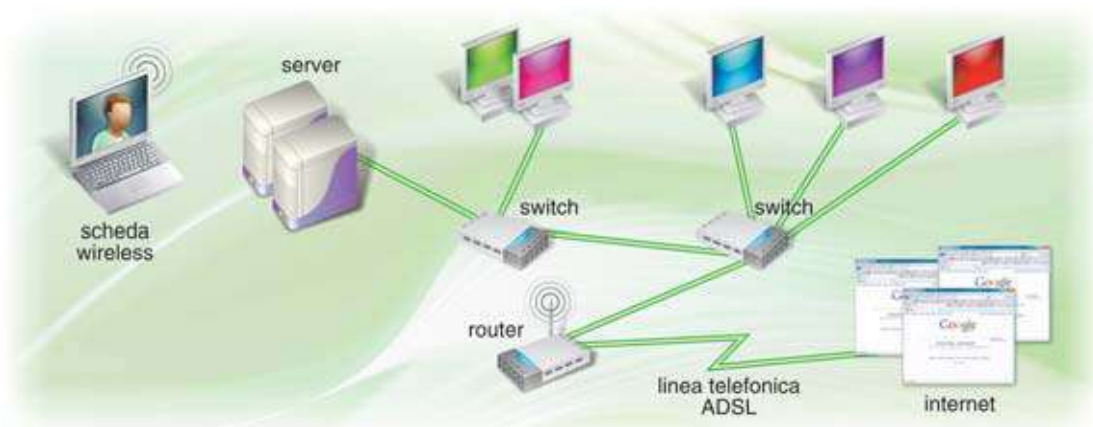
- La **scheda di rete** o **NIC** (*Network Interface Card*) viene installata nell'alloggiamento della scheda madre del computer. Le schede di rete più comuni sono dette **schede 10/100/1000**, in quanto possono trasmettere dati alla velocità di 10 *Mbps* (per le reti meno recenti) fino a 100 *Mbps* oppure 1 *Gbps*.



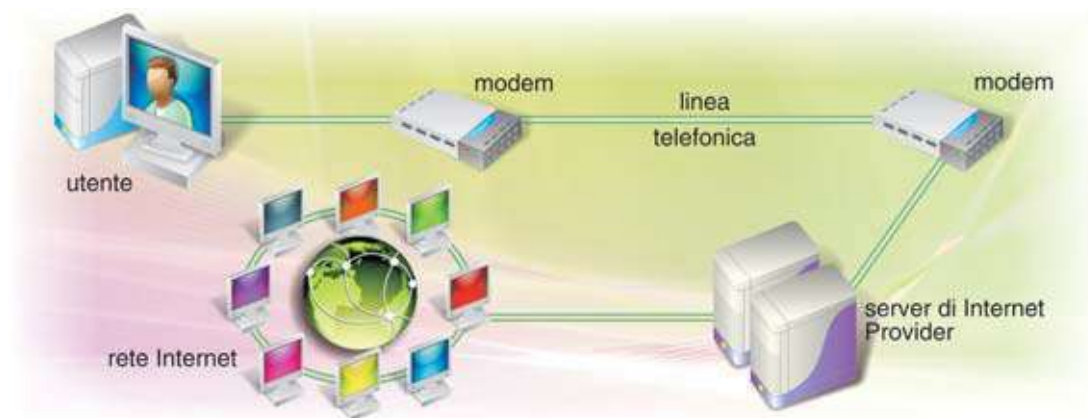
- Lo **switch** è un dispositivo dotato di prese (*porte*) nelle quali vengono inseriti i cavi provenienti dai computer della rete. Uno switch invia i pacchetti di dati alle porte specifiche dei destinatari, sulla base delle informazioni contenute nell'intestazione di ogni pacchetto. Di fatto rappresenta il centro di collegamento dei computer della rete e di smistamento dei messaggi.
- Il **router** è l'apparecchiatura che determina il nodo intermedio successivo nella rete che deve ricevere il pacchetto. Basandosi su una *tabella di routing*, consente ai pacchetti di raggiungere le loro destinazioni attraverso i percorsi più idonei. Poiché i router sono in grado di definire anche i collegamenti tra reti diverse, essi vengono tipicamente utilizzati per mettere in collegamento una rete locale con la rete Internet. Spesso, poi, i router sono dotati di porte che consentono di svolgere anche le funzionalità di switch (*switch-router*).

La figura seguente illustra una configurazione standard di una rete LAN collegata alla rete Internet: ogni PC è dotato di una *scheda di rete* e di un *cavo* per il collegamento allo *switch*, attraverso il quale può comunicare con gli altri PC. Lo switch è connesso al *router*, il quale a sua volta è fisicamente collegato alla rete telefonica per permettere ai PC di uscire all'esterno della rete verso la rete Internet.

Alcuni dei PC, normalmente portatili, ma anche telefoni cellulari di tipo *smartphone*, possono avere al loro interno una scheda *wireless* che consente il collegamento ad un *access point* per la connessione senza fili.



In una configurazione più semplice, l'utente di un personal computer può connettersi alla rete Internet attraverso un **modem** che è collegato da una parte a una porta del PC e dall'altra alla linea telefonica di casa. Questo tipo di collegamento presenta però velocità di trasmissione molto basse: la velocità tipica dei modem è di 56 K (*Kbps*). Il modem si chiama così perché è un *Modulatore/Demodulatore*, cioè il modem del computer di partenza trasforma i segnali digitali del computer in segnali analogici che possono viaggiare sulla linea telefonica (*modulazione*), mentre il modem collegato al computer di destinazione ritrasforma i segnali analogici in digitali (*demodulazione*).



Per la connessione a Internet occorre poi un abbonamento con un Provider dei servizi Internet (**ISP**, *Internet Service Provider*), cioè una società, che di solito coincide con una tra le aziende di servizi telefonici, e che fornisce il punto di accesso per Internet agli utenti.

## 4 Le reti pubbliche

### • PSTN

La soluzione più semplice, ma meno efficiente, per collegarsi a Internet è la normale linea telefonica di casa, la linea analogica detta **PSTN** (*Public Switched Telephone Network*) che usa il doppino telefonico come mezzo trasmissivo: la connessione è realizzata attraverso l'uso di un modem collegato al computer e alla linea telefonica. Il limite è rappresentato dal fatto che, mentre si è connessi a Internet via modem, non si può usare la linea per le comunicazioni vocali.

Le linee digitali messe a disposizione dai fornitori di accesso alle reti pubbliche sono ISDN e ADSL.

### • ISDN

**ISDN** (*Integrated Services Digital Network*) è uno standard internazionale predisposto dalle società concessionarie del servizio telefonico per un sistema di tipo digitale, che consente di integrare la trasmissione di messaggi vocali con quella di dati e immagini. La combinazione fondamentale prevede due canali digitali a 64 Kbps in modo da poter usare la stessa linea sia per la comunicazione vocale che per gli apparati di trasmissione dati.

Negli anni recenti questo tipo di linea è stato sempre meno utilizzato in quanto le linee ADSL risolvono in maniera molto più efficiente il problema della trasmissione contemporanea di voce e dati.

#### • ADSL

La tecnologia **ADSL** (*Asymmetrical Digital Subscriber Line*) è un servizio ad alta velocità che opera attraverso il tradizionale doppino telefonico. La ADSL riserva un canale separato al servizio vocale, quindi si ha la disponibilità della connessione a Internet 24 ore su 24, avendo contemporaneamente libera la linea telefonica. La ADSL si chiama così perché utilizza velocità *asimmetriche* di trasferimento dei dati, cioè i dati provenienti da Internet sono più veloci di quelli inoltrati verso Internet: i modem ADSL permettono di trasmettere le informazioni dal centro servizi verso l'utente (*download*) ad una velocità che può variare da 2 Mbps a 20 Mbps, mentre nella direzione opposta (*upload*) i dati viaggiano ad una velocità compresa tra 300 Kbps e 1 Mbps.

Le moderne reti sono basate sulla **banda larga** che consente una connessione sempre attiva e alte velocità di trasferimento di dati e dei documenti, anche di grandi dimensioni quali immagini, suoni e filmati. Spesso le connessioni hanno tariffe fisse indipendenti dalla durata. Si osservi che l'uso di connessioni in banda larga comporta maggiori rischi per la sicurezza con la possibilità di intrusioni indesiderate nel proprio computer: questi rischi possono essere eliminati con adeguate protezioni sui dati e con controlli sugli accessi.

Oltre che con reti di cavi o con reti wireless, è possibile la connessione a Internet anche attraverso i telefoni cellulari e i satelliti.



### INTRANET E EXTRANET

Le moderne tecnologie di rete hanno di fatto creato una forte integrazione tra le reti locali aziendali e la rete Internet, attraverso le cosiddette reti **Intranet**, cioè un insieme di reti locali appartenenti ad un'azienda tra loro interconnesse e aventi la possibilità di connettersi all'esterno con la rete Internet.

Lo scopo principale di una rete Intranet consiste nella distribuzione delle risorse e delle informazioni aziendali tra i dipendenti, rendendo possibili anche il lavoro di équipe e le teleconferenze tra persone appartenenti alle diverse filiali di un'azienda che si trovano a grandi distanze.

Le reti Intranet usano gli stessi protocolli della rete Internet: questo significa che gli utenti della rete possono usare le stesse modalità di lavoro sia quando elaborano dati aziendali residenti sulla rete locale sia quando accedono alle pagine e agli archivi disponibili nella rete Internet.

Inoltre parte della rete aziendale Intranet può essere resa accessibile ai clienti, alle aziende dello stesso gruppo, ai fornitori o ad altri utenti, per collegare sedi aziendali distanti, oppure per far colloquiare una sede centrale con i computer portatili dei venditori o dei tecnici di manutenzione.

In questi casi si parla di rete **Extranet**, per indicare l'uso di parti della rete aziendale per utenti selezionati esterni all'azienda.

Le aziende possono usare la rete Extranet per:

- scambio di grandi volumi di documenti elettronici;
- distribuzione di cataloghi di prodotti, riservati ai rappresentanti od operatori commerciali;
- collaborazione con altre aziende collegate per lo sviluppo di progetti;
- programmi di addestramento per il personale (*e-training*);
- accesso alle applicazioni fornite da altre aziende;
- distribuzione di informazioni di interesse comune con aziende partner.

## 5 I protocolli

Si è detto all'inizio che la caratteristica fondamentale delle reti consiste nella possibilità di mettere in comunicazione tra loro computer e server con hardware e sistemi operativi diversi, ma anche reti con architetture diverse, operanti a grandi distanze in tutti i paesi del mondo. Quindi è necessario stabilire, a livello internazionale, regole comuni che si traducono poi in principi teorici e modalità operative che stanno alla base dei software per la gestione della comunicazione tra i nodi di una rete.

Queste regole prendono il nome di **protocolli**.

Il protocollo più importante è **TCP/IP** (*Transmission Control Protocol / Internet Protocol*) che è più propriamente un insieme di protocolli. Esso è nato con la rete Internet, ma viene oggi utilizzato in tutte le reti, locali e geografiche, apportando un vantaggio fondamentale: gli utenti delle reti usano le stesse modalità di lavoro e le stesse applicazioni sia quando sono connessi ad una rete locale, sia quando sono connessi alla rete Internet.

In particolare, il protocollo IP si occupa di formare i pacchetti da trasmettere e di instradarli sulla rete, mentre il protocollo TCP ha il compito di trasferire i pacchetti.

In una rete viene associato a ogni computer un **indirizzo IP** (*IP address*) che permette di identificare il computer all'interno della rete stessa.

Gli indirizzi IP sono formati da 4 byte (32 bit) che per convenzione vengono separati da un punto, per esempio:

10010101.00101001.11001000.10100010

Nella rappresentazione pratica viene utilizzata la notazione decimale:

149.41.200.162

Quindi si può dire che un indirizzo IP è formato da quattro numeri, ognuno compreso tra 0 e 255. Vi sono alcuni indirizzi che vengono utilizzati per scopi particolari. Per esempio, gli indirizzi che hanno il primo numero uguale a 127, cioè del tipo 127.x.x.x, sono utilizzati per identificare reti virtuali all'interno di un nodo: in particolare l'indirizzo 127.0.0.1, viene utilizzato per identificare il computer stesso. All'indirizzo 127.0.0.1 viene poi associato il nome simbolico **localhost**.

Vi sono poi alcune altre classi di indirizzi, come 192.168.x.x oppure 172.16.x.x e 10.x.x.x, che, secondo le convenzioni internazionali, devono essere utilizzate solamente come classi di indirizzi interni nell'ambito di reti locali.

Per configurare un computer di una rete locale collegata a Internet tramite un router, si devono utilizzare le **impostazioni del protocollo TCP/IP**. Per esempio nel sistema operativo Windows: *Start, Pannello di Controllo, Rete e Internet, Centro Connessioni di rete e condivisione*, doppio clic sulla connessione, pulsante *Proprietà, Protocollo Internet (TCP/IP)*, pulsante *Proprietà*.

Anche non tenendo conto degli indirizzi IP destinati a usi particolari, si può calcolare che, utilizzando 4 numeri da 8 bit (32 bit), il numero di indirizzi disponibili non supera i 5 miliardi, numero che, quando questo sistema fu concepito, poteva sembrare irraggiungibile, ma che viene ora visto come limitativo per lo sviluppo di Internet. L'alternativa di indirizzamento che risolve i problemi legati ai grandi numeri di utenti Internet è rappresentata dal nuovo standard denominato **IPv6** (IP versione 6). Esso definisce indirizzi formati da 16 byte e quindi tali da garantire indirizzi teorici disponibili fino a  $3,4 \times 10^{38}$ .

Gli indirizzi IPv6 sono scritti con 8 gruppi di 4 cifre esadecimali, separati dal carattere due punti, come in questo esempio:

8000:0000:0000:0000:0123:4567:89AB:CDEF

La lunghezza dell'indirizzo IPv6 è quindi di 128 bit, perché ogni cifra esadecimale può essere rappresentata con 4 bit.



I protocolli TCP/IP forniscono il supporto diretto agli applicativi, offrendo inoltre agli utenti della rete la possibilità di utilizzare contemporaneamente più applicazioni Internet.

Al livello di applicazione si possono trovare i seguenti protocolli:

- **HTTP** (*HyperText Transfer Protocol*), per la trasmissione di informazioni ipertestuali sotto forma di *pagine Web*.
- **HTTPS**, il protocollo HTTP sicuro, per gestire le transazioni commerciali in rete oppure la trasmissione di dati riservati o protetti.
- **FTP** (*File Transfer Protocol*), per la trasmissione di file tra due sistemi con operazioni di *upload* e *download*.
- **SMTP** (*Simple Mail Transfer Protocol*), per il trasferimento di posta elettronica.
- **SSH** (*Secure Shell*), per accedere a un computer remoto come utente di quel sistema, pur essendo magari a migliaia di chilometri di distanza e garantendo la sicurezza tramite trasmissioni criptate.

#### AUTOVERIFICA

Domande da 1 a 9 pag. 526-527

Problemi da 1 a 4 pag. 529

## 6 Internet e Web

**Internet**, semplicemente indicata anche come “**la Rete**”, è un sistema di reti di computer distribuito su tutto il pianeta, una rete di reti nella quale gli utenti possono, avendo i permessi di accesso, ottenere le informazioni da altri computer e colloquiare direttamente con gli utenti collegati con altri computer.

Per il collegamento tra i computer sparsi nel mondo, Internet utilizza le risorse delle reti telefoniche pubbliche delle diverse Nazioni. La tecnologia di rete si basa poi sui protocolli standard **TCP/IP**.

L'aspetto di Internet più importante è sicuramente il **World Wide Web** (letteralmente “ragnatela estesa sul mondo”, abbreviato con **WWW** e indicato brevemente con **Web**): è una tecnologia prodotta alla fine degli anni '80 presso il CERN (*Centro Europeo per le Ricerche Nucleari*) di Ginevra, con lo scopo originario di fornire uno strumento di lavoro per gli ambienti di ricerca di tutto il mondo. Dal 1994 l'utilizzo del World Wide Web si è esteso al grande pubblico aumentando velocemente in pochi anni il numero di utenti.

Il WWW consiste nell'organizzazione delle informazioni in modo **ipertestuale**, con la possibilità di passare da un documento ad un altro che si trova anche su un computer diverso e lontano geograficamente. Il collegamento ipertestuale (**link**) viene rappresentato con parole o immagini. L'insieme delle informazioni, organizzate come un ipertesto e residenti su un computer, forma un **sito Web**: il computer diventa un **host** (o *server Web*) al quale gli utenti di altri computer (*client*) possono collegarsi per consultare le informazioni. L'architettura è quindi di tipo *client/server*. Il software che gestisce un host, consentendo l'accesso degli utenti e la visualizzazione delle pagine Web, si chiama **Web server** (per esempio, *IIS* di Microsoft oppure *Apache*).

Il protocollo standard utilizzato per la trasmissione delle informazioni si chiama **HTTP** (*HyperText Transfer Protocol*).

Le risorse residenti sui vari computer collegati alla rete vengono identificati con un indirizzo Internet detto **URL** (*Uniform Resource Locator*, localizzatore universale di risorse), che deve essere esplicitamente specificato dall'utente che vuole accedere ad un sito Internet. Inoltre l'accesso ipertestuale ai documenti e alle risorse disponibili viene realizzato associando un URL ai link costituiti da parole, icone o immagini.

L'indirizzo URL è formato in genere dal nome del protocollo seguito dal nome dell'organizzazione che gestisce il sito e termina con una sigla che identifica la Nazione o il tipo di organizzazione.

Per esempio:

<http://www.nasa.gov/>

<http://www.istruzione.it/>

|                                 |                                                                 |
|---------------------------------|-----------------------------------------------------------------|
| <i>http</i>                     | indica che il server utilizza il protocollo HTTP                |
| <i>www</i>                      | indica che il sito si trova sul World Wide Web                  |
| <i>nasa</i> o <i>istruzione</i> | indicano il nome dell'azienda od ente che gestisce il sito      |
| <i>gov</i>                      | indica che si tratta di un'organizzazione governativa americana |
| <i>it</i>                       | indica che è un sito italiano.                                  |

La barra / (*slash*) è un separatore che consente di costruire il percorso per arrivare al sito e agli archivi registrati sul disco del server.

L'attività con la quale ci si muove da un server all'altro sulla rete Internet, attraverso i link ipertestuali o indicando specifici URL, si indica con il termine **navigazione Web** (indicata spesso anche con il termine inglese *netsurfing*).

Il sito è formato da pagine che possono essere visualizzate sul video dei computer degli utenti attraverso appositi programmi detti **browser** (sfogliatori): la pagina iniziale del sito, che presenta l'azienda o l'organizzazione, insieme all'indice dei contenuti, si chiama **Home Page**.

Le pagine Web sono testi contenenti al loro interno caratteri di controllo o pezzi di codice che vengono interpretati dal browser per la corretta visualizzazione in forma grafica sul computer dell'utente: il linguaggio per la formazione di pagine Web si chiama **HTML** (*HyperText Markup Language*, cioè un linguaggio che usa contrassegni di controllo nel testo per costruire ipertesti).

Il browser può essere definito come il programma che visualizza i documenti ipertestuali in modalità grafica. I salti da un documento all'altro vengono attivati attraverso i **link**: essi possono essere parole sottolineate e in colore diverso dal resto del documento, oppure icone o immagini. Esempi di programmi browser sono:



## IL SISTEMA DEI NOMI DI DOMINIO

In Internet sono definiti domini e nomi di host all'interno di domini. Il **dominio** rappresenta una rete, solitamente locale o estesa a un raggio di qualche decina di chilometri, collegata a Internet, mentre l'**host** individua un particolare computer all'interno della rete stessa. In sostanza gli indirizzi sono del tipo *host.dominio*, separati dal punto.

Il **DNS** (*Domain Name System*) è il sistema che traduce i nomi dei domini Internet in indirizzi IP e viceversa. Un server DNS è un computer nella rete Internet che può essere interrogato da altri computer per ottenere l'indirizzo IP di un sito internet, permettendone la connessione. Comunemente, i server DNS sono presenti sui computer dei **Provider** della rete Internet.

Per esempio, scrivendo nella casella dell'indirizzo in un browser:

<http://www.museoscienza.org>

l'indirizzamento verso il sito del *Museo della scienza e della tecnica* avviene automaticamente con l'indirizzo IP numerico corretto: questo viene infatti fornito al computer dal server DNS che viene interrogato prima di stabilire la connessione con il server che ospita il sito del museo.

Tutto questo avviene in maniera trasparente all'utente che sta navigando in Internet.

Il DNS è organizzato in modo gerarchico: nell'esempio precedente *org* si chiama dominio di alto, o primo, livello (**TLD**, *top-level domain*), *museoscienza* è un dominio di secondo livello (*second-level domain*) all'interno di *org*.

I suffissi che rappresentano domini di alto livello presenti sulla rete Internet sono:

**com** per indicare le aziende  
**org** per le organizzazioni o le aziende non-profit  
**net** per gli enti che si occupano delle tecnologie e degli standard della rete Internet  
**edu** per le scuole e le università americane  
**mil** per gli enti militari americani  
**gov** per le istituzioni governative americane.

A questi si aggiungono: .eu (Unione europea), .aero, .biz, .coop, .info, .jobs, .mobi, .museum, .name, .pro, .travel.

Tutte le nazioni al di fuori degli Stati Uniti dispongono di domini con due lettere denominati *country code top-level domain* (**ccTLD**), per esempio:

**it** per l'Italia  
**fr** per la Francia  
**de** per la Germania.

Bisogna notare anche che tra *www* e il suffisso finale ci può essere più di una parola, per esempio: *www.provincia.milano.it*. In questo caso *provincia* rappresenta un dominio di terzo livello (*third-level domain*).

L'organizzazione internazionale che è responsabile dell'amministrazione e del coordinamento del DNS è l'**ICANN** (*Internet Corporation for Assigned Names and Numbers*, [www.icann.org](http://www.icann.org)). I nomi di dominio possono essere registrati attraverso diverse società accreditate dall'ICANN, presenti in ogni nazione, denominate **Registrars** (registratori). La registrazione di un nome di dominio può essere fatta da aziende, enti pubblici, scuole, ma anche da singole persone. Il costo per la registrazione può variare a seconda del tipo di dominio di primo livello (.com, .it, .info, ecc.) ed è comunque molto limitato, dell'ordine di qualche decina di euro all'anno. In genere comprende anche uno spazio sul server Web per la pubblicazione delle pagine del proprio sito.



#### MATERIALI ONLINE

### 1. Nomi di dominio

## 7 Le applicazioni di Internet e del Web

### Posta elettronica

La posta elettronica, indicata con **e-mail** (*electronic mail*), è senza dubbio il servizio più usato e che ha determinato la grande diffusione della rete Internet.

Il servizio di posta elettronica consiste nella possibilità di scambiare messaggi di natura elettronica tra due utenti di Internet i quali devono possedere entrambi una casella postale (*mailbox*) su un server di e-mail, cioè sul computer di un Internet Provider che gestisce il servizio di posta elettronica. L'invio di un messaggio di posta elettronica avviene in questo modo:

- un utente invia un messaggio in partenza al proprio server specificando l'indirizzo di e-mail del destinatario;
- il server lo spedisce, attraverso la rete Internet, al computer dove risiede la mailbox del destinatario;
- il server del destinatario lo memorizza nella casella postale;
- il destinatario si connette alla rete Internet tramite il proprio server, controlla la casella postale e scarica il nuovo messaggio in arrivo.



Ogni utente collegato a Internet ha a disposizione una propria casella di posta elettronica. Questa non è altro che un indirizzo Internet che identifica un dominio a cui è associato un particolare nome di utente (*username*) secondo un formato predefinito:

username@dominio

significa che l'utente *username* ha una casella postale presso il *dominio*, per esempio:

*bianchi@tin.it*

Per l'invio e il ricevimento dei messaggi si possono usare programmi di posta elettronica installati sul proprio computer, per esempio: *Outlook*, *Windows Mail*, *Eudora*, *Mozilla Thunderbird*, oppure si può accedere alla posta tramite il browser (**Web mail**) collegandosi al sito del Provider che fornisce la casella di posta elettronica.

Il testo del messaggio può essere trattato con evidenziazioni in neretto o corsivo, formattazione dei paragrafi, controllo ortografico. Si può creare una rubrica di indirizzi, inserendoli eventualmente in gruppi di persone alle quali normalmente dobbiamo inviare lo stesso messaggio. Inoltre nel messaggio si può inserire un file come allegato (*attachment*).

## Motore di ricerca

Un **motore di ricerca** (in inglese, *search engine*) è un sito Web, progettato al fine di fornire all'utente un elenco di siti che riguardano un particolare argomento. La lingua usata è solitamente l'inglese anche se esistono siti di tutte le Nazioni.

In Internet sono disponibili molti motori di ricerca; i più famosi e più usati sono:

*Google* (<http://www.google.it>), *Yahoo* (<http://www.yahoo.it>), *Bing* (<http://www.bing.it>).

Per collegarsi al sito contenente il motore scelto, basta digitare l'indirizzo completo nella casella *Indirizzo* del browser; all'utente viene presentata la *Home page* del sito.

L'utente deve inserire la parola, o le parole, che ritiene più significative per trovare le pagine o i documenti Web che trattano un determinato argomento; il motore di ricerca restituisce il numero di documenti trovati e un elenco riassuntivo dei titoli dei documenti. I documenti vengono elencati di solito dieci per pagina in ordine decrescente di importanza rispetto all'argomento richiesto.

Il titolo di ciascun documento è normalmente riportato in colore perché in effetti è un link al documento; premendo infatti sulla riga del titolo l'utente viene connesso direttamente con il sito Internet che possiede il documento prescelto.

## Commercio elettronico

Il commercio elettronico, in inglese **e-commerce** (*electronic commerce*), consiste nella vendita e nell'acquisto di prodotti e servizi sulla rete Internet utilizzando la tecnologia del World Wide Web.

Sia le aziende grandi che quelle piccole possono trarre grandi vantaggi dalla creazione di un sito Internet orientato al commercio elettronico, in quanto l'attività commerciale può raggiungere un vasto pubblico e interagire con i clienti. Inoltre il sito Internet non è soltanto una semplice presenza sulla rete, ma lo strumento per effettuare transazioni commerciali on line. Le aziende possono acquisire dati importanti per le ricerche di mercato attraverso la registrazione degli utenti che visitano il sito, la compilazione di questionari in linea o la raccolta di ordini.

Naturalmente la sicurezza delle transazioni commerciali è un problema di grande importanza: è essenziale introdurre meccanismi di autenticazione degli operatori commerciali che effettuano le transazioni, controllare gli accessi alle pagine Web per utenti selezionati e registrati, criptare le comunicazioni che viaggiano sulle linee della rete, assicurare la riservatezza e la consistenza delle transazioni.

## Home banking

Le banche e i promotori finanziari utilizzano la rete Internet non soltanto come vetrina delle attività aziendali, ma anche per fornire servizi agli utenti aziendali e privati: queste applicazioni sono indicate con **home banking** (oppure *e-banking*) e **servizi finanziari on line**. I servizi offerti ai clienti si dividono in due categorie:

- la gestione di un conto corrente, con tutte le possibilità offerte al correntista quali: visione del saldo, dei movimenti, delle condizioni di contratto, esecuzione di bonifici bancari, ecc.
- la gestione diretta di un portafoglio titoli, con la possibilità di avere le quotazioni azionarie in tempo reale e di acquistare e vendere titoli, con varie modalità operative; è possibile quindi fissare il prezzo di acquisto o di vendita e verificare che le operazioni siano svolte e che il portafoglio titoli registri le operazioni andate a buon fine.



## e-government

In generale l'uso delle tecnologie informatiche e della rete Internet per gli Enti governativi viene indicato con il termine **e-government**: esso non riguarda solo l'approvvigionamento di prodotti o servizi, ma anche lo scambio di informazioni, il miglioramento dei rapporti normativi e fiscali tra cittadini e Stato e tra aziende e Stato, la semplificazione della macchina amministrativa. Le applicazioni informatiche su larga scala riguardano i sistemi di memorizzazione e di elaborazione dei grandi archivi (anagrafi fiscali, censimenti, pubblico registro dei veicoli) oppure l'automazione nella gestione del fisco e delle dichiarazioni dei redditi.

## Scuola ed e-learning

L'uso delle reti nella scuola riguarda la parte gestionale e organizzativa degli Istituti scolastici:

- pubblicazione di notizie e informazioni sugli Istituti scolastici;
- comunicazione scuola-famiglia, con accesso riservato ai dati degli studenti da parte della famiglie (circolari, modulistica, assenze, voti, risultati finali).

Non meno importanti sono i metodi di insegnamento e i processi di apprendimento di studenti, adulti, dipendenti di aziende (**e-learning**):

- apprendimento assistito al computer;
- didattica attiva e collaborativa basata sull'infrastruttura delle reti scolastiche e della rete Internet;
- corsi di formazione e lauree universitarie a distanza.

## VoIP

**Voice over IP** (Voce sul protocollo standard di Internet, IP, *Internet Protocol*) indica la tecnologia che consente di telefonare usando la rete Internet, anziché la linea telefonica tradizionale. È possibile telefonare, cioè scambiare messaggi vocali in modalità sincrona, utilizzando un programma e la rete Internet. Il servizio *VoIP* converte il segnale analogico, catturato dal microfono del computer (la voce dell'utente), in un segnale digitale che può essere inviato sulla rete Internet utilizzando il protocollo IP. Grazie ad altri strumenti software, non solo si può parlare direttamente con tutti gli utenti della rete che hanno installato un software compatibile, ma anche chiamare i telefoni fissi e i cellulari, senza che il ricevente possa capire che la chiamata è effettuata via Internet, a costi contenuti e comunque più bassi rispetto alle reti telefoniche tradizionali.

## RSS

**RSS** (*Really Simple Syndication*) indica un formato standard per la rappresentazione e la distribuzione di notizie e contenuti nel Web. Un flusso RSS viene detto *feed* e i programmi che sono in grado di visualizzare un RSS sono detti **feed reader** (lettori di feed) o **aggregatori** di notizie. I siti Web che contengono RSS consentono agli utenti di rimanere aggiornati sulle ultime notizie pubblicate.

Per ricevere gli aggiornamenti in modo automatico un utente Internet deve “abbonarsi”, in modo del tutto gratuito, al feed di un sito, utilizzando il programma di *feed reader*.

I browser contengono al loro interno la funzionalità di gestione dei feed RSS. In ogni caso il feed RSS di un sito può essere visualizzato facendo clic sull'icona standard che lo rappresenta e che è presente nelle pagine dei siti che offrono il servizio RSS. Nei browser, la presenza di un feed RSS in un sito è segnalata dalla presenza dell'icona nella barra dell'indirizzo.



## Podcast

Un **podcast** è un file, normalmente musicale o video, messo a disposizione degli utenti Internet che possono scaricarlo liberamente e in modo automatico attraverso un programma di *podcasting* (per esempio *iTunes* o *Juice*) simile ai *feed reader* visti in precedenza e disponibile gratuitamente in Internet. Anche in questo caso infatti l'utente di un sito Web può “abbonarsi” gratuitamente in modo da ricevere direttamente sul proprio computer i motivi musicali e, in generale, i file audio e video disponibili. I file scaricati rimangono registrati sul disco del computer e possono essere ascoltati in tempi successivi oppure memorizzati su lettori portatili di file *mp3* o *mp4*, oppure in un telefono cellulare. Il termine *podcasting* deriva dal nome del più famoso di questi dispositivi portatili, *iPod* di Apple.

## Social networking e Web 2.0

La rete Internet offre la possibilità di utilizzare applicazioni per lavori e attività di ogni giorno, nelle professioni, nello studio o nel divertimento. L'utente non legge solo passivamente le pagine Web, ma può interagire e collaborare in modo attivo con altri utenti, condividere documenti, pubblicare propri file multimediali. Le nuove modalità di utilizzo della rete Internet sono indicate con il termine **Web 2.0**, volendo indicare con questo termine l'evoluzione della rete. La sfera sociale e comunicativa viene notevolmente ampliata: è possibile infatti conoscere e restare facilmente in contatto con persone residenti in tutto il mondo, attraverso **comunità virtuali** di rete:

- siti di reti sociali (in inglese, **social network**), per esempio *Facebook* o *Twitter*
- blog
- forum
- messaggistica istantanea (in inglese **IM**, *Instant Messaging*), per esempio *Messenger*
- chat
- giochi informatici in rete
- pubblicazione di fotografie o di clip video, per esempio *Flickr* o *YouTube*.



### MATERIALE ONLINE

#### 2. Ricerche avanzate in Internet

#### 3. La Netiquette in Internet

#### 4. Telefonare con il VoIP

#### 5. Impostare la privacy di Facebook

## 8 La sicurezza delle reti

Alcuni server Internet, contenenti informazioni riservate o che prevedono l'esecuzione di operazioni bancarie o commerciali, sono protetti dagli accessi non autorizzati. In questi casi l'accesso al sito è regolato da un **nome utente** e da una **password** che devono essere inseriti correttamente per poter visualizzare i contenuti del sito. Inoltre il server Web utilizza il protocollo **HTTPS** per la visualizzazione delle informazioni e la gestione dell'interazione con gli utenti. Poiché la rete Internet è diventata l'infrastruttura per effettuare transazioni commerciali e trasferimenti di documenti riservati, sono stati adottati dispositivi e tecniche per garantire la sicurezza delle operazioni.

Il certificato digitale o **firma digitale** è un codice assegnato ad una persona che vuole fare transazioni sulla rete, in modo che ne venga riconosciuta in modo inequivocabile l'identità. La firma digitale è garantita dagli **Enti di certificazione** (*Certification Authority*) internazionali.

La **crittografia** è la tecnica con cui, mediante una chiave di interpretazione nota solo ai due interlocutori, si può rendere illeggibile il messaggio tranne che agli interlocutori stessi. Essa consente di proteggere le informazioni scambiate sulla rete, evitando anche che vengano intercettate durante la trasmissione sulle linee di comunicazione. Le firme digitali sono basate sulle tecniche di crittografia.

Normalmente le operazioni commerciali effettuate sulla rete sono garantite dalle aziende venditrici che utilizzano banche di appoggio per assicurare la correttezza delle transazioni e la riservatezza delle informazioni. Esiste comunque la possibilità di usi fraudolenti dei numeri delle carte di credito utilizzate nelle operazioni in rete: è buona norma quindi porre attenzione alle caratteristiche di affidabilità dei siti che offrono il commercio elettronico.

Un computer o una rete aziendale connessa a Internet devono essere protetti dall'accesso di utenti non autorizzati. Per garantire la sicurezza degli archivi aziendali si usano appositi computer (**firewall**) che fanno da filtro per i messaggi che escono ed entrano nella rete aziendale.

Durante la navigazione in Internet o la consultazione della posta elettronica contenente allegati, può accadere che il proprio computer venga infettato da un programma contenente un **virus**. È quindi opportuno mantenere in funzione il programma antivirus, usare cautela nello scaricare file da siti Internet non affidabili, non aprire messaggi e allegati provenienti da mittenti sconosciuti.

Tramite la posta elettronica vengono diffusi messaggi ingannevoli che chiedono all'utente di collegarsi a un falso sito di una banca o delle Poste per inserire il proprio numero conto e il codice segreto (questa forma di truffa è indicata con il termine **phishing**): la regola di comportamento è non fornire mai informazioni personali e riservate e, in ogni caso, si deve sempre accedere tramite l'indirizzo Internet corretto della banca o delle Poste e nel dubbio telefonare alla propria agenzia.

Nell'accesso alle **comunità virtuali** si deve ricordare che esse si basano sulla comunicazione libera con altre persone e sulla disponibilità di quanto pubblicato per tutti gli utenti della rete: è quindi importante adottare le necessarie precauzioni contro i pericoli che la rete Internet nasconde, soprattutto per i minori.

È quindi opportuno:

- rendere privato il proprio profilo,
- limitare la quantità di informazioni personali fornite evitando la pubblicazione di indirizzo e numero di telefono,
- essere consapevoli del fatto che le informazioni fornite sono disponibili a tutti,
- fare attenzione agli sconosciuti,
- essere consapevoli che l'identità delle persone con cui si è in contatto non è necessariamente quella dichiarata, in termini di età o di genere.

## Norme di legge e contenuti Web

Le norme sul **diritto di autore** (*copyright*) si applicano anche ai materiali disponibili nella rete Internet. In questo caso le violazioni del diritto d'autore riguardano:

- copia non autorizzata di pagine Web;
- duplicazione abusiva di software prelevato dalla rete;
- scambio attraverso la rete di file musicali o multimediali protetti da *copyright*.

In particolare è diventata di grande rilevanza la normativa sui *file musicali*, anche perché implica notevoli interessi economici.

In Italia le opere musicali sono protette dal diritto d'autore. La SIAE (*Società Italiana degli Autori ed Editori*) è l'organo istituzionale che tutela il diritto d'autore. La SIAE prevede diverse licenze che consentono l'uso via Web di brani musicali:

- *Streaming on demand*: il titolare della licenza consente di selezionare contenuti permettendo all'utente il solo ascolto.
- *Download*: il titolare può consentire di scaricare file audio; il file diventa di possesso dell'utente come se avesse acquistato un CD musicale.
- *Webcasting*: in questo caso l'utente può ascoltare i brani, come avviene con le normali trasmissioni radiofoniche.

La normativa in materia di diritto d'autore stabilisce che è consentita la riproduzione privata di musica e video su supporti, effettuata da una persona per uso esclusivamente personale, escludendo quindi la diffusione, vendita o distribuzione.

Alcuni siti Internet mettono a disposizione degli utenti le loro librerie di file e di programmi software. La maggior parte dei programmi esistenti su Internet sono **freeware** (distribuibili liberamente) e per essi non è previsto alcun costo economico per il loro uso.

Altri sono definiti **shareware** (distribuibili per essere condivisi), ed esistono due modalità di distribuzione. Una prima modalità prevede che gli autori concedano l'uso gratuito del loro programma, per un tempo di prova (di solito uno o due mesi). Alla scadenza del periodo di prova, l'utente che desidera continuare ad utilizzare il software deve registrarsi come utente del software stesso, pagando all'autore una piccola somma, prevista dal contratto d'uso scritto dall'autore stesso. Una seconda modalità prevede l'utilizzo di una versione del programma, priva di alcune funzioni di base (tipicamente la possibilità di salvare il file, di stampare o di copiare i dati dall'applicazione ad altre).

In ogni caso i programmi *shareware* e *freeware* non includono il *codice sorgente* e pertanto non sono modificabili.

La **licenza d'uso EULA** (*End-User License Agreement*, o contratto con l'utente finale) è il permesso per l'uso di un prodotto software che l'utente ottiene con l'acquisto. I prodotti software sono contrassegnati da un **numero di serie** o da un identificativo del prodotto (**Product ID**), che è stampato sulla copertina del CD o del DVD e che viene richiesto durante l'installazione sul computer.

Il **software libero**, indicato anche con **Open Source** (letteralmente, sorgente aperto), indica il software che può essere liberamente eseguito, copiato, distribuito e modificato o migliorato: perciò di questo software è disponibile anche il codice sorgente.

Il software *Open Source* è regolato dalla licenza **GNU GPL** (*Gnu General Public License*), che definisce le libertà e le condizioni di utilizzo dei programmi. Il software libero è frutto di un intenso lavoro operato, anche a titolo gratuito, da migliaia di programmatori in tutto il mondo i quali, mettendo a disposizione le proprie specifiche competenze, oltre che idee e tempo, contribuiscono alla diffusione e alla crescita di programmi affidabili e ricchi di funzionalità.



### MATERIALI ONLINE

## 6. La licenza GNU GPL



### AUTOVERIFICA

Domande da 10 a 21 pag. 527-528  
Problemi da 5 a 14 pag. 529

## DOMANDE

### Le reti

- 1 Che cosa significa connettere dei computer in rete?
  - a) Consentire la condivisione di file e risorse tra gli utenti
  - b) Eliminare la necessità della posta elettronica
  - c) Rendere i computer più affidabili
  - d) Aumentare la sicurezza nell'uso dei computer
  
- 2 Quale tra le seguenti è la caratteristica più importante delle reti?
  - a) Costruzione di reti telefoniche
  - b) Lavoro cooperativo tra utenti diversi
  - c) Utilizzo della posta elettronica
  - d) Produzione di software
  
- 3 Quale delle seguenti unità di misura è utilizzata per la velocità di trasmissione?
  - a) Mbyte
  - b) Mbps
  - c) Kbyte
  - d) GHz
  
- 4 Quali di queste affermazioni sono vere (V) e quali false (F)?
 

|                                                                                                                           |            |
|---------------------------------------------------------------------------------------------------------------------------|------------|
| a) LAN indica una rete di dimensioni metropolitane                                                                        | <b>V F</b> |
| b) WAN indica una rete nella quale la connessione tra computer copre una zona corrispondente al territorio di una Nazione | <b>V F</b> |
| c) In una architettura client/server i ruoli di server e client sono predefiniti                                          | <b>V F</b> |
| d) Il nome Internet deriva da Internetworking che significa rete di reti interconnesse                                    | <b>V F</b> |
  
- 5 Quale delle seguenti frasi rappresenta la definizione di Intranet?
  - a) La connessione tra reti aziendali.
  - b) La rete interna aziendale che usa le modalità e i protocolli tipici di Internet.
  - c) L'uso di parti della rete aziendale per utenti selezionati esterni all'azienda.
  - d) L'estensione di una rete da locale a geografica.
  
- 6 Quali delle seguenti affermazioni esprimono i vantaggi nell'uso di una linea ADSL quando si lavora con Internet?
  - a) Disporre della banda larga per accedere velocemente ad Internet.
  - b) Possibilità di utilizzare ipertesti nella rete.
  - c) Disponibilità della connessione a Internet 24 ore su 24, avendo contemporaneamente libera la linea telefonica.
  - d) Connessione diretta alla rete senza utilizzare modem o altre apparecchiature di collegamento.
  
- 7 Associa ciascun termine della colonna di sinistra al relativo significato nella colonna di destra.
 

|             |                                                                                                         |
|-------------|---------------------------------------------------------------------------------------------------------|
| a) Intranet | 1) Rete mondiale che collega molte reti di computer                                                     |
| b) Internet | 2) Tipologia di rete che collega computer presenti all'interno di un ufficio                            |
| c) LAN      | 3) Tipologia di rete che collega i computer di un'azienda, collocati in uffici geograficamente distanti |
| d) WAN      | 4) Rete aziendale di computer, riservata ai dipendenti                                                  |

- 8 Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
- a) Un indirizzo Internet è formato da quattro numeri che possono assumere solo valori compresi tra due limiti V F
  - b) Il protocollo TCP/IP è uno standard delle reti V F
  - c) Le reti Intranet sono le reti che usano protocolli diversi da quelli della rete Internet V F
  - d) È possibile la connessione a Internet anche attraverso i telefoni cellulari V F
- 9 Associa alle sigle della colonna di sinistra il corrispondente significato della colonna di destra
- |           |                                                              |
|-----------|--------------------------------------------------------------|
| a) FTP    | 1) Protocollo per la visualizzazione di pagine Web           |
| b) TCP/IP | 2) Provider di servizi Internet                              |
| c) SMTP   | 3) Programma per il trasferimento di file                    |
| d) SSH    | 4) Protocollo per la posta elettronica                       |
| e) HTTP   | 5) Protocolli di base di Internet                            |
| f) ISP    | 6) Protocollo per la connessione sicura a un computer remoto |

## Internet e applicazioni Web

- 10 Quali delle seguenti frasi riferite al WWW (World Wide Web) NON sono corrette?
- a) Insieme di informazioni disponibili in rete e visualizzabili attraverso il browser.
  - b) Insieme dei computer collegati attraverso le reti e dei programmi residenti su di essi.
  - c) Insieme dei programmi grafici e dei protocolli utilizzati da Internet.
  - d) Insieme di documenti multimediali collegati tra loro e consultabili in Internet.
- 11 Quale delle seguenti definizioni corrisponde a WWW?
- a) Il servizio di posta elettronica che utilizza Internet come infrastruttura di comunicazione.
  - b) L'insieme delle apparecchiature hardware per mettere in comunicazione più computer.
  - c) L'organizzazione ipertestuale dei documenti, che possono risiedere anche su computer diversi e geograficamente lontani.
  - d) L'elenco degli indirizzi Internet appartenenti ad una Nazione.
- 12 Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
- a) Internet si basa su un'architettura di tipo client/server V F
  - b) Il protocollo di base di Internet si chiama FTP V F
  - c) Due reti possono essere connesse tra loro solo se sono dello stesso tipo V F
  - d) WWW è l'acronimo di World Wide Web V F
- 13 Completa le frasi seguenti utilizzando una tra le parole elencate alla fine della domanda.
- a) Le pagine Web possono essere visualizzate con un ..... sul video dei computer degli utenti.
  - b) In un sito Internet i salti da un documento all'altro vengono attivati attraverso un .....
  - c) La pagina iniziale di un sito Internet si chiama .....
  - d) L'indirizzo di un sito Internet si chiama ..... perché serve a identificare le risorse residenti sui vari computer collegati alla rete.
  - e) Il protocollo standard ..... è utilizzato per la trasmissione delle informazioni sul Web.
- Accesso remoto, email, browser, WWW, ipertesto, HTTP, Word, Home Page, URL, link, chat, username**



- 14 Quale tra i seguenti indirizzi per un sito Internet è scritto correttamente secondo la sintassi URL?
- http://www.meteo.it
  - www.meteo.it/http
  - http://www.meteo.it
  - www://http.meteo.it
- 15 Quale tra i seguenti indirizzi di posta elettronica è scritto correttamente, supponendo che il nome dell'utente sia *franz*, il provider Internet sia *infonet* e la nazione sia la Germania?
- franz@de.infonet
  - infonet.de@franz
  - franz.de@infonet
  - franz@infonet.de
- 16 Quale delle seguenti definizioni corrisponde a motore di ricerca?
- Un programma per cercare file in formato Web sul proprio computer.
  - Un programma per cercare gli indirizzi dei siti che non sono più attivi.
  - Un sito Internet che restituisce i numeri di telefono dei provider Internet.
  - Un sito Internet che restituisce un elenco di siti riguardanti un particolare argomento.
- 17 Quale delle seguenti frasi rappresenta meglio la definizione di e-commerce?
- La creazione di un sito Web per un'azienda.
  - La pubblicazione di cataloghi di prodotti su Internet.
  - La vendita di beni e servizi attraverso Internet e le tecnologie Web.
  - La distribuzione di software a pagamento.
- 18 Completa le frasi seguenti utilizzando una tra le parole elencate alla fine della domanda.
- I flussi ..... consentono di ottenere informazioni aggiornate da un sito Web.
  - Il termine ..... indica l'uso di Internet per la comunicazione telefonica.
  - Social networking, forum, chat e giochi in rete sono esempi di .....
  - La didattica in rete si chiama .....
- VoIP, e-learning, comunità virtuali, sincrona, asincrona, blog, RSS, username, browser, password, link, e-mail**
- 19 Quali tra le seguenti sono precauzioni importanti da prendere durante l'accesso alle comunità virtuali?
- Fornire molti dati personali.
  - Rendere privato il proprio profilo.
  - Essere consapevoli che le informazioni fornite sono disponibili a tutti.
  - Interagire anche con sconosciuti.
- 20 Quale delle seguenti modalità di distribuzione del software corrisponde al termine *freeware*?
- L'uso gratuito del software, per un tempo limitato di prova.
  - L'utilizzo di una versione del programma, priva di alcune funzioni di base.
  - L'uso di un programma distribuibile liberamente.
  - L'utilizzo del software con regolare licenza d'uso.
- 21 Quali delle seguenti affermazioni relative al *copyright* sono corrette?
- Esistono diverse licenze che consentono l'uso via Web di brani musicali.
  - La legge sul copyright è applicabile anche a fotografie e audio depositati su Internet.
  - È consentita la vendita di musica e video solo se effettuata da un privato.
  - Lo scambio attraverso la rete di file musicali o multimediali protetti da *copyright* non viola il diritto d'autore.



## PROBLEMI

### Le reti

- 1 Descrivere la tipologia di connessione utilizzata per l'accesso a Internet (rete aziendale o scolastica, linea telefonica di casa, linea a banda larga).
- 2 Descrivere con una breve relazione le caratteristiche della rete LAN utilizzata nella propria scuola e i dispositivi di rete utilizzati.
- 3 Utilizzando le informazioni contenute nei siti Internet dei Provider di servizi telefonici, costruire una tabella comparativa delle diverse offerte di accesso a Internet tramite linea ADSL in termini di velocità di connessione e di costi.
- 4 Determinare l'indirizzo IP che si sta usando per la connessione ad Internet utilizzando il Pannello di controllo di Windows.

### Internet e applicazioni Web

- 5 Aprire una pagina Web di un sito Internet e salvarla su disco con l'opzione Salva con nome... del menu File nel browser.
- 6 Aprire una pagina Web di un sito Internet e salvare su disco una delle immagini in essa contenute (clic con il tasto destro del mouse, nel menu di scelta rapida scegliere Salva immagine con nome...)
- 7 Installare un secondo programma browser sul proprio computer (*Internet Explorer, Mozilla Firefox, Safari, Google Chrome, Opera*).
- 8 Utilizzare un motore di ricerca per trovare le pagine che contengono riferimenti a due parole chiave prefissate.
- 9 Cercare su Internet siti Web che contengono corsi on line sull'uso di Internet e dei suoi servizi, utilizzando un motore di ricerca.
- 10 Cercare in Internet mediante un motore di ricerca i siti che contengono riferimenti ad una parola prefissata: ripetere l'operazione con diversi motori di ricerca e costruire una tabella comparativa sul numero di pagine che ciascun motore è in grado di trovare per la stessa parola.
- 11 Installare sul proprio computer un programma di posta elettronica (per esempio, *Windows mail, Eudora, Mozilla Thunderbird*).
- 12 Inviare un messaggio di posta elettronica attivando la *Web mail* tramite il browser.
- 13 Specificare l'indirizzo Internet di alcuni siti Web che offrono servizi per la comunicazione in rete, per l'accesso a comunità virtuali e per la pubblicazione di blog, foto o video:

| Sito Web      | Servizi offerti |
|---------------|-----------------|
| http:// ..... |                 |
| http:// ..... |                 |
| http:// ..... |                 |
| http:// ..... |                 |

- 14 Scrivere i nomi di quattro software installati nel proprio computer e specificare per ciascuno il tipo di licenza (shareware, freeware, EULA, software libero):

| Software | Tipo di licenza |
|----------|-----------------|
|          |                 |
|          |                 |
|          |                 |
|          |                 |



## SCHEDA DI AUTOVALUTAZIONE

### CONOSCENZE

- ☐ Le reti
- ☐ Le architetture client/server e peer to peer
- ☐ Reti LAN e WAN
- ☐ Mezzi trasmissivi
- ☐ Dispositivi di rete
- ☐ Reti pubbliche
- ☐ Intranet e Extranet
- ☐ Protocollo TCP/IP
- ☐ Indirizzi IP
- ☐ Internet
- ☐ Sistema dei nomi di dominio DNS
- ☐ Applicazioni di Internet
- ☐ Applicazioni Web
- ☐ Web 2.0 e social networking
- ☐ Aspetti della sicurezza in rete
- ☐ Diritto d'autore nel Web

### ABILITÀ

- ☐ Effettuare una connessione alla rete Internet usando un personal computer
- ☐ Individuare diverse modalità per la connessione a Internet
- ☐ Individuare l'indirizzo IP del proprio computer
- ☐ Identificare una risorsa Internet tramite URL
- ☐ Consultare le pagine dei siti Internet utilizzando un browser
- ☐ Inviare e ricevere messaggi di posta elettronica
- ☐ Allegare un file al messaggio di posta elettronica
- ☐ Utilizzare un motore di ricerca per trovare informazioni sulla rete
- ☐ Accedere a una comunità virtuale
- ☐ Adottare precauzioni e regole per la sicurezza nelle reti
- ☐ Individuare materiali Web protetti da diritto d'autore
- ☐ Distinguere tra diversi tipi di licenze software



**SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 663**

# 10

**parte quarta**

Reti e Web

## **Pagine Web e fogli di stile**

### **OBIETTIVI DI APPRENDIMENTO**

In questo capitolo imparerai a operare con informazioni, documenti e oggetti multimediali in formato Web da pubblicare nei siti Internet.

Sarai anche in grado di utilizzare strumenti e linguaggi per personalizzare il layout e lo stile delle pagine Web.

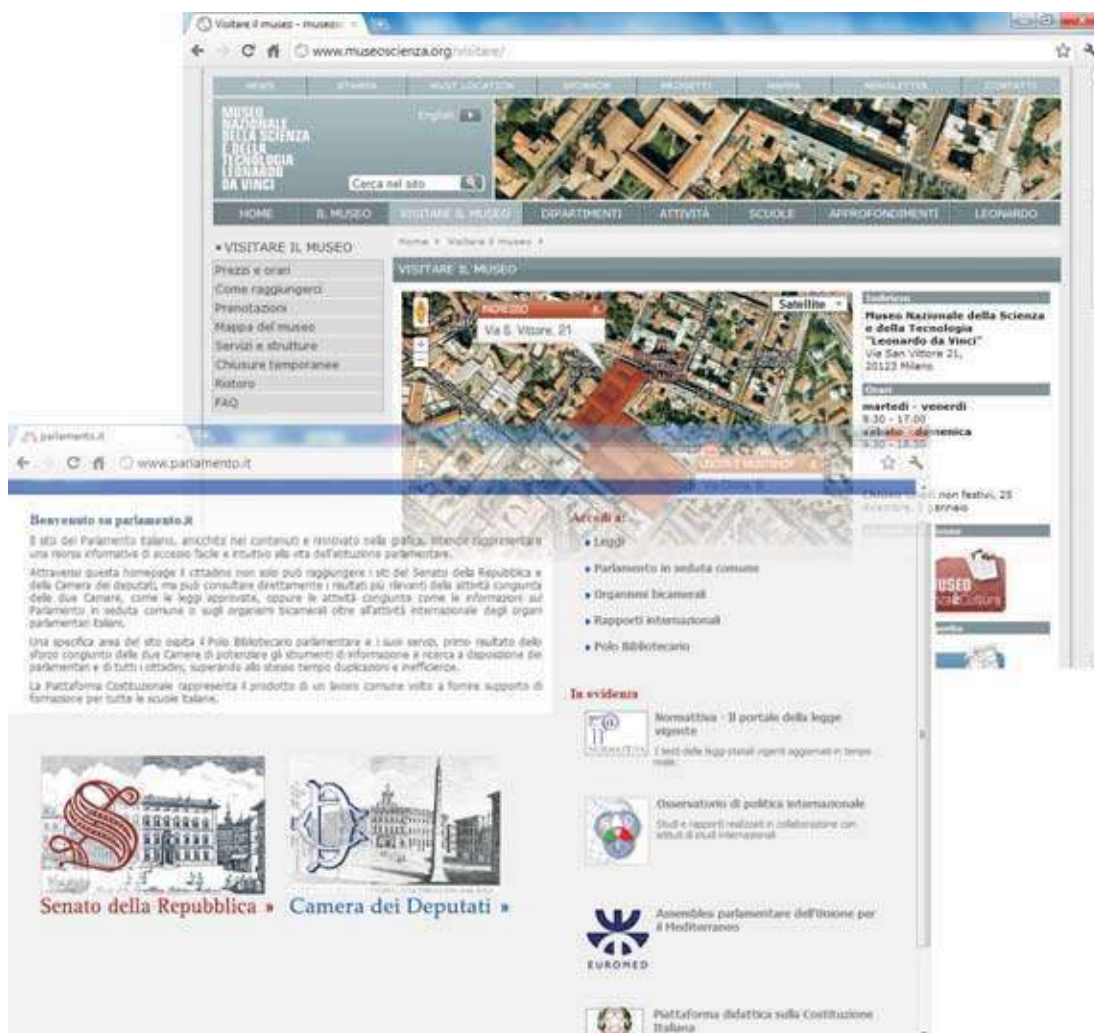
Le pagine per i siti Internet  
Formati multimediali  
Il linguaggio HTML  
La formattazione  
Link  
Liste e tabelle  
Elementi multimediali  
I form e l'interazione con  
l'utente  
Fogli di stile CSS  
Selettori e proprietà  
Classi e identificatori  
Accessibilità e usabilità

# 1 Le pagine per i siti Internet

La rapida espansione della rete Internet e del numero degli utenti ha comportato in pochi anni l'aumento dei siti raggiungibili e della quantità di informazioni disponibili. Moltissime aziende, agenzie di notizie, società di servizi, enti, università, scuole, ma anche singole persone, hanno creato un proprio sito Internet per mettere in condivisione progetti, informazioni o archivi di dati, per presentare prodotti commerciali, attività, servizi, iniziative, corsi o semplicemente interessi personali.

La tecnologia software mette a disposizione gli strumenti per realizzare con facilità pagine in formato Web, cioè pronte per essere caricate su un server Internet e visualizzabili con un browser dagli utenti che si collegano; molte aziende offrono la possibilità di creare il proprio sito Internet gratuitamente, fornendo lo spazio su un server e, in alcuni casi, anche gli strumenti necessari per la creazione delle pagine Web. Nel caso di necessità specifiche un utente o un'azienda possono acquistare uno spazio maggiore per la residenza delle pagine Web su un computer server.

L'insieme delle pagine che si riferiscono a uno stesso utente, a uno stesso argomento o azienda forma un **sito Internet** o **sito Web**.



Un sito può contenere pagine con brevi presentazioni, oppure informazioni, gallerie di immagini oltre a tutti gli altri oggetti multimediali, come suoni o filmati da scaricare (*download*) o poter vedere direttamente in rete (*streaming*).

Esaminiamo in questo primo paragrafo gli strumenti che servono per realizzare semplici pagine in formato Web da pubblicare in un sito Internet.

Per preparare pagine Web basta un qualsiasi personal computer con le caratteristiche minime necessarie a trattare documenti multimediali: processore con velocità superiore a 1GHz, memoria RAM di almeno 1GB, se si vuole usare lo scanner, disco fisso con la capacità di qualche centinaia di Gigabyte.

Le pagine, completate e verificate nel funzionamento, devono essere trasferite sul server dove si trova il sito e spesso devono essere modificate o aggiornate in momenti successivi; quindi è opportuno avere un collegamento a Internet veloce, allo scopo di diminuire i tempi per le operazioni di trasferimento dei file (operazioni di **upload**).



La realizzazione di pagine per Internet con contenuti multimediali richiede anche la disponibilità di apparecchiature **periferiche**, quali il lettore di CD-ROM o DVD, per installare nuovo software o per usare raccolte di immagini o fotografie, lo scanner, la macchina fotografica digitale, la scheda per l'acquisizione di filmati da videoregistratore o da telecamera digitale, oltre naturalmente alla scheda audio per trattare informazioni di tipo sonoro.

Le pagine da pubblicare su Internet sono realizzate usando il **linguaggio HTML** (*HyperText Markup Language*), in modo che possano essere correttamente interpretate dal browser e visualizzate in forma grafica.

Per la costruzione di un sito, è sufficiente utilizzare un programma qualsiasi di videoscrittura, per esempio il programma *Blocco note* o *WordPad*. Esistono però programmi, chiamati **Web editor**, che semplificano il lavoro di scrittura del codice.

I programmi di *Web editor* si suddividono in due categorie:

- **Web editor non visuali:** facilitano la scrittura del codice che però non viene generato automaticamente. Per esempio colorano alcuni termini specifici in maniera differente rispetto al resto del testo, completano automaticamente il codice oppure segnalano gli errori di sintassi.
- **Web editor visuali:** generano automaticamente il codice HTML utilizzando un'interfaccia visuale. Questi programmi vengono indicati con l'acronimo **WYSIWYG** (*What You See Is What You Get*, in italiano *quello che vedi è quello che ottieni*) per indicare che in ogni istante viene visualizzato l'aspetto finale del sito.

In Internet si possono creare siti in maniera gratuita, utilizzando uno spazio Web e una piattaforma in grado di generare siti, in maniera semiautomatica, partendo da *template* (modelli) predefiniti e senza conoscere il linguaggio HTML.

In ogni caso una conoscenza dei codici principali di HTML appare opportuna, per poterne comprendere il significato, per essere in grado di apportare modifiche a pagine create in precedenza e per correggere errori che il software non è in grado di sistemare automaticamente. Per questo motivo nei prossimi paragrafi presenteremo i fondamenti del linguaggio HTML e alcuni semplici esempi di codice.

L'uso di un Web Editor presuppone la disponibilità di un programma **browser** (*Firefox, Chrome, Opera, Safari, Internet Explorer*), per verificare la corretta visualizzazione delle pagine. Inoltre è sicuramente utile un **programma di grafica** per creare o trattare immagini e fotografie. Le pagine devono essere infatti attraenti per il visitatore e devono contenere poco testo e molti elementi grafici: disegni, anche scelti da collezioni già esistenti (**clipart**), animazioni, fotografie, sfondi, bottoni, linee di separazione.

Per il trasferimento (*upload*) dei file contenenti le pagine, dal disco del proprio computer al server Internet dove devono essere pubblicate, si usa un **programma per FTP** (*File Transfer Protocol*). Per effettuare questa operazione l'utente deve possedere il permesso di accesso (**account**), tramite un nome di identificazione e una password, forniti dall'amministratore del server Web.

Le pagine create possono essere ospitate in una sottocartella (*directory*) di un sito già esistente, oppure possono diventare un nuovo sito. In questo secondo caso il sito Internet viene identificato attraverso un indirizzo simbolico del tipo *www.prova.it*.



## I FORMATI DEGLI OGGETTI MULTIMEDIALI

Tutti gli oggetti creati o acquisiti con apparecchiature multimediali possono essere memorizzati su supporti di memoria di massa (dischi fissi, dischi ottici, memorie USB) in vista di successivi utilizzi. Dal punto di vista del sistema operativo essi sono comunque **file** che vengono identificati attraverso un **nome**, assegnato dall'utente al momento della registrazione, e da una **estensione** che ne specifica il formato. Il nome deve essere significativo, nel senso che deve ricordarne il contenuto anche a distanza di tempo: per esempio *GitaAgosto08.jpg* potrebbe essere il nome di un file contenente l'ottava fotografia di una raccolta sulla gita di agosto.

Lo sviluppo delle applicazioni multimediali ha prodotto negli ultimi anni numerosi formati in risposta a diverse necessità:

- trattare documenti di grosse dimensioni
- integrare diversi oggetti multimediali in uno stesso documento
- ottenere immagini e fotografie di alta qualità
- operare con tempi brevi per la loro visualizzazione
- registrare i documenti con minima occupazione di spazio sui supporti.

A questo proposito occorre ricordare il problema della **compressione** dei dati, cioè la riduzione delle dimensioni di un file di dati per:

- risparmiare spazio sui supporti
- ridurre i tempi di trasmissione dei file sulle linee telefoniche.

La compressione viene realizzata attraverso appositi algoritmi che determinano le modalità per la compressione e la successiva decompressione dei file.

Vi sono due tipi di compressione:

- **lossless** che non comporta perdita di informazione, cioè il file compresso può essere riportato nelle condizioni iniziali;
- **lossy**, con perdita di informazioni.

Per esempio un algoritmo *lossless* è utilizzato dai programmi che creano i file **zip**: scompattando un file compresso con questo algoritmo si ottiene un file identico a quello originale.

Un algoritmo *lossy* è quello utilizzato dai programmi che creano i file musicali **mp3**. La codifica in *mp3*, che fa risparmiare molto spazio, riduce la qualità del suono. In generale è bene impostare la codifica affinché vi sia un buon rapporto di compressione e una perdita minima di qualità.

Per i testi la compressione può essere una semplice rimozione di spazi bianchi superflui, oppure la sostituzione di caratteri ripetuti con un solo carattere, oppure la sostituzione di gruppi di caratteri con un solo simbolo, sulla base di analisi delle frequenze con le quali i gruppi di caratteri compaiono in un file di tipo testuale. Con questi algoritmi un testo può essere ridotto anche al 50% dell'originale. I programmi di compressione convertono i file in formati compressi, assegnando una nuova estensione al nome del file (per esempio *.zip*, *.gz*, *.arj*, *.7z*). Con questi programmi si può anche creare un unico file (detto **archivio**) come insieme di file compressi, per facilitare le operazioni di trasferimento o trasmissione da un computer all'altro.

Di seguito vengono illustrati i diversi **formati** con i quali gli oggetti vengono trattati dal sistema operativo, ai quali corrispondono diversi suffissi (*estensione*) nel nome del file. I diversi formati inoltre fanno riferimento a software diversi che consentono di acquisire, registrare e riprodurre i documenti: per visualizzare una fotografia sul video del computer serve un programma di grafica, per ascoltare un motivo musicale occorre usare un programma che riproduce il suono mediante la scheda audio, ecc.

#### a) Testi

Un testo è un insieme di caratteri, parole, frasi utilizzato per rappresentare titoli, descrizioni, messaggi. I formati principali sono:

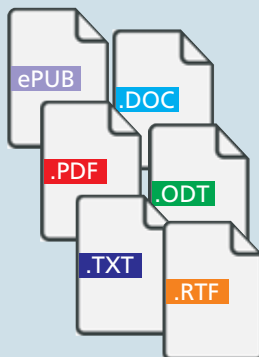
- **TXT** (*Text*), corrispondente a un semplice insieme di caratteri, senza particolari evidenziazioni o formattazioni. È il tipo di documento più semplice ed è compatibile con tutte le piattaforme senza la necessità di installare programmi particolari.
- **DOC** e **DOCX** (*Document*) testi creati con il programma *Microsoft Word*.
- **ODT** (*Open Document Format*, in italiano *formato aperto per i documenti*) testi creati con il programma *Writer* di *OpenOffice.org*.
- **RTF** (*Rich Text Format*), consente di mantenere tutta la formattazione, convertendola in istruzioni che possono essere lette e interpretate da altre applicazioni, offrendo la possibilità di scambiare file di testo tra differenti programmi di word processing; per esempio si può creare un file usando *Word*, salvare il file nel formato RTF (il file assume il suffisso *.rtf*) e inviarlo a un utente che usa il programma *OpenOffice.org* o una differente versione di *Word*.

Tra i formati per i file di testo va citato anche il formato **PDF** (*Portable Document Format*), che in realtà consente anche l'integrazione con altri formati: questi file possono essere creati con il programma *Adobe Acrobat* e visualizzati con il programma *Adobe Reader* che è disponibile per tutte le piattaforme. È un formato molto diffuso perché rappresenta l'impaginazione del documento pronto per la stampa e i file creati conservano sempre l'aspetto (*layout*) dei documenti originali, ottenuti con conversione di altri formati oppure con la digitalizzazione mediante scanner. Inoltre i documenti *pdf* possono essere visualizzati con qualsiasi sistema operativo (*Windows*, *Linux*, *OSX*) e con qualsiasi dispositivo.

Il formato *pdf* è prevalentemente utilizzato per i documenti con molta grafica e *layout* complessi, per esempio brochure, notiziari, dispense e volantini.



Per relazioni o libri con molto testo si utilizza il formato **ePub** (abbreviazione di *electronic publication*, pubblicazione elettronica): è uno standard aperto per la produzione di **e-book** (*electronic book*, libro elettronico o libro digitale) e funziona con qualsiasi dispositivo mobile di lettura, *e-reader*, *tablet* o *smartphone*.



## b) Immagini

I formati seguenti riguardano la rappresentazione di disegni, grafici e fotografie. Poiché i file di tipo grafico tendono a diventare molto grandi, i formati per le immagini sono strettamente legati agli algoritmi di compressione dei dati per rendere le dimensioni dei file il più possibile piccole.

Il **pixel** (abbreviazione di *picture element*) è la più piccola unità che un computer può visualizzare su un monitor; un insieme di pixel organizzati in una griglia sullo schermo forma i caratteri, le parti di un grafico e le immagini.

Ciascun pixel porta con sé una quantità prefissata di informazioni che dipende dal numero di bit ad esso associati: un pixel può contenere 1, 4, 8, 16, 24, 32 bit di informazione. Per esempio i pixel con 4 bit di informazioni sulla scala di grigi possono visualizzare 16 tipi di grigio ( $2^4$ ), i pixel con 8 bit possono visualizzare 256 colori ( $2^8$ ) e così via.

Lo standard attualmente più diffuso è 24 bit (8 bit per ogni colore primario) che permette la visualizzazione di milioni di colori.

Il numero dei colori determina la profondità del colore (*color depth*), cioè la definizione del colore. Queste caratteristiche del colore sono determinate dalle caratteristiche della scheda video.

Anche la **risoluzione video** dipende dalla scheda: essa rappresenta il numero dei pixel che definiscono la larghezza e l'altezza dell'immagine video. La risoluzione presa come standard è chiamata **VGA** (*Video Graphic Array*) e corrisponde a un rettangolo di 640x480 pixel. Tuttavia, considerando smartphone, tablet, portatili e computer fissi si hanno dispositivi con caratteristiche (risoluzione, frequenza di aggiornamento e profondità di colore) diverse. Nelle tabelle vengono illustrate le risoluzioni più utilizzate.

| Nome                                        | Risoluzione |
|---------------------------------------------|-------------|
| <b>QVGA</b> (quarter VGA, un quarto di VGA) | 320 x 240   |
| <b>HVGA</b> (half VGA, metà VGA)            | 480 x 320   |
| <b>VGA</b>                                  | 640 x 480   |
| <b>SVGA</b> (super VGA)                     | 800 x 600   |
| <b>XGA</b> (eXtended Graphic Array)         | 1024 x 768  |
| <b>SXGA</b> (super XGA)                     | 1280 x 1024 |
| <b>UXGA</b> (ultra XGA)                     | 1600 x 1200 |
| <b>QXGA</b> (quad XGA)                      | 2048 x 1536 |



A queste risoluzioni si aggiungono le versioni estese, con fattori di forma differenti, che vengono indicate con la lettera **W**.

| Nome                                                     | Risoluzione |
|----------------------------------------------------------|-------------|
| <b>WQVGA</b> (wide quarter VGA, un quarto di VGA estesa) | 432 x 240   |
| <b>WVGA</b> (wide VGA, VGA estesa)                       | 800 x 480   |
| <b>WSVGA</b> (wide super VGA)                            | 1024 x 600  |
| <b>WXGA</b> (wide eXtended Graphic Array)                | 1280 x 768  |
| <b>WSXGA</b> (wide super XGA)                            | 1680 x 1050 |
| <b>WUXGA</b> (wide ultra XGA)                            | 1920 x 1200 |
| <b>WQXGA</b> (wide quad XGA)                             | 2560 x 1600 |

Per le risoluzioni verticali maggiori di 720 pixel si utilizza il termine **HD** (*High Definition*, in italiano *alta definizione*) mentre per le risoluzioni verticali maggiori di 1080 pixel si utilizza il termine **FullHD**.

Durante la progettazione di un sito occorre prestare molta attenzione alla possibile visualizzazione in diverse risoluzioni in modo tale da assicurarsi che tutti gli utenti (che utilizzano display differenti) possano visualizzare il sito in maniera corretta.

Oltre alla risoluzione occorre prestare attenzione al formato con cui si sta lavorando. La classificazione dei formati suddivide le immagini in:

- bitmap (detto anche *raster*)
- vettoriali.

Per le immagini **bitmap** il formato definisce la posizione e il colore per ogni pixel.

Le dimensioni dell'immagine in pixel e il numero di colori stabiliscono le dimensioni del file su disco per registrare l'immagine in formato bitmap. I byte occupati dal file si possono calcolare con la formula:

$$\text{dimensione in byte} = \text{numero di pixel} \times \text{profondità di colore} : 8$$

tenendo presente che

$$\text{numero di pixel} = \text{altezza} \times \text{larghezza}$$

Per esempio un'immagine bitmap che occupa un quarto di video in risoluzione 1024x768 con colori a 24 bit, quindi ha la misura di 512x384 pixel, ha bisogno di un numero di byte pari a:

$$512 \text{ pixel} \times 384 \text{ pixel} \times 24 \text{ bit} : 8 = 589.824 \text{ byte (circa 590K)}.$$

Nel formato **vettoriale**, a differenza del formato bitmap, l'immagine non è descritta pixel per pixel, ma attraverso alcune forme geometriche fondamentali chiamate **primitive** (punto, segmento, poligono, cerchio, ecc.). In questo caso le immagini possono essere ridimensionate, senza perdere in qualità, adattandosi alle nuove misure, ottenute allargando o restringendo l'immagine originale. Con esso non si descrive il disegno, ma i comandi per creare il disegno.

I formati più usati per i *bitmap* sono:

- **BMP** (*BitMaP*), formato di *Microsoft* per i programmi *Windows* attualmente poco utilizzato.
- **TIFF** (*Tagged Image File Format*), è un formato comune nello scambio di immagini *bitmap* tra programmi applicativi diversi, in particolare quelli che usano le immagini acquisite con scanner; i file con questo formato hanno il suffisso *.tiff*, *.tif* o *.tff*.

- **JPEG** (*Joint Photographic Experts Group*), è un formato basato su algoritmi di compressione che devono conciliare l'esigenza di alta qualità dell'immagine con la minima dimensione del file; è un formato molto usato nelle pagine Internet, in particolare per le immagini di tipo fotografico. I file con questo formato hanno l'estensione *.jpg* o *.jpeg*.
- **GIF** (*Graphics Interchange Format*) è un formato molto usato per gli oggetti grafici in Internet ed è diventato di fatto uno degli standard più diffusi; è basato su un metodo di compressione denominato LZW. Questo formato permette anche di comporre una breve sequenza di immagini in un singolo file GIF per ottenere le cosiddette *GIF animate*.
- **PNG** (*Portable Network Graphics*), è un formato di compressione delle immagini destinato a sostituire il formato GIF rispetto al quale presenta notevoli miglioramenti e un grado di compressione maggiore.

Tra i formati *vettoriali* i più usati sono:

- **EPS** (*Encapsulated Postscript*), è il formato dei file che contengono codice *Postscript* per descrivere le immagini.
- **WMF** (*Windows MetaFile*), è un formato di file per il sistema operativo *Windows* che contiene informazioni per descrivere un altro file, in sostanza una sequenza di comandi al sistema operativo il cui risultato è la visualizzazione dell'immagine sullo schermo. Questo formato consente una notevole riduzione dello spazio su disco rispetto alle immagini *bitmap*.

### c) suoni

Anche per i suoni valgono le considerazioni fatte in precedenza a proposito degli algoritmi di compressione: in particolare gli algoritmi standard di compressione dei file audio sono *MPEG Audio* e *RealAudio*, per la riproduzione dei suoni associati alle pagine di Internet. Negli anni recenti **MP3** si è consolidato come formato standard per canzoni e motivi musicali sulla rete Internet. Esso è basato su un algoritmo di compressione della musica digitale (come quella registrata sui CD), in modo tale da ridurre fino a dieci volte le dimensioni dei documenti sonori, mantenendo un suono ad alta fedeltà. Le piccole dimensioni dei file rendono il formato MP3 adatto per trasferire o prelevare motivi musicali dalla rete Internet con tempi molto ridotti.

Altri formati utilizzati per la memorizzazione di dati audio sono **WMA** (*Windows Media Audio*), **AIFF** (*Audio Interchange File Format*) definiti rispettivamente da *Microsoft* e *Apple*: tutti gestiscono i formati mono e stereo con frequenze variabili.

Il protocollo **MIDI** (*Musical Instrument Digital Interface*) è stato progettato per la registrazione e la riproduzione di musica su sintetizzatori digitali che sono supportati dalle schede audio dei computer. Un file MIDI (avente il suffisso *.mid*) rappresenta le informazioni sul suono, piuttosto che il suono stesso e per questo motivo i file di questo formato occupano meno spazio su disco con lo svantaggio di una perdita del controllo sulla qualità del suono a seconda del dispositivo hardware installato nel computer.

### d) filmati

Il formato **MPEG** (*Moving Picture Experts Group*), già citato per i file sonori, fornisce anche la compressione dei file video digitali: la cosa appare ovvia dal momento che il filmato è sempre accompagnato dal sonoro. Esistono diverse versioni di questo standard, corrispondenti a esigenze diverse, e riconosciute a livello internazionale.

L'uso di file MPEG richiede un computer con processore veloce, una buona quantità di memoria centrale e grande quantità di spazio su disco, poiché i file contenenti filmati (aventi estensione *.mpg*) sono di dimensioni elevate anche per filmati di pochi minuti.

Un altro formato utilizzato per i filmati è denominato con **MOV** (*movie*): i file di questo tipo (aventi suffisso *.mov*) possono essere visualizzati sul monitor con il software *QuickTime*. Per presentare i documenti video su Internet si usano i formati **AVI** (*audio/visual*) e **WMV** (*Windows Media Video*), che permettono di utilizzare algoritmi che forniscono un elevato grado di compressione (dell'ordine di 250 a 1), necessario per far viaggiare i filmati sulla rete con tempi di trasmissione ridotti. Per esempio gli algoritmi di compressione **DivX** o **XviD** vengono utilizzati per ridurre le dimensioni di un intero lungometraggio in modo da essere memorizzati su un CD-ROM (760 Mbyte).

I file contenenti i video sono riproducibili sul computer usando il software *Windows Media Player* oppure installando nel browser gli appositi *plug-in*.

Questi software possono usare una tecnologia denominata **streaming**, con la quale i suoni e le immagini vengono decompressi e riprodotti appena arrivano dalla rete al computer che li ha richiesti: il risultato è la diminuzione complessiva dei tempi di attesa, in quanto l'utente non deve aspettare che l'intero file venga scaricato sul proprio computer prima di poter ascoltare o vedere il documento audio/video.

### e) animazioni

Tra i prodotti software per creare animazioni nelle pagine Web si è imposto come standard il software **Flash** che consente di creare interi siti Web interattivi, con documenti multimediali, video, grafica e animazioni. I file creati da questo programma hanno formato **SWF**, facilmente integrabile come oggetto all'interno delle pagine Web. Per visualizzare questi file, il browser deve avere il *plug-in* *Flash Player*, scaricabile liberamente da Internet.

Il *video streaming* viene ottenuto includendo un video compresso in formato **FLV** in un'animazione in formato *Flash*. Per esempio il sito *Youtube.com* utilizza questa tecnologia per lo *streaming* dei video.



### AUTOVERIFICA

Domande da 1 a 5 pag. 590

Problemi da 1 a 4 pag. 593

## 2 Il linguaggio HTML

Le pagine di Internet, in particolare tutti i documenti scritti per la tecnologia WWW, sono redatti con un linguaggio studiato per la gestione degli ipertesti: non si tratta di un linguaggio di programmazione in quanto non viene utilizzato per svolgere elaborazioni, ma di un **linguaggio di formattazione della pagina**.

Un **ipertesto** è un insieme di documenti che ha la possibilità di essere consultato in modo non sequenziale, passando da un documento all'altro attraverso collegamenti (**link**) realizzati mediante parole o immagini, sui quali occorre fare clic con il mouse.

Esso può contenere elementi non solo testuali ma anche immagini, suoni, filmati, e qualunque tipo di oggetto trattabile da un sistema informatico. In ambiente Internet, in particolare, utilizzando i riferimenti **URL** (*Uniform Resource Locator*), cioè gli indirizzi dei siti, si può accedere da un documento a un altro registrato su un computer ubicato in qualsiasi parte del mondo, realizzando così un'attività che viene comunemente definita **navigazione** nella rete (*netsurfing*).

Il linguaggio **HTML** (*HyperText Markup Language*, cioè linguaggio che utilizza dei contrassegni per la formazione di ipertesti) consente di sfruttare i vantaggi e le prestazioni derivanti dall'organizzazione ipertestuale e dall'uso degli oggetti multimediali: questo linguaggio è diventato lo standard nell'architettura WWW per creare e riconoscere i documenti ipermediali.

Le specifiche del linguaggio sono stabilite dal **W3C** (*World Wide Web Consortium*), l'Ente che regola gli standard del Web ([www.w3.org](http://www.w3.org) oppure [www.w3c.it](http://www.w3c.it) per la versione in lingua italiana). Le esigenze di inserimento di informazioni sempre più sofisticate e la necessità di rendere interattive con l'utente le pagine stesse, hanno introdotto via via numerose modifiche e ampliamenti al linguaggio stesso, creando versioni diverse. La versione consolidata è **HTML4** che è compatibile con tutti i browser. La successiva versione **HTML5** risponde alle necessità delle applicazioni che utilizzano la banda larga e il Web 2.0 e aggiunge nuove funzionalità (per esempio includere in una pagina in HTML elementi multimediali audio-video, menu, script) cercando comunque di rimanere compatibile con la versione precedente.



Nel seguito del capitolo, se non diversamente indicato, vengono presentati gli elementi compatibili con entrambe le versioni. Per visualizzare pagine Web contenenti elementi propri di HTML5 occorre usare i browser (*Internet Explorer*, *Chrome* o *Firefox*) nelle loro versioni più aggiornate.

I documenti WWW, tipicamente scritti in HTML, sono dei normali testi di caratteri, e di conseguenza sono visibili e modificabili con qualunque programma di trattamento dei testi, dal word processor come *Word* al semplice editor di testi come *Blocco note*.

Un documento in formato Web può essere aperto con un browser, attraverso un collegamento a Internet (*on line*) oppure caricandolo dal disco del proprio computer senza connettersi alla rete (*off line*). Sullo schermo viene visualizzata una pagina in formato grafico: la visualizzazione è il risultato di un'elaborazione del browser che interpreta i codici contenuti nel file e li trasforma in comandi per la costruzione della pagina.

I testi scritti in HTML si distinguono dai normali file con estensione *txt* o comunque da file di testo *non HTML* tramite l'estensione **.html** nel nome del file; spesso i testi HTML hanno il suffisso abbreviato **.htm**. Il suffisso non è un elemento obbligatorio, ma è utilizzato normalmente per identificare il tipo di file. In Windows i documenti in HTML vengono rappresentati con un'icona che raffigura il logo del browser predefinito nel computer dell'utente.

Tali documenti, quindi, non sono altro che dei normali file di caratteri; in più vengono aggiunti i codici di formazione della pagina che prendono il nome di **tag** (cioè contrassegno, da cui deriva il termine *MarkUp* della sigla di HTML) consistenti in sequenze di caratteri proprie del linguaggio HTML che non fanno parte del testo normale e che consentono di realizzare gli elementi caratteristici dell'ipertesto: i *link* (salti ipertestuali) e gli oggetti grafici e multimediali. Altri codici servono per semplificare l'impaginazione dei documenti (stili del testo, titolo dei documenti, paragrafi, liste), così da rendere il linguaggio utile anche per creare documenti complessi.

In ogni caso un testo normale (per esempio un file con l'estensione *txt*), senza codici HTML, è accettato e trascritto tale e quale da un interprete HTML, cioè dal browser di pagine Internet.

## PROGETTO 1

### Creare un documento HTML con i programmi Office.

Le più recenti versioni dei prodotti Microsoft Office (*Word*, *Excel*, *Access* e *PowerPoint*) offrono la possibilità di creare documenti che possono essere convertiti in modo automatico in testi con codice HTML, per ottenere una o più pagine visualizzabili con un browser. Per esempio per creare un documento HTML con il programma Word:

1. Aprire il programma *Microsoft Word* dalla scelta **Tutti i programmi** nel pulsante *Start* di *Windows*.
2. Scrivere alcune righe di testo.
3. Nel menu **File** fare clic su **Salva con nome...**: fornire il nome per il file da registrare e scegliere il tipo di file **Pagina Web (\*.htm; \*.html)**.

Il documento viene salvato come un testo in formato HTML e può essere successivamente letto e visualizzato con un browser.

I *tag* del linguaggio HTML sono costituiti da stringhe di caratteri comprese tra i simboli di *minore* (<) e *maggiore* (>):

```
<CODICE parametro1 parametro2 ... parametroN>
```

In HTML un codice è nella maggior parte dei casi abbinato a un altro codice corrispondente, che segna la fine dell'azione prevista da quel codice. Questo secondo tag ha una barra (*slash*) all'inizio:

```
</CODICE>
```

Per esempio, il codice **<B>** attiva il neretto nella scrittura dei caratteri; per specificare la fine del neretto si usa il codice che indica la fine dell'azione: **</B>**.

Tutti i caratteri del testo compresi tra **<B>** e **</B>** saranno visualizzati in neretto dal browser.

Il seguente frammento di codice HTML:

```
Cognome: Rossi Nome: Giovanni
```

viene interpretato dal browser come mostrato in figura.

**Cognome: Rossi Nome: Giovanni**

Tutti i codici possono essere indifferentemente specificati in caratteri maiuscoli o minuscoli: nel seguito useremo i tag sempre in maiuscolo per evidenziarli e distinguerli dalle frasi del testo.

Di norma un testo HTML viene suddiviso in almeno due **sezioni**:

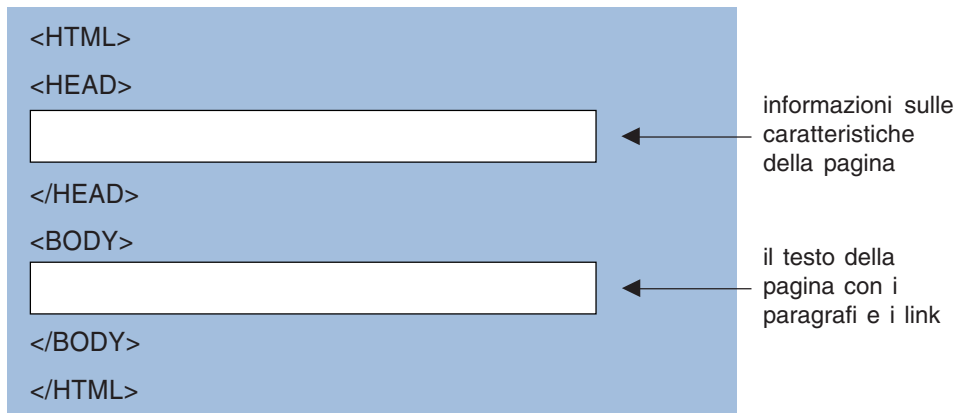
- la sezione *intestazione* (in inglese *head*) contenente informazioni sulla pagina;
- la sezione *corpo* (in inglese *body*), contenente il documento vero e proprio con tutti gli elementi caratteristici di un ipertesto.

La codifica di queste due sezioni avviene attraverso i tag:

- **<HEAD>** per l'intestazione
- **<BODY>** per il corpo.

Inoltre un documento HTML è di norma delimitato dai tag **<HTML>** e **</HTML>**, perché alcuni browser devono essere informati su quale porzione di testo è necessario applicare l'interpretazione del linguaggio HTML.

Quindi un documento HTML ha la seguente struttura di base:



All'interno della sezione di intestazione HEAD viene descritta la pagina, in particolare viene inserito il titolo e, utilizzando particolari tag chiamati *meta tag*, le informazioni sulla pagina (o *metadati*).

## Titolo

A ogni pagina HTML può essere associato un titolo che ne descrive il contenuto e che viene riportato nella barra del titolo della finestra del browser di Internet. Il titolo è usato principalmente per identificare il documento (per esempio quando si cerca un documento tra tanti attraverso un motore di ricerca oppure quando una pagina viene inserita in un *bookmark* di siti preferiti): è opportuno quindi che ogni titolo sia diverso dagli altri all'interno dell'insieme di pagine che formano un sito e che sia formato da una frase significativa per ricordarne il contenuto.

Il titolo viene racchiuso tra la coppia di tag **<TITLE>** **</TITLE>** e viene mostrato sulla barra del titolo della finestra oppure sulle linguette delle pagine Web se, nel browser, esse sono organizzate a schede.

## Meta tag

Nella sezione di intestazione della pagina HTML, tra **<HEAD>**... **</HEAD>**, sono inseriti anche i **meta tag**: sono i tag che contengono i **metadati** della pagina, cioè informazioni sul contenuto della pagina, le parole chiave e l'autore. I meta tag sono identificati dal tag **<META>**.

La struttura generale della frase è:

```
<META NAME="....." CONTENT=".....">
```

I valori più usati per l'attributo **NAME** sono:

- **description** per la descrizione,
- **keywords** per le parole chiave,
- **author** per l'autore.

Per esempio, la pagina di un sito Web scolastico può contenere metadati come i seguenti:

```
<META NAME="description" CONTENT="sito Web dell'Istituto Mazzini">
<META NAME="keywords" LANG="it" CONTENT="scuola, Mazzini, Istituto,
Informatica, Web, Tecnologie, POF, studenti, docenti, genitori">
<META NAME="author" CONTENT="Webmaster, redazione">
```

Un'importante informazione riguarda la specifica della codifica della pagina. Se non si specifica la codifica corretta può accadere che alcuni caratteri non standard, per esempio le lettere accentate, non vengano visualizzate correttamente.

La codifica utilizzata per rappresentare i simboli delle numerose lingue esistenti nel mondo si chiama **Unicode** ([www.unicode.org](http://www.unicode.org)). Di questo codice esistono poi versioni ridotte, per esempio *UTF-8*.

Per specificare la codifica **UTF-8**, che contiene tutti i caratteri occidentali, si utilizza il codice:

```
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

Nella versione HTML5 la dichiarazione è più concisa:

```
<META charset="UTF-8" />
```



## CARATTERI SPECIALI

Il linguaggio HTML è stato originariamente progettato per la comunicazione anche tra terminali che usano solo codifiche ASCII a 7 bit, cioè da 0 a 127; di conseguenza, se non si specifica una codifica estesa, tutti i caratteri con codice ASCII formato da 8 bit (per esempio le lettere accentate della lingua italiana, che sono identificate con un codice ASCII superiore al 127 decimale) devono essere rappresentati in modo particolare.

In HTML, per visualizzare i caratteri speciali, si utilizzano le **sequenze di escape**, ovvero delle stringhe particolari che all'atto dell'interpretazione da parte del browser vengono tradotte con i caratteri desiderati.

Una prima categoria di caratteri riguarda quelli che il linguaggio HTML utilizza per la rappresentazione dei tag, quali **<**, **&** o le virgolette, che verrebbero quindi considerati come facenti parte del codice del linguaggio e perciò ignorati.

Per questo motivo si usano, preceduti dal carattere **&** (*escape sequence*), le sequenze:

|                   |       |
|-------------------|-------|
| <b>&amp;lt;</b>   | per < |
| <b>&amp;gt;</b>   | per > |
| <b>&amp;amp;</b>  | per & |
| <b>&amp;quot;</b> | per " |

(con i punti e virgola finali).

Per le lettere accentate, si usano le seguenti rappresentazioni:

|                     |       |
|---------------------|-------|
| <b>&amp;egrave;</b> | per è |
| <b>&amp;Egrave;</b> | per È |
| <b>&amp;eacute;</b> | per é |

Ci sono rappresentazioni analoghe per le altre vocali a, o, i, u.

Altri simboli di uso comune sono:

|                     |                                        |
|---------------------|----------------------------------------|
| <b>&amp;euro;</b>   | per il simbolo dell'euro €             |
| <b>&amp;copy;</b>   | per il simbolo del copyright ©         |
| <b>&amp;reg;</b>    | per il simbolo di marchio registrato ® |
| <b>&amp;frac12;</b> | per la frazione 1/2                    |
| <b>&amp;sum;</b>    | simbolo di sommatoria                  |
| <b>&amp;infin;</b>  | simbolo di infinito                    |
| <b>&amp;alfa;</b>   | lettera greca alfa                     |
| <b>&amp;beta;</b>   | lettera greca beta                     |
| <b>&amp;gamma;</b>  | lettera greca gamma                    |



In alternativa si possono usare anche i valori numerici dei codici dei caratteri **Unicode** preceduti dai simboli **&** e **#** e seguiti dal punto e virgola finale.

Per esempio la lettera **ç** (c con cediglia), usata nella lingua francese, corrisponde al codice numerico

**&#199;**

che è equivalente al codice testuale

**&ccedil;**

Il simbolo cinese dell'acqua 水 può essere rappresentato con la sequenza **&#27700;**

Si osservi che utilizzando la codifica UTF-8 la maggior parte dei caratteri speciali viene interpretata correttamente.

## Doctype

È buona norma indicare all'inizio della pagina Web il linguaggio utilizzato e la versione. Queste informazioni si possono inserire utilizzando il tag **<!DOCTYPE>**.

Il codice di base è

```
<!DOCTYPE html>
```

che semplicemente specifica che la pagina è scritta in HTML.

Si possono aggiungere ulteriori indicazioni riguardanti il codice:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

dove:

- **PUBLIC** indica che il documento è pubblico.
- **W3C** indica che vengono utilizzate le specifiche rilasciate dal W3C.
- **DTD HTML 4.01 Transitional** (*Document Type Definition*, definizione del tipo di documento) significa che il documento segue le specifiche della versione *4.01 Transitional*.
- **EN** indica che il documento è scritto in inglese.

Segue poi l'URL dove si trova il documento DTD con le specifiche.

Come si vede, oltre alla versione (4.01) viene specificato anche il termine *Transitional*. La versione 4.01 ammette infatti tre specifiche differenti:

- **Strict**, è la versione più pura di HTML, che contiene esclusivamente tag e attributi ed è senza *frame*. Sono ammessi solo i tag ufficiali. Non ci può essere nessuna descrizione del *layout*.
- **Transitional**, è la versione più elastica in cui si può descrivere il *layout* della pagina e utilizzare tag sconsigliati (in inglese *deprecated*), cioè non più in uso.
- **Frameset**, indica la presenza di *frame*.

Il termine **frame** indica una porzione di pagina dove viene visualizzata una pagina Web differente. Per esempio in una zona laterale della finestra viene mostrato un menu e nella parte centrale i contenuti situati in altre pagine Web. Anche se questo metodo è da considerarsi superato dall'utilizzo dei fogli di stile *CSS* e del linguaggio *Javascript*, alcuni siti ne fanno ancora uso. In HTML4 la specifica del *doctype* è facoltativa mentre nella versione HTML5 è obbligatoria e, non essendoci di fatto la necessità di utilizzare sottoversioni, si utilizza il codice abbreviato:

```
<!DOCTYPE html>
```



## PROGETTO 2

### Visualizzare il codice HTML di una pagina Web.

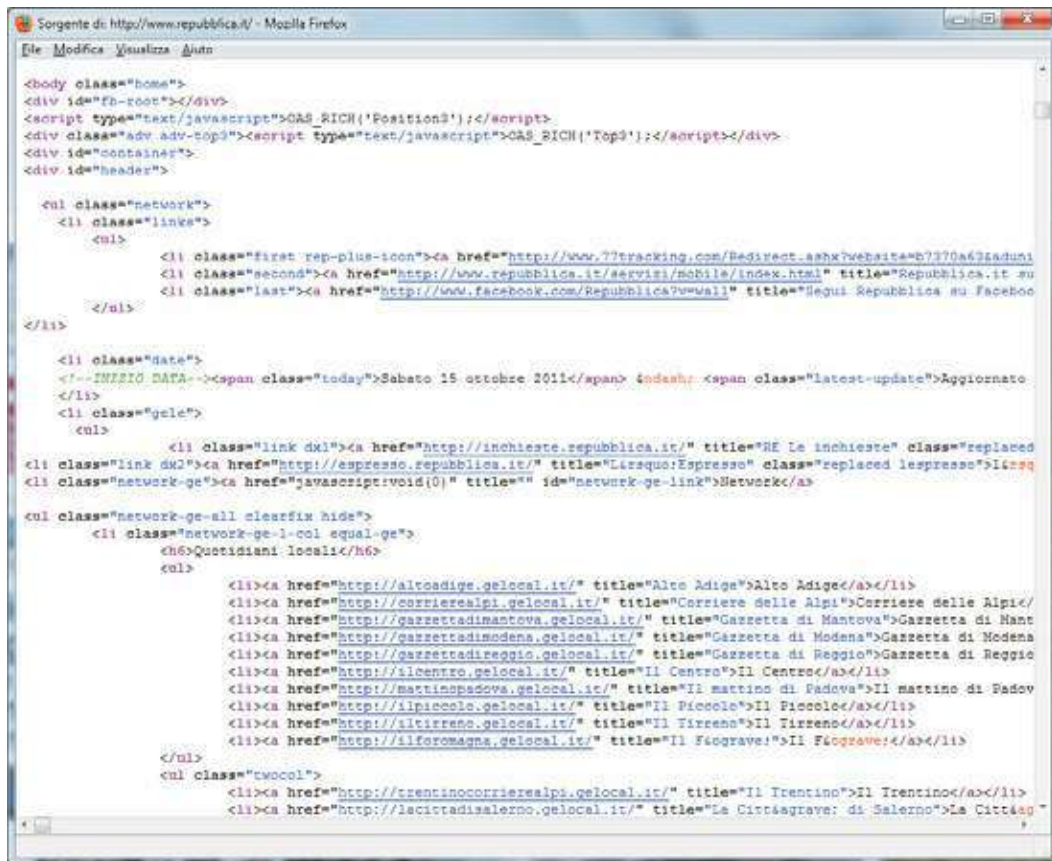
1. Aprire il browser *Firefox*.
2. Caricare una pagina Web.
3. Premere il tasto destro del mouse sulla pagina e selezionare l'opzione **Visualizza sorgente pagina** oppure fare clic sul pulsante



4. Selezionare **Sviluppo Web** e poi **Visualizza sorgente pagina** oppure utilizzare la scorciatoia con la combinazione di tasti **Ctrl + U**.

Modalità del tutto analoghe si possono usare per gli altri browser (*Internet Explorer*, *Chrome*, ecc.).

Viene aperta una nuova finestra che mostra il *codice sorgente* della pagina in linguaggio HTML.



```
<body class="home">
<div id="fb-root"></div>
<script type="text/javascript">OAS_RICH('Position3');</script>
<div class="adv adv-top3"><script type="text/javascript">OAS_RICH('Top3');</script></div>
<div id="container">
<div id="header">

<ul class="network">
<li class="links">

<li class="first rep-plus-icon">
<li class="second">
<li class="last"><a href="http://www.facebook.com/Repubblica?w=wall" title="Segui Repubblica su Facebo

<li class="date">
<!-- INIZIO DATA -->Sabato 15 ottobre 2011 <dash> Aggiornato

<li class="gale">

<li class="link dw1"><a href="http://inchieste.repubblica.it/" title="RE le inchieste" class="replaced
<li class="link dw1">L'Espresso
<li class="network-ge">Network

<ul class="network-ge-all clearfix hide">
<li class="network-ge-l-col equal-ge">
<h6>Quotidiani locali</h6>

Alto Adige
Corriere delle Alpi
Gazzetta di Mantova
Gazzetta di Modena
Gazzetta di Reggio
Il Centro
Il mattino di Padova
Il Piccolo
Il Tirreno
Il Fogliore

<ul class="twocol">
Il Trentino
La Cittàadisaalerno


```

Nel caso in cui sia stata selezionata una parte della pagina Web ne viene mostrato solo il codice relativo (**Visualizza sorgente selezione**).

MODELLO

## Atene, mito a due velocità

CULTURA: «Democrazia realizzata in età classica, idealizzata dal 700 ad oggi. Fu governo di popolo e dominio accettato di sottoporsi al controllo dei cittadini»

ANDYMOUS: In Sh

CULTURA: «Anonymous» con Edward Hoag, Rhys Ifans. Il

Photographs Foto

11 e il 12 novembre a Parigi

Visualizza sorgente selezione

```
<p>
CULTURADemocrazia realizzata in età classica,
idealizzata dal 700 ad oggi. Fu governo di popolo e dominio di signori. Ma l'élite raffinata
accettò di sottoporsi al controllo dei cittadini
</p>
```

Il significato dei tag di uso comune sarà spiegato nei prossimi paragrafi del capitolo.

#### AUTOVERIFICA

Domande da 6 a 7 pag. 591-592  
Problemi da 5 a 6 pag. 593



#### MATERIALE ONLINE

1. Verifica della compatibilità del browser con la versione HTML5
2. Programmi per l'editing di pagine Web con guida pratica per il loro utilizzo

## 3 La formattazione

La soluzione più efficace per formattare le pagine Web si basa sull'uso dei  *fogli di stile* , che saranno descritti nel successivo paragrafo 6. Il linguaggio HTML comprende alcuni tag che permettono di cambiare lo stile del testo e della pagina (**tag stilistici**). Di seguito vengono illustrati i tag più importanti che servono per definire un testo in linguaggio HTML.

### Dimensione e stile dei caratteri

Solitamente è necessario differenziare le dimensioni dei caratteri, anche solo per distinguere il titolo dal resto del testo o comunque per definire dei sottotitoli o elementi testuali evidenziati. Per questo è prevista una serie di sei codici differenti per altrettante dimensioni differenti dei caratteri. I valori di queste differenti dimensioni sono preceduti nel tag dalla lettera H:

**<Hn>...</Hn>**, dove *n* indica il codice numerico corrispondente alla dimensione del carattere.

Il numero tra 1 e 6 specifica in effetti l'importanza del sottotitolo (1 risulta essere il più importante).

Per esempio con il seguente codice

```
<H1>Eccezionale ritrovamento archeologico</H1>
<H2>Scoperta una tomba di 2.700 anni fa nel Cile</H2>
<H3>All'interno della camera funeraria trovati monili in oro e ossidiana</H3>
```

si ottiene nel browser la seguente pagina:

## Eccezionale ritrovamento archeologico

### Scoperta una tomba di 2.700 anni fa nel Cile

All'interno della camera funeraria trovati monili in oro e ossidiana

Alcune parti di testo possono essere evidenziate usando uno stile diverso dal testo normale, con il corsivo, il neretto o il sottolineato. La parte di testo deve essere racchiusa tra una delle seguenti coppie di tag:

`<I>...</I>` per ottenere il testo in corsivo (*italic*)

`<B>...</B>` per il neretto o grassetto (*bold*)

`<U>...</U>` per il sottolineato (*underline*).

Per esempio, aggiungendo al codice precedente le seguenti righe:

```
<U>La tomba</U>. Si trova all'interno di una piramide, potrebbe essere "<I>la piu' antica del suo tipo in tutta l'area dell' America Centrale</I>", ha precisato oggi l'Istituto nazionale di antropologia e storia (Inah), rilevando che l'importante scoperta e' stata fatta nell'area archeologica di <I>Chiapa de Corzo</I>, circa 850 km a sud-est di Citta' del Messico.
```

si ottiene:

La tomba. Si trova all'interno di una piramide, potrebbe essere "*la piu' antica del suo tipo in tutta l'area dell' America Centrale*", ha precisato oggi l'*Istituto nazionale di antropologia e storia* (Inah), rilevando che l'importante scoperta e' stata fatta nell'area archeologica di *Chiapa de Corzo*, circa 850 km a sud-est di Citta' del Messico.

## Paragrafi

Un paragrafo è una porzione di testo separata dalle altre da una riga di spazio o semplicemente da un ritorno a capo.

Per specificare un paragrafo si utilizza il tag `<P>...</P>`; l'inserimento di questo codice, oltre a creare una riga di separazione tra paragrafi, viene utilizzata per assegnare le impostazioni tipografiche al paragrafo stesso.

Il codice `<P>` è anche utilizzato per imporre un salto di paragrafo quando serve.

È importante sottolineare che, a differenza di quanto avviene nei programmi di *word processing*, in HTML l'imporre un ritorno a capo con il tasto *Invio* per portare a riga nuova il testo scritto, non produce l'effetto desiderato: quando si edita un testo, se si finisce una frase in mezzo a una riga con un punto e si va a capo, quello che si scrive nella riga seguente apparirà subito dopo il punto.

Per inserire invece un ritorno a capo si usa il tag `<BR>` per l'interruzione di riga, che si può scrivere ovunque serva. A differenza dei tag visti finora il tag `<BR>` non prevede un tag di chiusura (cioè non esiste `</BR>`). Per maggiore chiarezza del codice è buona norma usare il tag nella forma `<BR />`.

Talvolta può essere utile separare un paragrafo da un altro attraverso una linea orizzontale: questa riga viene inserita con il tag **<HR>**. Anche questo tag non ha chiusura, quindi viene usato nella forma **<HR />**.

Più spazi di seguito vengono raggruppati in un unico spazio (l'indentazione tramite spazi non funziona in HTML).

HTML adotta il *word-wrapping*, cioè ridimensiona la finestra in cui il documento appare, scalando il testo per permetterne la visione integrale.

Per evitare questa operazione occorre racchiudere la parte di testo tra la coppia di tag indicanti il testo preformattato

**<PRE> ... </PRE>**

che consente anche di mantenere il formato originale di un testo creato con un *word processor*, comprese le tabulazioni e i rientri. Il tag **<PRE>** è da considerarsi obsoleto per cui è da utilizzare il meno possibile.

## Colori

Per rappresentare i colori dei caratteri o dello sfondo di una pagina, il linguaggio HTML utilizza una combinazione dei tre colori fondamentali **RGB** (*Red, Green, Blue*) attraverso tre numeri, compresi ciascuno tra 0 e 255, che specificano la gamma cromatica del rosso, del verde e del blu. Questi valori sono scritti in formato esadecimale (base 16) e quindi variano da 00 a FF, essendo FF la rappresentazione in base 16 del numero 255.

Per esempio:

| Codice | Colore | Spiegazione                                                                                             |
|--------|--------|---------------------------------------------------------------------------------------------------------|
| 000000 | Nero   | Livello di rosso, di verde e di blu a zero; nessun colore corrispondente al nero                        |
| FFFFFF | Bianco | Livello di rosso, di verde e di blu al valore massimo; la somma di tutti i colori corrisponde al bianco |
| FF0000 | Rosso  | Livello di rosso al massimo, verde e blu a zero: rosso puro                                             |
| 00FF00 | Verde  | Livello di verde al massimo, rosso e blu a zero: verde puro                                             |
| 0000FF | Blu    | Livello di blu al massimo, rosso e verde a zero: blu puro                                               |

I colori intermedi corrispondono a valori compresi tra 00 e FF per i tre colori.

Per esempio:

| Codice | Colore      | Spiegazione                                            |
|--------|-------------|--------------------------------------------------------|
| FF9999 | Rosa        | Livello massimo di rosso, livello medio di blu e verde |
| 33CC99 | Verde acqua | Livello alto di verde, medio di blu e basso di rosso   |
| 999999 | Grigio      | Tutti i colori allo stesso livello                     |

Per assegnare un colore di sfondo a una pagina si usa l'attributo **BGCOLOR** del tag **<BODY>**. Per esempio il tag

```
<BODY BGCOLOR="#FFFFFF">
```

consente di visualizzare una pagina con lo sfondo bianco.

Se le cifre esadecimali del colore sono uguali, si può usare una formula abbreviata: per esempio, #000 equivale a #000000.

I colori possono essere specificati anche con un nome simbolico predefinito (in inglese) al posto del valore esadecimale.

Per esempio per ottenere uno sfondo giallo chiaro, si può usare uno dei seguenti tag:

```
<BODY BGCOLOR= "#FFFFE0">
```

oppure

```
<BODY BGCOLOR= "lightyellow">.
```

In Appendice si può trovare la tabella completa dei colori utilizzabili in una pagina HTML con il nome simbolico e il corrispondente valore in esadecimale.

A questo punto possiamo provare a scrivere un semplice esempio di documento in formato HTML, controllandone il funzionamento con il browser.

### PROGETTO 3

#### Scrivere una pagina in HTML.

1. Aprire un programma di trattamento del testo, per esempio **Blocco note**, negli **Accessori** di *Windows*.

Scrivere il seguente testo:

```
<!DOCTYPE html>
<HTML>
 <HEAD>
 <TITLE>Un nuovo documento</TITLE>
 </HEAD>
 <BODY BGCOLOR ="#FFFFE0">
 <H1>La rete per comunicare</H1>
 Una rete si collega a un'altra rete e forma una rete

 di reti.
 <P>I computer delle reti comunicano tra loro.</P>
 </BODY>
</HTML>
```

Si osservi che l'indentazione del codice HTML non viene mantenuta nella visualizzazione con il browser, tuttavia favorisce l'ordine e la leggibilità del codice.

2. Nel menu **File** di *Blocco Note* scegliere **Salva con nome...**; nella finestra di dialogo che si apre, fornire, come **Nome File**, *prova.htm* e nella casella **Salva come** scegliere la dicitura **Tutti i file**. Chiudere il programma *Blocco Note*.
3. Con **Risorse del computer** individuare la cartella contenente il testo appena salvato: si noti che il file HTML ha assunto come icona il logo del browser normalmente utilizzato in *Windows*.
4. Fare doppio clic sull'icona: si apre il programma browser e la pagina viene visualizzata in forma grafica e in modalità *Non in linea*.

## La rete per comunicare

Una rete si collega a un'altra rete e forma una rete di reti.

I computer delle reti comunicano tra loro.

5. Fare clic sul menu **Visualizza** del browser, e poi clic sulla scelta **HTML** per vedere il sorgente della pagina, oppure nel menu di scelta rapida che si apre facendo clic con il tasto destro del mouse in un punto qualsiasi della pagina, fare clic su **HTML**.
6. Correggere l'errore presente nel testo: il tag **<BR />**, che serve per inserire un ritorno a capo, è stato messo in una posizione errata e spezza la frase in modo inatteso. Cancellare il tag **<BR />**, salvare il testo nella versione corretta (scelta **Salva** nel menu **File**) e chiudere *Blocco Note* (scelta **Esci** dal menu **File**).
7. Nel browser fare clic sull'icona **Aggiorna** nella barra degli strumenti: la pagina viene visualizzata nella versione corretta.

## Link

La principale caratteristica di un **ipertesto** è la possibilità di consultare il documento in modo non sequenziale attraverso l'uso di collegamenti ad altri oggetti o pagine. Con i **link** si effettuano riferimenti a file remoti o ad altre parti del documento stesso oppure si può passare velocemente da un indice alla voce richiesta.

Il link appare nella pagina come un'immagine, un'animazione o semplicemente una parola (o una breve frase) di norma sottolineata e di colore azzurro: passando su di esso il puntatore del mouse si trasforma diventando una piccola mano con il dito indice alzato; questo simbolo indica che si può attivare il collegamento premendo il tasto del mouse. Un link già visitato, di solito, assume il colore rosso porpora.

La struttura del tag che definisce un link ipertestuale è la seguente:

**<A HREF = "nomerisorsa" > parola </A>**

Il *nomerisorsa* indica la destinazione del link, e deve essere indicato con la sintassi degli indirizzi URL.

*Parola* indica la parte di testo che compare sottolineata e in colore per rappresentare il link. La lettera A, all'inizio e alla fine del tag, sta per **àncora** (*anchor*), cioè punto di attracco, destinazione del salto ipertestuale; quindi questo link specifica il riferimento ipertestuale (*HREF*) di un'àncora.

Per esempio la riga:

```
Prova
```

produce sul video la parola *Prova* sottolineata e in colore; se con il mouse si fa clic su di essa, il browser richiama il file *prova.htm*: abbiamo quindi realizzato un link ipertestuale a un'altra pagina.

Con i link definiti dal tag **<A HREF> ... </A>**, si può produrre un salto ipertestuale a un altro sito Internet, specificandone l'indirizzo URL:

```
Vai su Google
```

Sul video la frase *Vai su Google* appare sottolineata e in colore; passando sopra di essa il puntatore del mouse diventa una mano e, facendo clic su di essa, il browser richiama la pagina di apertura del sito richiesto.



Si può anche specificare il nome della finestra di destinazione del salto ipertestuale, dove deve essere visualizzata la nuova pagina, attraverso l'attributo **TARGET**. In particolare, il valore **\_blank** apre la pagina del link in una nuova finestra del browser:

```
Vai su Google
```

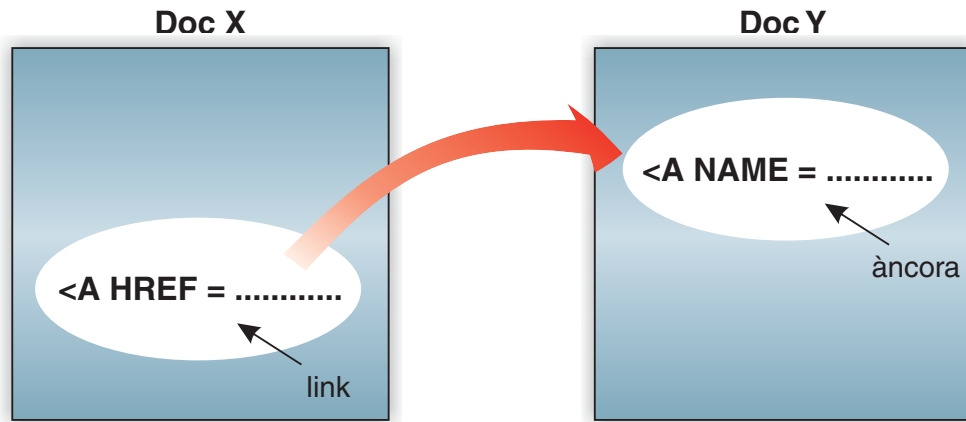
I link possono riguardare anche parti specifiche di un documento oppure possono essere utilizzati per spostarsi all'interno dello stesso documento, oppure a una determinata posizione interna a un altro documento, facendo riferimento alle sue sezioni.

Si supponga di voler creare un link da un documento X a una specifica sezione del documento Y. La sezione a cui si fa riferimento nel documento Y deve iniziare con un tag HTML definito nel modo seguente:

```

```

Il punto dove viene inserito il tag `<A NAME>... </A>` specifica il nome dell'**àncora** (*anchor name*), cioè del punto di arrivo di un salto ipertestuale.



Nel documento X si deve inserire il seguente link:

```
parola
```

ove *docy.htm* è l'indirizzo in formato URL che localizza il documento Y e *parola* è la parte sottolineata e in colore, mostrata nel documento X, per segnalare all'utente, che consulta la pagina, la presenza di un link.

Quando si produce una pagina in Internet può essere utile creare il collegamento a un indirizzo di posta elettronica (*e-mail*), per esempio dell'autore del documento.

Il codice

```
Scrivete a Rossi Giacomo
```

consente di attivare un link con l'autore. Se l'utente fa un clic sulla scritta *Scrivete a Rossi Giacomo* viene aperto un modulo vuoto di posta elettronica da completare, che verrà inviato all'indirizzo di e-mail specificato (*rossi@societax.it*).

L'uso di **mailto** (*spedisci a*) indica che il riferimento ipertestuale è un indirizzo di posta elettronica anziché un file o un indirizzo Web.

## I NOMI DEI FILE

I documenti in rete vengono identificati univocamente utilizzando l'**URL** (*Uniform Resource Locator*) chiamato comunemente *indirizzo* della pagina Web.

Per esempio `http://www.sito.it/pagina2.html` indica il documento HTML `pagina2.html` del sito `www.sito.it`.

I nomi che si utilizzano per i testi in HTML sono di norma liberi, e hanno di solito l'estensione `.htm` oppure `.html`. Ma esiste un identificativo particolare:

`index.html`

(in alternativa si può usare anche `home.html`).

Esso sta a indicare la pagina di apertura di un sito Internet (**Home Page**), cioè la prima pagina che viene presentata all'utente che visita il sito.

Se questo file è presente nella directory del sito Internet sul server, esso viene aperto automaticamente quando l'utente specifica l'indirizzo del sito, anche senza specificare il nome del file.

Per esempio, se si scrive `http://www.sito.it/prova` nella casella *Indirizzo*, il browser cerca il file `index.html` nella directory `prova` del sito `http://www.sito.it/`, e, se presente, lo visualizza. È comodo quindi per l'utente usare tale nome per la *home page* del proprio sito o per la pagina di partenza di ogni sottodirectory contenuta nel sito stesso.

Occorre anche osservare che alcuni sistemi operativi che gestiscono i server Internet (per esempio *Linux*) usano di solito la distinzione tra lettere maiuscole e minuscole (sono cioè *case sensitive*): questo significa che se un file è salvato su disco con il nome `prova.htm` (in minuscolo) e il testo HTML contiene il tag

```
Prova
```

il file richiesto non potrà essere ritrovato, perché, avendo la lettera iniziale maiuscola, viene considerato con un nome diverso.

È buona norma quindi uniformare il metodo di rappresentazione per i nomi dei file: la soluzione più semplice consiste nell'usare sempre le lettere minuscole, che sono anche lo standard in ambiente Internet per gli indirizzi URL.

## Liste

Il linguaggio HTML mette a disposizione i codici per definire le **liste**, cioè righe di un elenco, in particolare liste puntate o numerate.

Per ottenere una lista puntata si usa la seguente sintassi:

- si inizia con un tag di apertura **<UL>**
- si mette il tag **<LI>...</LI>** per ogni riga della lista
- si chiude la lista con il tag **</UL>**.

Per esempio, la seguente porzione di codice HTML

```

 Roma
 Milano
 Napoli

```

produce nella visualizzazione della pagina la lista puntata:

- Roma
- Milano
- Napoli



Per ottenere la stessa lista come elenco numerato, basta sostituire i tag `<UL>...</UL>` con `<OL>...</OL>`.

1. Roma
2. Milano
3. Napoli

## Tabelle

In una pagina Web si possono inserire le **tabelle** per organizzare i dati su righe e colonne, secondo una griglia che facilita la visualizzazione di informazioni. Le tabelle sono però utili anche per impaginare testo e immagini in modo ordinato: in questo caso i bordi delle celle devono essere assenti, usando la tabella solo come traccia per disporre gli elementi.

I tag del linguaggio HTML che definiscono una tabella sono `<TABLE> ... </TABLE>`.

La riga di intestazione delle colonne, con i caratteri in grassetto, è identificata dai tag `<TH>...</TH>`.

All'interno vengono definite le righe, ciascuna racchiusa tra `<TR>...</TR>`.

All'interno della riga ogni cella è racchiusa tra la coppia di tag `<TD>...</TD>`.

Per esempio una tabella di due righe e tre colonne, con una riga di intestazione, si rappresenta in HTML nel seguente modo:

```
<TABLE WIDTH="70%" BORDER="1">
<TR>
<TH WIDTH="40%">Fiume</TH>
<TH WIDTH="40%">Lunghezza (km)</TH>
<TH WIDTH="20%">Foce</TH>
</TR>
<TR>
<TD>Volga</TD>
<TD>3531</TD>
<TD>Mar Caspio</TD>
</TR>
<TR>
<TD>Danubio</TD>
<TD>2858</TD>
<TD>Mar Nero</TD>
</TR>
<TR>
<TD>Reno</TD>
<TD>1320</TD>
<TD>Mare del Nord</TD>
</TR>
<TR>
<TD>Loira</TD>
<TD>1020</TD>
<TD>Oceano Atlantico</TD>
</TR>
<TR>
<TD>Po</TD>
<TD>652</TD>
<TD>Mar Adriatico</TD>
</TR>
</TABLE>
```

| Fiume   | Lunghezza (km) | Foce             |
|---------|----------------|------------------|
| Volga   | 3531           | Mar Caspio       |
| Danubio | 2858           | Mar Nero         |
| Reno    | 1320           | Mare del Nord    |
| Loira   | 1020           | Oceano Atlantico |
| Po      | 652            | Mar Adriatico    |

All'interno di ogni cella si possono inserire testo, immagini o qualsiasi altro oggetto.  
Altre opzioni riguardano:

- **<TABLE BORDER = valore>** per aggiungere il bordo alla tabella;
- **<TABLE WIDTH = valoreInPercentuale>** per stabilire la percentuale di occupazione della pagina;
- **<TH WIDTH = valoreInPercentuale>** o **<TD WIDTH = valoreInPercentuale>** per fissare la larghezza di ciascuna colonna espressa in percentuale.
- **<TABLE CELLPADDING = valore>** per aggiungere spazio tra le celle della tabella
- **<TABLE CELLSPACING = valore>** per aggiungere dello spazio tra il contenuto delle celle e i bordi.

Per capire il significato di **CELLPADDING** e **CELLSPACING** vediamo alcuni esempi:

```
<TABLE BORDER="1" CELLSPACING="0" CELLPADDING="0">
 <TR>
 <TD>Cuori</TD>
 <TD>Quadri</TD>
 </TR>
 <TR>
 <TD>Fiori</TD>
 <TD>Picche</TD>
 </TR>
</TABLE>
```

fornisce come risultato la seguente tabella

|       |        |
|-------|--------|
| Cuori | Quadri |
| Fiori | Picche |

Modificando il valore di *CELLSPACING* nella prima linea

```
<TABLE BORDER="1" CELLSPACING="15" CELLPADDING="0">
```

si ottiene

|       |        |
|-------|--------|
| Cuori | Quadri |
| Fiori | Picche |

Le altre combinazioni sono:

```
<TABLE BORDER="1" CELLSPACING="0" CELLPADDING="15">
```

|       |        |
|-------|--------|
| Cuori | Quadri |
| Fiori | Picche |

```
<TABLE BORDER="1" CELLSPACING="15" CELLPADDING="15">
```

|       |        |
|-------|--------|
| Cuori | Quadri |
| Fiori | Picche |

## UNIONE DI CELLE NELLE TABELLE

Nelle tabelle i tag **<TD>** e **<TH>** possono essere accompagnati da due attributi: **COLSPAN** e **ROWSPAN**.

Questi attributi specificano il numero di colonne (o di righe) sul quale si deve estendere una cella: *COLSPAN* definisce il numero di colonne occupate da una cella, mentre *ROWSPAN* definisce il numero di righe occupate da una cella.

Nell'esempio seguente l'intestazione *Primo trimestre* occupa lo spazio di tre colonne:

```
<TABLE WIDTH="200" BORDER="1">
 <TR>
 <TH COLSPAN="3">Primo trimestre</TH>
 </TR>
 <TR>
 <TD>Gennaio</TD>
 <TD>Febbraio</TD>
 <TD>Marzo</TD>
 </TR>
</TABLE>
```

| Primo trimestre |          |       |
|-----------------|----------|-------|
| Gennaio         | Febbraio | Marzo |

L'attributo *ROWSPAN* si comporta allo stesso modo: se la prima colonna deve occupare lo spazio di quattro righe, occorre inserire l'attributo *ROWSPAN="4"*.

```
<TABLE WIDTH="200" BORDER="1">
 <TR>
 <TH ROWSPAN="4">Italia</TH>
 <TH>Zona</TH>
 <TH>Agente</TH>
 </TR>
 <TR>
 <TD>Nord</TD>
 <TD>Rossi</TD>
 </TR>
 <TR>
 <TD>Centro</TD>
 <TD>Bianchi</TD>
 </TR>
 <TR>
 <TD>Sud e Isole</TD>
 <TD>Verdi</TD>
 </TR>
 <TR>
 <TH ROWSPAN="2">Esteros</TH>
 <TD>Europa</TD>
 <TD>Smith</TD>
 </TR>
 <TR>
 <TD>Asia</TD>
 <TD>Ching Sai</TD>
 </TR>
</TABLE>
```

| Italia  | Zona        | Agente    |
|---------|-------------|-----------|
|         | Nord        | Rossi     |
|         | Centro      | Bianchi   |
|         | Sud e Isole | Verdi     |
| Esteros | Europa      | Smith     |
|         | Asia        | Ching Sai |

Si osservi che l'attributo *ROWSPAN* deve essere inserito solo nella prima riga (all'interno del primo tag **<TR>****</TR>**). Nelle righe successive si inseriscono le colonne per ogni riga senza ripetere la prima colonna di intestazione.

## 4 Elementi multimediali

### Immagini

I browser per Internet sono in grado di visualizzare immagini che sono state memorizzate in un formato grafico.

In particolare i formati grafici più usati sono:

- **GIF** (*Graphics Interchange Format*)
- **JPEG** (*Joint Photographic Experts Group*)
- **PNG** (*Portable Network Graphics*).

Le immagini possono essere inserite all'interno di un documento HTML. Ogni immagine richiede un certo tempo di elaborazione, se è di notevoli dimensioni, causando un rallentamento del sistema nel momento della sua acquisizione. Nel progettare le pagine HTML è opportuno usare immagini di dimensioni contenute, al fine di ridurre i tempi necessari per il caricamento delle pagine e quindi i tempi di attesa per l'utente.

Per includere un'immagine nel documento si deve usare il tag **IMG** (abbreviazione di *image*) con l'attributo **SRC** (abbreviazione di *source*):

```

```

dove *nomeimmagine* indica il file sorgente dell'immagine, cioè l'indirizzo URL del file contenente l'immagine.

La barra finale chiude il tag IMG che non ha tag di chiusura.

Di norma un'immagine incorporata in una pagina è registrata nella stessa directory della pagina, e quindi l'indirizzo URL è riferito a file locali. In alternativa si possono raccogliere tutte le immagini utilizzate nelle pagine del sito in un'unica sottodirectory di nome *images*. In questo caso il tag per l'immagine diventa:

```

```

Il formato dell'immagine è scelto in base all'estensione nel nome del file contenente l'immagine; quindi se il file immagine è in formato GIF il suo nome termina con l'estensione *.gif*; se il formato è JPEG termina con l'estensione *.jpg*.

Di norma, se non viene specificato nulla (per *default*) l'angolo in basso a destra dell'immagine è allineato con il testo che segue nel codice HTML. Nella zona indicata con *parametri* nella sintassi del tag, si specifica una delle opzioni per l'allineamento con il parametro **ALIGN**:

```
ALIGN="top"
ALIGN="middle"
ALIGN="bottom"
```

se si vuole allineare il testo rispettivamente vicino al bordo superiore dell'immagine, al centro dell'immagine o al bordo inferiore. L'ultima opzione è quella di *default* e quindi può essere omessa.

Un'immagine (per esempio un bottone o una fotografia) può diventare anche un **link** ipertestuale per saltare a un'altra pagina oppure a un altro sito; per fare questo la parola o la frase del link ipertestuale è sostituita dal tag **<IMG SRC="...">** come nel seguente esempio:

```
.
```

Quando l'utente fa un clic sopra l'immagine, il browser passa alla visualizzazione della nuova pagina identificata con il nome *pag2.htm*.

Può accadere che l'utente, che si collega al sito Internet, disponga di terminali non grafici; tali terminali non sono in grado di visualizzare le immagini nei diversi formati grafici. Per ovviare a questo problema è stata messa a disposizione l'opzione **ALT** (che sta per *alternate*) che permette di visualizzare un testo al posto dell'immagine qualora questa non possa essere visualizzata.

Per esempio, la seguente specifica:

```

```

fa apparire la scritta *Descrizione alternativa* qualora il sistema non sia in grado di visualizzare le immagini.

In pratica, poiché i terminali non grafici sono dispositivi obsoleti, nelle versioni recenti dei browser, la descrizione specificata dopo ALT viene usata per visualizzare una scritta che appare sopra l'immagine (in fondo giallo) quando il mouse passa sopra di essa: questa opportunità può essere sfruttata per presentare al visitatore una breve descrizione dell'immagine stessa oppure del contenuto della pagina che sarà visualizzata facendo clic sull'immagine, nel caso in cui l'immagine funzioni come link ipertestuale.

Un accorgimento che serve a rendere più veloce la visualizzazione delle pagine contenenti immagini, evitando lunghe attese per l'utente, consiste nell'indicare, all'interno del tag per le immagini, le misure in pixel della larghezza (**WIDTH**) e dell'altezza (**HEIGHT**) dell'immagine. Tali misure sono facilmente rilevabili caricando l'immagine in un programma per la grafica. Per esempio:

```

```

Aggiungendo queste opzioni, il browser visualizza subito la parte testuale della pagina lasciando degli spazi a forma di rettangolo che verranno riempiti con le immagini. Mentre le immagini vengono caricate, l'utente ha la possibilità di leggere qualcosa anziché guardare lo schermo vuoto.

Inoltre, quando nel sito si vogliono utilizzare immagini di grandi dimensioni oppure fotografie ad alta risoluzione, che provocano un rallentamento nel caricamento delle pagine, si può decidere di visualizzare le immagini solo su richiesta dell'utente creando un link per l'immagine; questa verrà quindi visualizzata in una nuova finestra e non più all'interno della pagina corrente.

Per includere un riferimento a un'immagine si usa la seguente sintassi HTML:

```
descrizione dell'immagine.
```

Il link può essere realizzato anche tramite una versione rimpicciolita dell'immagine originale, ottenuta riducendone le dimensioni con un programma di grafica.

La stessa sintassi è quella di norma usata per specificare un link con animazioni e suoni esterni, in quanto questi elementi multimediali richiedono lunghi tempi di caricamento.

È importante notare che il riconoscimento del formato del file, a cui un link si riferisce, viene effettuato tramite le estensioni del nome di file, che risultano quindi non più facoltative, ma necessarie per specificare il formato degli oggetti multimediali.

## Video

La maggior parte dei video viene inserita nelle pagine Web utilizzando un *video player* in *Adobe Flash* e integrandolo nel codice HTML con il tag **OBJECT**. Non esiste uno standard fisso né per il *video player* né per il codice utilizzato per inserirlo: per questo tipo di oggetto ogni sito può utilizzare un proprio metodo. In più il *browser* deve avere installato un *plug-in* per visualizzare gli elementi multimediali in *Flash*.

In generale si utilizza una sintassi come la seguente:

```
<OBJECT opzioni>
 <PARAM NAME=nomeParametro VALUE=valoreParametro>
 . . .
 <EMBED SRC=nomeFile TYPE="application/x-shockwave-flash" opzioni>
 </EMBED>
</OBJECT>
```

Per esempio per inserire in una pagina HTML un video di *YouTube*, in modo da visualizzarlo in *streaming*, si utilizza il seguente codice:

```
<OBJECT WIDTH=larghezza HEIGHT=altezza>
 <PARAM NAME="movie" VALUE=indirizzo></param>
 <PARAM NAME="wmode" VALUE="transparent"></param>
 <EMBED SRC="indirizzo" TYPE="application/x-shockwave-flash"
 WMODE="transparent" WIDTH=larghezza HEIGHT=altezza>
 </EMBED>
</OBJECT>
```

dove

- *larghezza* e *altezza* sono le dimensioni del video
- *indirizzo* è l'URL del video sul sito *www.youtube.com*.

La versione HTML5 semplifica notevolmente la procedura perchè permette di inserire un video con l'apposito tag **<VIDEO>**. La sintassi è molto semplice e ogni browser compatibile con HTML5 può visualizzare il filmato, senza dover installare *plug-in* aggiuntivi.

Il formato utilizzato per il video si chiama **Webm** che è un formato compresso e gratuito.

Per aggiungere un video alla pagina si utilizza il codice seguente:

```
<VIDEO SRC=nomeFileVideo POSTER=nomeFileCopertina CONTROLS=
 messaggioErrore>
</VIDEO>
```

- *nomeFileVideo* è l'URL del video,
- *nomeFileCopertina* è l'URL dell'immagine che viene visualizzata prima dell'avvio del video,
- *messaggioErrore* è il testo che viene mostrato se il *browser* non è compatibile con il tag VIDEO.

Oltre a **POSTER** e a **SRC**, altri attributi del tag **<VIDEO>** sono:

- **CONTROLS="controls"** per mostrare i controlli (*play/pausa*)
- **AUTOPLAY="autoplay"** per avviare il file automaticamente
- **LOOP="loop"** per ripetere il brano in continuo
- **PRELOAD="preload"** per caricare il brano prima dell'avvio.

Tutti gli attributi precedenti sono di tipo *booleano*. Quindi i codici:

- `<VIDEO CONTROLS>`
- `<VIDEO CONTROLS="controls">`
- `<VIDEO CONTROLS="">`

sono equivalenti.

## Audio

Un discorso simile ai video vale anche per i file audio: non c'è uno standard preciso. Quindi spesso si utilizza un *audio player* scritto in *Flash* integrandolo nella pagina con il tag **<OBJECT>**.

In alcuni casi si può inserire il file audio semplicemente come link con il tag **<A>**:

```
Fai clic qui per ascoltare l'audio
```

In questo caso il file non viene aperto all'interno della pagina Web ma, a seconda delle impostazioni del browser, può essere aperto in un'altra finestra oppure viene salvato sul disco fisso.

In ogni caso il file audio non è integrato all'interno del sito.

Analogamente ai video, la versione HTML5 utilizza per i suoni il tag **<AUDIO>**. Si possono aggiungere file in formato compresso *mp3* o *ogg*.

La sintassi è:

```
<AUDIO opzioni>
 <SOURCE SRC=nomeFile TYPE=typefile>
 Messaggio errore
</AUDIO>
```

dove **SOURCE** indica il file sorgente dell'audio: **SRC** è il percorso del file (URL) e **TYPE** specifica il formato del file audio.

Per esempio, per aggiungere alla pagina Web il file *musicaPreferita.mp3* si utilizza il codice seguente:

```
<AUDIO CONTROLS="controls">
 <SOURCE SRC="./musicaPreferita.mp3" TYPE="audio/mp3" />
 Attenzione: il tuo browser non supporta il tag AUDIO.
</AUDIO>
```

Il risultato è la visualizzazione di un semplice lettore audio in grado di aprire il file *.mp3*.



Altri attributi del tag **<AUDIO>** sono:

- **CONTROLS="controls"** per mostrare i controlli (*play/pausa*)
- **AUTOPLAY="autoplay"** per avviare il file automaticamente
- **LOOP="loop"** per ripetere il brano in continuo
- **PRELOAD="preload"** per caricare il brano prima dell'avvio.

Tutti gli attributi precedenti sono di tipo *booleano*. Quindi i codici:

- `<AUDIO CONTROLS>`
- `<AUDIO CONTROLS="controls">`
- `<AUDIO CONTROLS="">`

sono equivalenti.

Per esempio, per inserire una musica di sottofondo, si aggiunge l'attributo *AUTOPLAY*=*"autoplay"* per l'avvio automatico e si toglie l'attributo *CONTROLS*=*"controls"* per nascondere i controlli. All'apertura della pagina non viene mostrato nulla, ma viene avviato il sottofondo musicale.

#### AUTOVERIFICA

Domande da 8 a 15 pag. 591-592

Problemi da 7 a 18 pag. 593-594



#### MATERIALI ONLINE

### 3. Mappe sensibili al mouse

## 5 I form e l'interazione con l'utente

I siti Web non sono soltanto un insieme di pagine da leggere con testo e immagini, ma rappresentano uno strumento per interagire con gli utenti: per esempio un'azienda può utilizzare un sito Web non solo per presentare i suoi prodotti o servizi, ma anche per raccogliere ordini o richieste di informazioni sui prodotti oppure per effettuare vendite e transazioni commerciali.

Per l'invio di informazioni da parte dell'utente si usano i **form** (moduli) che in HTML sono racchiusi tra la coppia di tag **<FORM> ... </FORM>**.

All'interno del *form* si possono inserire oggetti per acquisire i dati o le scelte dall'utente. I componenti del form per inserire un valore o effettuare una scelta vengono aggiunti utilizzando il tag **<INPUT>**. Per specificare il tipo di input si utilizza l'attributo **TYPE**.

#### Casella di testo (text)

```
<INPUT TYPE="text" NAME=nome SIZE=valore MAXLENGTH=valore VALUE=valore>
```

Tra i parametri utilizzabili ci sono:

- **SIZE** per la dimensione della casella di testo
- **MAXLENGTH** per il numero massimo dei caratteri in input
- **VALUE** per l'eventuale valore predefinito.

Per questo e per gli oggetti grafici successivi, in generale, il parametro **NAME** identifica l'oggetto all'interno del *form* e il parametro **VALUE** specifica il valore che può essere successivamente utilizzato da procedure eseguibili dal lato client o dal lato server.

#### Password (password)

È simile al tipo testo e viene utilizzata per campi speciali contenenti una password.

```
<INPUT TYPE="password" NAME="pwd" SIZE="10" MAXLENGTH="8">
```

In questo caso i caratteri inseriti dall'utente sono sostituiti durante la digitazione da una sequenza di asterischi o di punti.



### Casella di controllo (checkbox):

Viene utilizzato per effettuare una scelta. Si può selezionare più di una voce contemporaneamente. Per esempio:

```
<FORM>
Lingue conosciute

<INPUT TYPE="checkbox" NAME="CB1" VALUE="Ita" CHECKED>Italiano

<INPUT TYPE="checkbox" NAME="CB2" VALUE="Ing">Inglese

<INPUT TYPE="checkbox" NAME="CB3" VALUE="Fra">Francese

<INPUT TYPE="checkbox" NAME="CB4" VALUE="Ted">Tedesco

</FORM>
```

Ciascun elemento può essere selezionato o deselezionato con un click del mouse. L'attributo **CHECKED**, se presente, indica una selezione predefinita.

### Pulsanti di opzione (radio)

Come la casella di controllo, i pulsanti di opzione vengono utilizzati per effettuare una scelta. A differenza dei precedenti, non si possono effettuare più scelte contemporaneamente.

```
<FORM>
Carta di credito

<INPUT TYPE="radio" NAME="carta" VALUE="VISA" CHECKED>VISA

<INPUT TYPE="radio" NAME="carta" VALUE="MasterCard">MasterCard

</FORM>
```

Per consentire la scelta alternativa tra quelle possibili, gli oggetti di tipo *radio* devono avere lo stesso nome specificato con il parametro *NAME*. L'elemento selezionato per *default* è indicato con l'attributo **CHECKED**.

### Campo nascosto (hidden)

È un campo nascosto che non viene visualizzato. Viene utilizzato per inviare i dati in maniera trasparente all'utente:

```
<INPUT TYPE="hidden" NAME="codicePost" value="143354_2">
```

### Pulsanti di comando per l'invio (*submit*) o la cancellazione (*reset*) dei dati

Nella parte inferiore del form occorre inserire due pulsanti che permettono l'invio o la cancellazione dei dati.

```
<INPUT TYPE="submit" NAME="B1" VALUE="Invia">
<INPUT TYPE="reset" NAME="B2" VALUE="Annulla">
```

Per questi oggetti grafici la proprietà **VALUE** indica la parola che deve comparire sul pulsante. Facendo clic sul pulsante di tipo *submit* si attiva una procedura eseguibile, specificata come parametro del tag `<FORM>`, alla quale vengono anche inviati i dati che l'utente ha inserito tramite gli oggetti grafici del *form*.

Vediamo ora gli altri tag utilizzabili all'interno di un form.

### Area di testo

Il tag **<TEXTAREA>** permette di creare un'area nella quale inserire testo che può occupare anche più righe.

```
<TEXTAREA NAME="richiesta" ROWS="3" COLS="10">
Scrivi qui la tua domanda</TEXTAREA>
```

I parametri del tag specificano il numero di righe (proprietà **ROWS**) e di colonne (proprietà **COLS**) che vengono visualizzate come area di digitazione e un eventuale valore iniziale predefinito (testo compreso tra `<TEXTAREA>``</TEXTAREA>`).

### Casella combinata

Con il tag **<SELECT>** (*lista di selezione*) si crea un casella combinata con la quale si sceglie un elemento da un elenco di valori:

```
<FORM>
<SELECT NAME="regione">
<OPTION VALUE="1" SELECTED>Nord</OPTION>
<OPTION VALUE="2">Centro</OPTION>
<OPTION VALUE="3">Sud e Isole</OPTION>
</SELECT>
</FORM>
```

Aggiungendo l'attributo **SELECTED** al tag `<OPTION>` si specifica un elemento selezionato per default.

Per consentire all'utente di effettuare scelte multiple all'interno della casella combinata (*lista di selezione multipla*), occorre aggiungere il parametro **MULTIPLE** subito dopo il tag `SELECT`:

```
<SELECT MULTIPLE NAME="scelte">
...
</SELECT>
```

La versione 5 del linguaggio HTML assegna grande importanza all'aspetto semantico e quindi presenta un maggior numero di elementi e di parametri per i form. A prima vista può sembrare che l'aumento del numero dei tag e degli attributi utilizzabili renda più complicata la scrittura del codice: in realtà si hanno più funzionalità.

Supponiamo per esempio di creare un form dove l'utente deve inserire la propria *home page*. In HTML4 si utilizza un campo testo nel seguente modo:

```
<FORM>
Homepage: <INPUT TYPE="text" name="homepage">
<INPUT TYPE="submit" NAME="Invia" VALUE="Invia">
</FORM>
```

Il campo è di tipo testo per cui l'utente può inserire una sequenza di caratteri qualsiasi senza alcun controllo.

Homepage:

In HTML5 invece esiste il tipo **url** per cui si può scrivere il codice nel seguente modo:

```
<FORM>
Homepage: <INPUT TYPE="url" name="homepage">
<INPUT TYPE="submit" NAME="Invia" VALUE="Invia">
</FORM>
```

All'apparenza non cambia nulla. Tuttavia, quando l'utente inserisce il dato, viene effettuata la **validazione** dei dati e, se il testo inserito non è un indirizzo Web, l'utente riceve un avviso.

Homepage:

Inserire un URL

Altri tipi utilizzabili per l'attributo *TYPE* sono:

- **email** quando il campo deve contenere un indirizzo di posta elettronica
- **date** quando il campo deve contenere una data (in un formato riconoscibile, per esempio 23-07-1996)
- **time** quando il campo deve contenere un orario (per esempio 13:34)
- **number** quando il campo deve contenere un valore numerico.

Per rendere il campo obbligatorio si aggiunge l'attributo **REQUIRED**.

Il controllo della validità del codice HTML di una pagina Web può essere effettuato utilizzando appositi programmi, detti **validatori**, disponibili sulla rete Internet. In particolare il validatore del W3C si trova all'indirizzo <http://validator.w3.org/>. Una breve guida per l'uso operativo del validatore è contenuta nei *Materiali on line*.



#### AUTOVERIFICA

Domande da 16 a 17 pag. 592  
Problemi da 19 a 21 pag. 594



#### MATERIALI ON LINE

### 4. Validazione del codice HTML

## 6 Fogli di stile CSS

Il foglio **CSS** (*Cascading Style Sheet*, foglio di stile a cascata) è un documento in formato testuale che definisce il controllo sulla visualizzazione della pagina HTML e sul suo **layout**, cioè la disposizione degli elementi nella pagina, con le caratteristiche di **stile** riguardanti i font, i colori, gli allineamenti e le spaziature.

Come si è visto nei paragrafi precedenti, il linguaggio HTML consente di definire, utilizzando opportuni *tag*, diversi attributi per personalizzare la visualizzazione del documento HTML, per esempio il font e la dimensione di un carattere oppure il colore di sfondo di una pagina. Benché questa tecnica consenta di definire con una certa accuratezza le necessità di presentazione per una singola pagina, essa risulta inadeguata nel momento in cui è necessario un maggior controllo sulla visualizzazione di più pagine, anche di un intero sito Web. Infatti lo sviluppatore di pagine Web, volendo per esempio modificare l'aspetto (font, dimensione, colore) di ogni titolo, definito dal tag `<H1>`, di una singola pagina o dell'intero sito, è costretto a intervenire manualmente nel codice di ogni pagina per effettuare le opportune modifiche. Queste procedure comportano necessariamente uno sforzo di aggiornamento molto elevato all'aumentare del numero delle pagine da modificare.

Il CSS serve a eliminare questo lavoro gravoso: separa i contenuti della pagina Web dal *layout* e dalle caratteristiche di rappresentazione e, grazie a differenti livelli di astrazione, consente una trasformazione controllata, e nello stesso tempo elegante, di uno o più documenti Web.

In particolare i vantaggi nell'uso dei fogli di stile sono:

- Separazione dei contenuti dalle caratteristiche di visualizzazione: i tag di struttura permettono di definire logicamente lo schema dei contenuti, per esempio il titolo di primo livello con `<H1>`, il titolo di secondo livello con `<H2>`, mentre i fogli di stile consentono di definire come essi devono essere visualizzati.
- Maggiore controllo sulla rappresentazione di documenti omogenei: permette di modificare la visualizzazione di ogni documento modificando un solo foglio di stile; ogni modifica si ripercuote direttamente su ogni documento ad esso associato.

- Indipendenza dal dispositivo e quindi migliore accessibilità: garantisce una trasformazione del documento in presenza di differenti dispositivi di visualizzazione o particolari necessità dell'utente. In pratica si definisce uno stile differente a seconda della piattaforma utilizzata dai visitatori.

I fogli di stile sono scritti usando una sintassi specifica e fissata nei documenti del **W3C** (*World Wide Web Consortium*), l'ente internazionale per gli standard nel Web. La versione più recente è **CSS3**.

Esistono tre principali metodi per utilizzare i CSS all'interno dei documenti HTML:

- Fogli di stile in linea
- Fogli di stile incorporati
- Fogli di stile collegati.

I **fogli di stile in linea** non presentano significative modifiche rispetto all'utilizzo di specifiche proprietà dei tag HTML. Infatti le proprietà di stile in linea vengono innestate direttamente in corrispondenza del tag che si desidera controllare utilizzando l'attributo **STYLE**. Lo stile viene applicato solo a quel particolare tag. Per esempio il seguente paragrafo viene visualizzato dal browser secondo le seguenti impostazioni: tipo di carattere (**font-family**) *Arial*, dimensione (**font-size**) di 9 pixel (9px), colore del testo (**color**) rosso (#FF0000), colore di sfondo (**background-color** o, in forma abbreviata, **background**) giallo (#FFFF33).

Le impostazioni dello stile sono racchiuse tra virgolette e separate da punto e virgola.

```
<P STYLE="font-family: Arial; font-size: 9px; color: #FF0000; background-color: #FFFF33">
 Le Regioni italiane
</P>
```

Le Regioni italiane

I **fogli di stile incorporati** permettono di stabilire un livello di astrazione maggiore: la struttura di ogni stile viene definita, attraverso il tag **<STYLE>**, all'interno dell'intestazione del documento definita dai tag **<HEAD> ... </HEAD>**. A differenza dei fogli di stile in linea, utilizzando questo sistema lo stile viene applicato a tutti i tag presenti in quella pagina Web: si definisce ogni rappresentazione che si desidera utilizzare all'interno del documento in modo che abbia sempre lo stesso comportamento; per esempio si può stabilire il carattere predefinito per ogni titolo di livello 1 (tag **<H1>**).

```
<HTML>
 <HEAD>
 <STYLE TYPE="text/css">
 H1 { font-family: Arial; font-size: 23px; color: #0099CC }
 </STYLE >
 </HEAD>
 <BODY>
 <H1>Le regioni italiane</H1>
 <H2>Dal 1948 a oggi</H2>
 ...
 <H1>Le province</H1>
 ...
 <H1>I comuni</H1>
 ...
 </BODY>
</HTML>
```

Ogni titolo di livello 1 è visualizzato dal browser utilizzando caratteri con font Arial, dimensione di 23 pixel e colore azzurro.

Si noti che le impostazioni dello stile sono racchiuse tra parentesi graffe e separate da punto e virgola.

Le parentesi graffe si ottengono con la combinazione di tasti **Maiusc + AltGr + [** e **Maiusc + AltGr + ]**.

Mentre con gli stili in linea, per modificare l'aspetto di ogni titolo marcato nel documento come elemento <H1>, sarebbe stato necessario modificare ogni singolo elemento, con lo stile incorporato è sufficiente modificare la definizione degli stili CSS una volta sola nella sezione di intestazione della pagina.



I **fogli di stile collegati** stabiliscono un livello di astrazione ancora più importante, perché permettono la definizione di un foglio di stile separato dal documento e registrato in un file esterno avente estensione **.css**. Questo file viene poi collegato alla singola pagina attraverso il tag **<LINK>**.

Questa modalità permette di strutturare un foglio di stile, contenente tante definizioni per le diverse caratteristiche della pagina, che può essere condiviso da più pagine o da un intero sito Web. Per esempio si modifica l'aspetto di ogni paragrafo di tutte le pagine del sito eseguendo una sola modifica alla definizione della sua rappresentazione nel foglio di stile collegato.

```
<HTML>
 <HEAD>
 <LINK REL="stylesheet" HREF="stile.css" TYPE="text/css">
 </HEAD>
 <BODY>
 <P>
 Il 13 dicembre 1947 nella seduta pomeridiana della seconda
 sottocommissione della Commissione per la Costituzione, il presidente Mauro
 Gennari legge il testo definitivo dell'articolo 22: Le Regioni sono: Piemonte,
 Lombardia, Trentino-Alto Adige, Veneto, Friuli, Venezia-Giulia, Liguria, Emilia
 e Romagna, Toscana, Umbria, Marche, Lazio, Abruzzi, Molise, Campania, Puglia,
 Salento, Lucania, Calabria, Sicilia, Sardegna, Valle d'Aosta, ma il Testo
 coordinato dal Comitato di redazione prima della votazione finale in Assemblea
 e distribuito ai Deputati il 20 dicembre 1947 recitava: Art. 131. Sono
 costituite le seguenti Regioni: Piemonte; Valle d'Aosta; Lombardia; Trentino-
 Alto Adige; Veneto; Friuli-Venezia Giulia; Liguria; Emilia-Romagna; Toscana;
 Umbria; Marche; Lazio; Abruzzi e Molise; Campania; Puglia; Basilicata;
 Calabria; Sicilia; Sardegna.
 </P>
 </BODY>
</HTML>
```

Nel codice HTML il nome *stile.css* rappresenta il file contenente le definizioni del foglio di stile collegato.

Un documento HTML può utilizzare contemporaneamente le tre modalità di uso dei fogli CSS esposte precedentemente, però *a cascata*, cioè secondo regole predefinite di priorità per l'unione di più fogli di stile (in ordine crescente di priorità):

1. Impostazione predefinite del browser Web
2. Fogli di stile collegati
3. Fogli di stile incorporati
4. Fogli di stile in linea.

Secondo questa definizione, un foglio di stile in linea ha quindi la precedenza su ogni altro foglio di stile (foglio di stile incorporato, collegato, impostazioni del browser): un foglio di stile in linea, innestato in un elemento HTML, nasconde ogni altro stile dichiarato in precedenza.

Come si è visto, un foglio di stile è un insieme di regole che permette di associare *tag* ed elementi del linguaggio HTML a dettagliate caratteristiche di presentazione.

Il linguaggio CSS prevede una sintassi che identifica esplicitamente un **selettore** e le regole di presentazione ad esso associate. Per ogni selettore si dichiara una o più proprietà di stile: ogni proprietà di stile va separata da un punto e virgola. L'insieme delle proprietà è raggruppato da una coppia di parentesi graffe.

```
selettore {
 proprietà: valore;
 proprietà: valore;
 proprietà: valore;
 ...;
}
```

Un selettore può essere:

- un **tag predefinito** del linguaggio HTML (<H1>, <P>, <TD>, ecc.)
- una **classe** definita dall'utente; lo stile viene poi richiamato nella pagina HTML con l'attributo **CLASS**:

CLASS="nomeclasse"

- un **identificatore** definito dall'utente che individua le caratteristiche di una parte del documento; l'applicazione dello stile viene richiamata nella pagina HTML con l'attributo **ID**:

ID="identificatore"

Mentre la classe definisce le caratteristiche di paragrafi e singole parti di una pagina HTML, gli identificatori vengono solitamente utilizzati per definire il layout di una pagina suddivisa in **contenitori** (o sezioni), identificati con i tag <DIV> ... <DIV>.

Uno schema generale dell'uso di classi e identificatori all'interno della pagina HTML è il seguente:

```
<DIV ID="identificatore">
 ...
 <P CLASS="nomeclasse">
 ...
 </P>
 ...
</DIV>
```

Un selettore ID può essere usato una sola volta all'interno di una pagina, mentre una classe può essere richiamata più volte in punti diversi.

Nel foglio di stile il nome degli identificatori è preceduto dal carattere #, mentre il nome delle classi è preceduto dal punto (.).

Il linguaggio CSS prevede anche la possibilità di inserire **commenti** che saranno ignorati dal browser: un commento inizia con /\* e termina con \*/.

Essendo un file di testo, il foglio di stile (file con estensione .css) può essere creato con qualsiasi editor di testi, per esempio *Blocco note* di Windows, anche se l'uso di un programma specifico (*CSS editor*) facilita il lavoro di scrittura del codice CSS, perché le proprietà vengono indicate con un colore diverso ed eventuali errori nel codice vengono evidenziati.

## PROGETTO 4

### Progettare la pagina di notizie per un sito scolastico.

La creazione del foglio di stile è preceduta dalla progettazione della pagina, con il layout e la definizione delle caratteristiche dei diversi elementi. La pagina è organizzata con un titolo, un piè di pagina (*footer*) e due colonne nella parte centrale, con un menu di scelte a sinistra e la finestra di visualizzazione dei contenuti a destra.

Ogni voce del menu a sinistra è un link a una pagina Web diversa.

|                                  |                                          |
|----------------------------------|------------------------------------------|
| titolo                           |                                          |
| sinistra<br>menu<br>menu<br>menu | contenuto<br>notizia<br>notizia<br>..... |
| footer                           |                                          |

Nella figura sono stati indicati i nomi utilizzati nel foglio di stile.

Vediamo ora in dettaglio le definizioni degli stili.

Il tag per i titoli di secondo livello <H2> viene ridefinito con evidenziazione in grassetto (**bold**) dimensioni maggiori (114% rispetto alle dimensioni predefinite del testo nel browser), font *Arial sans-serif* e colore azzurro (codice RGB in esadecimale 006699).

```
H2 {
 font: bold 114% Arial,sans-serif;
 color: #006699;
}
```

La parte in alto della pagina (identificatore *titolo*) ha un'altezza (**height**) di 60 pixel, evidenziazione in grassetto, 50% più grande, font *Arial*, colore bianco su fondo blu.

```
#titolo {
 height: 60px;
 font: bold 150% Arial,sans-serif;
 color: #FFFFFF;
 background-color: #0000FF;
}
```

La colonna di sinistra (identificatore *sinistra*) ha una larghezza (**width**) pari al 20% della pagina, ed è collocata 0 pixel a sinistra e 80 pixel a partire dal bordo superiore della pagina; il colore di sfondo è bianco. Il posizionamento è assoluto (**position**) secondo le distanze dai margini superiore e laterale, impostate con le proprietà **top** e **left**.

```
#sinistra {
 position: absolute;
 width: 20%;
 left: 0px;
 top: 80px;
 background-color: #FFFFFF;
}
```

La classe *menu* definisce lo stile per le voci del menu nella colonna di sinistra: testo allineato al centro (**text-align**), con dimensione 110% (**font-size**).

```
.menu {
 text-align: center;
 font-size: 110%;
}
```

La parte a destra per i contenuti (identificatore *contenuto*) ha i caratteri neri su fondo bianco; il posizionamento è indicato con i margini: il 25% a sinistra e il 20% a destra dello spazio a disposizione, con 0 pixel sia per il margine in alto che in basso.

```
#contenuto {
 color: #000000;
 background-color: #FFFFFF;
 margin-right: 20%;
 margin-top: 0px;
 margin-left: 25%;
 margin-bottom: 0px;
}
```

Le notizie del contenitore precedente (classe *notizia*) sono scritte con caratteri più piccoli (80%).

```
.notizia {
 font-size: 80%;
}
```

Il piè di pagina (identificatore *footer*) ha un'altezza di 30 pixel, caratteri più grandi del 14%, font *Arial*, allineamento al centro, colore bianco su fondo blu.

```
#footer {
 height: 30px;
 font: bold 114% Arial,sans-serif;
 text-align: center;
 color: #FFFFFF;
 background-color: #0000FF;
}
```

Si possono ridefinire i tag HTML associati a un identificatore. Per esempio, un link nel *footer*, che ha fondo blu, sarebbe invisibile, perché di solito i link nelle pagine HTML sono indicati in blu sottolineato, mentre i link già visitati diventano colore porpora. Per mantenerli comunque visibili, si assegna il colore bianco ai collegamenti ipertestuali (**a:link**) e ai link visitati (**a:visited**).

```
#footer a:link, #footer a:visited {
 color: #FFFFFF;
}
```

I selettori *a:link* e *a:visited* si chiamano **pseudoselettori**.

La dichiarazione precedente mostra anche che si possono assegnare le stesse proprietà a due o più selettori, scrivendoli uno accanto all'altro separati dalla virgola.

Le precedenti dichiarazioni vengono salvate in un unico file di testo con estensione **.css**.



### Foglio di stile (*DueColonne.css*)

```
H2 {
 font: bold 114% Arial,sans-serif;
 color: #006699;
}
#titolo {
 height: 60px;
 font: bold 150% Arial,sans-serif;
 color: #FFFFFF;
 background-color: #0000FF;
}
#sinistra {
 position: absolute;
 width: 20%;
 left: 0px;
 top: 80px;
 background-color: #FFFFFF;
}
.menu {
 text-align: center;
 font-size: 110%;
}
#contenuto {
 color: #000000;
 background-color: #FFFFFF;
 margin-right: 20%;
 margin-top: 0px;
 margin-left: 25%;
 margin-bottom: 0px;
}
.notizia {
 font-size: 80%;
}
#footer {
 height: 30px;
 font: bold 114% Arial,sans-serif;
 text-align: center;
 color: #FFFFFF;
 background-color: #0000FF;
}
#footer a:link, #footer a:visited {
 color: #FFFFFF;
}
```

Le dichiarazioni di stile devono poi essere inserite nella pagina HTML per determinare il layout degli elementi e le loro caratteristiche di visualizzazione. All'inizio della pagina il tag **LINK** specifica il nome del file contenente il foglio di stile da utilizzare:

```
<LINK REL="stylesheet" HREF="DueColonne.css" TYPE="text/css" />
```

Per comodità di lettura, i nomi degli identificatori e delle classi sono stati evidenziati in neretto.

## Pagina Web (Notizie.htm)

```
<!DOCTYPE html>
<HTML>
<HEAD>
<TITLE>Sito della scuola</TITLE>
<LINK REL="stylesheet" HREF="DueColonne.css" TYPE="text/css" />
</HEAD>
<BODY>
<DIV ID="titolo">Ultime notizie</DIV>
<DIV ID="sinistra">
<P CLASS="menu">News</P>
<P CLASS="menu">Orari</P>
<P CLASS="menu">Docenti</P>
<P CLASS="menu">POF</P>
</DIV>
<DIV ID="contenuto">
<H2>Gara di nuoto</H2>
<P CLASS="notizia">Giovedì 12 marzo alle ore 10 si svolgeranno le gare di nuoto
di Istituto presso la Piscina comunale.

I primi tre classificati passano alle fasi regionali.
</P>
<H2>Conferenza per le classi quinte</H2>
<P CLASS="notizia">Martedì 2 aprile alle ore 9 nell'aula magna incontro con il
prof. Bianchi sul tema "Le nuove tecnologie del Web". Alla conferenza
partecipano le classi 5A, 5B e 5C.
</P>
</DIV>
<DIV ID="footer">
<P>scrivi a
Webmaster</P>
</DIV>
</BODY>
</HTML>
```

La figura mostra la pagina creata con questo foglio di stile.



Si provi ora a modificare il foglio di stile *DueColonne.css*, per cambiare il colore dell'intestazione e del piè di pagina da blu ad arancione e per allineare a destra le voci del menu laterale (le modifiche da apportare sono evidenziate in neretto).

```
#titolo {
 height: 60px;
 font: bold 150% Arial,sans-serif;
 color: #FFFFFF;
 background-color: #FF6600;
}
.menu {
 text-align: right;
 font-size: 110%;
}
#footer {
 height: 30px;
 font: bold 114% Arial,sans-serif;
 text-align: center;
 color: #FFFFFF;
 background-color: #FF6600;
}
```

La visualizzazione della pagina con il browser mostra i cambiamenti apportati.



I fogli di stile realizzano la **separazione tra layout e contenuti** nelle pagine Web: modificando gli stili, le pagine di un intero sito possono assumere un aspetto completamente diverso, pur mantenendo inalterati i contenuti.

Inoltre si possono adattare le pagine alle esigenze di specifiche categorie di utenti, per esempio si può creare un foglio di stile che assegna dimensioni più grandi ai testi di una pagina per utenti con difficoltà visive.

Un'altra funzionalità fondamentale dei fogli di stile è rappresentato dalle opzioni, riguardanti l'aspetto delle pagine, che non sarebbero possibili utilizzando i normali tag disponibili nel linguaggio HTML.

Per esempio, la proprietà **z-index** permette di definire quale elemento, tra due sovrapposti, deve stare sopra e quale sotto: l'oggetto con *z-index* minore si trova sotto all'oggetto con *z-index* maggiore.

Per esempio supponiamo di sovrapporre un testo all'immagine di un pulsante in modo da utilizzarlo per diverse funzioni.

Utilizzando il solo codice HTML possiamo inserire i due elementi (testo e immagine) ma non riusciremmo in nessun caso a sovrapporli. Utilizzando le definizioni di stile all'interno della pagina HTML, si ottiene la visualizzazione con i due elementi, testo e immagini, uno di seguito all'altro. Partiamo dall'immagine di un pulsante:



e sovrapponiamo il testo *INVIA*. Usando il linguaggio HTML si scrive il seguente frammento di pagina Web:

```
<DIV CLASS="zona_sotto">

</DIV>
<DIV CLASS="zona_sopra">
INVIA
</DIV>
```

Come si vede, ci sono due contenitori `<DIV>` chiamati *zona\_sotto* (che contiene l'immagine) e *zona\_sopra* (che contiene il testo). Se non si usa alcun foglio di stile CSS, il testo viene visualizzato dopo l'immagine del pulsante.



Definiamo ora lo stile:

```
.zona_sotto {
 position: absolute;
 top: 0px;
 left: 0px;
 width: 336px;
 height: 118px;
 z-index: 0;
}
.zona_sopra {
 position: absolute;
 top: 20px;
 left: 140px;
 z-index: 1;
 font-family: verdana;
 font-size: 35px;
 font-weight: bold;
 text-align: center;
 vertical-align: bottom;
 color: white;
}
```

Applicando lo stile si ottiene la visualizzazione in figura.



Lo schema seguente spiega il significato del codice utilizzato nel foglio di stile CSS.

| Codice                                                                             | Significato                                                               |
|------------------------------------------------------------------------------------|---------------------------------------------------------------------------|
| <pre>position: absolute; top: 20px; left: 140px;</pre>                             | Posizione dell'oggetto                                                    |
| <pre>width: 336px; height: 118px; z-index: 0; z-index: 1;</pre>                    | Dimensione dell'oggetto<br><br>L'oggetto sta sotto<br>L'oggetto sta sopra |
| <pre>font-family: verdana; font-size: 35px; font-weight: bold; color: white;</pre> | Stile e colore del testo                                                  |

Anche per i fogli di stile sono disponibili i **validatori** on line, in particolare il validatore del W3C si trova all'indirizzo <http://jigsaw.w3.org/css-validator/>. Una breve guida per l'uso operativo del validatore è contenuta nei *Materiali on line*.

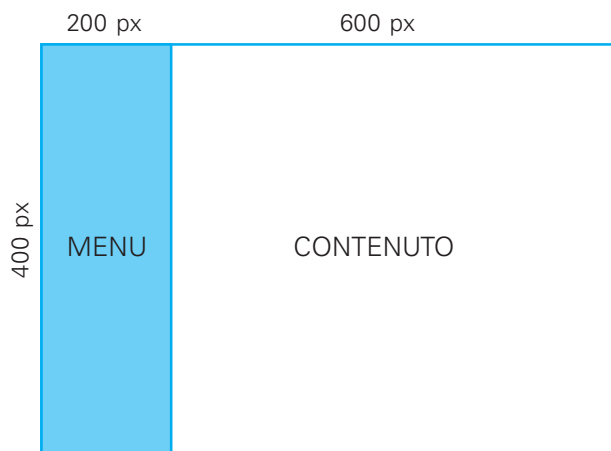
## PROGETTO 5

### Creare un sito sul poeta Ungaretti con informazioni sulla vita e le opere.

Il progetto utilizza i fogli di stile CSS per creare un semplice sito con un menu fisso posto a sinistra della pagina. La pagina è di dimensioni costanti (800x400 pixel) e divisa in due parti:

- a sinistra si trova il menu con larghezza 200 pixel;
- a destra si trovano i contenuti del sito in un riquadro di dimensioni 600x400.

Nel caso in cui il contenuto non stia interamente nella pagina vengono mostrate le barre di scorrimento.



A differenza del Progetto 4, facendo clic su una voce del menu, non viene aperta una nuova pagina, ma vengono visualizzati i contenuti nella zona a destra.  
Il menu è un elenco puntato ed è composto da tre voci:

- Vita
- Opere
- Poetica.

### Pagina Web (*index.html*)

```
<!DOCTYPE html>
<HTML>
<HEAD>
 <TITLE>Ungaretti</TITLE>
 <META CHARSET="utf-8">
 <LINK REL="stylesheet" HREF="stile.css" TYPE="text/css">
</HEAD>
<BODY>
 <DIV class="menu">
 <H1>Giuseppe Ungaretti</H1>

 Vita
 Opere
 Poetica

 </DIV>

 <DIV class="centro">
 <H1>La vita</H1>
 Nasce nel 1888 ad Alessandria d'Egitto da genitori lucchesi che vi si erano
 trasferiti in cerca di lavoro. Nel 1912 si trasferisce a Parigi per studiare
 alla Sorbona.

 Vive nella capitale francese molti anni dove matura le sue prime esperienze
 letterarie entrando in contatto con l'ambiente simbolista. Frequenta
 esponenti di spicco della classe intellettuale europea, tra i quali
 <I>Apollinaire</I>, <I>De Chirico</I>, <I>Modigliani</I>, <I>Picasso</I>,
 <I>Braque</I>. Nel 1914 partecipa come volontario alla Prima Guerra Mondiale.
 Nel 1915 pubblica le prime poesie e nel 1916 la sua prima raccolta di
 liriche, <I>Il porto sepolto</I> alla quale seguirà <I>Allegria di naufragi</
 I> nel 1919. Le due raccolte vengono poi riproposte in un unico volume dal
 titolo <I>L'Allegria</I> (1931). Al termine della guerra ritorna alcuni anni
 a Parigi lavorando per l'ambasciata italiana. Nel 1936 viene nominato docente
 di Letteratura italiana all'Università di San Paolo in Brasile dove rimane
 fino 1942.

 <H1>Le opere principali</H1>

 Natale, Napoli, 26 dicembre 1916;
 II Porto Sepolto, Stabilimento tipografico friulano, Udine, 1917;
 Allegria di naufragi, Vallecchi, Firenze, 1919;
 Il Porto Sepolto Stamperia Apuana, La Spezia, 1923;
 L'Allegria, Preda, Milano, 1931;
 Sentimento del Tempo, Vallecchi, Firenze, 1933;
 Il Dolore, Milano, 1947;
 Demiers Jours. 1919, Milano, 1947;
```

```

Gridasti: Soffoco..., Milano, 1950;
La Terra Promessa, Milano, 1950;
Un grido e Paesaggi, Milano, 1952;
Poesie disperse (1915-1927), Milano, 1959;
Il Taccuino del Vecchio, Milano, 1960;
Dialogo, Milano, 1968;
Vita d'un uomo. Tutte le poesie, Milano, 1969.

Fonte: <I>Wikipedia</I>

```

```

<H1>La Poetica</H1>
L'<I>Allegria</I> segna un momento chiave della storia della letteratura
italiana: Ungaretti rielabora in modo molto originale il messaggio formale
dei simbolisti (in particolare dei versi spezzati e senza punteggiatura dei
<I>Calligrammes</I> di <I>Guillaume Apollinaire</I>), coniugandolo con
l'esperienza atroce del male e della morte nella guerra. Al desiderio di
fraternità nel dolore si associa la volontà di ricercare una nuova "armonia"
con il cosmo che culmina nella citata poesia <I>Mattina</I> (1917), o in
<I>Soldati</I>. Questo spirito mistico-religioso si evolverà nella
conversione in <I>Sentimento del Tempo</I> e nelle opere successive, dove
l'attenzione stilistica al valore della parola (e al recupero delle radici
della nostra tradizione letteraria), indica nei versi poetici l'unica
possibilità dell'uomo, o una delle poche possibili, per salvarsi
dall'"universale naufragio".

Fonte: <I>Wikipedia</I>

```

```

</DIV>
</BODY>
</HTML>

```

Il documento è composto da due blocchi principali, chiamati *menu* e *centro*:

- *menu* contiene l'elenco puntato con le tre voci (*Vita*, *Opere* e *Poetica*)
- *centro* contiene il resto della pagina; ogni titolo di paragrafo (tag <H1>) è un'ancora che è la destinazione di un link del menu.

### Foglio di stile (*stile.css*)

```

.menu {
 position: absolute;
 top: 0;
 bottom: 0;
 left: 0;
 width: 200px;
 height: 400px;
 overflow: hidden;
 background: navy;
 color: white;
 padding: 5px;
}

.menu a {
 color: white;
}

```

```
.centro {
 position: fixed;
 top: 0;
 left: 200px;
 right: 0;
 bottom: 0;
 overflow: auto;
 width: 600px;
 height: 400px;
 background: white;
 padding: 5px;
}
```

Aperto il documento *index.html* nel browser, viene visualizzato il menu a sinistra su sfondo blu e il contenuto nella parte centrale (con le barre di scorrimento). Facendo clic su una voce del menu, la zona centrale scorre fino a visualizzare il contenuto desiderato.

## Giuseppe Ungaretti

- [Vita](#)
- [Opere](#)
- [Poetica](#)

## La vita

Nasce nel 1888 ad Alessandria d'Egitto da genitori lucchesi che vi si erano trasferiti in cerca di lavoro. Nel 1912 si trasferisce a Parigi per studiare alla Sorbona. Vive nella capitale francese molti anni dove matura le sue prime esperienze letterarie entrando in contatto con l'ambiente simbolista. Frequenta esponenti di spicco della classe intellettuale europea, tra i quali *Apollinaire*, *De Chirico*, *Modigliani*, *Picasso*, *Braque*. Nel 1914 partecipa come volontario alla Prima Guerra Mondiale. Nel 1915 pubblica le prime poesie e nel 1916 la sua prima raccolta di liriche, *Il porto sepolto* alla quale seguirà *Allegria di naufragi* nel 1919. Le due raccolte vengono poi riproposte in un unico volume dal titolo *L'Allegria* (1931). Al termine della guerra ritorna alcuni anni a Parigi lavorando per l'ambasciata italiana. Nel 1936 viene nominato docente di Letteratura italiana all'Università di San Paolo in Brasile dove rimane fino 1942.

## Le opere principali

- Natale, Napoli, 26 dicembre 1916;
- Il Porto Sepolto, Stabilimento tipografico friulano, Udine, 1917;
- Allegria di naufragi, Vallecchi, Firenze, 1919;

Lo schema seguente spiega il significato del codice utilizzato nel foglio di stile CSS.

| Codice                                 | Significato                                                                                                                                                                                                                                                                                                                                               |
|----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>.menu a {   color: white; }</pre> | Il menu ha lo sfondo blu quindi occorre specificare il colore dei link (che solitamente vengono visualizzati in azzurro), in questo caso bianco. Tuttavia gli eventuali link nella pagina non possono essere bianchi (visto che sarebbero invisibili). Il codice permette di specificare lo stile del tag <i>a</i> all'interno dell'oggetto <i>menu</i> . |
| <pre>overflow: hidden;</pre>           | Il contenuto che non sta nella pagina non viene visualizzato. Non vengono mostrate le barre di scorrimento.                                                                                                                                                                                                                                               |
| <pre>overflow: auto;</pre>             | Soltanto se necessario vengono mostrate le barre di scorrimento.                                                                                                                                                                                                                                                                                          |



Uno degli strumenti per l'interattività con gli utenti è la pagina per raccogliere suggerimenti o commenti dai visitatori, formata da moduli sul video da riempire con il nome del mittente, l'indirizzo di posta elettronica e il testo del commento. Queste pagine si chiamano **moduli di feedback**.

Il progetto seguente mostra un modulo di feedback realizzato con la versione **HTML5** in modo da effettuare la **validazione** dei dati. Inoltre per lo stile vengono utilizzate alcune caratteristiche disponibili dalla versione **CSS3**.

## PROGETTO 6

### Creare un modulo di feedback.

La pagina Web *modulo.html* utilizza le specifiche di stile descritte nel foglio *feedback.css*.

#### Pagina Web (*modulo.html*)

```
<!DOCTYPE html>
<HTML>
<HEAD>
 <META CHARSET="utf-8">
 <LINK REL="stylesheet" HREF="feedback.css" TYPE="text/css">
 <TITLE>Osservazioni e commenti</TITLE>
</HEAD>
<BODY>
<DIV ID="contenitore">
 <FORM METHOD="get" ACTION="invia.php" NAME="formCommenti" ID="commenti">
 <FIELDSET>
 <LEGEND>Inserisci i tuoi dati</LEGEND>
 <LABEL FOR="nome">Nome (obbligatorio)</LABEL>
 <INPUT TYPE="text" NAME="nome" ID="nome" PLACEHOLDER="Scrivi il tuo
nome" REQUIRED/>

 <LABEL FOR="cognome">Cognome</LABEL>
 <INPUT TYPE="text" NAME="cognome" ID="cognome" PLACEHOLDER="Scrivi il
tuo cognome"/>

 <LABEL FOR="email">Email (obbligatorio)</LABEL>
 <INPUT TYPE="email" NAME="email" ID="email" PLACEHOLDER="Scrivi la tua
Email" REQUIRED/>

 </FIELDSET>
 <FIELDSET>
 <LEGEND>Mandaci le tue osservazioni</LEGEND>
 <LABEL FOR="oggetto">Oggetto (obbligatorio)</LABEL>
 <INPUT TYPE="text" NAME="oggetto" ID="oggetto" PLACEHOLDER="Oggetto"
REQUIRED/>

 <LABEL FOR="messaggio">Messaggio</LABEL>
 <TEXTAREA COLS="40" ROWS="20" NAME="messaggio" ID="messaggio"
PLACEHOLDER="Il tuo messaggio"></TEXTAREA>

 </FIELDSET>
 <BUTTON TYPE="reset" NAME="reset" CLASS="bottone">Annulla</BUTTON>
 <BUTTON TYPE="submit" NAME="invia" CLASS="bottone">Invia</BUTTON>
 </FORM>
</DIV>
</BODY>
</HTML>
```

**Foglio di stile** (*feedback.css*)

```
label span {
 color: #0000FF;
 font-size: 85%;
}

#contenitore {
 width: 500px;
 height: 400px;
 margin: 0 auto;
}

#commenti {
 font-family: verdana;
 font-size: 100%;
 color: #000000;
}

#commenti label {
 float: left;
 display: block;
 margin: 3px;
 clear: both;
}

#commenti input {
 color: #000000;
 margin: 0 0 7px 0;
 border: 1px solid #CCCCCC;
 width: 220px;
 float: right;
 -moz-border-radius: 9px;
 -Webkit-border-radius: 9px;
 padding: 3px 10px;
}

#commenti select {
 color: #000000;
 margin: 0 0 7px 0;
 border: 1px solid #CCCCCC;
 width: 240px; float: right;
 -moz-border-radius: 9px;
 -Webkit-border-radius: 9px;
 padding: 3px 5px 3px 10px;
}

#commenti option {
 display: block;
 color: #666666;
}
```

```
#commenti textarea {
 color: #666666;
 float: right;
 font-size: 115%;
 font-family: verdana;
 height: 90px;
 border: 1px solid #CCCCCC;
 width: 220px;
 -moz-border-radius: 12px;
 -Webkit-border-radius: 12px;
 padding: 2px 10px;
 margin-bottom: 10px;
}

.bottone {
 clear: both;
 -moz-border-radius: 10px;
 -Webkit-border-radius: 10px;
 border: 0;
 cursor: pointer;
 margin-top: 15px;
 float: right;
 font-weight: bold;
 font-size: 110%;
 padding: 5px 15px;
 width: 200px;
}
```

I due file devono essere salvati nella stessa cartella.  
Aprendo la pagina Web con un browser, il modulo viene visualizzato come in figura.

The image shows a web form with two main sections. The first section, titled 'Inserisci i tuoi dati', contains three input fields: 'Nome (obbligatorio)' with placeholder text 'Scrivi il tuonome', 'Cognome' with placeholder text 'Scrivi iltuo cognome', and 'Email (obbligatorio)' with placeholder text 'Scrivi la tuaEmail'. The second section, titled 'Mandaci le tue osservazioni', contains two input fields: 'Oggetto (obbligatorio)' with placeholder text 'Oggetto' and a larger text area for 'Messaggio' with placeholder text 'Il tuo messaggio'. At the bottom right of the form are two buttons: 'Annulla' and 'Invia'.

Vediamo il significato del codice HTML della pagina Web (*form.html*).

```
<!DOCTYPE html>
<HTML>
<HEAD>
 <META CHARSET="utf-8">
 <LINK REL="stylesheet" HREF="feedback.css" TYPE="text/css">
 <TITLE>Osservazioni e commenti</TITLE>
</HEAD>
<BODY>
```

La prima parte è standard: si indica che il codice che segue è scritto in HTML, si specifica che il documento utilizza la codifica UTF-8 e che il foglio di stile è *feedback.css*. Il titolo della pagina è *Osservazioni e commenti*.

```
<DIV ID="contenitore">
```

Tutto il codice del modulo è contenuto in un tag `<DIV>` chiamato *contenitore*. In questo modo si possono specificare nel foglio di stile la posizione e le dimensioni di tutto il *form*.

```
<FORM METHOD="get" ACTION="invia.php" NAME="formCommenti" id="commenti">

<FIELDSET>
<LEGEND>Inserisci i tuoi dati</LEGEND>
```

Il tag **<SPAN>** viene utilizzato per assegnare lo stile a un testo con i CSS. Il tag `<SPAN>` da solo non effettua alcuna modifica allo stile, per esempio non va a capo come fanno `<P>` o `<DIV>`. Viene utilizzato quindi per applicare stili differenti che stanno sulla stessa linea. Il form invia i dati a una pagina Web lato server scritta con il linguaggio Php oppure ASP.NET o JSP: il nome del file (*invia.php*) è assegnato al parametro **ACTION** del form e i dati sono inviati con il metodo **get**.

(Il testo della pagina Php è riportato dopo la spiegazione della pagina Web e del foglio di stile CSS).

Il form è composto da due *fieldset*. Un **fieldset** è un gruppo di dati collegati logicamente. Il primo *fieldset* contiene i dati sull'utente, mentre il secondo il testo delle osservazioni.

```
<LABEL FOR="nome">Nome (obbligatorio)</LABEL>
<INPUT TYPE="text" NAME="nome" ID="nome" PLACEHOLDER="Scrivi il tuo
nome" REQUIRED/>

<LABEL FOR="cognome">Cognome</LABEL>
<INPUT TYPE="text" NAME="cognome" ID="cognome" PLACEHOLDER="Scrivi il
tuo cognome"/>

<LABEL FOR="email">Email (obbligatorio)</LABEL>
<INPUT TYPE="email" NAME="email" ID="email" PLACEHOLDER="Scrivi la tua
Email" REQUIRED/>
```

Per ogni oggetto si utilizza il tag **<LABEL>** (etichetta) e il tag **<INPUT>** che vengono collegati dall'attributo **FOR**. I primi due input sono di testo generico, mentre il terzo è un indirizzo email e quindi viene eseguita la validazione. Nel primo e nel terzo è specificato l'attributo **REQUIRED** e quindi l'utente è obbligato ad inserire i dati.

L'attributo **PLACEHOLDER** permette di mostrare un messaggio nella casella di testo come suggerimento per l'utente.

L'attributo *PLACEHOLDER* non è compatibile con Internet Explorer.

```
<FIELDSET>
<LEGEND>Mandaci le tue osservazioni</LEGEND>
 <LABEL FOR="oggetto">Oggetto (obbligatorio)</LABEL>
 <INPUT TYPE="text" NAME="oggetto" ID="oggetto" PLACEHOLDER="Oggetto"
 REQUIRED/>

 <LABEL for="messaggio">Messaggio</LABEL>
 <TEXTAREA COLS="40" ROWS="20" NAME="messaggio" ID="messaggio"
 PLACEHOLDER="Il tuo messaggio"></TEXTAREA>
</FIELDSET>
```

L'*oggetto* è un campo di testo generico (**INPUT TYPE="text"**) mentre il messaggio è un'*area di testo* (**TEXTAREA**).

Si osservi che l'attributo **NAME** specifica il nome dell'elemento del form (e il nome della variabile che viene passata alla pagina lato server), mentre l'attributo **ID** identifica univocamente l'elemento (utilizzato per i CSS). In questo modo due oggetti differenti, per esempio due caselle di testo, aventi le stesse caratteristiche di stile possono avere *name* diversi e *class* uguali.

```
<BUTTON TYPE="reset" NAME="reset" CLASS="bottone"/>Annulla</BUTTON>
<BUTTON TYPE="submit" NAME="invia" CLASS="bottone"/>Invia</BUTTON>
</FORM>
</DIV>
</BODY>
</HTML>
```

Gli ultimi elementi sono i due pulsanti *Annulla* e *Invia* realizzati con due oggetti **BUTTON**, uno di tipo **reset** e l'altro di tipo **submit**.

Nel file CSS (*feedback.css*) viene specificato lo stile dei vari componenti della pagina Web secondo la seguente sintassi.

```
label span {...}
```

descrive i tag generici **<LABEL>** e **<SPAN>**.

```
#commenti NOMETAG {
```

descrive lo stile del tag presente nel *form*.

```
#contenitore {
```

descrive lo stile del *contenitore*.

```
.bottone {
```

descrive l'identificatore *bottone* (che viene utilizzato per i due pulsanti).

Lo schema seguente spiega il significato del codice utilizzato nel foglio di stile CSS.

| Codice                                                                          | Significato                                                                                                                                                                                                                                                             |
|---------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>color:#0000FF;</code>                                                     | colore dell'oggetto (in esadecimale)                                                                                                                                                                                                                                    |
| <code>font-size:85%;</code>                                                     | dimensione del testo in percentuale rispetto alla dimensione predefinita nel browser; se l'utente modifica le dimensioni dei caratteri le proporzioni vengono rispettate;                                                                                               |
| <code>font-family:verdana;</code>                                               | font del carattere                                                                                                                                                                                                                                                      |
| <code>width:500px;</code><br><code>height:400px;</code>                         | dimensioni dell'oggetto                                                                                                                                                                                                                                                 |
| <code>margin:0 auto;</code>                                                     | in questo modo l'oggetto viene centrato                                                                                                                                                                                                                                 |
| <code>float:left;</code>                                                        | l'oggetto si posiziona il più possibile a sinistra                                                                                                                                                                                                                      |
| <code>display:block;</code>                                                     | l'elemento viene trattato come blocco (cioè non ammette altri oggetti a fianco)                                                                                                                                                                                         |
| <code>margin:3px;</code>                                                        | fissa la distanza (sopra, sotto, a destra e a sinistra) dagli altri oggetti                                                                                                                                                                                             |
| <code>margin:0 0 7px 0;</code>                                                  | fissa la distanza dagli altri oggetti:<br>superiore = 0    destro = 0    inferiore = 7 px    sinistro = 0                                                                                                                                                               |
| <code>border:1px solid #D8D8D8;</code>                                          | descrive il bordo:<br>spesso 1px    stile solido    di colore grigio (D8D8D8)                                                                                                                                                                                           |
| <code>-moz-border-radius:9px;</code><br><code>-webkit-border-radius:9px;</code> | definisce l'arrotondamento del bordo dell'oggetto (questa specifica potrebbe non essere compatibile con il browser in uso); <i>moz</i> e <i>Webkit</i> indicano famiglie di browser ( <i>moz</i> per <i>Mozilla</i> e <i>Webkit</i> per <i>Safari</i> e <i>Chrome</i> ) |

Creare un foglio di stile partendo da zero può essere complesso. Per capire meglio il codice è buona abitudine provare a fare degli esperimenti: se per esempio non riusciamo a capire cosa fa una porzione di codice, la si può cancellare o modificare e andare a vedere nel browser i cambiamenti nella pagina.

Dopo la pagina Web e il foglio di stile, occorre creare la pagina *invia.php* che tratti i dati inseriti dall'utente. Solitamente questo tipo di informazioni viene inviato a un indirizzo email oppure salvato in un database. Vediamo entrambe le soluzioni.

### Inviare i dati tramite mail

Si può accedere ai dati inseriti dall'utente utilizzando l'array associativo di Php **\$\_GET**:

```
$_GET["nome"]
$_GET["cognome"]
$_GET["mail"]
$_GET["oggetto"]
$_GET["messaggio"]
```

Il linguaggio *PHP* permette di inviare un messaggio email con il comando **mail**:

```
mail(destinatario, oggetto, testo, [intestazione]);
```

### Pagina Php (*invia.php*)

(Le parti in grassetto devono essere modificate opportunamente con lo specifico nome del sito e l'indirizzo email del destinatario).

```
<!DOCTYPE html>
<HTML>
```

```
<HEAD>
 <META CHARSET="utf-8">
 <TITLE>Invio dati</TITLE>
</HEAD>
<BODY>
<?php
$nome = $_GET["nome"];
$cognome = $_GET["cognome"];
$mail = $_GET["email"];
$oggetto = $_GET["oggetto"];
$messaggio = $_GET["messaggio"];

$intestazione = "MIME-Version: 1.0\r\n";
$intestazione .= "Content-type: text/html; charset=iso-8859-1\r\n";
$intestazione .= "From: <$mail>";

$oggetto = "Mail automatica dal form di www.prova.it - ".$oggetto;

$contento = $messaggio."
Inviato da: ".$cognome." ".$nome."
Mail:
".$mail."
Data: ".date("m/d/y H:i")."
 IP: ".$_SERVER['REMOTE_ADDR'];

if(mail("destinatario", $contento, $messaggio, $intestazione)) {
 echo "email inviata correttamente";
}
else {
 echo "ci sono stati degli errori, riprovare più tardi";
}
?>
</BODY>
</HTML>
```

La prima parte del codice recupera i dati inseriti dall'utente e inviati tramite il form. La variabile *\$intestazione* viene creata per poter inviare la mail in formato HTML. Poi si creano le variabili *\$oggetto* e *\$contento*.

- *\$oggetto* è la stringa di testo "Mail automatica dal form ..."
- *\$contento* è composto da
  - Il *messaggio* inserito dall'utente
  - Il suo *nome* e il *cognome*
  - La *mail*
  - La *data* e l'*ora* con la funzione

```
date("m/d/y H:i")
```

- L'*indirizzo IP* del mittente

```
$_SERVER['REMOTE_ADDR']
```

Il seguente frammento di codice invia la mail:

```
if(mail("destinatario", $contento, $messaggio, $intestazione)) {
 echo "email inviata correttamente";
}
else {
 echo "ci sono stati degli errori, riprovare più tardi";
}
```

Se l'operazione va a buon fine, viene mostrato il messaggio *email inviata correttamente*, altrimenti viene mostrato un messaggio di errore.

### Inserire i dati in un database

In alternativa è possibile effettuare una connessione a un database MySQL e inserire i dati in una tabella. Supponiamo che nel database chiamato *datiSito* ci sia una tabella *datiForm* formata dai seguenti campi:

- data
- IP
- email
- nome
- cognome
- oggetto
- messaggio.

### Pagina Php (*inviadb.php*)

(Le parti in grassetto devono essere modificate opportunamente con il nome del server e l'account dell'utente autorizzato ad accedere al database).

```
<!DOCTYPE html>
<HTML>
<HEAD>
 <META CHARSET="utf-8">
 <TITLE>Invio dati</TITLE>
</HEAD>
<BODY>
<?php
$nome = $_GET["nome"];
$cognome = $_GET["cognome"];
$mail = $_GET["email"];
$oggetto = $_GET["oggetto"];
$messaggio = $_GET["messaggio"];
$data = date("m/d/y H:i");
$IP = $_SERVER['REMOTE_ADDR'];

$hostname = "hostname";
$username = "username";
$password = "password";

$conn = mysql_connect($hostname, $username, $password);
if (! $conn){
 echo ("Errore durante la connessione a MySQL.");
 exit();
}
echo ("Connessione a MySQL effettuata con successo.");
mysql_select_db('datiSito', $conn);

$strSQL = "INSERT INTO datiForm (data, IP, email, nome, cognome, oggetto,
 messaggio) ";
$strSQL .= "VALUES ('$data', '$IP', '$email', '$nome', '$cognome', '$oggetto',
 '$messaggio')";
if (mysql_query($strSQL, $conn)) {
 echo "Dati memorizzati correttamente";
} else {
 echo "Attenzione, si è verificato un errore: " . mysql_error();
}
```



```
mysql_close($conn);
?>
</BODY>
</HTML>
```

I dati da trattare sono simili al caso precedente. Aniché creare il messaggio di posta elettronica, si effettua una connessione al database.

```
$hostname = "hostname";
$username = "username";
$password = "password";

$conn = mysql_connect($hostname, $username, $password);
if (! $conn){
 echo ("Errore durante la connessione a MySQL.");
 exit();
}
echo ("Connessione a MySQL effettuata con successo.");
```

La riga seguente seleziona il database:

```
mysql_select_db('datiSito', $conn);
```

Il comando SQL inserisce una nuova riga nella tabella del database con i dati ricevuti dal form della pagina HTML:

```
$strSQL = "INSERT INTO datiForm (data, IP, email, nome, cognome, oggetto,
 messaggio) ";
$strSQL .= "VALUES ('$data', '$IP', '$email', '$nome', '$cognome', '$oggetto',
 '$messaggio')";
if (mysql_query($strSQL, $conn)) {
 echo "Dati memorizzati correttamente";
} else {
 echo "Attenzione, si è verificato un errore: " . mysql_error();
}
```

Se l'operazione va a buon fine viene mostrato il testo *Dati memorizzati correttamente*, altrimenti viene visualizzato un messaggio di errore.

I form nelle pagine Web possono essere utilizzati anche per l'identificazione degli utenti nell'accesso ad aree riservate dei siti (moduli per il **login**): normalmente si devono fornire nome utente (*username*) e *password*. In questo caso l'attributo **ACTION** del form richiama l'attivazione di un programma sul server per il controllo e l'autenticazione dell'utente.



#### AUTOVERIFICA

Domande da 18 a 20 pag. 592

Problemi da 22 a 23 pag. 594-595



#### MATERIALE ONLINE

### 5. Validatori dei fogli di stile CSS

### 6. Software per creare fogli di stile CSS

## 7 Accessibilità e usabilità

I termini **accessibilità** e **usabilità**, utilizzati nel contesto della progettazione di documenti multimediali e soprattutto di siti Web, si riferiscono alle regole da seguire per realizzare contenuti che siano facilmente fruibili da tutte le persone, comprese quelle con disabilità fisiche. Tuttavia i due termini hanno significati e obiettivi diversi.

Costruire un sito **accessibile** significa rispettare alcune norme e raccomandazioni sviluppate all'interno del progetto **WAI** (*Web Accessibility Initiative*), promosso dal **W3C** (*World Wide Web Consortium*, l'ente che regola il Web e i suoi standard tecnologici). Le informazioni e i documenti sul progetto si trovano all'indirizzo <http://www.w3.org/WAI/>.

I gruppi di lavoro del WAI hanno rilasciato negli anni diversi documenti contenenti le specifiche da seguire per creare siti Web accessibili al maggior numero di utenti possibile.

Nel 1999 sono state pubblicate le prime linee-guida ufficiali del WAI, raccolte in un documento chiamato **WCAG 1.0** (*Web Content Accessibility Guidelines*), e nel 2008 è seguita la seconda versione del documento, la **2.0**, riveduta e aggiornata dal punto di vista tecnologico e con una ricca documentazione a supporto.

Le guide WCAG forniscono uno standard per l'accessibilità dei contenuti Web che possa essere condiviso da tutti: singoli, società, organizzazioni, governi.

Le linee guida sono indirizzate sia ai *Web content developers* (sviluppatori di contenuti Web), sia agli sviluppatori di *authoring tools*, cioè di software per generare pagine, documenti e contenuti per il Web.

L'idea alla base del lavoro del WAI è che gli utenti possono visualizzare uno stesso sito Web in modo differente (a causa di una disabilità o di un sistema operativo o browser diverso, o con un mezzo portatile), ma il contenuto deve essere leggibile e chiaro per tutti.

I contenuti dovrebbero essere resi accessibili in modo particolare alle persone con disabilità fisiche (momentanee o permanenti) o cognitive, apportando una maggiore facilità nella visualizzazione, nella navigazione e nella fruizione dei contenuti, e agli utenti con computer datati e poco potenti o con dispositivi portatili con schermi molto piccoli.

La prima versione del WCAG prevedeva che i siti Web fossero progettati tenendo conto delle linee guida divise secondo due criteri:

1. assicurare un'elegante trasformazione della pagina, che deve essere riconosciuta correttamente da qualsiasi browser;
2. creare contenuti comprensibili e navigabili.

Per il primo aspetto, le raccomandazioni vertono sulla separazione dei contenuti dalla formattazione, attraverso l'uso del linguaggio HTML per il contenuto e dei CSS per la rappresentazione (come abbiamo visto in precedenza), sull'inserimento di testi alternativi per le immagini, con la descrizione di ciò che nell'immagine è rappresentato, e sulla possibilità di fruire di un contenuto attraverso più sensi, come la presenza di testi con le trascrizioni di audio e video per permetterne la fruizione a utenti con disabilità visive o uditive.

Il secondo punto si riferisce alla necessità di usare un linguaggio semplice, che possa essere compreso da tutti, e di inserire nelle pagine strumenti di navigazione chiari, sia per scorrere i contenuti all'interno di una pagina, sia per navigare tra le pagine di un sito.

Nella versione 2.0 tutte le linee guida sono state ricollocate all'interno di 4 principi di accessibilità: *Percepibile*, *Utilizzabile*, *Comprensibile* e *Robusto*.

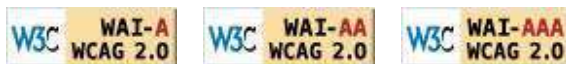
- **Percepibile:** le informazioni devono essere chiare e riconoscibili da tutti gli utenti attraverso tutti i sensi. È necessario fornire testi alternativi alle immagini, inserire sottotitoli e trascrizioni per i contenuti audio/video, rendere il contenuto interpretabile da tecnologie assistive (strumenti hardware e software, come *screen reader* o lettori Braille, che consentono a chi ha difficoltà motorie o sensoriali di poter accedere alle informazioni visualizzate da strumenti informatici) e usare un contrasto sufficiente perché le informazioni siano leggibili e chiare.

- **Utilizzabile:** l'interfaccia deve poter essere usata da chiunque e non può richiedere operazioni che l'utente non sia in grado di svolgere (persone diverse hanno necessità diverse). Per questo è necessario che tutte le funzionalità siano accessibili da tastiera, che ci siano strumenti di navigazione adeguati, che ci sia tempo sufficiente per leggere o utilizzare il contenuto.
- **Comprensibile:** contenuti e interfaccia devono poter essere compresi e usati senza difficoltà, il testo deve essere leggibile e non devono verificarsi comportamenti imprevedibili dei contenuti o dell'interfaccia.
- **Robusto:** il contenuto deve essere interpretabile a prescindere dallo strumento e dalla tecnologia usata per la visualizzazione. È importante quindi che il sito Web sia compatibile con le tecnologie presenti e future, con i browser e i sistemi operativi più diffusi e che sia portabile, cioè sia visualizzato correttamente su qualsiasi dispositivo.

Tutte le linee guida sono state suddivise in tre livelli di conformità (*success criteria*), che rappresentano i tre possibili livelli di applicazione delle regole del WCAG.

Il raggiungimento del primo livello di conformità viene contrassegnato con una **A**, mentre il secondo e il terzo livello di conformità sono contraddistinti rispettivamente da **AA** e **AAA**.

Ottenuta la validazione, è possibile inserire nel proprio sito Web il logo corrispondente al livello di aderenza alle specifiche.



In Italia dal 2004 è in vigore una legge riguardante l'accessibilità, "*Disposizioni per favorire l'accesso dei soggetti disabili agli strumenti informatici*", nota come *Legge Stanca* dal nome del ministro promotore (*Legge 4 del 2004*). Essa recepisce ampiamente le indicazioni contenute nelle guide del W3C.

Per sottoporre un sito Web alla validazione sono disponibili numerose risorse, in particolare i **validatori** del W3C:

<http://validator.w3.org/>

Sono poi disponibili in rete numerosi tool e strumenti specifici per l'accessibilità, relativi anche alla validazione cromatica e alla validazione dei contenuti multimediali.

A differenza dell'accessibilità, l'**usabilità** non prevede regole precise: l'obiettivo è che un sito risponda alle esigenze della tipologia di utenza per cui è stato creato.

L'idea di usabilità dei siti Web si esprime in una teoria, chiamata **Web usability**, caratterizzata dalla presenza dell'utente finale al centro del processo di progettazione di un sito Web.

Non ci sono linee guida predefinite: si deve porre attenzione a creare un prodotto che risponda alle aspettative di chi lo userà, con un'interfaccia in cui l'utente sia in grado di esplorare i contenuti senza difficoltà e senza dover acquisire delle abilità e delle conoscenze che non ha. L'attività dell'utente, quindi, deve essere concentrata sulle informazioni che cerca, non sul modo per trovarle: in un sito usabile, l'utente ha ben chiari quali siano gli strumenti che ha a disposizione, per esempio i pulsanti, conosce la loro posizione, e non vi sono ambiguità legate alle azioni che può compiere facendo clic su di essi.

L'utente dovrebbe essere coinvolto durante la fase di progettazione e poi di testing del sito Web: interviste, sondaggi, simulazioni e questionari sono strumenti utili per avere un riscontro diretto da un campione degli utenti che useranno il sito e grazie ai quali il progetto può diventare **ergonomico**.

Per questo motivo il percorso di un sito verso l'usabilità non si esaurisce con l'applicazione di regole, ma è in continua evoluzione: solo seguendo i *feedback* degli utenti e riparando gli errori, i siti Web possono migliorare e diventare più ergonomici.

Usando un software o un sito Web, un utente si costruisce un modello mentale del prodotto che sta usando e quindi si crea delle aspettative sul suo funzionamento, per esempio sulla posizione delle informazioni o dei pulsanti. Se queste aspettative vengono confermate, il sito può dirsi *ergonomico*, perché non impone la modifica del proprio modello mentale e della propria aspettativa.

Un altro aspetto dell'usabilità di un sito Web riguarda la progettazione e la gestione del sito stesso da parte del *Web designer*. Perché sia usabile non solo dal lato dell'utente, ma anche dal lato dello sviluppatore, un sito Web deve essere accompagnato da istruzioni e note, cioè dalla documentazione del progetto che permetta a un altro progettista di poter continuare il lavoro fatto, effettuare modifiche, aggiornamenti e la manutenzione necessaria senza dover rivedere il codice dall'inizio o andare a tentativi.

Dal punto di vista aziendale l'usabilità nei siti Web è un fattore determinante per la realizzazione della vendita e quindi del business. Il sito Web rappresenta una vetrina della **brand identity** (identità della marca) di un'azienda che deve contraddistinguersi anche attraverso un'interfaccia curata e di effetto, tramite elementi di design e soluzioni tecnologiche. Si tratta di due logiche opposte: usabilità e funzionalità da un lato, immagine ed estetica dall'altro. Compito del *Web designer* è conciliare le esigenze del business aziendale con la realizzazione di un sito ergonomico e usabile.

#### AUTOVERIFICA

Domande da 21 a 23 pag. 593

Problemi da 24 a 26 pag. 595



#### MATERIALI ONLINE

**7. Linee guida e norme per l'accessibilità**

**8. Tool per l'accessibilità**



#### CREAZIONE DI SITI WEB

Come si è visto nei paragrafi precedenti le tecnologie Web permettono di costruire un insieme di pagine di informazioni con le caratteristiche dell'interfaccia grafica, collegabili tra loro attraverso i link: in sostanza possiamo realizzare **ipertesti**, con un vantaggio aggiuntivo offerto dalla possibilità di pubblicare l'ipertesto sulla rete Internet, rendendolo visibile a un vasto pubblico di utenti, e di stabilire collegamenti anche con le pagine e le informazioni presenti in altri siti nel mondo.

Ovviamente l'attività di progettazione del sito Web deve avere la massima attenzione: occorre infatti definire con precisione gli obiettivi e i contenuti di ciò che si vuole comunicare, prima di passare all'implementazione sul computer e alla scelta degli strumenti software e multimediali da utilizzare.

Dal punto di vista operativo, la creazione di un sito Web richiede le seguenti *fasi di lavoro*:

1. definire obiettivi e struttura del sito
2. scrivere i testi o selezionare testi già esistenti
3. acquisire immagini con lo scanner o selezionarne di già esistenti
4. determinare quali oggetti multimediali sono importanti per la comunicazione
5. costruire le pagine usando un programma di *Web editor*
6. validare la sintassi delle pagine e del foglio di stile
7. validare il funzionamento delle pagine con il browser
8. verificare i requisiti di accessibilità e usabilità
9. trasferire le pagine su un server Internet.

Al termine del lavoro si può accedere alle pagine tramite la rete: in sostanza otteniamo un **sito Internet**.

Naturalmente un sito Web può essere consultato anche da un computer non connesso alla rete, usando il browser in modalità *Non in linea*, oppure può essere messo a disposizione degli utenti di una rete locale che utilizza le tecnologie Internet (rete *Intranet*).

La pagina di partenza del sito, detta *indice* o *home page*, assume il nome **index.html**: questa è la pagina che si apre quando l'utente Internet scrive nella casella del browser l'indirizzo Web (URL) del sito dove si trovano le pagine.

Poiché questa pagina è quella che viene vista per prima dal visitatore di un sito, è opportuno che contenga sommari, titoli ed elementi grafici che devono suscitare attrattiva e interesse, invogliando l'utente a guardare le altre pagine del sito.

Un'altra buona regola di carattere generale consiste nell'evitare pagine contenenti immagini di grandi dimensioni oppure piccole ma in numero elevato: esse provocano lunghi tempi di attesa per il caricamento delle pagine e una perdita di interesse nell'utente che si è collegato al sito.

I siti Internet possono essere raggiunti da persone che usano computer con sistemi operativi diversi e apparecchiature hardware diverse. Il linguaggio HTML garantisce la visualizzazione corretta con diversi tipi di computer. Però ci possono essere degli inconvenienti quando per scrivere le pagine si usano tipi di font poco standard, che non sono installati sul computer dell'utente, oppure quando il monitor usa una risoluzione grafica diversa da quella che abbiamo usato nella progettazione delle pagine. Perciò è bene usare tipi di caratteri con i font più diffusi e specificare nella *Home Page* la risoluzione video più adatta (per esempio 1024x768) per la migliore visualizzazione delle pagine.

Tutti i siti Internet contengono alcuni elementi standard:

- il logo e il sommario nella *Home Page*
- un link alla *Home Page* nelle altre pagine
- una guida all'uso del sito o una mappa dell'intero sito
- l'indicazione degli eventuali collaboratori o sponsor
- un programma di ricerca all'interno del sito
- una pagina di link ad altri siti di interesse generale o che trattano argomenti simili
- un modulo con il quale l'utente può inviare richieste, messaggi o osservazioni (*feedback*).

Un sito Internet è raggiungibile da qualsiasi nazione del pianeta: perciò, se esso contiene informazioni di interesse per un pubblico vasto ed eterogeneo, è meglio produrre due versioni delle pagine, in italiano e in inglese, offrendo all'utente la possibilità di scelta della lingua a partire dalla pagina iniziale.



#### **AUTOVERIFICA**

Problemi da 27 a 29 pag. 595

## DOMANDE

### Siti Web e oggetti multimediali

- 1 Quali tra i seguenti sono formati per le immagini nel Web?
  - a) pdf
  - b) gif
  - c) doc
  - d) jpg
  - e) txt
  
- 2 Associa alle estensioni della colonna di sinistra il corrispondente tipo di file scegliendo dalla colonna di destra.

|         |                       |
|---------|-----------------------|
| a) txt  | 1) file musicale      |
| b) doc  | 2) video              |
| c) png  | 3) immagine compressa |
| d) midi | 4) documento di testo |
| e) avi  | 5) Microsoft Word     |
  
- 3 Associa a ciascun termine della colonna di sinistra il corrispondente significato scegliendo dalla colonna di destra.

|                 |                                   |
|-----------------|-----------------------------------|
| a) Lossless     | 1) con perdita di informazioni    |
| b) Archivio     | 2) file contenenti file compressi |
| c) Lossy        | 3) senza perdita di informazioni  |
| d) Compressione | 4) formato di file compresso      |
| e) Zip          | 5) diminuzione di dimensioni      |
  
- 4 Quali delle seguenti affermazioni sono vere (V) e quali false (F)?

|                                                                    |                                                       |
|--------------------------------------------------------------------|-------------------------------------------------------|
| a) L'estensione è associata al tipo di file                        | <input type="checkbox"/> V <input type="checkbox"/> F |
| b) L'estensione determina con quale programma verrà aperto un file | <input type="checkbox"/> V <input type="checkbox"/> F |
| c) L'estensione indica la grandezza di un file                     | <input type="checkbox"/> V <input type="checkbox"/> F |
| d) Per aprire alcuni file sono necessari programmi particolari     | <input type="checkbox"/> V <input type="checkbox"/> F |
  
- 5 Quale programma si usa per trasferire i file contenenti le pagine Web dal proprio computer al server Internet dove devono essere pubblicate?
  - a) Un programma di posta elettronica
  - b) Un programma browser
  - c) Un programma di compressione
  - d) Un programma di FTP

### Linguaggio HTML, formattazione e multimedia

- 6 Qual è l'ordine dei tag che definiscono la struttura fondamentale della pagina HTML?
  - a) BODY, HEAD, HTML
  - b) HEAD, HTML, BODY
  - c) HTML, HEAD, BODY
  - d) HTML, BODY, HEAD

- 7 Quali delle seguenti porzioni di codice HTML sono prive di errori?
- `<HTLM><BODY>ciao</BODY><HTLM>`
  - `<HTML BODY>ciao</BODY /HTML>`
  - `<HTML><BODY>ciao</BODY></HTML>`
  - `<HTML><BODY>ciao</HTML></BODY>`
  - `<HTML></HTML>`
- 8 Quale delle seguenti frasi HTML visualizza la scritta "ciao **a tutti voi**".
- ciao `<B>a tutti <l> voi`
  - ciao `<l> a tutti </l> <b> voi </B>`
  - ciao `<B>a tutti</B> <l>voi</l>`
  - ciao `<l> a tutti <B> voi`
  - ciao `<l><B> a tutti </B></l>`
- 9 Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
- `<l>` significa "scrivi in corsivo"
  - `<B>` significa "grassetto"
  - `</B>` significa "grassetto"
  - Il codice `<l><B>ciao</B></l>` genera un errore
- 10 Associa a ciascun colore della colonna di sinistra il corrispondente codice RGB scegliendo dalla colonna di destra.
- |           |             |
|-----------|-------------|
| a) nero   | 1) FF FF FF |
| b) bianco | 2) 00 00 FF |
| c) rosso  | 3) 00 FF 00 |
| d) verde  | 4) 00 00 00 |
| e) blu    | 5) FF 00 00 |
- 11 Quale tra le seguenti è la sintassi corretta del tag che definisce un link ipertestuale in HTML?
- `</A HREF = "....."> parola <A>`
  - `<A HREF = "....."> parola </A>`
  - `<A URL = "....."> parola </A>`
  - `</A URL = "....."> parola <A>`
- 12 Quali delle seguenti affermazioni sono vere (V) e quali false (F)?
- Il tag `<A>` crea un link
  - Il tag `<LINK>` crea un link ipertestuale
  - È possibile creare un link a un indirizzo e-mail
  - È possibile creare un link a un punto specifico di una pagina
- 13 Quale tra le seguenti frasi HTML consente di impostare un'immagine come link a un'altra pagina?
- `<IMG ><A HREF="pagina.htm"><SRC="immagine.jpg"></A>.`
  - `<IMG SRC="immagine.jpg"><A HREF="pagina.htm"></A>.`
  - `<A HREF=<IMG SRC=" immagine.jpg">pagina.htm</A>.`
  - `<A HREF="pagina.htm"><IMG SRC=" immagine.jpg"></A>.`



- 14** Quale dei seguenti tag produce una lista puntata in HTML?
- a) `<UL><LI> ... </LI></UL>`
  - b) `<UL><LO> ... </LO></UL>`
  - c) `<UL><OL> ... </OL></UL>`
  - d) `<UL><IL> ... </IL></UL>`
- 15** Quale delle seguenti porzioni di codice HTML produce una tabella con le lettere A e B su 1 riga e 2 colonne?
- a) `<TABLE><TR><TD>A</TD></TR><TR><TD>B</TD></TR></TABLE>`
  - b) `<TABLE><TD><TR>A</TR></TD><TD><TR>B</TR></TD></TABLE>`
  - c) `<TABLE><TR><TD>A</TD><TD>B</TD></TR></TABLE>`
  - d) `<TABLE><TR><TR>A</TR><TR>B</TR></TR></TABLE>`

### I form e l'interazione con l'utente

- 16** Quali delle seguenti affermazioni sono vere (V) e quali false (F)?

- a) Un form permette di trasferire dati dall'utente al Web
- b) Un form permette di eseguire semplici programmi
- c) Un form permette di inviare dati da una pagina a un'altra
- d) Tutti i siti devono contenere un form
- e) Tutti i browser supportano il tag `<FORM>`



- 17** Associa agli oggetti grafici di una pagina Web della colonna di sinistra il corrispondente significato scegliendo dalla colonna di destra.
- |             |                      |
|-------------|----------------------|
| a) form     | 1) annulla           |
| b) textarea | 2) invia             |
| c) select   | 3) area di testo     |
| d) reset    | 4) modulo            |
| e) submit   | 5) casella combinata |

### Fogli di stile CSS

- 18** Quale tra i seguenti è il tag che definisce un foglio di stile incorporato?
- a) `<TYPE STYLE = "text/css"> ..... </TYPE>`
  - b) `<CSS TYPE="text/css"> ..... </CSS >`
  - c) `<STYLE SHEET="text/css"> ..... </STYLE >`
  - d) `<STYLE TYPE="text/css"> ..... </STYLE >`
- 19** Quali dei seguenti sono attributi per utilizzare i selettori di un foglio di stile?
- a) link
  - b) class
  - c) id
  - d) type
- 20** Che cosa è un modulo di feedback?
- a) La ricevuta di un pagamento di commercio elettronico.
  - b) La ricevuta dell'avvenuta ricezione di un messaggio di posta elettronica.
  - c) Il modulo per acquistare una connessione a Internet.
  - d) Una pagina per raccogliere suggerimenti o commenti dai visitatori di un sito Internet.



## Accessibilità e usabilità

- 21** Chi ha promosso il progetto WAI (Web Accessibility Initiative)?
- a) Un gruppo di Web designer indipendenti.
  - b) Il CERN di Ginevra.
  - c) Il W3C.
  - d) Il governo degli Stati Uniti.
- 22** Quale delle seguenti frasi esprime meglio la definizione di accessibilità?
- a) La capacità dei sistemi informatici di interagire con computer aventi sistemi operativi diversi.
  - b) La capacità dei sistemi informatici di controllare l'accesso degli utenti tramite username e password.
  - c) La capacità dei sistemi informatici di erogare servizi e fornire informazioni anche a persone con disabilità fisiche.
  - d) La capacità dei sistemi informatici di erogare servizi e fornire informazioni in tempi molto rapidi.
- 23** Come deve essere un sito Web accessibile secondo le specifiche WCAG?
- a) percepibile, utilizzabile, comprensibile, robusto
  - b) visibile, navigabile, comprensibile, multilingua
  - c) scalabile, leggibile, orientabile, conforme
  - d) distinguibile, solido, comprensibile, validato
  - e) riconoscibile, semplice, modificabile, ottimizzato

## PROBLEMI

### Siti Web e oggetti multimediali

- 1** Verificare la presenza sul proprio computer degli strumenti software per creare e pubblicare pagine Web.
- 2** Aprire una pagina Web e verificare il formato degli oggetti multimediali in essa contenuti.
- 3** Individuare sul disco del proprio computer file nei diversi formati (testi, immagini, suoni, video).
- 4** Controllare se il browser può visualizzare documenti *pdf* e animazioni *Flash*.

### Linguaggio HTML, formattazione e multimedia

- 5** Ricercare sul disco del computer i file in formato HTML, aprirne uno e controllarne il codice HTML.
- 6** Preparare una pagina HTML con i tag principali che compaiono in tutte le pagine (HTML, HEAD, BODY) lasciando alcune righe bianche da riempire successivamente, in modo da poterla usare come base di lavoro, senza dover riscrivere ogni volta tutti i tag.
- 7** Creare tre pagine HTML aventi colori di sfondo diversi.
- 8** Preparare una pagina Web con un titolo principale e due paragrafi. Ogni paragrafo ha un sottotitolo.

- 9 Creare due pagine HTML con qualche riga di testo: nella prima una parola diventa link ipertestuale alla seconda pagina; nella seconda un altro link consente di tornare alla prima.
- 10 Inserire all'interno della pagina i collegamenti ipertestuali a due siti Internet di cui si conosce l'indirizzo nel formato URL.
- 11 Realizzare una pagina Web contenente un indice con i nomi di tre squadre di calcio. Facendo clic sul nome di una squadra, si apre una nuova pagina con una breve descrizione della squadra.
- 12 Organizzare gli indirizzi dei siti preferiti o più usati in un testo scritto con Word. Trasformare poi il testo in una pagina Web di bookmark per accedere velocemente ai siti elencati.
- 13 Suddividere un testo in parti, assegnando a ciascuna parte un sottotitolo. Copiare i sottotitoli all'inizio del testo per formare un sommario di link ipertestuali, che consentano di saltare direttamente alla parte scelta, all'interno del documento, identificata con un'àncora (tag `<A NAME = ".....">`). Alla fine di ogni parte, un link deve dare la possibilità di ritornare all'inizio della pagina per vedere il sommario.
- 14 Costruire una rubrica di indirizzi di posta elettronica di amici e conoscenti con un testo Word. Trasformare poi il testo in una pagina Web di indirizzi email che consenta di inviare velocemente messaggi di posta con il tag `<A HREF="mailto: .....">`.
- 15 Costruire in una pagina Web la lista puntata con i nomi delle città capoluogo di una Regione. Modificare poi il codice HTML per ottenere una lista numerata.
- 16 Costruire una pagina Web contenente una tabella con l'elenco degli studenti di una classe e i voti ottenuti nell'ultima prova di Informatica.
- 17 Creare tre pagine Web aventi colori di sfondo diversi e contenenti ciascuna un'immagine. Inserire all'interno della prima pagina una figura che diventa link alla seconda pagina.
- 18 Creare una pagina Web e inserire in essa i tag per riprodurre un audio e per visualizzare un video.

### I form e l'interazione con l'utente

- 19 Preparare, all'interno di una pagina Web, un modulo formato da caselle di testo o altri campi modulo, per chiedere agli utenti che visitano il sito alcune informazioni sui loro gusti personali riguardo ai cibi.
- 20 Predisporre un modulo per l'identificazione dell'utente che accede a un sito Web con le caselle per il nome utente e la password, oltre ai pulsanti *Invia* e *Annulla*.
- 21 Effettuare il collegamento al sito delle Ferrovie dello Stato ([www.trenitalia.it](http://www.trenitalia.it)) e verificare il funzionamento dei form per la richiesta di informazioni sugli orari ferroviari.

### Fogli di stile CSS

- 22 Preparare alcune pagine Web per illustrare le caratteristiche di una città, dopo averne progettato la struttura, usando il foglio di stile per definire il layout e la visualizzazione dei contenuti.

- 23** Progettare la struttura delle pagine Web per descrivere i componenti di un personal computer, predisporre il foglio di stile e costruire le pagine.

### **Accessibilità e usabilità**

- 24** Acquisire da Internet la documentazione sugli standard internazionali e sulle normative nazionali relative all'accessibilità e all'usabilità dei siti Web.
- 25** Ricercare in Internet gli strumenti per la validazione dell'accessibilità.
- 26** Controllare il livello di accessibilità di una pagina Web, utilizzando i validatori disponibili in Internet, e verificare le eventuali difformità rispetto alle linee guida.

### **Siti Web**

- 27** Effettuare il collegamento al sito del Corriere della sera ([www.corriere.it](http://www.corriere.it)) e individuare all'interno della home page gli elementi caratteristici delle pagine Web (logo, banner, menu, forum, casella per la ricerca di parole, ...).
- 28** Cercare in Internet le modalità e i costi per aprire un sito Web.
- 29** Attivare lo spazio per la pubblicazione di pagine Web che i Provider della connessione a Internet mettono a disposizione gratuitamente per gli utenti registrati.



## SCHEDA DI AUTOVALUTAZIONE

### CONOSCENZE

- ☐ Sito Web
- ☐ Formati dei file di testo, immagini e oggetti multimediali
- ☐ Linguaggio HTML
- ☐ Tag del linguaggio
- ☐ Struttura generale di una pagina HTML
- ☐ Attributi dei tag
- ☐ Formattazione del testo
- ☐ Titoli, sottotitoli, paragrafi
- ☐ Elenchi puntati e numerati
- ☐ Tabelle
- ☐ Collegamenti ipertestuali
- ☐ Immagini, audio, video
- ☐ Moduli per l'interazione con l'utente
- ☐ Fogli di stile CSS
- ☐ Fogli di stile in linea, incorporati, collegati
- ☐ Selettore, classe, identificatore
- ☐ Accessibilità e usabilità

### ABILITÀ

- ☐ Riconoscere i formati standard per testi, immagini e oggetti multimediali
- ☐ Visualizzare una pagina Web in modalità off-line
- ☐ Visualizzare il codice HTML di una pagina Web
- ☐ Creare una semplice pagina HTML
- ☐ Inserire titoli e sottotitoli
- ☐ Inserire un paragrafo
- ☐ Inserire una barra orizzontale
- ☐ Creare elenchi puntati e numerati
- ☐ Inserire una tabella
- ☐ Creare un collegamento
- ☐ Inserire un'immagine in una pagina
- ☐ Inserire un video o un suono
- ☐ Creare pagine con i form
- ☐ Inserire fogli di stile in linea, incorporati, collegati
- ☐ Creare una classe
- ☐ Definire un identificatore
- ☐ Validare l'accessibilità di un sito Web



**SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 663**

# Programmazione Web con JavaScript e HTML5

## 1 Il linguaggio JavaScript

All'interno di una pagina Web, oltre ai tag HTML, si possono incorporare comandi organizzati in funzioni che vengono poi interpretati ed eseguiti dal browser al momento dell'apertura della pagina: si ottengono così i *JavaScript*, cioè *script* che usano parole e sintassi simili al *linguaggio Java*.

**JavaScript** è un *linguaggio interpretato*, cioè non viene compilato, e ogni comando viene eseguito passo passo direttamente dal *browser*. In questo caso non si utilizza il termine *programma* ed è più corretto usare il termine *script*.

Il termine **script** in informatica indica, in generale, un testo che, anziché contenere una descrizione o un messaggio, è formato dai comandi di un linguaggio, organizzati in una sequenza o in una selezione del tipo *se ... allora ...* oppure in una ripetizione.

I JavaScript possono essere inseriti nelle pagine HTML per questi scopi:

- gestire gli eventi che possono accadere durante la consultazione delle pagine, provocati dall'utente con il clic del mouse o dalla navigazione all'interno di un sito;
- fornire messaggi di avvertimento o risposte alle scelte effettuate;
- acquisire le scelte dell'utente all'interno di un elenco e lanciare l'esecuzione dell'azione conseguente;
- controllare la validità dei dati inseriti in un modulo (*form*);
- apportare modifiche all'aspetto di una pagina Web durante la sua visualizzazione con il browser.

Gli script consentono di realizzare pagine Web con contenuti dinamici dal **lato client**, perché l'interpretazione dei comandi viene fatta direttamente dal browser Web nel computer dell'utente (*client*), a differenza delle pagine Web **lato server** (scritte per esempio usando i linguaggi *ASP.NET* o *Php*) che contengono codice eseguito sul server con invio del risultato dell'elaborazione al browser dell'utente.

I linguaggi ASP.NET e Php, con le modalità operative per la realizzazione di pagine Web lato server, compreso l'accesso ai database residenti sul server, sono presentati nel testo:  
*A. Lorenzi, E. Cavalli – Progettazione dei database, Linguaggio SQL, Dati in rete (Access, MySQL, Pagine ASP, Pagine Php) – Atlas – ISBN 978-88-268-1675-3*

Di solito i JavaScript sono costituiti da una o più procedure, dette *function*, scritte all'interno della pagina HTML nella parte iniziale, e raggruppati dalla coppia di tag **<SCRIPT> ... </SCRIPT>**, indicanti l'inizio e la fine della zona riservata al codice JavaScript.

```
<SCRIPT language="javascript">
...
...
</SCRIPT>
```

La **function** è identificata da un nome e da eventuali parametri (racchiusi tra due parentesi tonde) che le vengono inviati al momento dell'esecuzione.

```
function nomeFunzione(parametri) {
...
...
}
```

Le istruzioni da eseguire sono racchiuse tra una coppia di parentesi graffe (le parentesi graffe si ottengono con le combinazioni di tasti **Maiuscolo + AltGr + [** e **Maiuscolo + AltGr + ]**).

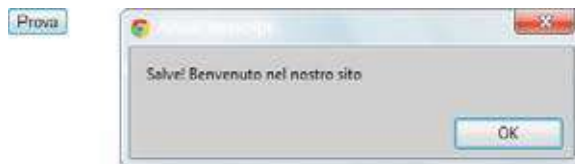
## PROGETTO 1

### Attivare una finestra di messaggio in una pagina Web.

Lo script produce un messaggio all'interno di una finestra quando l'utente fa clic sul pulsante con la scritta *Prova*.  
(Nei primi progetti il codice JavaScript è evidenziato in grassetto).

#### Pagina Web (*messaggio.html*)

```
<!DOCTYPE html>
<HTML>
<HEAD>
 <META CHARSET="utf-8">
 <TITLE>Avviso</TITLE>
 <SCRIPT language="javascript">
 function mostraMessaggio() {
 alert("Salve! Benvenuto nel nostro sito");
 }
 </SCRIPT>
</HEAD>
<BODY>
 <INPUT TYPE="button" VALUE="Prova" NAME="B1" onClick="mostraMessaggio()">
</BODY>
</HTML>
```



Il codice JavaScript racchiuso tra `<SCRIPT>...</SCRIPT>` è stato scritto all'interno di un testo HTML salvato su disco con l'estensione *.html*. È composto da una sola funzione chiamata *mostraMessaggio*:

```
function mostraMessaggio() {
 alert("Salve! Benvenuto nel nostro sito");
}
```

Dopo la parola **function** c'è il nome della procedura e la coppia di parentesi tonde per eventuali parametri.

Il comando **alert** produce sul video una finestra di avvertimento con il messaggio indicato tra virgolette.

Aperto la pagina nel browser, si ottiene l'effetto desiderato: la function viene attivata in risposta all'evento prodotto dall'utente quando fa clic (**onClick**), con il mouse, sul pulsante che ha come descrizione *Prova*. Quando l'utente ha terminato di leggere il messaggio, può chiudere la finestra facendo clic sul pulsante **OK**.

Questo esempio rappresenta quindi una modalità di gestione degli eventi che accadono durante la consultazione delle pagine da parte degli utenti: la procedura JavaScript fornisce la risposta all'evento in modo interattivo.

Come si è visto nel progetto precedente, le funzioni possono essere attivate in seguito a un particolare evento, per esempio un clic su un pulsante. Nella tabella seguente sono mostrati i nomi e le descrizioni degli eventi più utilizzati con i tag HTML dove possono essere usati.

| Nome               | Descrizione                                             | Tag                                             |
|--------------------|---------------------------------------------------------|-------------------------------------------------|
| <i>onLoad</i>      | Un elemento viene caricato                              | <BODY>, <IMG>                                   |
| <i>onSubmit</i>    | Invio di informazioni                                   | <FORM>                                          |
| <i>onMouseOver</i> | Passaggio del mouse sopra un oggetto                    | <A>, <AREA> e la maggior parte dei tipi <INPUT> |
| <i>onMouseOut</i>  | Uscita del mouse da un oggetto                          | <A>, <AREA> e la maggior parte dei tipi <INPUT> |
| <i>onChange</i>    | Un elemento viene modificato dall'utente                | <INPUT>, <SELECT>                               |
| <i>onFocus</i>     | L'elemento viene selezionato (prende il <i>focus</i> )  | <A>, <INPUT>, <BODY>                            |
| <i>onBlur</i>      | L'elemento viene deselezionato (perde il <i>focus</i> ) | <A>, <INPUT>, <BODY>                            |

Gli eventi provocano l'attivazione delle funzioni che possono modificare alcuni elementi di una pagina in modo dinamico, sulla base delle azioni dell'utente.

## PROGETTO 2

### Visualizzare le informazioni sulle componenti di un PC.

La pagina Web deve mostrare un'immagine di un computer (monitor, tastiera, mouse e unità centrale) e visualizzare le informazioni sui componenti indicati dall'utente tramite il mouse. Le informazioni vengono visualizzate in un paragrafo sotto l'immagine e variano a seconda della posizione del mouse sulla fotografia. Il file dell'immagine è memorizzato in una sottocartella *IMG*.

La pagina Web utilizza le **mappe immagine** sensibili al mouse (tag MAP del linguaggio HTML) che sono state presentate nei *Materiali on line* del Capitolo 10.



#### Pagina Web (*computer.html*)

```
<!DOCTYPE html>
<HTML>
<HEAD>
 <META CHARSET="utf-8">
 <TITLE>Il mio PC</TITLE>
 <SCRIPT language="javascript">
 function scriviTesto(testo) {
 document.getElementById("descrizione").innerHTML=testo;
 }
 </SCRIPT>
</HEAD>
```

```
<BODY>

<MAP NAME="PCmap">
 <AREA SHAPE="poly" COORDS ="80,0,260,0,260,200,80,170"
 onmouseover="scriviTesto('Monitor. Dispositivo di output che permette di
visualizzare immagini e video.')"
 HREF="http://it.wikipedia.org/wiki/Monitor" TARGET="_blank" ALT="Monitor" >
 <AREA SHAPE="poly" COORDS="0,280,75,230,290,275,230,325"
 onmouseover="scriviTesto('Tastiera. Dispositivo di input che permette di
inserire testo.')"
 HREF="http://it.wikipedia.org/wiki/Tastiera_(informatica)" TARGET="_blank"
ALT="Tastiera" >
 <AREA SHAPE="poly" COORDS="300,275,360,280,360,310,295,300"
 onmouseover="scriviTesto('Mouse. Dispositivo di input che permette di
sposare un cursore sullo schermo.')"
 HREF="http://it.wikipedia.org/wiki/Mouse" TARGET="_blank" ALT="Mouse" >
 <AREA SHAPE="poly" COORDS="275,60,365,30,440,40,440,220,360,270,275,250"
 onmouseover="scriviTesto('Case. Dispositivo contenente i componenti
principali del computer.')"
 HREF="http://it.wikipedia.org/wiki/Case_(hardware)" TARGET="_blank"
ALT="Case" >
</MAP>
<P ID="descrizione">Sposta il mouse sull'immagine per visualizzare le
informazioni. Se fai clic troverai le informazioni di Wikipedia.</P>
</BODY>
</HTML>
```



Monitor. Dispositivo di output che permette di visualizzare immagini e video.

Lo script è composto da una sola funzione, chiamata *scriviTesto* che, a sua volta, è formata da un solo comando:

```
function scriviTesto(testo) {
 document.getElementById("descrizione").innerHTML=testo;
}
```



- **document** è l'intero documento HTML;
- **getElementById** ("*descrizione*") è l'oggetto, nella pagina HTML, avente **ID** uguale a *descrizione*, cioè il paragrafo sotto l'immagine (letteralmente l'elemento ottenuto, *get*, tramite l'*Id*, *by id*);
- **innerHTML** è il codice all'*interno* dell'oggetto, nel nostro caso il testo.

In pratica la funzione cambia il testo all'interno del paragrafo chiamato *descrizione*.

Sull'immagine viene ricreata la mappa immagine (**AREA**) che permette di individuare gli oggetti. Per esempio, il poligono che identifica il monitor è rappresentato dalla forma grafica (**SHAPE**) di forma poligonale (**poly**) con il seguente codice:

```
<AREA SHAPE="poly" COORDS="80,0,260,0,260,200,80,170"
onmouseover="scriviTesto('Monitor. Dispositivo di output che permette di
visualizzare immagini e video.')"
HREF="http://it.wikipedia.org/wiki/Monitor" TARGET="_blank" ALT="Monitor" >
```

La tabella seguente spiega il significato del codice.

| Codice                                      | Significato                                                                                                                                         |
|---------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| SHAPE="poly"                                | l'area è un poligono formato da segmenti consecutivi                                                                                                |
| COORDS="80,0,260,0,260,200,80,170"          | il poligono ha come vertici i punti con coordinate (80;0), (260;0), (260;200), (80;170)                                                             |
| onMouseOver="scriviTesto('...')"            | quando l'utente passa con il mouse sopra l'area viene richiamata la funzione <i>scriviTesto</i> , alla quale viene passato il testo da visualizzare |
| HREF="http://it.wikipedia.org/wiki/Monitor" | se l'utente fa clic viene rimandato alla pagina di cui è specificato l'URL                                                                          |

Alla fine della pagina Web si trova il testo (con *ID*="descrizione") che viene cambiato per visualizzare le informazioni:

```
<P ID="descrizione">Sposta il mouse sull'immagine per visualizzare le
informazioni. Se fai clic troverai le informazioni di Wikipedia.</P>
```

In questo modo vengono visualizzate le informazioni riguardanti un'immagine senza fare clic, ma semplicemente passando il mouse sopra l'area grafica.

In modo simile si potrebbe visualizzare una mappa dell'Italia e mostrare informazioni sulle varie regioni, oppure un'immagine di un motore e le sue componenti, un organo e le sue parti, uno schema complesso e la sua spiegazione.

Javascript può essere utilizzato per assegnare valori a variabili o effettuare calcoli. L'istruzione di **assegnazione** di un valore a una variabile ha la seguente sintassi generale:

```
var = valore
```

Gli operatori aritmetici, di confronto, logici e di assegnazione sono simili a quelli dei linguaggi di programmazione.

| Operatore | Tipo         | Descrizione                    |
|-----------|--------------|--------------------------------|
| +         | Aritmetico   | Addizione                      |
| -         |              | Sottrazione                    |
| *         |              | Moltiplicazione                |
| /         |              | Divisione                      |
| >         | Confronto    | Maggiore                       |
| >=        |              | Maggiore o uguale              |
| <         |              | Minore                         |
| <=        |              | Minore o uguale                |
| !=        |              | Diverso                        |
| ==        |              | Uguale                         |
| !         | Logico       | NOT                            |
| &&        |              | AND                            |
|           |              | OR                             |
| =         | Assegnazione | Assegnazione                   |
| ++        |              | Incremento e assegnazione      |
| --        |              | Decremento e assegnazione      |
| +=        |              | Addizione e assegnazione       |
| -=        |              | Sottrazione e assegnazione     |
| *=        |              | Moltiplicazione e assegnazione |
| /=        |              | Divisione e assegnazione       |

Altre funzioni matematiche più complesse (per esempio l'estrazione di radice o l'elevamento a potenza) sono disponibili come metodi della classe **Math**.

| Nome                                   | Significato                                                            | Utilizzo                                                        |
|----------------------------------------|------------------------------------------------------------------------|-----------------------------------------------------------------|
| <b>abs</b>                             | Valore assoluto                                                        | <code>Math.abs(valore)</code>                                   |
| <b>PI</b>                              | PI greco                                                               | <code>Math.PI</code>                                            |
| <b>sin, cos, tan, asin, acos, atan</b> | Seno, coseno, tangente, arcseno, arccoseno, arctangente                | <code>Math.sin(angolo)</code><br><code>Math.asin(valore)</code> |
| <b>max, min</b>                        | Valore massimo, valore minimo                                          | <code>Math.max(valore1, valore2)</code>                         |
| <b>exp</b>                             | Potenza                                                                | <code>Math.exp(base, esponente)</code>                          |
| <b>sqrt</b>                            | Radice quadrata                                                        | <code>Math.sqrt(valore)</code>                                  |
| <b>random</b>                          | Numero casuale                                                         | <code>Math.random()</code>                                      |
| <b>round, ceil, floor</b>              | Arrotondamento, arrotondamento per eccesso, arrotondamento per difetto | <code>Math.round(valore)</code>                                 |

In particolare gli operatori e le funzioni possono essere utilizzati con i dati inseriti dall'utente tramite un *form*.

## PROGETTO 3

### Calcolare gli interessi e il montante.

Il progetto mostra l'uso di JavaScript per effettuare un calcolo sui valori forniti dall'utente tramite un modulo (*form*) con caselle di testo; il risultato viene scritto in un'altra casella di testo della pagina.

Si richiamano alcuni concetti sulla *capitalizzazione composta*:

- $C$  è il capitale iniziale, in euro, che viene investito
- $t$  è la durata, misurata in anni, dell'investimento
- $i$  è il tasso di interesse annuo espresso in percentuale
- $I$  è l'interesse in euro
- $M$  è il montante in euro, cioè il capitale iniziale al quale vengono aggiunti gli interessi.

Le formule da utilizzare sono le seguenti:

$$M = C \left( 1 + \frac{i}{100} \right)^t \quad I = M - C$$

Lo script accetta i dati forniti per i valori del tasso  $i$ , del tempo  $t$  e del capitale iniziale  $C$  ed effettua il calcolo come risposta all'evento che l'utente produce quando fa clic sul pulsante con la descrizione *Calcola tasso*. I vari elementi sono stati inseriti in una tabella per aumentare l'ordine di presentazione.

### Pagina Web (*montante.html*)

```
<!DOCTYPE html>
<HTML>
<HEAD>
<TITLE>Interesse composto</TITLE>
<SCRIPT Language="javascript">
function calcola(form){
 tassoUnit = form.tasso.value / 100;
 montante = form.capitale.value * Math.pow((1 + tassoUnit),form.tempo.value);
 form.montante.value = Math.round(montante*100)/100;
 form.interesse.value = Math.round((montante - form.capitale.value)*100)/100;
}
</SCRIPT>
<STYLE type="text/css">
TD {
text-align: center;
}
</style>
</HEAD>
<BODY>
<H2>Calcolo dell'Interesse Composto</H2>
<FORM>
 <TABLE border="0">
 <TR><TD>Capitale iniziale</TD>
 <TD>C</TD>
 <TD><INPUT TYPE="Text" NAME="capitale" SIZE="10"></TD>
 <TD>€</TD></TR>
 <TR><TD>Tasso di interesse</TD>
 <TD>i</TD>
```

```
<TD><INPUT TYPE="Text" NAME="tasso" SIZE="10">
</TD><TD>%</TD></TR>
<TR><TD>Tempo</TD>
<TD>t</TD>
<TD><INPUT TYPE="Text" NAME="tempo" SIZE="10">
</TD><TD>anni</TD></TR>
<TR><TD COLSPAN="3">
<INPUT TYPE="button" VALUE="Calcola tasso" NAME="Invia"
OnClick="calcola(this.form)">
<INPUT TYPE="reset" VALUE="Annulla" NAME="Rifare">
</TD></TR>
<TR><TD>Interesse</TD>
<TD>I</TD>
<TD><INPUT TYPE="Text" NAME="interesse" SIZE="10"></TD>
<TD>€</TD></TR>
<TR><TD>Montante</TD>
<TD>M</TD>
<TD><INPUT TYPE="Text" NAME="montante" SIZE="10"></TD>
<TD>€</TD></TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

| Calcolo dell'Interesse Composto |          |                                                                                     |      |
|---------------------------------|----------|-------------------------------------------------------------------------------------|------|
| Capitale iniziale               | <b>C</b> | <input type="text" value="1600"/>                                                   | €    |
| Tasso di interesse              | <b>i</b> | <input type="text" value="4"/>                                                      | %    |
| Tempo                           | <b>t</b> | <input type="text" value="10"/>                                                     | anni |
|                                 |          | <input type="button" value="Calcola tasso"/> <input type="button" value="Annulla"/> |      |
| Interesse                       | <b>I</b> | <input type="text" value="768.39"/>                                                 | €    |
| Montante                        | <b>M</b> | <input type="text" value="2368.39"/>                                                | €    |

L'evento **onClick** provoca la chiamata della funzione *calcola* passando come parametro **this.form**. In generale per JavaScript la parola chiave **this** (letteralmente *questo*) indica l'oggetto che chiama un determinato metodo; inoltre ogni elemento di un form possiede una proprietà che fa riferimento all'oggetto **form**. Quindi **this.form** indica il form del bottone sul quale è stato fatto il clic.

La function riceve come parametro il nome del *form*, contenente le caselle con i dati da usare per il calcolo e la casella *tasso* nella quale scrivere il risultato.

Si noti che le proprietà degli oggetti di un form sono identificati con la scrittura

*nomeform.nomeoggetto.proprietà*

(separati dal punto)

per esempio: *form.i1.value*.

Nello script la funzione **eval** restituisce il valore numerico del contenuto delle caselle del form. JavaScript non possiede un operatore per l'elevamento a potenza; il calcolo deve essere fatto attraverso il metodo **Math.pow**:

**Math.pow**(*base*, *esponente*)

I risultati devono essere arrotondati alla seconda cifra decimale. In *JavaScript* non esiste una funzione che permette di arrotondare specificando le cifre decimali; tuttavia il metodo **Math.round** restituisce il valore arrotondato all'intero più vicino:

```
Math.round(valore)
```

Volendo arrotondare alla seconda cifra decimale basta moltiplicare per 100, arrotondare e poi dividere per 100:

```
Math.round(valore*100)/100;
```

In modo analogo si possono fare i seguenti arrotondamenti:

```
Math.round(valore*10)/10; arrotonda a 1 cifra
Math.round(valore*1000)/1000; arrotonda a 3 cifre
Math.round(valore*1000000)/1000000; arrotonda a 6 cifre
```

In JavaScript sono presenti anche le **strutture di controllo** fondamentali. La struttura di **selezione binaria** è rappresentata in JavaScript secondo questo schema generale:

```
if(condizione){
 istruzioni
} else {
 istruzioni
}
```

Per la **selezione multipla** si utilizza invece la struttura **switch**:

```
switch(espressione){
 case valore1:
 istruzioni1
 break;
 case valore2:
 istruzioni2
 break;
 ...
 istruzioni predefinite
}
```

Per la **ripetizione enumerativa** si utilizza la struttura **for**:

```
for(contatore=valoreIniziale; contatore<valoreFinale; incremento){
 istruzioni
}
```

Per la **ripetizione con condizione** si utilizza la struttura **while**:

```
while(condizione){
 istruzioni
}
```

## PROGETTO 4

### Invio di un messaggio di saluto per l'utente.

Lo script acquisisce dal sistema la data e l'ora del giorno e manda un messaggio di saluto all'utente a seconda dell'ora (*Buongiorno, Buon pomeriggio oppure Buona sera*).

#### Pagina Web (*saluto.html*)

```
<!DOCTYPE html>
<HTML>
<HEAD>
<TITLE>Benvenuti!</TITLE>
<SCRIPT language="javascript">
function Saluto() {
 var d = new Date();
 var h = d.getHours();
 if (h < 12)
 document.write("Buongiorno!")
 else
 if (h < 17)
 document.write("Buon pomeriggio!")
 else
 document.write("Buona sera!");
}
</SCRIPT>
</HEAD>
<BODY onLoad = "Saluto()">
</BODY>
</HTML>
```

La *function* usa la selezione di JavaScript con una **struttura annidata** *if ... else if ... else*.

```
if (h < 12)
 document.write("Buongiorno!")
else
 if (h < 17)
 document.write("Buon pomeriggio!")
 else
 document.write("Buona sera!");
```

Schematizzando:

**SE** è prima di mezzogiorno  
    **ALLORA** scrivi "Buongiorno!"  
**ALTRIMENTI**  
    **SE** è prima delle 17  
        **ALLORA** scrivi "Buon pomeriggio!"  
    **ALTRIMENTI** scrivi "Buona sera!"

La *function* viene attivata al momento del caricamento della pagina (evento **onLoad**): è un esempio di JavaScript che modifica in modo dinamico l'aspetto della pagina durante la sua visualizzazione.

La data di sistema è fornita dall'oggetto **Date()** con i suoi metodi per estrarre: ora (metodo **getHours()**), minuti (**getMinutes()**), giorno (**getDate()**), numero del mese (**getMonth()**) e anno (**getFullYear()**).

Quando si scrive codice complesso è buona abitudine inserire righe di **commento** per spiegare le varie funzionalità implementate. Le modalità con cui si può inserire un commento negli script sono tre.

Il doppio *slash* inizia un commento che termina automaticamente alla fine della linea. Quindi è adatto per commenti molto brevi.

```
// commento1
```

Se il commento è molto lungo si può iniziare con i simboli */\** e terminare con i simboli *\*/*.

```
/* questo occupa
due linee di codice */
```

In alternativa si possono utilizzare i caratteri di commento del linguaggio HTML *<!--* e *-->*

```
<!-- anche questo commento
occupa più linee
di codice -->
```

## PROGETTO 5

### Calcolatrice per le quattro operazioni fondamentali (+, -, \*, /).

Il progetto è formato dalla pagina HTML che disegna la calcolatrice con i tasti organizzati in una tabella e da un file esterno che contiene il codice JavaScript. Il file separato ha l'estensione **.js** e viene incorporato all'interno della pagina Web, al momento del suo caricamento, con questa modalità sintattica (**src** è l'abbreviazione di *source*, sorgente):

```
<SCRIPT language="javascript" src="fileEsterno.js">
</SCRIPT>
```

L'uso di file esterni presenta alcuni vantaggi importanti nella programmazione con JavaScript:

- semplifica la lettura della pagina Web
- protegge il codice nella visualizzazione del codice della pagina
- consente di utilizzare lo stesso codice in pagine Web diverse senza inserirlo in ogni pagina.

La tabella HTML contiene 5 righe e 4 colonne. Le celle della prima linea vengono unite a formare il display della calcolatrice, attraverso l'attributo **ROWSPAN**:

| Display |   |   |   |
|---------|---|---|---|
| 7       | 8 | 9 | / |
| 4       | 5 | 6 | X |
| 1       | 2 | 3 | - |
| 0       | C | = | + |

Il display è rappresentato da una casella di testo, mentre i tasti sono elementi di tipo **button**.

|   |   |   |   |
|---|---|---|---|
| 7 | 8 | 9 | / |
| 4 | 5 | 6 | x |
| 1 | 2 | 3 | - |
| 0 | C | = | + |

Il codice HTML che definisce il tasto con la cifra 1 è il seguente:

```
<INPUT TYPE="button" VALUE="1" style="width:20px; height:20px;"
onClick="document.getElementById('display').value+=1">
```

Si osservi che nel tag <INPUT> sono stati utilizzati sia il linguaggio dei fogli di stile CSS:

```
style="width:20px; height:20px;"
```

sia il linguaggio JavaScript:

```
onClick="document.getElementById('display').value+=1">
```

Facendo clic sulla cifra 1, questa viene aggiunta al display attraverso l'operatore **+=** (addizione e assegnazione).

L'operatore += aggiunge la cifra 1 alla stringa di caratteri della casella del display, ma non incrementa il valore: infatti l'attributo **value** della casella è di tipo testo.

In modo analogo si possono creare i bottoni per le altre cifre.

Quando si fa clic sul pulsante con il segno di un'operazione viene chiamata la funzione relativa (*piu*, *meno*, *per*, *diviso*). In questo caso si deve memorizzare il valore nella casella di testo, ricordare l'operazione selezionata e cancellare il display.

Per esempio, il codice della funzione per la divisione è il seguente:

```
function diviso(){
 val1 = eval(document.getElementById("display").value);
 op = "diviso";
 document.getElementById('display').value='';
}
```

Quando si fa clic sul pulsante con il segno =, si memorizza il secondo valore e si calcola il risultato dell'operazione selezionata. La selezione multipla è rappresentata con la struttura **switch**:

```
function uguale(){
 val2 = eval(document.getElementById("display").value);
 switch(op){
 case "diviso":
 risultato = val1 / val2;
 break;
 case "per":
 risultato = val1 * val2;
 break;
 case "piu":
 risultato = val1 + val2;
 break;
 case "meno":
 risultato = val1 - val2;
 break;
 default:
 risultato = "errore";
 }
 document.getElementById('display').value = risultato;
}
```



Di seguito sono presentati i codici completi della pagina HTML e del file esterno di JavaScript.

**Pagina Web** (*calcolatrice.html*)

```
<!DOCTYPE html>
<HTML>
<HEAD>
<SCRIPT language="javascript" src="calcola.js">
</SCRIPT>
</HEAD>
<TITLE>Calcolatrice</TITLE>
<BODY>
<TABLE CELLSPACING="0" CELLPADDING="0">
<TR>
<TD HEIGHT="40" COLSPAN="4" align="right">
<INPUT TYPE="text" ID="display" style="width:80px; height:20px;">
</TD>
</TR>
<TR>
<TD><INPUT TYPE="button" VALUE="7" style="width:20px; height:20px;"
onClick="document.getElementById('display').value+=7"></TD>
<TD><INPUT TYPE="button" VALUE="8" style="width:20px; height:20px;"
onClick="document.getElementById('display').value+=8"></TD>
<TD><INPUT TYPE="button" VALUE="9" style="width:20px; height:20px;"
onClick="document.getElementById('display').value+=9"></TD>
<TD><INPUT TYPE="button" VALUE="/" style="width:20px; height:20px;"
onClick="diviso()"></TD>
</TR>
<TR>
<TD><INPUT TYPE="button" VALUE="4" style="width:20px; height:20px;"
onClick="document.getElementById('display').value+=4"></TD>
<TD><INPUT TYPE="button" VALUE="5" style="width:20px; height:20px;"
onClick="document.getElementById('display').value+=5"></TD>
<TD><INPUT TYPE="button" VALUE="6" style="width:20px; height:20px;"
onClick="document.getElementById('display').value+=6"></TD>
<TD><INPUT TYPE="button" VALUE="x" style="width:20px; height:20px;"
onClick="per()"></TD>
</TR>
<TR>
<TD><INPUT TYPE="button" VALUE="1" style="width:20px; height:20px;"
onClick="document.getElementById('display').value+=1"></TD>
<TD><INPUT TYPE="button" VALUE="2" style="width:20px; height:20px;"
onClick="document.getElementById('display').value+=2"></TD>
<TD><INPUT TYPE="button" VALUE="3" style="width:20px; height:20px;"
onClick="document.getElementById('display').value+=3"></TD>
<TD><INPUT TYPE="button" VALUE="-" style="width:20px; height:20px;"
onClick="meno()"></TD>
</TR>
<TR>
<TD><INPUT TYPE="button" VALUE="0" style="width:20px; height:20px;"
onClick="document.getElementById('display').value+=0"></TD>
<TD><INPUT TYPE="button" VALUE="C" style="width:20px; height:20px;"
onClick="document.getElementById('display').value='' ;val1=0;"></TD>
```

```

 <TD><INPUT TYPE="button" VALUE="=" style="width:20px; height:20px;"
onClick="uguale()"></TD>
 <TD><INPUT TYPE="button" VALUE="+" style="width:20px; height:20px;"
onClick="piu()"></TD>
 </TR>
</TABLE>
</BODY>
</HTML>

```

### Codice JavaScript (*calcola.js*)

```

function piu(){
 val1 = eval(document.getElementById("display").value);
 op = "piu";
 document.getElementById('display').value='';
}
function meno(){
 val1 = eval(document.getElementById("display").value);
 op = "meno";
 document.getElementById('display').value='';
}
function per(){
 val1 = eval(document.getElementById("display").value);
 op = "per";
 document.getElementById('display').value='';
}
function diviso(){
 val1 = eval(document.getElementById("display").value);
 op = "diviso";
 document.getElementById('display').value='';
}
function uguale(){
 val2 = eval(document.getElementById("display").value);
 switch(op){
 case "diviso":
 risultato = val1 / val2;
 break;
 case "per":
 risultato = val1 * val2;
 break;
 case "piu":
 risultato = val1 + val2;
 break;
 case "meno":
 risultato = val1 - val2;
 break;
 default:
 risultato = "errore";
 }
 document.getElementById('display').value = risultato;
}

```

## PROGETTO 6

### Controllo della risposta a un quiz.

La seguente pagina Web presenta all'utente la domanda a risposta multipla di un quiz e controlla la correttezza della risposta, inviando un messaggio di errore o di conferma.

#### Pagina Web (*quiz.html*)

```
<!DOCTYPE html>
<HTML>
<HEAD>
<TITLE> Quiz </TITLE>
<SCRIPT language="javascript">
// questa funzione avvisa della risposta errata
function Errore() {
 alert("Risposta sbagliata, riprova!");
}
// questa funzione avvisa che la risposta è corretta
function Esatto() {
 alert("Risposta esatta, bene!");
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
 <H2>Come si calcola l'area del triangolo ?</H2>
 <P>
 <INPUT TYPE="radio" VALUE="1" NAME="q1" onClick="Errore()">base x altezza

 <INPUT TYPE="radio" VALUE="2" NAME="q1" onClick="Esatto()">base x altezza: 2

 <INPUT TYPE="radio" VALUE="3" NAME="q1" onClick="Errore()">base + altezza

 <INPUT TYPE="radio" VALUE="4" NAME="q1" onClick="Errore()">base + altezza x 2
 </P>
</FORM>
</BODY>
</HTML>
```

### Come si calcola l'area del triangolo ?

- ☒ base x altezza
- ☐ base x altezza: 2
- ☐ base + altezza
- ☐ base + altezza x 2

Per presentare il quiz è stato utilizzato un modulo (*form*), con l'oggetto **radio** che rappresenta il pulsante di opzione con un piccolo cerchio: i quattro pulsanti devono avere tutti lo stesso nome. Quando l'utente preme il pulsante scelto, esso si annerisce e viene attivata una delle due function *Esatto()* oppure *Errore()*: ovviamente una sola delle opzioni può attivare la function *Esatto()*. Le due function mandano il messaggio all'utente in una finestra, che si apre al centro della pagina, con il comando **alert**.

Anche questo progetto mostra come sia possibile utilizzare i JavaScript per controllare i dati inseriti dall'utente oppure per lanciare l'esecuzione di azioni come conseguenza delle scelte fatte all'interno di un elenco.

## PROGETTO 7

### Cambio dell'immagine al passaggio del mouse sopra di essa.

Pagina Web (*immagini.html*)

```
<!DOCTYPE html>
<HTML>
<HEAD>
<TITLE>Cambio immagine</TITLE>
<SCRIPT language="javascript">
OnImage = new Image();
OnImage = "pic1.gif";
OffImage = new Image();
OffImage = "pic2.gif";
function displayOnImage () {
 document.image.src = OnImage;
}
function displayOffImage() {
 document.image.src = OffImage;
}
</SCRIPT>
</HEAD>
<BODY>
 <H1>Cambio immagine</H1>
 <A HREF="#" onMouseOver="displayOnImage()" onMouseOut="displayOffImage()"

 Muovi il mouse sopra l'immagine
per vedere come cambia
</BODY>
</HTML>
```

Lo script utilizza due immagini *pic1.gif* e *pic2.gif*: all'inizio viene visualizzata la seconda; quando l'utente passa con il mouse sopra di essa viene visualizzata la prima; spostando il mouse in un'altra posizione del video si ritorna alla situazione iniziale. L'intera pagina è identificata dall'oggetto **document**. All'interno di essa è posizionato l'oggetto **image**.

La scrittura

```
document.image.src = ...
```

indica la modifica della proprietà **src** dell'oggetto **image** all'interno di **document**, cioè del file origine dell'immagine.

Il passaggio del mouse sopra l'immagine e il suo spostamento da essa sono rappresentati con gli eventi **onMouseOver** e **onMouseOut**.

Il linguaggio JavaScript possiede molti comandi per introdurre elementi di interattività e di dinamicità nelle pagine Web. Per approfondire l'uso di JavaScript si possono consultare i numerosi siti Internet che presentano *on line* spiegazioni tutoriali ed esempi di JavaScript. È anche utile osservare il sorgente HTML delle pagine Web che si consultano durante la navigazione in Internet, per capire come vengono realizzati gli oggetti che appaiono interessanti: studiando il funzionamento dei JavaScript e usando il codice come traccia, si possono modificare gli script e le function per adattare alle proprie esigenze.

## 2 Le canvas

Il tag **<CANVAS>**, presente nella versione HTML5, costruisce un'immagine all'interno di una pagina Web.

In generale, in informatica, il termine **canvas** (letteralmente *tela*) indica una superficie su cui disegnare oggetti statici o dinamici.

La struttura generale del tag per la definizione della canvas è la seguente:

```
<CANVAS ID="canvasProva" WIDTH="400" HEIGHT="300">
 Attenzione: il tuo browser non supporta il tag CANVAS.
</CANVAS>
```

Il codice precedente crea uno spazio di disegno di dimensioni 400x300 pixel vuoto oppure, se il browser non è compatibile, avvisa l'utente con un messaggio.

Il sistema di coordinate di una canvas ha l'**origine (0, 0)** nell'angolo in alto a sinistra della superficie.

Dopo aver creato il tag, tramite *JavaScript*, si possono aggiungere, togliere o modificare elementi grafici che possono essere segmenti, rettangoli, archi di circonferenza, curve, ombre, altre immagini o video.

Per esempio, il seguente frammento di codice JavaScript disegna un arco di circonferenza.

```
mioCanvas = document.getElementById("canvasProva");
context = mioCanvas.getContext("2d");
context.arc(200, 200, 100, 0, Math.PI, true);
context.stroke();
```

Vediamo il significato del codice.

```
mioCanvas = document.getElementById("canvasProva");
context= mioCanvas.getContext("2d");
```

rende disponibile l'oggetto *mioCanvas* in *JavaScript*. Si specifica che il contesto è **2d** cioè, di fatto, non si utilizza la coordinata Z della terza dimensione.

```
context.arc(200, 200, 100, 0, Math.PI, true);
```

crea un arco avente:

- centro nel punto di coordinate (200; 200)
- raggio uguale a 100
- angolo di partenza uguale a 0 radianti
- angolo di arrivo uguale a  $\pi$  radianti (180°)
- in senso antiorario (**true** significa senso antiorario, **false** senso orario).

Il metodo **stroke** disegna la linea:

```
context.stroke();
```

Per *default* (in mancanza di una dichiarazione specifica) la linea viene disegnata in colore nero e con spessore 1 pixel.

Lo spessore diverso da 1 è definito con il metodo **lineWidth**. Per esempio per una linea di spessore uguale a 3 pixel:

```
context.lineWidth = 3;
```

L'impostazione del colore per lo *stroke* è realizzata con il metodo **strokeStyle**:

```
context.strokeStyle=colore;
```

Il *colore* può essere espresso con la combinazione RGB (colore *blue* = 0, 0, 255 ):

```
context.strokeStyle = "rgba(0, 0, 255, 1)";
```

oppure con il valore in esadecimale (colore *blue* = 0000FF):

```
context.strokeStyle = "#0000FF";
```

Si può usare anche il nome simbolico del colore.

Nella notazione **rgba** sono forniti i tre valori decimali per i colori *Red*, *Green* e *Blue* (tra 0 e 255); il quarto parametro è un numero decimale che rappresenta il valore di trasparenza (od opacità) del colore. L'intervallo di valori è tra 0.0 (massima trasparenza) e 1.0 (massima opacità).

In alternativa la figura può essere riempita con un colore:

```
context.fillStyle=colore;
context.fill();
```

Il metodo **fillStyle** definisce le modalità di riempimento (*fill*); il metodo **fill** disegna la forma riempita.

Per esempio il riempimento con il colore rosso opaco (combinazione RGB = 255, 0, 0 ):

```
context.fillStyle = "rgba(255, 0, 0, 1)";
```

oppure in esadecimale (colore rosso = FF0000):

```
context.fillStyle = "#FF0000";
```

Il codice seguente rappresenta la pagina Web completa (HTML e funzione JavaScript) per disegnare un semicerchio di colore rosso con un bordo blu.

#### Pagina Web (*semicerchio.html*)

```
<!DOCTYPE html>
<HTML>
<HEAD>
<TITLE>Canvas: semicerchio</TITLE>
<SCRIPT language="javascript">
function creaFigura() {
 mioCanvas = document.getElementById("canvasProva");
 context = mioCanvas.getContext("2d");
 context.arc(200, 200, 100, 0, Math.PI, true);
 context.lineWidth = 3;
 context.strokeStyle = "rgba(0, 0, 255, 1)";
 context.stroke();
}
```

```
context.fillStyle = "rgba(255, 0, 0, 1)";
context.fill();
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
 <H1>Semicerchio</H1>
 <INPUT TYPE="button" ID="btnOK" value="Crea" onClick="creaFigura()">
 <P><CANVAS ID="canvasProva" WIDTH="400" HEIGHT="300">
 Attenzione: il tuo browser non supporta il tag CANVAS.
 </CANVAS></P>
</FORM>
</BODY>
</HTML>
```

## Semicerchio

Crea



Eliminando le due righe di codice per il riempimento:

```
context.fillStyle = "rgba(255, 0, 0, 1)";
context.fill();
```

si ottiene la sola linea della semicirconferenza.

Vediamo ora come si creano le principali forme geometriche.

## Segmento

```
context.moveTo(Ax,Ay);
context.lineTo(Bx,By);
context.stroke();
```

si posiziona nel punto di coordinate (Ax,Ay) e traccia una linea fino al punto di coordinate (Bx,By). Il metodo **stroke** applica lo stile e disegna il segmento (se non si specifica nulla il segmento viene disegnato nero e spesso 1 pixel).

```
context.lineWidth = valore;
```

specifica lo spessore del segmento.

```
context.strokeStyle= colore;
```

specifica il colore del segmento.

```
context.lineCap = tipo;
```

specifica lo stile delle estremità del segmento, per esempio arrotondato (*round*) o squadrato (*square*).

## Arco di circonferenza

```
context.arc(Cx, Cy, r, angoloPartenza, angoloArrivo, direzione);
```

disegna un arco:

- avente come centro il punto di coordinate (Cx,Cy)
- avente *r* come raggio
- da *angoloPartenza* a *angoloArrivo* (espressi in radianti)
- con *direzione* in senso orario (*false*) o antiorario (*true*).

## Percorsi

Un percorso (**path**) è un insieme di linee e curve consecutive. Si inizia un percorso con **beginPath()** e si termina con **closePath()**.

```
context.beginPath(); // inizio del percorso
context.moveTo(100,100); // partenza da (100;100)
context.lineTo(400,100); // segmento da (100;100) a (400;100)
context.lineTo(150,300); // segmento da (400;100) a (150;300)
context.lineTo(250,0); // segmento da (150;300) a (250;0)
context.lineTo(350,300); // segmento da (250;0) a (350;300)
context.closePath(); // il percorso si chiude
```

## Figure (shapes)

Fanno parte di questa classe rettangoli, cerchi, semicerchi o poligoni definiti attraverso più segmenti consecutivi.

```
context.rect(Ax, Ay, larghezza, altezza);
```

crea un rettangolo avente come vertice superiore sinistro il punto di coordinate (Ax,Ay) e le dimensioni *larghezza* e *altezza*.

Per generare una circonferenza di centro (Cx,Cy) e raggio *r*, si crea un arco di un angolo giro ( $360^\circ = 2\pi$  radianti):

```
context.arc(Cx, Cy, r, 0, 2*Math.PI, false);
```

Per riempire una forma grafica si utilizza:

```
context.fillStyle=colore;
context.fill();
```



## PROGETTO 8

### Creare un grafico a torta (areogramma).

La pagina Web contiene un form per acquisire tre valori attraverso tre caselle di testo: lo script traccia l'areogramma con colori diversi per i settori.

Il form contiene tre caselle di testo (chiamate *val1*, *val2* e *val3*) e un pulsante per richiamare la funzione *creaAreo()*.

```
<FORM>
 <INPUT TYPE="number" ID="val1">
 <INPUT TYPE="number" ID="val2">
 <INPUT TYPE="number" ID="val3">
 <INPUT TYPE="button" ID="btnOK" value="Crea" onClick="creaAreo()">
</FORM>
```

Il codice *JavaScript* prima di tutto deve recuperare i valori nelle caselle di testo

```
valore1 = eval(document.getElementById("val1").value);
valore2 = eval(document.getElementById("val2").value);
valore3 = eval(document.getElementById("val3").value);
```

Poi occorre calcolare le ampiezze degli angoli ricordando la proporzione:

$$\text{valore} : \text{totale} = \text{angolo} : 2\pi$$

per cui:

$$\text{angolo} = \frac{\text{valore} \times 2\pi}{\text{totale}}$$

Il codice per il calcolo è il seguente:

```
totale = valore1 + valore2 + valore3;
angolo1 = 2* Math.PI * valore1/totale;
angolo2 = 2* Math.PI * valore2/totale;
angolo3 = 2* Math.PI * valore3/totale;
```

Seguono le istruzioni per accedere alla *canvas*:

```
mioCanvas = document.getElementById("canvasAreo");
context= mioCanvas.getContext("2d");
```

Per ogni valore occorre creare il settore circolare corrispondente. Si utilizza un percorso così composto:

- tracciamento dell'arco di circonferenza
- ritorno al centro del cerchio.

Il codice per il primo settore circolare disegna l'arco di cerchio che parte da 0 radianti e arriva fino a *angolo1*:

```
context.beginPath();
 context.arc(200, 200, 100, 0, angolo1, false);
 context.lineTo(200, 200);
context.closePath();
```

Il settore viene colorato con riempimento rosso:

```
context.fillStyle = "rgba(200, 0, 0, 1)";
context.fill();
```

Il secondo settore circolare deve partire da dove finisce il primo (*angolo1*) e deve avere ampiezza *angolo2*, per cui deve terminare in *angolo1+angolo2*. Viene poi riempito con il colore verde.

```
context.beginPath();
 context.arc(200, 200, 100, angolo1, angolo1+angolo2, false);
 context.lineTo(200, 200);
context.closePath();
context.fillStyle = "rgba(0, 200, 0, 1)";
context.fill();
```

Il terzo settore parte da *angolo1+angolo2* e termina in *angolo1+angolo2+angolo3=2π*. Il colore di riempimento è blu.

```
context.beginPath();
 context.arc(200, 200, 100, angolo1+angolo2, angolo1+angolo2+angolo3, false);
 context.lineTo(200, 200);
context.closePath();
context.fillStyle = "rgba(0, 0, 200, 1)";
context.fill();
```

Di seguito è presentato il codice completo.

#### Pagina Web (*areogramma.html*)

```
<!DOCTYPE html>
<HTML>
<HEAD>
<SCRIPT language="javascript">
function creaAreo(){
 // Prende i valori dalle caselle di testo
 valore1 = eval(document.getElementById("val1").value);
 valore2 = eval(document.getElementById("val2").value);
 valore3 = eval(document.getElementById("val3").value);
 totale = valore1 + valore2 + valore3;
 angolo1 = 2* Math.PI * valore1/totale;
 angolo2 = 2* Math.PI * valore2/totale;
 angolo3 = 2* Math.PI * valore3/totale;
 mioCanvas = document.getElementById("canvasAreo");
 context= mioCanvas.getContext("2d");
 // primo settore
 context.beginPath();
 context.arc(200, 200, 100, 0, angolo1, false);
 context.lineTo(200, 200);
 context.closePath();
 context.fillStyle = "rgba(200, 0, 0, 1)";
 context.fill();
 // secondo settore
 context.beginPath();
```

```

 context.arc(200, 200, 100, angolo1, angolo1+angolo2, false);
 context.lineTo(200, 200);
 context.closePath();
 context.fillStyle = "rgba(0, 200, 0, 1)";
 context.fill();
 // terzo settore
 context.beginPath();
 context.arc(200, 200, 100, angolo1+angolo2, angolo1+angolo2+angolo3, false);
 context.lineTo(200, 200);
 context.closePath();
 context.fillStyle = "rgba(0, 0, 200, 1)";
 context.fill();
 }
</SCRIPT>
</HEAD>
<BODY>
<FORM>
 <INPUT TYPE="number" ID="val1">
 <INPUT TYPE="number" ID="val2">
 <INPUT TYPE="number" ID="val3">
 <INPUT TYPE="button" ID="btnOK" value="Crea" onClick="creaAreo()">
 <P><CANVAS ID="canvasAreo" width="400" height="400">
 Attenzione: il tuo browser non supporta il tag CANVAS.
 </CANVAS></P>
</FORM>
</BODY>
</HTML>

```

23 45 90 Crea



In modo analogo si può creare un *istogramma* utilizzando rettangoli aventi larghezza uguale e altezza proporzionale al valore da rappresentare.

Esaminiamo ora la situazione nella quale si deve ricreare un'immagine aggiornata dopo un determinato periodo di tempo. Per questo scopo si può usare la funzione *setTimeout*. La funzione **setTimeout** attiva un contatore e, allo scadere del tempo indicato, invoca la funzione specificata. Il *timeout* non agisce ripetitivamente: quindi, se si imposta il *timeout* con il valore di dieci secondi, la funzione associata viene richiamata dopo dieci secondi e non ogni dieci secondi. La funzione può essere comunque resa ripetitiva attraverso un procedimento ricorsivo, come si vede nel progetto seguente. Ovviamente questa funzione non è specifica per le immagini, ma può essere utilizzata in contesti diversi.

## PROGETTO 9

### Creare un orologio analogico.

Come si è visto in precedenza, l'oggetto **Date()** fornisce, tramite i suoi metodi, la data e l'ora corrente. Si vuole costruire un'immagine che rappresenta il quadrante di un orologio analogico.

L'orologio è formato da:

- un cerchio (il quadrante),
- tre segmenti (le lancette delle ore, dei minuti e dei secondi).

Di seguito è riportato il codice completo della pagina che viene poi descritto in dettaglio.

#### Pagina Web (*orologio.html*)

```
<!DOCTYPE html>
<HTML>
<HEAD>
<TITLE>Orologio</TITLE>
<SCRIPT language="javascript">
function cambia_ora(){
 d = new Date();
 // Estrae le ore, i minuti e i secondi
 h = d.getHours();
 m = d.getMinutes();
 s = d.getSeconds();
 ms = d.getMilliseconds();
 s += ms/1000;
 m += s/60;
 h += m/60;
 var canvas = document.getElementById("mioCanvas");
 var context = canvas.getContext("2d");

 // Disegna il quadrante
 context.beginPath();
 context.arc(200, 200, 150, 0, 2 * Math.PI, false);
 context.closePath();
 context.fillStyle = "#EEEEFF";
 context.fill();
 context.lineWidth = 1;
 context.strokeStyle = "#000055";
 context.stroke();

 // Lancetta dei secondi
 var Sangolo = s * Math.PI / 30;
 var Sx = 200 + 140 * Math.sin(Sangolo);
 var Sy = 200 - 140 * Math.cos(Sangolo);
 context.beginPath();
 context.moveTo(200 + 75 * Math.sin(Sangolo), 200 - 75 * Math.cos(Sangolo));
 context.lineTo(Sx, Sy);
 context.closePath();
 context.lineWidth = 1;
 context.stroke();
}
```

```
// Lancetta dei minuti
var Mangolo = m * Math.PI / 30;
var Mx = 200 + 145 * Math.sin(Mangolo);
var My = 200 - 145 * Math.cos(Mangolo);
context.beginPath();
 context.moveTo(200, 200);
 context.lineTo(Mx, My);
context.closePath();
context.lineWidth = 3;
context.stroke();

// Lancetta delle ore
var Hangolo = h * Math.PI / 6;
var Hx = 200 + 100 * Math.sin(Hangolo);
var Hy = 200 - 100 * Math.cos(Hangolo);
context.beginPath();
 context.moveTo(200, 200);
 context.lineTo(Hx, Hy);
context.closePath();
context.lineWidth = 5;
context.stroke();

// Scrive il numero 12 in alto al centro
context.font = "32pt Arial";
context.strokeText("12", 171, 100);

// L'orologio si aggiorna ogni 25 millisecondi
setTimeout("cambia_ora()", 25);
}
</SCRIPT>
</HEAD>
<BODY onLoad="cambia_ora()">
 <CANVAS ID="mioCanvas" width="400" height="400">
 Attenzione: il tuo browser non supporta il tag CANVAS.
 </CANVAS>
</BODY>
</HTML>
```



Al caricamento della pagina Web (evento **onLoad**) viene avviata la funzione *cambia\_ora*. Nel **<BODY>** è definita la *canvas* di dimensioni 400x400 pixel.

```
<BODY onLoad="cambia_ora()">
 <CANVAS ID="mioCanvas" width="400" height="400">
 Attenzione: il tuo browser non supporta il tag CANVAS.
 </CANVAS>
</BODY>
```

Vediamo ora, passo passo, il codice *JavaScript*.

L'oggetto *d* contiene la data corrente:

```
d = new Date();
```

L'oggetto *h* contiene le ore (da 0 a 23), le variabili *m* e *s* i minuti e i secondi (da 0 a 59). Per avere la possibilità di aggiornare l'orologio più di una volta al secondo occorre tenere conto anche dei millisecondi (da 0 a 999).

```
h = d.getHours();
m = d.getMinutes();
s = d.getSeconds();
ms = d.getMilliseconds();
```

Il movimento dell'orologio deve essere il più continuo possibile quindi occorre tener conto dei valori decimali:

```
s += ms/1000;
m += s/60;
h += m/60;
```

Le due seguenti righe di codice effettuano l'accesso alla canvas:

```
var canvas = document.getElementById("mioCanvas");
var context = canvas.getContext("2d");
```

Il quadrante è un cerchio (arco di 360°) avente centro nel punto (200; 200) e raggio 150 pixel.

Viene colorato in azzurro chiaro (#EEEEFF) con il bordo blu scuro (#000055).

```
// Quadrante
context.beginPath();
context.arc(200, 200, 150, 0, 2 * Math.PI, false);
context.closePath();
context.fillStyle = "#EEEEFF";
context.fill();
context.lineWidth = 1;
context.strokeStyle = "#000055";
context.stroke();
```

Per le lancette occorre calcolare l'angolo in base all'orario, operazione che si può fare con una proporzione.

Per i secondi e i minuti un giro intero ( $2\pi$ ) corrisponde a 60:

$$\text{secondi} : \text{angolo} = 60 : 2\pi$$

cioè

$$\text{angolo} = \frac{2\pi \times \text{secondi}}{60} = \frac{\pi \times \text{secondi}}{30}$$

Mentre per le ore un giro corrisponde a 12:

$$\text{ore} : \text{angolo} = 12 : 2\pi$$

cioè

$$\text{angolo} = \frac{2\pi \times \text{ore}}{12} = \frac{\pi \times \text{ore}}{6}$$

Il codice per i calcoli è il seguente:

```
var Sangolo = s * Math.PI / 30;
var Mangolo = m * Math.PI / 30;
var Hangolo = h * Math.PI / 6;
```

Il seguente frammento di codice disegna la lancetta delle ore. La lancetta parte dal centro del quadrante; il punto di arrivo della lancetta si ottiene utilizzando le funzioni goniometriche *seno* e *coseno*.

```
// Lancetta delle ore
var Hangolo = h * Math.PI / 6;
var Hx = 200 + 100 * Math.sin(Hangolo);
var Hy = 200 - 100 * Math.cos(Hangolo);
context.beginPath();
 context.moveTo(200, 200);
 context.lineTo(Hx, Hy);
context.closePath();
context.lineWidth = 5;
context.stroke();
```

In modo analogo si possono creare le lancette dei minuti e dei secondi.

Si scrive poi il numero 12 in alto al centro dell'orologio con il metodo **font** per impostare il tipo di carattere e il metodo **strokeText** per scrivere una stringa di caratteri:

```
context.font = "32pt Arial";
context.strokeText("12", 171, 100);
```

Il metodo *strokeText* ha 3 argomenti: la stringa da scrivere e le coordinate del punto dove posizionare la stringa.

L'ultimo comando richiama periodicamente, in maniera ricorsiva, la funzione *cambia\_ora* e permette di aggiornare l'orario:

```
setTimeout("cambia_ora()", 25);
```

Ponendo uguale a 1000 il periodo di aggiornamento, la lancetta dei secondi scatta ogni secondo.

Diminuendo questo valore (per esempio ponendolo uguale a 50 oppure 25) il movimento diventa più fluido.

## PROGETTO 10

### Realizzare un conto alla rovescia.

L'utente inserisce il numero dei minuti in una casella di testo e preme il pulsante *Set* per impostare il tempo. Il pulsante *Start* avvia il conto alla rovescia. Oltre all'indicatore testuale, il tempo viene rappresentato attraverso un arco di circonferenza che si riduce via via che il tempo scorre. Il pulsante *Stop* termina il conto alla rovescia.



La pagina HTML contiene una casella di testo (*minuti*), tre pulsanti (*Set*, *Start*, *Stop*), un paragrafo per visualizzare il tempo (*tempo*) e il canvas (*canvasRovescia*) per la forma grafica.

```
Minuti:<INPUT TYPE="text" ID="minuti">
<INPUT TYPE="button" ID="set" VALUE="Set" onClick="set()">
<INPUT TYPE="button" ID="start" VALUE="Start" onClick="start()">
<INPUT TYPE="button" ID="stop" VALUE="Stop" onClick="stop()">
<P ID="tempo">00:00</P>
<CANVAS ID="canvasRovescia" WIDTH="400" HEIGHT="400">
 Attenzione: il tuo browser non supporta i canvas.
</CANVAS>
```

La funzione **set** imposta il tempo: legge il valore inserito nella casella di testo e lo assegna alla variabile *m*, mentre la variabile *s* viene posta uguale a zero. Successivamente viene visualizzato nel paragrafo del tempo il numero dei minuti. Vengono inizializzate anche le variabili *angolo* ( $360^\circ$ ) e *secondi* ( $m \cdot 60$ ) che saranno utilizzate per la rappresentazione grafica del tempo.

```
function set() {
 m = eval(document.getElementById("minuti").value);
 s = 0;
 document.getElementById("tempo").innerHTML=m+":00";
 angolo = 2 * Math.PI;
 secondi = m * 60;
}
```

La funzione **start** avvia il contatore: le prime righe decrementano il tempo; viene decrementato il numero dei secondi (se è diverso da zero) oppure, nel caso in cui il numero dei secondi sia nullo, viene decrementato il numero dei minuti e posto il numero del secondo pari a 59. Successivamente viene disegnato l'arco di circonferenza. Ad ogni secondo l'angolo viene decrementato. Le ultime linee richiamano ricorsivamente la funzione fino a quando il tempo si esaurisce.



```
function start() {
 if(s!=0){
 s--;
 }else{
 s=59;
 m--;
 }
 angolo -= 2 * Math.PI/secondi;
 mioCanvas = document.getElementById("canvasRovescia");
 context= mioCanvas.getContext("2d");
 context.clearRect(0, 0, 400, 400);
 context.arc(200, 200, 100, 0, angolo, false);
 context.lineWidth = 150;
 context.strokeStyle = "red";
 context.stroke();
 document.getElementById("tempo").innerHTML=m+": "+s;
 if(s!=0 || m!=0){
 setTimeout("start()", 1000);
 }else{
 alert("Tempo scaduto!");
 }
}
```

Il comando `context.clearRect(0,0,400,400)` viene utilizzato per pulire l'area e poter aggiornare correttamente l'immagine. In questo caso un'area rettangolare che occupa tutto il *canvas*. La funzione **stop** ferma il contatore: modifica il tempo a 00:01 in modo che, al secondo successivo, il tempo si esaurisca.

```
function stop() {
 s=1;
 m=0;
}
```

Di seguito viene presentato il codice completo della pagina Web.

#### Pagina Web (*contoRovescia.html*)

```
<!DOCTYPE html>
<HTML>
<HEAD>
<TITLE>Conto alla rovescia</TITLE>
<SCRIPT language="javascript">
function set() {
 m = eval(document.getElementById("minuti").value);
 s = 0;
 document.getElementById("tempo").innerHTML=m+":00";
 angolo = 2 * Math.PI;
 secondi = m * 60;
}
function start() {
 if(s!=0){
 s--;
 } else {
 s=59;
 }
}
```

```

 m--;
 }
 angolo -= 2 * Math.PI/secondi;
 mioCanvas = document.getElementById("canvasRoveschia");
 context= mioCanvas.getContext("2d");
 context.clearRect(0, 0, 400, 400);
 context.arc(200, 200, 100, 0, angolo, false);
 context.lineWidth = 150;
 context.strokeStyle = "red";
 context.stroke();
 document.getElementById("tempo").innerHTML=m+": "+s;
 if(s!=0 || m!=0){
 setTimeout("start()", 1000);
 } else {
 alert("Tempo scaduto!");
 }
}
function stop() {
 s=1;
 m=0;
}
</SCRIPT>
</HEAD>
<BODY>
Minuti:<INPUT TYPE="text" ID="minuti">
<INPUT TYPE="button" ID="set" VALUE="Set" onClick="set()">
<INPUT TYPE="button" ID="start" VALUE="Start" onClick="start()">
<INPUT TYPE="button" ID="stop" VALUE="Stop" onClick="stop()">
<P ID="tempo">00:00</P>
<CANVAS ID="canvasRoveschia" WIDTH="400" HEIGHT="400">
 Attenzione: il tuo browser non supporta i canvas.
</CANVAS>
</BODY>
</HTML>

```

### 3 Cookie e Webstorage

Un **cookie** (letteralmente *biscotto*) è un pacchetto di informazioni che viene salvato sul computer dell'utente. In successive sessioni di collegamento a Internet (normalmente per un periodo di tempo limitato) il browser può accedere ed utilizzare questi dati.

Utilizzare i *cookie* permette a un sito di ricordare alcune informazioni sull'utente: in alcuni casi può essere molto comodo (per esempio si evita che un utente debba inserire ogni volta il nome utente e la password), ma può generare problemi di sicurezza.

Si possono creare *cookie* in vari modi, per esempio utilizzando il linguaggio *PHP* oppure il linguaggio *JavaScript*.

L'utilizzo dei *cookie* presenta due limiti fondamentali:

- La dimensione massima di un *cookie* è di soli 4 Kbyte e un dominio solitamente non può utilizzare più di 20 *cookie*. Se fino a qualche anno fa i *cookie* venivano utilizzati solo per pochi dati (nome utente, ecc.), con lo sviluppo delle applicazioni Web questo limite è diventato troppo restrittivo.

- Non si possono impostare *cookie* diversi per sessioni contemporanee sullo stesso browser. In passato i *browsers* non facevano uso di schede per la visualizzazione delle pagine, per cui raramente l'utente poteva tenere aperte più pagine Web contemporaneamente. Ora la disponibilità di banda larga e le capacità dei computer permettono la navigazione contemporanea su decine di pagine Web: l'utente può, per esempio, tenere aperte nello stesso momento le caselle di posta elettronica su due *account* differenti.

Nella versione HTML5 la funzionalità di **Webstorage** permette di memorizzare dati in maniera simile ai *cookie* e risolve i problemi appena visti.

*Webstorage* comprende due oggetti principali che possono sostituire le funzionalità dei *cookie*:

- **localStorage**: può contenere dati che sono comuni a tutto il dominio (esattamente come i *cookie*). Se vengono aperte due finestre su uno stesso sito, i dati memorizzati da una pagina del sito sono disponibili nell'altra pagina.
- **sessionStorage**: i dati sono disponibili solamente alla pagina che li ha creati.

Per esempio, supponiamo di aprire in due finestre il sito di *Gmail*. Effettuiamo l'accesso con l'account di posta elettronica:

- Se i dati sono memorizzati in *localStorage*, ci si trova automaticamente collegati anche nella seconda finestra. Effettuando il logout nella seconda pagina, il collegamento viene chiuso anche nella prima pagina.
- Se i dati sono memorizzati in *sessionStorage*, si può effettuare il login nella seconda finestra con un account differente.

Questo non significa che il secondo metodo sia migliore del primo, perchè in alcuni casi può essere utile condividere le informazioni da una finestra all'altra. Per esempio, in un sito di *e-commerce*, i prodotti scelti e presenti nel carrello devono essere gli stessi in tutte le finestre.

Vediamo ora i metodi dell'oggetto *localStorage*.

Per memorizzare un valore si utilizza il metodo **setItem**:

```
localStorage.setItem(nomeVariabile, valore);
```

Per richiamare un valore si utilizza il metodo **getItem**:

```
localStorage.getItem(nomeVariabile);
```

Per cancellare un valore si utilizza il metodo **removeItem**:

```
localStorage.removeItem(nomeVariabile);
```

Per cancellare tutti i valori si utilizza il metodo **clear**:

```
localStorage.clear();
```

Metodi analoghi sono disponibili per l'oggetto *sessionStorage*:

```
sessionStorage.setItem(nomeVariabile, valore);
sessionStorage.getItem(nomeVariabile);
sessionStorage.removeItem(nomeVariabile);
sessionStorage.clear();
```

È sempre opportuno prevedere il comportamento della pagina Web nel caso in cui il browser utilizzato non sia compatibile con *Webstorage*. Al caricamento della pagina Web, si può eseguire la verifica:

```
if (typeof(localStorage) == 'undefined') {
 alert('Attenzione: il tuo browser non supporta Webstorage.');
```

La seguente pagina Web informa se il browser in uso è compatibile con *Webstorage*. La procedura di controllo viene avviata al caricamento della pagina (evento **onLoad**) senza alcun intervento da parte dell'utente.

```
<!DOCTYPE html>
<HTML>
<HEAD>
<SCRIPT language="javascript">
function controllo(){
 if (typeof(localStorage) == 'undefined') {
 alert('Attenzione: il tuo browser non supporta Webstorage.');
```

## PROGETTO 11

### Gestire i dati inseriti dall'utente con un form.

Il form contiene due campi di tipo testo in cui inserire il nome e il cognome e un pulsante per annullare i dati inseriti e svuotare le caselle. L'utente attiva le operazioni con tre pulsanti per: salvare i dati, mostrare i dati ed eliminare i dati (*Salva, Mostra, Elimina*). Quando la pagina viene caricata, oppure quando l'utente fa clic su uno dei pulsanti, i dati vengono aggiornati e visualizzati.

La gestione dei dati viene fatta con l'oggetto **localStorage**.

Le seguenti righe di codice rappresentano il form HTML.

```
<FORM>
 <P>
 Nome:<INPUT TYPE="text" ID="nome">
 Cognome:<INPUT TYPE="text" ID="cognome">
 <INPUT TYPE="reset" VALUE="Annulla" />
 </P>
 <P>
 <INPUT TYPE="button" onClick="salvaDati()" VALUE="Salva" />
 <INPUT TYPE="button" onClick="mostraDati()" VALUE="Mostra" />
 <INPUT TYPE="button" onClick="eliminaDati()" VALUE="Elimina" />
 </P>
</FORM>
```

Le due caselle di testo hanno ID *nome* e *cognome*. Il pulsante *Annulla* svuota le caselle di testo.

Dei tre pulsanti sotto le caselle, il primo avvia la funzione *salvaDati*, il secondo la funzione *mostraDati* e il terzo *eliminaDati*.

Alla fine del form è inserito un paragrafo vuoto chiamato *datiSalvati* per visualizzare i dati:

```
<P ID="datiSalvati"></P>
```

Nel codice JavaScript della funzione *salvaDati*, le prime due righe memorizzano il contenuto delle caselle di testo nelle variabili *nomeDaSalvare* e *cognomeDaSalvare*. Il metodo **setItem** effettua poi la memorizzazione.

```
function salvaDati() {
 var nomeDaSalvare = document.getElementById("nome").value;
 var cognomeDaSalvare = document.getElementById("cognome").value;
 localStorage.setItem("nome", nomeDaSalvare);
 localStorage.setItem("cognome", cognomeDaSalvare);
 mostraDati();
}
```

La funzione *mostraDati* visualizza i dati in un paragrafo sotto il form: le prime due righe recuperano i dati tramite il metodo **getItem**, mentre l'ultima riga visualizza i dati nel paragrafo chiamato *datiSalvati*.

```
function mostraDati() {
 var nome = localStorage.getItem("nome");
 var cognome = localStorage.getItem("cognome");
 document.getElementById("datiSalvati").innerHTML = nome + " " + cognome;
}
```

Infine la funzione *eliminaDati* richiama il metodo **clear** per cancellare tutti i dati.

```
function eliminaDati() {
 localStorage.clear();
 mostraDati();
}
```

#### Pagina Web (*gestioneDati.html*)

```
<!DOCTYPE html>
<HTML>
<HEAD>
<SCRIPT language="javascript">
function salvaDati() {
 var nomeDaSalvare = document.getElementById("nome").value;
 var cognomeDaSalvare = document.getElementById("cognome").value;
 localStorage.setItem("nome", nomeDaSalvare);
 localStorage.setItem("cognome", cognomeDaSalvare);
 alert("Dati salvati.");
 mostraDati();
}
```

```
function mostraDati() {
 var nome = localStorage.getItem("nome");
 var cognome = localStorage.getItem("cognome");
 document.getElementById("datiSalvati").innerHTML = nome + " " + cognome;
}

function eliminaDati() {
 localStorage.clear();
 mostraDati();
}
</SCRIPT>

</HEAD>
<BODY onload="mostraDati()">
<FORM>
 <P>
 Nome:<INPUT TYPE="text" ID="nome">
 Cognome:<INPUT TYPE="text" ID="cognome">
 <INPUT TYPE="reset" VALUE="Annulla" />
 </P>
 <P>
 <INPUT TYPE="button" onClick="salvaDati()" VALUE="Salva" />
 <INPUT TYPE="button" onClick="mostraDati()" VALUE="Mostra" />
 <INPUT TYPE="button" onClick="eliminaDati()" VALUE="Elimina" />
 </P>
</FORM>
<P ID="datiSalvati"></P>
</BODY>
</HTML>
```

All'apertura della pagina Web (evento **onLoad**), viene chiamata la funzione *mostraDati*.

Nome:  Cognome:

Egidio Garibaldi

Chiudendo e riaprendo la finestra (oppure ricaricando la pagina con il pulsante *Aggiorna* del browser), si può osservare che i dati rimangono memorizzati.

Se non ci sono dati memorizzati viene mostrata la stringa *null*.

Si può intercettare questa situazione e gestirla con una struttura *if*, modificando il codice della funzione *mostraDati*:

```
function mostraDati() {
 if(localStorage.getItem("nome")==null || localStorage.getItem("cognome")==null){
 document.getElementById("datiSalvati").innerHTML = "nome o cognome non
 memorizzato";
 } else {
 var nome = localStorage.getItem("nome");
 var cognome = localStorage.getItem("cognome");
 document.getElementById("datiSalvati").innerHTML = nome + " " + cognome;
 }
}
```

## 4 Validazione dei dati di un form

Il linguaggio *JavaScript* veniva utilizzato in passato per effettuare la **validazione** dei dati *lato client*, che è molto più funzionale della validazione *lato server*, in quanto i dati non necessitano di essere inviati e, di conseguenza, l'utente può essere avvisato di eventuali errori o mancanze durante l'inserimento dei dati.

La seguente tabella illustra la differenza tra i due diversi tipi di validazione.

| Validazione lato server                                                                                                                                                                                                                                                                                                                                                                             | Validazione lato client                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ol style="list-style-type: none"> <li>1. L'utente inserisce i dati.</li> <li>2. I dati vengono inviati al server.</li> <li>3. Il server effettua la validazione.</li> <li>4. Se ci sono errori o mancanze si avvisa l'utente che deve correggere o aggiungere i dati mancanti e ripetere l'invio. Se ci sono altri errori la procedura viene ripetuta finché i dati risultano corretti.</li> </ol> | <ol style="list-style-type: none"> <li>1. L'utente inserisce i dati.</li> <li>2. I dati vengono validati sul computer dell'utente.</li> <li>3. Se i dati sono validi, vengono inviati al server.</li> </ol> |

La validazione lato client presenta problemi di sicurezza, nel senso che un malintenzionato potrebbe sfruttare questa vulnerabilità per acquisire dati in modo non autorizzato.

Vediamo la modalità per effettuare la validazione *lato client* con il linguaggio JavaScript per un dato inserito dall'utente con una casella di testo.

Il form contiene una casella di testo, il pulsante di tipo *submit* per l'invio dei dati e il pulsante di tipo *reset* per svuotare la casella.

Il pulsante *Invia* deve richiamare l'esecuzione di una *pagina Php* sul server; prima però viene effettuato il controllo sul dato inserito.

```
<FORM NAME="form1" ACTION="/elaboraDati.php" METHOD="post"
 onSubmit="return validazione()">
 Codice: <INPUT TYPE="text" ID="codice">
 <INPUT TYPE="submit" VALUE="Invia">
 <INPUT TYPE="reset" VALUE="Annulla">
</FORM>
```

Al momento dell'invio (evento **onSubmit**) viene chiamata la funzione *validazione*:

```
function validazione()
{
 var codice = document.getElementById("codice").value;
 if (codice == null || codice == "")
 {
 alert("Attenzione: occorre inserire il codice.");
 return false;
 }
}
```

La funzione recupera il contenuto della casella di testo e, in caso sia *null* oppure vuota, invia un *alert* all'utente impedendo l'invio dei dati al server.

Di seguito è presentato il codice completo.

```
<!DOCTYPE html>
<HTML>
<HEAD>
<SCRIPT language="javascript">
```

```
function validazione()
{
var codice = document.getElementById("codice").value;
if (codice == null || codice == "")
{
alert("Attenzione: occorre inserire il codice.");
return false;
}
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="form1" ACTION="/elaboraDati.php" METHOD="post"
onSubmit="return validazione()">
Codice: <INPUT TYPE="text" ID="codice">
<INPUT TYPE="submit" VALUE="Invia">
<INPUT TYPE="reset" VALUE="Annulla">
</FORM>
</BODY>
</HTML>
```

Come abbiamo già visto nel Capitolo 10, con la versione HTML5 sono stati introdotti alcuni tipi per il tag **<INPUT>** dei form che definiscono criteri di validazione implicita sui dati inseriti dall'utente in una casella.

Per esempio, se si definisce una casella per l'inserimento di un indirizzo di posta elettronica con il tag **<INPUT>** e l'attributo **TYPE** uguale a **email**, il browser effettua una validazione sulla stringa inserita e segnala un messaggio di errore nel caso in cui l'indirizzo non abbia il formato di un indirizzo email (nome, @, dominio):

```
<INPUT TYPE="email" NAME="indirizzo" />
```

giovanni#prova

❗ Inserisci un indirizzo email.

Si osservi che non tutte le versioni dei browser supportano i tipi della versione HTML5 per le caselle di inserimento dei dati in un form. Comunque, nel caso in cui il browser non supporti il tipo, la casella viene considerata come una normale casella di testo.

I principali tipi che introducono la validazione per le caselle di inserimento dei dati in un form sono elencati nella tabella seguente.

| TYPE          | Significato                                                                       | Esempio                                                                                                                      |
|---------------|-----------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| <b>email</b>  | indirizzo di posta elettronica                                                    | <code>&lt;INPUT TYPE="email" NAME="indirizzo" /&gt;</code>                                                                   |
| <b>url</b>    | URL di una risorsa Web                                                            | <code>&lt;INPUT TYPE="url" NAME="sitoWeb" /&gt;</code>                                                                       |
| <b>number</b> | dato numerico (anche con intervallo di valori possibili)                          | <code>&lt;INPUT TYPE="number" NAME="età" /&gt;</code><br><code>&lt;INPUT TYPE="number" NAME="età" min=15 max=30 /&gt;</code> |
| <b>range</b>  | selettore ( <i>slider</i> ) per impostare un valore compreso tra minimo e massimo | <code>&lt;INPUT TYPE="range" NAME="temp" min=5 max=28 /&gt;</code>                                                           |
| <b>date</b>   | data                                                                              | <code>&lt;INPUT TYPE="date" NAME="datanascita" /&gt;</code>                                                                  |



**MATERIALE ONLINE**

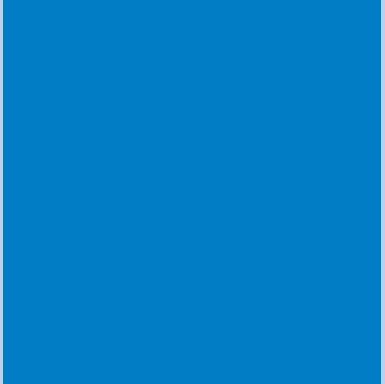
**Codice HTML e JavaScript dei progetti presentati nelle pagine precedenti**



## PROBLEMI

- 1 Visualizzare in una pagina Web l'ora corrente, indicando le ore, i minuti e i secondi. La pagina si deve aggiornare ogni secondo.
- 2 Fare in modo che nella barra del titolo di una pagina Web compaia l'orario (ore e minuti) aggiornato ogni minuto.
- 3 Visualizzare in una pagina Web la data odierna. Utilizzando il costrutto *switch*, mostrare per esteso il nome del mese (se il mese è 1 visualizza *Gennaio*, per 2 visualizza *Febbraio*, ecc.).
- 4 Visualizzare in una pagina Web quanti giorni mancano alla fine dell'anno.
- 5 Creare una pagina Web con un cronometro. Scrivere le funzioni per avviare il cronometro, fermarlo o azzerare il tempo.
- 6 Utilizzare un'immagine dell'Italia per creare una mappa che identifichi il Nord, il Centro e il Sud utilizzando un poligono. Quando l'utente passa con il mouse sopra una zona, vengono mostrate alcune informazioni sulla zona.
- 7 Sull'immagine del problema precedente creare una mappa che, utilizzando piccole circonferenze, identifichi Milano, Roma e Napoli. Quando l'utente passa con il mouse sopra una zona vengono mostrate alcune informazioni sulla città.
- 8 Creare una pagina Web che simuli il lancio di una moneta il cui esito può essere *Testa* o *Croce* (utilizzare il metodo *Math.random()* che genera un numero casuale).
- 9 Creare una pagina Web che simuli il lancio di due dadi, sommi i due valori e mostri il risultato.
- 10 Visualizzare la data dell'ultima modifica della pagina Web, usando la proprietà *lastModified* dell'oggetto *document*.
- 11 Mostrare un messaggio in una finestra di dialogo che si apre quando l'utente preme il tasto del mouse sopra un pulsante.
- 12 Mostrare un messaggio in una finestra di dialogo che si apre quando l'utente passa con il cursore sopra un pulsante (senza necessità di fare clic).
- 13 Per ciascuno dei seguenti problemi, costruire uno script per consentire all'utente di effettuare i calcoli indicati:
  - a. Dato il prezzo di un prodotto, calcolare l'importo dello sconto secondo una percentuale prefissata e il prezzo scontato.
  - b. Data la base e l'altezza di un rettangolo, calcolare l'area.
  - c. Dato il raggio di un cerchio, calcolare la lunghezza della circonferenza.
  - d. Dato un reddito, calcolare l'imposta considerando una percentuale del 20% fino a 30.000 euro e una percentuale del 30% oltre.
- 14 Creare una pagina che conta i clic su un pulsante.
- 15 Creare un pulsante che cambia colore quando il cursore passa sopra di esso.
- 16 Inserire in una pagina una casella combinata tramite l'elemento *Select* di un form HTML. La pagina deve poi mostrare in una finestra di dialogo la voce che l'utente ha scelto dall'elenco (l'oggetto che gestisce la casella combinata contiene la proprietà *selectedIndex* che specifica l'indice della voce selezionata).
- 17 Un atleta inserisce in una pagina Web i chilometri percorsi e il tempo di percorrenza in ore. Tramite uno script calcolare la velocità media di percorrenza espressa in km/h.

- 18 Realizzare un gioco in cui l'utente deve indovinare un numero compreso tra 1 e 100, scelto casualmente da JavaScript con la funzione *Math.random()*. Il gioco inizia quando l'utente preme un pulsante di avvio. Successivamente l'utente tenta di indovinare inserendo il numero in una casella di testo. Ogni volta che inserisce un numero, lo script scrive un messaggio in una finestra di dialogo con uno tra i seguenti tipi di risposta:
  - a. il numero è troppo basso,
  - b. il numero è troppo alto,
  - c. hai indovinato.
- 19 Costruire una pagina Web per visualizzare una raccolta di cinque fotografie. La pagina non deve mostrare tutte le fotografie, ma soltanto una per volta. L'utente può scorrere tutte le fotografie premendo su uno di cinque pulsanti numerati che richiamano il numero dell'immagine.
- 20 In una pagina Web creare un form per l'inserimento del peso di un oggetto. Al momento dell'invio l'utente deve essere avvisato se:
  - a. Non ha inserito alcun valore
  - b. Ha inserito un valore non numerico.
- 21 In una pagina Web creare un form per il calcolo del BMI (*body mass index*) di una persona. L'indice viene calcolato dividendo il peso (in kg) per l'altezza (in metri) al quadrato. La pagina informa poi se il valore è:
  - a. basso (<18)
  - b. corretto (tra 18 e 25)
  - c. alto (>25).
- 22 Utilizzando il tag <CANVAS>, generare un rettangolo avente come dimensioni due valori forniti dall'utente.
- 23 Aggiungere, nel problema precedente, il calcolo dell'area e del perimetro del rettangolo.
- 24 Utilizzando il tag <CANVAS>, generare una circonferenza, con le coordinate del centro e il raggio forniti dall'utente.
- 25 Aggiungere, nel problema precedente, il calcolo dell'area del cerchio e della lunghezza della circonferenza.
- 26 Sovrapporre le forme grafiche del problema 22 e del problema 24 utilizzando colori semi-trasparenti.
- 27 Utilizzando il tag <CANVAS>, creare una areogramma che rappresenti la percentuale di maschi e di femmine della classe.
- 28 Utilizzando il tag <CANVAS>, creare un istogramma che rappresenti tre valori forniti dall'utente.
- 29 Simulare il tiro di un dado e generare l'immagine corrispondente.
- 30 Creare una pagina con un conto alla rovescia grafico (all'avvio viene mostrato un settore circolare che con il passare del tempo diminuisce di ampiezza).
- 31 Creare una pagina Web che memorizzi il numero di telefono di un utente (separatamente prefisso e numero).
- 32 Creare una pagina Web che richieda all'utente il genere (M / F). Modificare lo sfondo della pagina a seconda della risposta. Il dato deve essere memorizzato in modo da mantenere la visualizzazione.
- 33 Creare una pagina Web per dare la possibilità all'utente di scegliere tra tre lingue (italiano, spagnolo e inglese). Il dato deve essere memorizzato in *localStorage*. Da quel momento viene visualizzata una tra le scritte "Ciao!", "Hello!", "¡Hola!".

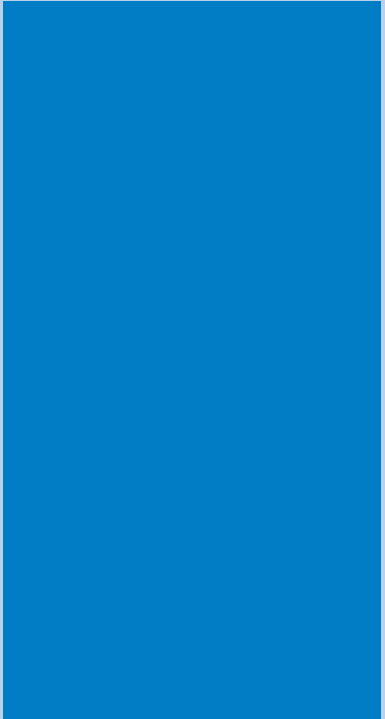


# **FOCUS NOTES**

**LESSICO E TERMINOLOGIA IN LINGUA INGLESE**



**Operating systems**  
**Computer programming**  
**Object Oriented Programming**  
**Computer networks**  
**Internet and Web**  
**Publishing Web pages**  
**Glossary**  
**Acronyms**



# OPERATING SYSTEMS

## Architecture

The operating system architecture, at its highest level, acts as an intermediary between the underlying hardware and the user applications. The applications rarely talk to the underlying hardware directly. Instead, applications communicate with the hardware through a set of well-defined system interfaces that make the user application independent of hardware modifications. This abstraction makes it easy to write applications that work consistently on devices with different hardware capabilities.

The implementation of an operating system can be viewed as a set of layers. At the lower layers of the system are the fundamental services and technologies which work with all applications; higher-level layers contain more sophisticated services and technologies.

## Memory and storage

A computer memory refers to the electronic holding place for instructions and data where the computer's microprocessor can quickly reach (RAM and ROM). Computer storage refers to the permanent computer memory that stores data files and instructions even after the computer system is turned off (magnetic tapes, magnetic and optical disks, flash memories).

## Virtual memory

In a virtual memory, the memory system is organised at two levels: the primary memory, consisting of only the main memory (RAM), and the secondary memory, consisting of the storage devices (disks). The information not immediately required by the CPU for the execution of processes can be moved, and retained, to the auxiliary memory and brought back into the main memory (from the disk) when required by the CPU. The main memory contains only the parts relative to the running computations. It is so that the operating system can accommodate significantly more processes than the main memory would normally do and can initiate new processes even if there is not sufficient space in the main memory to hold the entire process.

## Operating systems for mobile devices

A mobile operating system is the operating system that controls a mobile device such as notebook, smartphone or tablet computer. The most important are:

- *iOS*

iOS is Apple's mobile operating system for iPhone, iPod Touch and iPad. This operating system manages the device hardware and provides various system applications, such as notifications, mail, Safari browser, message services, newsstand for books and magazines, camera, photos, social networks and other standard system services. The user interface is based on touch controls.

- *Android*

Android, developed by Google, is a software stack for mobile devices that includes an operating system, middleware and applications. Android relies on Linux for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

- *Windows Phone*

Windows Phone is a mobile operating system developed by Microsoft for smartphones. It is a compact operating system combined with a suite of basic applications and features for mobile devices. These features include multitouch screen, a user interface with a new modern design, social networking services, e-mail and a mobile version of Microsoft Office programs.

## COMPUTER PROGRAMMING

An *algorithm* is a procedure for solving a problem. In computer science an algorithm is a finite, deterministic, and effective problem-solving method suitable for implementation as a computer program.

*Implementation* defines how something is done by a computer. In computer science terms, *implementation* is the code written in a programming language.

The three tasks of any computer program are: reading input data, processing it and displaying the results.

The data is read from the *standard input* device (keyboard).

The results are displayed on the *standard output* device (screen).

A data value is assigned to the variable with an *assignment* statement.

In a basic program, the instructions are executed sequentially in the order in which they appear in the program (*sequence*).

Decision-making statements alter the sequence of the instructions depending upon certain conditions and transfer the control from one part to other parts of the program (*selection*).

A loop is a group of statements, which are repeated for a certain number of times (*repetition*).

The loops are of two types:

- *counter-controlled loop*, the number of iterations to be performed is defined in advance in the program itself.
- *condition-controlled loop*, loop termination happens on the basis of certain conditions.

In *Bohm-Jacopini's* theory (*theorem*) a well-structured program is built combining the instructions in only three specific control structures: sequence, selection and repetition.

Sequence means statements appearing one after another. Selection is implemented using *if . . . else* and *switch* statements. Repetition is implemented using *for* and *do . . . while* statements.

## OBJECT ORIENTED PROGRAMMING

An *object* is a software construct that encapsulates state and behavior.

A *class* defines the common attributes and behaviors shared by a type of object.

*Attributes* (or properties) are the characteristics of a class.

A *behavior* is an action taken by an object when passed a message or in response to a state change: it's what an object does.

*Method* is a function that defines the behavior of an object. An object can expose an attribute by providing an internal variable or by returning the value through a method.

A class is used to create object instances.

Constructors are methods used to initialize objects during their instantiation.

Objects communicate with one another through *messages*. Messages cause an object to do something. The four main principles of object-oriented programming are: data abstraction, encapsulation, inheritance and polymorphism.

*Data abstraction* is the development of a class as a model of real objects.

*Encapsulation* allows the programmer to build self-contained pieces of software by taking some functionality and hiding its implementation from the outside world.

*Inheritance* allows the programmer to define a new class upon the definition of an existing class. The new class inherits all of the attributes and behaviors (methods) from the existing class.

*Polymorphism*: the same name of a method on objects of different classes represents different code to express many different behaviors.

C++ and Java are the most popular examples of object-oriented computer languages.

# COMPUTER NETWORKS

## Client/Server and P2P

In Client/Server networks, each computer is either a client or a server. A Client requests a service or a resource from the Server and the Server responds by providing that service or resource. The network functions, user administration and applications are centralised on the server. C/S networks provide a high level of security.

A P2P network does not rely for communication on dedicated servers; instead, it uses direct connections between clients (peers). A P2P network has equal peer nodes that simultaneously function both as client and server to the other nodes on the network. A P2P network lacks for proper security policies.

P2P networks were designed primarily for small LANs. Presently most file and video-sharing networks can be classified as P2P network.

## OSI Model

The ISO/OSI Model is the standard model network developers use to visualise the architecture of a computer network. The OSI consists of seven layers:

Layer 1 is the physical layer;

Layer 2 is the data link layer;

Layer 3 is the network layer;

Layer 4 is the transport layer;

Layer 5 is the session layer;

Layer 6 is the presentation layer;

Layer 7 is the application layer.

By using a standard and dividing the network up into layers, it is easier for network administrators:

- to conceptualise and implement a network
- to fix a network problem.

## IPv4 and IPv6

The current version of IP, also known as IP version 4 (IPv4), is showing signs of ageing due to the growth of the Internet and the variety of connection devices. This protocol was designed in the late 1970s and has risen above all other networking protocols to become the world standard for data communications.

The most visible and urgent problem with using IPv4 in the modern Internet is the limited space for public addresses, due to the address length being limited to 32 bits.

IPv6 solves the problems of IPv4 by providing a huge address space. IPv6 addresses are 128 bits long, resulting in an address space with  $3.4 \times 10^{38}$  possible addresses. This is a great amount of address space for the future and allows numerous devices to connect to the Internet supporting mobility more efficiently.

## WiMAX

WiMax is similar to WiFi (wireless network), but broadcasts, using the 802.16 standard, over an entire metropolitan area rather than just at a single location. A wireless Internet service provider (WISP) provides WiMax Internet service. The WISP, like other ISPs, needs to connect to the Internet via a high-speed backbone, so that it can deliver the Internet service to its subscribers. The IEEE 802.16 is the standard technology to send voice, audio, video, and data transmissions across distances of up to 30 miles. In 2001 a group of companies formed a not-for-profit trade organisation called the WiMAX Forum to satisfy the need of connection services over wide areas.

# INTERNET AND WEB

## Digital divide

The digital divide is the gap between those with, and those without the ability to acquire skills and gain access to digital information and the Internet through computers and networks. The causes of digital divide are economic, political, social or generational. Also gender, geographic localisation and culture can result in a divide. In the globalisation era, people who have access to the Internet are at a significant advantage over those who do not.

## Browsing and cookie

Web browsing leaves a trail of cookies, temporary files, Web page history, form data, and other traces that remain on our PC long after we are finished with the browsing. The page history can be visible to other users of that same PC, while the cookies can be viewed by other Web sites. The most recent browser versions create a new environment where all of the potentially sensitive data is held temporarily and then discarded when the browser window is closed. For example, cookies are held in the memory and are cleared when the browser shuts down. Temporary files are stored on the hard disk, but are deleted when the browser shuts down.

## Netiquette rules

In any chat area, message board, instant message, social network or email you must not:

- Defame, harass or threaten others
- Make any hateful or racially offensive statements
- Discuss illegal activities with the intent to commit them
- Post or distribute any material that violates any third party right or any law
- Post or distribute any vulgar, obscene, discourteous, or indecent language or images
- Post material or make statements that do not generally pertain to the designated topic or theme of a chat room or bulletin board.

## Twitter: an information network

Twitter is a real-time information network that connects you to the latest information about what you find interesting. Simply identify the public streams you find most compelling and follow the conversations.

At the heart of Twitter are small bursts of information called Tweets. Each Tweet is 140 characters in length, but do not be fooled by the small size: you can share a lot within a little space. Connected to each Tweet is a rich details pane that provides: additional information, deeper context and embedded media. You can tell your story within your Tweet, or you can think of a Tweet as the headline, and use the details pane to tell the rest with photos, videos and other media content.

Following someone on Twitter means: you are subscribing to their Tweets as a Follower; their updates will appear in your timeline; that person has permission to send you private Tweets. Followers are people who receive your Tweets. If someone follows you, they: become your follower and will display in the 'newest follower' section; receive your Tweets in their home page. If you follow someone, you are their follower.

(*twitter.com*)

# PUBLISHING WEB PAGES

## HTML5

HTML was primarily designed as a language for semantically describing scientific documents, however its general design and the adaptations over the years have enabled it to be used to describe a variety of document types.

HTML5 defines the 5th major revision of the core language of the World Wide Web: the HyperText Markup Language (HTML). In this version, new features are introduced to assist Web application authors, new elements are introduced based on research into prevailing authoring practices, and special attention has been given to defining clear conformance criteria for user agents in an effort to improve interoperability ([www.w3.org](http://www.w3.org)).

HTML5 provides a semantic-level markup language for authoring accessible pages on the Web. A basic HTML document looks like this:

```
<!doctype html>
<html>
 <head>
 <title>Sample page</title>
 </head>
 <body>
 <h1>Sample page</h1>
 <p>This is a simple sample.</p>
 <!-- this is a comment -->
 </body>
</html>
```

HTML documents consist of a tree of elements and text. Each element is denoted in the source by a start tag, such as "<body>", and an end tag, such as "</body>". Certain end tags can occasionally be omitted and are implied by other tags.

## CSS3

CSS (Cascading Style Sheets) is a language for describing the on screen rendering of HTML and XML documents. CSS is used to control the style and layout of Web pages. CSS3 is the latest standard for CSS.

CSS provides web browsers with the information they need to control the visual aspect of a web page, such as: the position of HTML elements, text styles, backgrounds, colours and images, and much more.

Advanced CSS techniques give website authors the ability to tailor layouts and designs for mobile web browsers, as well as the skills they need to create websites for regular desktop browsers. To set the text color of the H1 elements to red, we can write the following CSS rule:

```
h1 { color: red }
```

A CSS rule consists of two main parts: selector ('h1') and declaration ('color: red'). Selectors can be HTML tags or CSS defined selectors, such as a 'class' or an 'ID'.

External style sheets can be used for multiple Web pages. To apply styles, the external CSS file (for example style.css) is referenced by adding a link tag in HTML head section as follows:

```
<head>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
```



## GLOSSARY

*Access point:* In a wireless LAN the device through which several computers access the connection to the network.

*Algorithm:* a process or a set of rules used for calculation or problem solving. In computer programming the term is used referring to instructions given to a computer.

*Anchor:* An anchor is another word for an internal link or bookmark on a web page. An anchor is a link that links within the same page of the document.

*Apache:* Apache is a Web server, the software that manages a server for access via HTTP. More than 50% of Web sites worldwide use Apache. It is free, it is powerful, and is supported by Apache Software Foundation, an active team of developers, to update the Web server and its features. Apache is typically the Web server on a Linux computer but Windows also supports Apache.

*Apps:* Web applications, or web apps, combine the power of the Internet with the simplicity of the multi-touch technology of smartphones. Visiting Apple Store or Android market, a smartphone user can download a web app on his mobile device.

*Attribute:* a part of a class whose value contributes to the state definition of a class.

*Backbone:* A high-speed connection, or group of connections, that serves as the major data artery in a network.

*Behavior:* the way an object acts and reacts, the way it changes its state and passes the message.

*<body>...</body>:* Enclose everything visible on Web page.

*Boot:* Term that means to start the computer. Booting a computer generally refers to loading the basic files, such as the operating system, required to run it.

*Buffer:* A temporary storage area in a computer's memory, usually RAM, that holds recent modifications to files and other information to be written later-on to the hard drive.

*Compiler:* a computer program which translates a program, written in a *high level language* (programming language), into an equivalent *machine language* program.

*Constructor:* an operation that creates an object and initialises its state.

*Data structure:* a way of storing data in a computer so that it can be organised and retrieved efficiently.

*Debugging:* a process of finding and removing the *bugs* in a computer program that produces an incorrect or unexpected result.

*Declaration:* A corresponding pair of property and value parameters which suggest a CSS style for a given selector.

*Destructor:* an operation that destroys the object and frees the space in the computer memory.

*Friend:* a function of a class that may reference the private parts of another class.

*Gateway:* The point in a network that serves as an entrance to another network.

*<head>...</head>:* Starts the header part of any HTML page. Everything between these is mainly used to help the browser and search engines to classify Web pages.

## GLOSSARY

*<html>...</html>*: Standard opening and closing tags for any HTML page.

*Host*: A computer that shares information and services with other computers (client). A host can also be a Web server (Web host).

*Implementation*: converting an algorithm into a computer program using a particular programming language and an operating system.

*Information hiding*: hiding the structure of an object, as well as the implementation of its methods.

*Interface*: the outside view of a class while hiding its structure and its behavior.

*Interrupt*: A command that tells the processor to stop what it is doing and wait for further instruction.

*Iteration*: repetition of a process within a computer program (synonymous with repetition or loop).

*Kernel*: The main part of an operating system. The kernel handles the most basic, but important tasks, such as starting processes and managing interrupts.

*Link*: An image or portion of text that is high-lighted in some way (usually underlined on the Web) and connects the current document to another. Hyperlinks are what make plain text into hypertext.

*MAC address*: The MAC (Media Access Control) address is a unique, factory-assigned address for every Ethernet and Wi-Fi adapter. A MAC address consists of six two-digit hexadecimal numbers separated by colons, such as 0B:E2:44:03:01:BF.

*Machine language*: the binary language in which the CPU and the hardware of a computer are to be programmed.

*Method*: an operation on an object, defined as part of the declaration of a class.

*Multitasking*: A function of the operating system: it means that the operating system can simultaneously perform multiple tasks.

*Operand*: an operand is a data item on which operators perform operations.

*Operating system*: System software that controls and manages all of the hardware and software. The resources managed by an operating system include memory, files, processes, and input and output.

*Operator*: an operator indicates an operation to be performed on data.

*Optical fibre*: Optical fibre is the transparent material used to transmit light pulses. Fiber-optic networks are more expensive to install and maintain, but optical fibres are more durable than copper wires. Optical fibers deliver a huge amount of bandwidth at high speeds.

*Packet switching*: Source computers split the message over a network or the Internet into smaller parts called packets, each one taking the fastest possible route as it travels to its final destination. The destination computer reassembles the packets into the original transmission as they arrive.

## GLOSSARY

*Phishing*: The act of trying to trick users into giving up personal information by making them think they're dealing with a real bank or office. The email claims to be from a legitimate company and the user is solicited to click on a URL and supply password, credit card number or other personal information. The link is similar to that of the real site, but the URL is on a server controlled by the phisher.

*Podcast*: Podcasting is a Web technology to publish files on Internet allowing users to subscribe to a feed and receive new files automatically by way of a subscription. Podcast files are typically audio files in mp3 format or episodes of radio broadcasts.

*Private*: a declaration that is not visible to any other classes or objects.

*Pseudo class*: A syntax for selecting markup elements which have subsidiary attributes in the context of document navigation or positioning, including hyperlinks.

*Public*: a declaration that is visible to all other classes and objects.

*Router*: A router is a specialized device that connects a network to the Internet and decides the best route the packet should use to reach its destination.

*Script*: A text that contains a sequence of commands that the processor executes in batch mode.

*Selector*: A logical identifier by which style declarations are associated with specific elements of an HTML document.

*Shell*: A program that provides an interface between the operating system and the user.

*Spam*: Unsolicited email on the Internet sent to a large number of users.

*Stylesheet*: A collection of one or more style rules, specified in an HTML style element or external CSS file attached by an HTML link element.

*Superclass*: the class from which another class inherits.

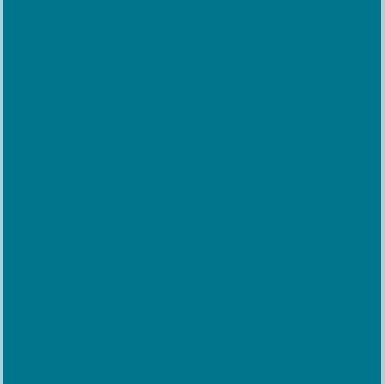
*Tag*: Tags are the commands that give HTML documents their functionality. They consist of keywords placed within < and >.

*Thread*: A small, independent set of operations belonging to a larger program being executed.

*Unified Modeling Language (UML)*: A standard notation used during object-oriented analysis and design to model class and superclass.

## ACRONYMS

|         |                                                                 |         |                                                 |
|---------|-----------------------------------------------------------------|---------|-------------------------------------------------|
| ADSL    | Asymmetrical Digital Subscriber Line                            | JPEG    | Joint Photographic Experts Group                |
| ccTLD   | country code Top-Level Domain                                   | LAN     | Local Area Network                              |
| CPU     | Central Processing Unit                                         | Mbps    | Megabit per second                              |
| CSP     | Cloud Service Provider                                          | MIME    | Multipurpose Internet Mail Extensions           |
| CSS     | Cascading Style Sheets                                          | MIPS    | Million Instructions Per Second                 |
| DHCP    | Dynamic Host Configuration Protocol                             | MPEG    | Moving Picture Experts Group                    |
| DLL     | Dynamic Link Library                                            | NIC     | Network Interface Card                          |
| DNS     | Domain Name System                                              | OS      | Operating system                                |
| DTD     | Document Type Definition                                        | PAN     | Personal Area Network                           |
| EDGE    | Enhanced Data rates for GSM Evolution                           | PDF     | Portable Document Format                        |
| FS      | File System                                                     | PNG     | Portable Network Graphics                       |
| FTP     | File Transfer Protocol                                          | POP     | Post Office Protocol                            |
| Gbps    | Gigabit per second                                              | RAM     | Random Access Memory                            |
| GIF     | Graphics Interchange Format                                     | RGB     | Red Green Blue                                  |
| GPRS    | General Packet Radio Service                                    | ROM     | Read Only Memory                                |
| GSM     | Global System for Mobile communications                         | RSS     | Really Simple Syndication                       |
| gTLD    | generic Top-Level Domain                                        | SaaS    | Software as a Service                           |
| GUI     | Graphical User Interface                                        | SMTP    | Simple Mail Transfer Protocol                   |
| HSPA    | High Speed Packet Access                                        | SPOOL   | Simultaneous Peripheral Operation On Line       |
| HTML    | HyperText Markup Language                                       | SSH     | Secure Shell                                    |
| HTTP    | HyperText Transfer Protocol                                     | TCP/IP  | Transmission Control Protocol/Internet Protocol |
| HTTPS   | HyperText Transfer Protocol Secure                              | TLD     | Top-Level Domain                                |
| I/O     | Input/Output                                                    | UMTS    | Universal Mobile Telecommunications System      |
| ICANN   | Internet Corporation for Assigned Names and Numbers             | URL     | Uniform Resource Locator                        |
| IEEE    | Institute of Electrical and Electronics Engineers               | UTF     | Unicode Transformation Format                   |
| IIS     | Internet Information Services                                   | UTP     | Unshielded Twisted Pair                         |
| IM      | Instant Messaging                                               | VoIP    | Voice Over IP                                   |
| IMAP    | Internet Message Access Protocol                                | VPN     | Virtual Private Network                         |
| IPL     | Initial Program Loader                                          | W3C     | World Wide Web Consortium                       |
| ISDN    | Integrated Services Digital Network                             | WAI     | Web Accessibility Initiative                    |
| ISO/OSI | International Standard Organization/Open System Interconnection | WAN     | Wide Area Network                               |
| ISP     | Internet Service Provider                                       | WCAG    | Web Content Accessibility Guidelines            |
|         |                                                                 | WiMAX   | Worldwide Interoperability for Microwave Access |
|         |                                                                 | WLAN    | Wireless Local Area Network                     |
|         |                                                                 | WWW     | World Wide Web                                  |
|         |                                                                 | WYSIWYG | What You See Is What You Get                    |



# APPENDICE



**Tabella dei caratteri**

**Pseudocodifica**

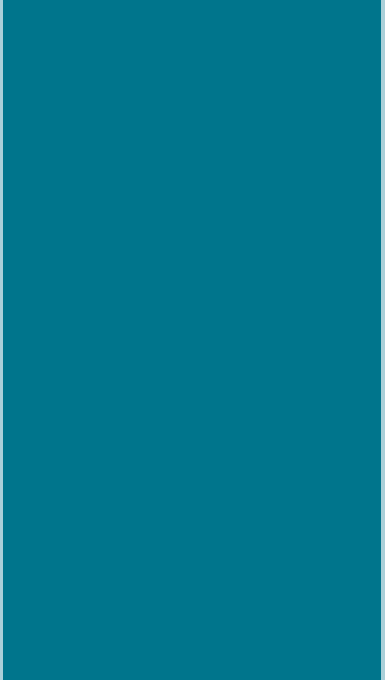
**Linguaggio C++**

**Tag HTML**

**Tabella dei colori**

**Soluzioni alle domande  
di autoverifica**

**Indice analitico**



## Tabella dei caratteri

### Codice ASCII

| Valore<br>decim. | Caratt. | Nome<br>HTML | Valore<br>decim. | Caratt. | Nome<br>HTML | Valore<br>decim. | Caratt. | Nome<br>HTML |
|------------------|---------|--------------|------------------|---------|--------------|------------------|---------|--------------|
| 0                | NUL     |              | 43               | +       |              | 86               | V       |              |
| 1                | SOH     |              | 44               | ,       |              | 87               | W       |              |
| 2                | STX     |              | 45               | -       |              | 88               | X       |              |
| 3                | ETX     |              | 46               | .       |              | 89               | Y       |              |
| 4                | EOT     |              | 47               | /       |              | 90               | Z       |              |
| 5                | ENQ     |              | 48               | 0       |              | 91               | [       |              |
| 6                | ACK     |              | 49               | 1       |              | 92               | \       |              |
| 7                | BEL     |              | 50               | 2       |              | 93               | ]       |              |
| 8                | BS      |              | 51               | 3       |              | 94               | ^       |              |
| 9                | HT      |              | 52               | 4       |              | 95               | _       |              |
| 10               | LF      |              | 53               | 5       |              | 96               | `       |              |
| 11               | VT      |              | 54               | 6       |              | 97               | a       |              |
| 12               | FF      |              | 55               | 7       |              | 98               | b       |              |
| 13               | CR      |              | 56               | 8       |              | 99               | c       |              |
| 14               | SO      |              | 57               | 9       |              | 100              | d       |              |
| 15               | SI      |              | 58               | :       |              | 101              | e       |              |
| 16               | DLE     |              | 59               | ;       |              | 102              | f       |              |
| 17               | DC1     |              | 60               | <       | &lt;         | 103              | g       |              |
| 18               | DC2     |              | 61               | =       |              | 104              | h       |              |
| 19               | DC3     |              | 62               | >       | &gt;         | 105              | i       |              |
| 20               | DC4     |              | 63               | ?       |              | 106              | j       |              |
| 21               | NAK     |              | 64               | @       |              | 107              | k       |              |
| 22               | SYN     |              | 65               | A       |              | 108              | l       |              |
| 23               | ETB     |              | 66               | B       |              | 109              | m       |              |
| 24               | CAN     |              | 67               | C       |              | 110              | n       |              |
| 25               | EM      |              | 68               | D       |              | 111              | o       |              |
| 26               | SUB     |              | 69               | E       |              | 112              | p       |              |
| 27               | ESC     |              | 70               | F       |              | 113              | q       |              |
| 28               | FS      |              | 71               | G       |              | 114              | r       |              |
| 29               | GS      |              | 72               | H       |              | 115              | s       |              |
| 30               | RS      |              | 73               | I       |              | 116              | t       |              |
| 31               | US      |              | 74               | J       |              | 117              | u       |              |
| 32               | SP      |              | 75               | K       |              | 118              | v       |              |
| 33               | !       |              | 76               | L       |              | 119              | w       |              |
| 34               | "       | &quot;       | 77               | M       |              | 120              | x       |              |
| 35               | #       |              | 78               | N       |              | 121              | y       |              |
| 36               | \$      |              | 79               | O       |              | 122              | z       |              |
| 37               | %       |              | 80               | P       |              | 123              | {       |              |
| 38               | &       | &amp;        | 81               | Q       |              | 124              |         |              |
| 39               | '       |              | 82               | R       |              | 125              | }       |              |
| 40               | (       |              | 83               | S       |              | 126              | ~       |              |
| 41               | )       |              | 84               | T       |              | 127              | DEL     |              |
| 42               | *       |              | 85               | U       |              |                  |         |              |

## Tabella dei caratteri

### ISO 8859-1, Unicode Latin-1

| Valore<br>decim. | Caratt. | Nome<br>HTML | Valore<br>decim. | Caratt. | Nome<br>HTML | Valore<br>decim. | Caratt. | Nome<br>HTML |
|------------------|---------|--------------|------------------|---------|--------------|------------------|---------|--------------|
| 128              | €       | &euro;       | 171              | «       | &laquo;      | 214              | Ö       | &Ouml;       |
| 129              |         |              | 172              | ¬       | &not;        | 215              | ×       | &times;      |
| 130              | ,       | &sbquo;      | 173              |         | &shy;        | 216              | Ø       | &Oslash;     |
| 131              | f       | &fnof;       | 174              | ®       | &reg;        | 217              | Ù       | &Ugrave;     |
| 132              | ”       | &bdquo;      | 175              | –       | &macr;       | 218              | Ú       | &Uacute;     |
| 133              | ...     | &hellip;     | 176              | °       | &deg;        | 219              | Û       | &Ucirc;      |
| 134              | †       | &dagger;     | 177              | ±       | &plusmn;     | 220              | Ü       | &Uuml;       |
| 135              | ‡       | &Dagger;     | 178              | ²       | &sup2;       | 221              | Ý       | &Yacute;     |
| 136              | ^       | &circ;       | 179              | ³       | &sup3;       | 222              | Þ       | &THORN;      |
| 137              | ‰       | &permil;     | 180              | ´       | &acute;      | 223              | ß       | &szlig;      |
| 138              | Š       | &Scaron;     | 181              | μ       | &micro;      | 224              | à       | &agrave;     |
| 139              | ‹       | &lsaquo;     | 182              | ¶       | &para;       | 225              | á       | &aacute;     |
| 140              | Œ       | &OElig;      | 183              | ·       | &middot;     | 226              | â       | &acirc;      |
| 141              |         |              | 184              | ¸       | &cedil;      | 227              | ã       | &atilde;     |
| 142              | Ž       |              | 185              | ¹       | &sup1;       | 228              | ä       | &auml;       |
| 143              |         |              | 186              | º       | &ordm;       | 229              | å       | &aring;      |
| 144              |         |              | 187              | »       | &raquo;      | 230              | æ       | &aelig;      |
| 145              | ‘       | &lsquo;      | 188              | ¼       | &frac14;     | 231              | ç       | &ccedil;     |
| 146              | ’       | &rsquo;      | 189              | ½       | &frac12;     | 232              | è       | &egrave;     |
| 147              | “       | &ldquo;      | 190              | ¾       | &frac34;     | 233              | é       | &eacute;     |
| 148              | ”       | &rdquo;      | 191              | ¿       | &quest;      | 234              | ê       | &ecirc;      |
| 149              | •       | &bull;       | 192              | À       | &Agrave;     | 235              | ë       | &euml;       |
| 150              | —       | &ndash;      | 193              | Á       | &Aacute;     | 236              | ì       | &igrave;     |
| 151              | —       | &mdash;      | 194              | Â       | &Acirc;      | 237              | í       | &iacute;     |
| 152              | ~       | &tilde;      | 195              | Ã       | &Atilde;     | 238              | î       | &icirc;      |
| 153              | ™       | &trade;      | 196              | Ä       | &Auml;       | 239              | ï       | &iuml;       |
| 154              | š       | &scaron;     | 197              | Å       | &Aring;      | 240              | ð       | &eth;        |
| 155              | ›       | &rsaquo;     | 198              | Æ       | &AElig;      | 241              | ñ       | &ntilde;     |
| 156              | œ       | &oelig;      | 199              | Ç       | &Ccedil;     | 242              | ò       | &ograve;     |
| 157              |         |              | 200              | È       | &Egrave;     | 243              | ó       | &oacute;     |
| 158              | ž       |              | 201              | É       | &Eacute;     | 244              | ô       | &ocirc;      |
| 159              | ÿ       | &yuml;       | 202              | Ê       | &Ecirc;      | 245              | õ       | &otilde;     |
| 160              |         | &nbsp;       | 203              | Ë       | &Euml;       | 246              | ö       | &ouml;       |
| 161              | ¡       | &iexcl;      | 204              | Ì       | &Igrave;     | 247              | ÷       | &divide;     |
| 162              | ¢       | &cent;       | 205              | Í       | &Iacute;     | 248              | ø       | &oslash;     |
| 163              | £       | &pound;      | 206              | Î       | &Icirc;      | 249              | ù       | &ugrave;     |
| 164              | £       | &curren;     | 207              | Ï       | &Iuml;       | 250              | ú       | &uacute;     |
| 165              | ¥       | &yen;        | 208              | Ð       | &ETH;        | 251              | û       | &ucirc;      |
| 166              | ¡       | &brvbar;     | 209              | Ñ       | &Ntilde;     | 252              | ü       | &uuml;       |
| 167              | §       | &sect;       | 210              | Ò       | &Ograve;     | 253              | ý       | &yacute;     |
| 168              | ”       | &uml;        | 211              | Ó       | &Oacute;     | 254              | þ       | &thorn;      |
| 169              | ©       | &copy;       | 212              | Ô       | &Ocirc;      | 255              | ÿ       | &yuml;       |
| 170              | ®       | &ordf;       | 213              | Õ       | &Otilde;     |                  |         |              |

# Pseudocodifica

## Regole generali

### ■ Tipi di dati standard

- Numerico intero
- Numerico reale, non intero
- Carattere, dato formato da un solo carattere
- Stringa (o *alfanumerico*), sequenza qualsiasi di caratteri
- Logico (o *booleano*), variabile che può assumere solo uno tra i due valori *Vero* o *Falso*.

### ■ Commenti

/\* ..... \*/

### ■ Assegnazione

assegna variabile = valore

calcola variabile = espressione

### ■ Operatori aritmetici

- + addizione
- sottrazione
- \* moltiplicazione
- / divisione
- % resto

### ■ Operatori di relazione

- = uguale
- < minore di
- <= minore o uguale di
- > maggiore
- >= maggiore o uguale di
- <> diverso

## Parole chiave

### ■ Operatori logici

- and prodotto logico (coniunzione)
- or somma logica (disgiunzione)
- not negazione
- xor or esclusivo

### ■ Inizio e termine dell'algoritmo

inizio  
fine

### ■ Operazioni di ingresso/uscita

immetti variabile  
scrivi variabile  
scrivi messaggio

### ■ Strutture di controllo

*Sequenza*

.....  
istruzione1  
istruzione2  
istruzione3  
.....

*Alternativa (o Selezione)*

se condizione  
allora  
    istruzione1  
altrimenti  
    istruzione2  
fine se

*Selezione multipla*

caso variabile =  
    lista valori1: istruzioni1  
    lista valori2: istruzioni2  
    .....  
    lista valorin: istruzionin  
altrimenti  
    istruzioni  
fine caso

*Ripetizione (o Iterazione)*

esegui  
    istruzioni  
ripeti finché condizione  
  
esegui mentre condizione  
    istruzioni  
ripeti

per variabile-contatore da min a max  
    istruzioni  
ripeti

### ■ Funzioni

funzione *nome* (ref *parametri formali*)  
inizio

.....  
.....  
fine

funzione *nome* (val *parametri formali*)  
inizio

.....  
.....  
fine



# Linguaggio C++

## Parole riservate

|                   |                     |                         |                    |                 |
|-------------------|---------------------|-------------------------|--------------------|-----------------|
| <b>auto</b>       | <b>double</b>       | <b>int</b>              | <b>short</b>       | <b>typeid</b>   |
| <b>bool</b>       | <b>dynamic_cast</b> | <b>long</b>             | <b>signed</b>      | <b>typename</b> |
| <b>break</b>      | <b>else</b>         | <b>main</b>             | <b>sizeof</b>      | <b>union</b>    |
| <b>case</b>       | <b>enum</b>         | <b>mutable</b>          | <b>static</b>      | <b>unsigned</b> |
| <b>catch</b>      | <b>explicit</b>     | <b>namespace</b>        | <b>static_cast</b> | <b>using</b>    |
| <b>char</b>       | <b>extern</b>       | <b>new</b>              | <b>struct</b>      | <b>virtual</b>  |
| <b>class</b>      | <b>false</b>        | <b>operator</b>         | <b>switch</b>      | <b>void</b>     |
| <b>const</b>      | <b>float</b>        | <b>private</b>          | <b>template</b>    | <b>volatile</b> |
| <b>const_cast</b> | <b>for</b>          | <b>protected</b>        | <b>this</b>        | <b>while</b>    |
| <b>continue</b>   | <b>friend</b>       | <b>public</b>           | <b>throw</b>       |                 |
| <b>default</b>    | <b>goto</b>         | <b>register</b>         | <b>true</b>        |                 |
| <b>delete</b>     | <b>if</b>           | <b>reinterpret_cast</b> | <b>try</b>         |                 |
| <b>do</b>         | <b>inline</b>       | <b>return</b>           | <b>typedef</b>     |                 |

## Frasi di commento

// .....

## Istruzioni di input e output standard

*unità\_input* >> *variabile*;  
*unità\_output* << *variabile*;

## Tipi di dati

|           | Tipo                      | Intervallo                                                                        | Bit utilizzati |
|-----------|---------------------------|-----------------------------------------------------------------------------------|----------------|
| booleani  | <b>bool</b>               | true/false                                                                        | 8 bit          |
| caratteri | <b>unsigned char</b>      | Da 0 a 255                                                                        | 8 bit          |
|           | <b>char</b>               | Da -128 a 127                                                                     | 8 bit          |
|           | <b>wchar_t</b>            | carattere esteso                                                                  | 16 bit         |
| interi    | <b>unsigned short int</b> | Da 0 a 65.535                                                                     | 16 bit         |
|           | <b>short int</b>          | Da -32.768 a 32.767                                                               | 16 bit         |
|           | <b>int</b>                | Da -2.147.483.648 a 2.147.483.647                                                 | 32 bit         |
|           | <b>unsigned long int</b>  | Da 0 a 4,294,967,295                                                              | 32 bit         |
|           | <b>long int</b>           | Da -2.147.483.648 a 2.147.483.647                                                 | 32 bit         |
| reali     | <b>float</b>              | Da $1,17549435 \cdot 10^{-38}$ a $3,40282347 \cdot 10^{+38}$                      | 32 bit         |
|           | <b>double</b>             | Da $2,2250738585072014 \cdot 10^{-308}$<br>a $1,7976931348623157 \cdot 10^{+308}$ | 64 bit         |
|           | <b>long double</b>        | Da $3,4 \cdot 10^{-4932}$ a $1,1 \cdot 10^{4932}$                                 | 80 bit         |

## APPENDICE

## Operatori algebrici

|   |                                              |
|---|----------------------------------------------|
| + | addizione                                    |
| - | sottrazione                                  |
| * | moltiplicazione                              |
| / | divisione                                    |
| % | calcolo del resto della divisione tra interi |

## Operatori di assegnamento

| Operatore | Utilizzo                     | Esempio | Istruzione equivalente |
|-----------|------------------------------|---------|------------------------|
| ++        | Incremento unitario          | x++;    | x = x + 1;             |
| --        | Decremento unitario          | x--;    | x = x - 1;             |
| +=        | Incremento                   | x+= y;  | x = x + y;             |
| -=        | Decremento                   | x-= y;  | x = x - y;             |
| *=        | Moltiplicazione              | x*= y;  | x = x * y;             |
| /=        | Divisione                    | x/= y;  | x = x / y;             |
| %=        | Resto della divisione intera | x%=y;   | x = x % y;             |

## Operatori di confronto

|    |                   |
|----|-------------------|
| == | uguale            |
| >  | maggiore          |
| <  | minore            |
| <= | minore o uguale   |
| >= | maggiore o uguale |
| != | diverso           |

## Connettivi logici

|    |                                           |
|----|-------------------------------------------|
| && | operazione di congiunzione ( <i>And</i> ) |
|    | operazione di disgiunzione ( <i>Or</i> )  |
| !  | negazione ( <i>Not</i> ).                 |

## Sequenze di escape

| Sequenza di escape | Descrizione                      | Terminologia inglese   |
|--------------------|----------------------------------|------------------------|
| \n                 | A capo riga                      | <i>new line</i>        |
| \t                 | Tabulazione                      | <i>tab</i>             |
| \r                 | Ritorno a capo della stessa riga | <i>carriage return</i> |
| \"                 | Doppi apici                      | <i>double quote</i>    |
| \\                 | Barra contraria                  | <i>backslash</i>       |
| \b                 | Una battuta indietro             | <i>backspace</i>       |
| \'                 | Apice singolo                    | <i>single quote</i>    |
| \?                 | Punto di domanda                 | <i>question mark</i>   |
| \a                 | Segnalazione acustica            | <i>bell</i>            |
| \0                 | Fine stringa                     | <i>end of string</i>   |
| \f                 | Salto pagina                     | <i>form feed</i>       |

## Strutture di controllo

### Sequenza

```
{
.....
.....
.....
}
```

### Selezione

```
if (condizione) {
 istruzioni-a;
}
else {
 istruzioni-b;
}
```

```
if (condizione) istruzione-a;
else istruzione-b;
```

```
if (condizione) {
 istruzioni-a;
}
```

### Selezione multipla

```
switch(variabile) {
 case valore-1:
 istruzioni-1;
 break;
 case valore-2:
 istruzioni-2;
 break;

 case valore-n:
 istruzioni-n;
 break;
 default:
 istruzioni;
 break;
}
```

### Ripetizione

```
do {
 :
 ;
} while (condizione);
```

```
while (condizione) {
 :
 ;
}
```

```
for (i=min; i<=max; i++) {
 :
 ;
}
```

## Manipolatori di I/O formattato (*io manip*)

| Manipolatore                  | Descrizione                                                             |
|-------------------------------|-------------------------------------------------------------------------|
| <b>endl</b>                   | carattere di fine riga (equivalente a '\n')                             |
| <b>ends</b>                   | carattere di fine stringa (equivalente a '\0')                          |
| <b>dec</b>                    | imposta la base decimale                                                |
| <b>oct</b>                    | imposta la base ottale                                                  |
| <b>hex</b>                    | imposta la base esadecimale                                             |
| <b>setw(<i>b</i>)</b>         | imposta il numero di battute (default <i>b</i> =0, senza formattazione) |
| <b>setfill(<i>c</i>)</b>      | imposta il carattere di riempimento (default <i>c</i> =spazio)          |
| <b>setprecision(<i>d</i>)</b> | imposta il numero di decimali (default <i>d</i> =6)                     |
| <b>flush</b>                  | scarica il buffer di output ed effettua la scrittura                    |

## Librerie standard del linguaggio

|                         |                                      |
|-------------------------|--------------------------------------|
| <b>&lt;iostream&gt;</b> | Input e output standard              |
| <b>&lt;cstdlib&gt;</b>  | Funzioni miste                       |
| <b>&lt;cstring&gt;</b>  | Array di caratteri                   |
| <b>&lt;cctype&gt;</b>   | Tipi carattere                       |
| <b>&lt;cmath&gt;</b>    | Matematica in <i>floating point</i>  |
| <b>&lt;ctime&gt;</b>    | Data e ora                           |
| <b>&lt;fstream&gt;</b>  | Flussi di dati                       |
| <b>&lt;iomanip&gt;</b>  | Manipolatori per l'output formattato |
| <b>&lt;string&gt;</b>   | Stringhe                             |

## Operatori per i quali è consentito l'overloading

|    |    |     |        |     |     |
|----|----|-----|--------|-----|-----|
| +  | −  | *   | /      | %   | ^   |
| !  | =  | <   | >      | +=  | -=  |
| ^= | &= | =   | <<     | >>  | <<= |
| <= | >= | &&  |        | ++  | --  |
| () | [] | new | delete | &   |     |
| ~  | *= | /=  | %=     | >>= | ==  |
| != | ,  | ->  | ->*    |     |     |

## Classi

```
class NomeClasse {
 // attributi
 tipo1 attributo1;
 tipo2 attributo1;
 ...
 // metodi
 tipo1 funzione1;
 tipo2 funzione2;
 ...
};

tipo NomeClasse::funzione1
{
 ...
 return;
}
```

## Metodi di <string>

|                   |                            |                           |                  |
|-------------------|----------------------------|---------------------------|------------------|
| <b>append()</b>   | <b>copy()</b>              | <b>find_last_of()</b>     | <b>rend()</b>    |
| <b>assign()</b>   | <b>data()</b>              | <b>find_last_not_of()</b> | <b>replace()</b> |
| <b>at()</b>       | <b>empty()</b>             | <b>get_allocator()</b>    | <b>reserve()</b> |
| <b>begin()</b>    | <b>end()</b>               | <b>getline()</b>          | <b>resize()</b>  |
| <b>c_str()</b>    | <b>erase()</b>             | <b>insert()</b>           | <b>rfind()</b>   |
| <b>capacity()</b> | <b>find()</b>              | <b>length()</b>           | <b>size()</b>    |
| <b>clear()</b>    | <b>find_first_of()</b>     | <b>max_size()</b>         | <b>substr()</b>  |
| <b>compare()</b>  | <b>find_first_not_of()</b> | <b>rbegin()</b>           | <b>swap()</b>    |

**Standard Template Library (STL)**

|              |
|--------------|
| <algorithm>  |
| <deque>      |
| <functional> |
| <iterator>   |
| <list>       |
| <map>        |
| <memory>     |
| <numeric>    |
| <queue>      |
| <set>        |
| <stack>      |
| <utility>    |
| <vector>     |

**Contenitori della libreria STL**

| Sequenze                                     |
|----------------------------------------------|
| vector<br>list<br>deque (double-ended queue) |
| Adattatori                                   |
| stack<br>queue<br>priority queue             |
| Contenitori associativi                      |
| bitset<br>map<br>multimap<br>set<br>multiset |

**Metodi dei contenitori vector, list e map**

| <vector>        | <list>          | <map>           |
|-----------------|-----------------|-----------------|
| assign()        | assign()        | begin()         |
| at()            | back()          | clear()         |
| back()          | begin()         | count()         |
| begin()         | clear()         | empty()         |
| capacity()      | empty()         | end()           |
| clear()         | end()           | equal_range()   |
| empty()         | erase()         | erase()         |
| end()           | front()         | find()          |
| erase()         | get_allocator() | get_allocator() |
| front()         | insert()        | insert()        |
| get_allocator() | max_size()      | key_comp()      |
| insert()        | merge()         | lower_bound()   |
| max_size()      | pop_back()      | max_size()      |
| pop_back()      | pop_front()     | rbegin()        |
| push_back()     | push_back()     | rend()          |
| rbegin()        | push_front()    | size()          |
| rend()          | rbegin()        | swap()          |
| reserve()       | remove()        | upper_bound()   |
| resize()        | remove_if()     | value_comp()    |
| size()          | rend()          |                 |
| swap()          | resize()        |                 |
|                 | reverse()       |                 |
|                 | size()          |                 |
|                 | sort()          |                 |
|                 | splice()        |                 |
|                 | swap()          |                 |
|                 | unique()        |                 |

**Iteratori**

|                        |
|------------------------|
| iterator               |
| input_iterator         |
| output_iterator        |
| forward_iterator       |
| bidirectional_iterator |
| random_iterator        |
| reverse_iterator       |
| istream_iterator       |
| ostream_iterator       |
| ostream_iterator       |

**Algoritmi della libreria standard (<algorithm>)**

|                      |                      |                                 |
|----------------------|----------------------|---------------------------------|
| <b>accumulate</b>    | <b>for_each</b>      | <b>Search</b>                   |
| <b>adjacent</b>      | <b>inner_product</b> | <b>search_n</b>                 |
| <b>difference</b>    | <b>max</b>           | <b>set_difference</b>           |
| <b>adjacent_find</b> | <b>max_element</b>   | <b>set_intersection</b>         |
| <b>binary_search</b> | <b>merge</b>         | <b>set_symmetric_difference</b> |
| <b>copy</b>          | <b>min</b>           | <b>set_union</b>                |
| <b>copy_backward</b> | <b>min_element</b>   | <b>sort</b>                     |
| <b>count</b>         | <b>mismatch</b>      | <b>swap</b>                     |
| <b>equal</b>         | <b>partition</b>     | <b>transform</b>                |
| <b>equal_range</b>   | <b>remove</b>        | <b>unique</b>                   |
| <b>fill</b>          | <b>replace</b>       |                                 |
| <b>find</b>          | <b>reverse</b>       |                                 |

**Stream I/O**

| <b>Operatori (&lt;iostream&gt;)</b>                                                            | <b>Classi stream (&lt;fstream&gt; e &lt;strstream&gt;)</b>                                                                                                    |
|------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <<<br>>>                                                                                       | fstream (file I/O)<br>ifstream (file input)<br>ofstream (file output)<br>strstream (stringa I/O)<br>istrstream (stringa input)<br>ostrstream (stringa output) |
| <b>Oggetti stream aperti automaticamente</b>                                                   |                                                                                                                                                               |
| cin (standard input)<br>cout (standard output)<br>cerr (standard error)<br>clog (standard log) |                                                                                                                                                               |

**Modalità di apertura per stream (<fstream>)**

| <b>Modalità di apertura</b> | <b>Descrizione</b>                                                                                                                                                                                                             |
|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ios::app</b>             | Apre un file di output per l'inserimento in coda al file.                                                                                                                                                                      |
| <b>ios::ate</b>             | Apre un file esistente, sia di input sia di output, e si posiziona alla fine del file.                                                                                                                                         |
| <b>ios::in</b>              | Apre un file in input. Utilizzando <i>ios::in</i> con un <i>ofstream</i> si ha una sovrascrittura carattere per carattere, conservando i dati successivi.                                                                      |
| <b>ios::out</b>             | Apre un file in output. Utilizzando <i>ios::out</i> con un <i>ofstream</i> (è il <i>default</i> ) senza aggiungere <i>ios::app</i> , <i>ios::ate</i> , oppure <i>ios::in</i> implicitamente si effettua un <i>ios::trunc</i> . |
| <b>ios::nocreate</b>        | Apre un file solamente se questo esiste. Questa modalità è presente solamente nel file d'intestazione <i>fstream.h</i> .                                                                                                       |
| <b>ios::noreplace</b>       | Apre un file solamente se questo non esiste. Questa modalità è presente solamente nel file d'intestazione <i>fstream.h</i> .                                                                                                   |
| <b>ios::trunc</b>           | Apre un file distruggendone la copia precedente se già esistente.                                                                                                                                                              |
| <b>ios::binary</b>          | Apre un file in modalità binaria. Se omissso l'apertura avviene in modalità testo.                                                                                                                                             |

**Metodi di `<fstream>`**

|                |                  |                    |                          |                 |
|----------------|------------------|--------------------|--------------------------|-----------------|
| <b>bad()</b>   | <b>flags()</b>   | <b>ignore()</b>    | <b>rdstate()</b>         | <b>tellg()</b>  |
| <b>clear()</b> | <b>flush()</b>   | <b>open()</b>      | <b>read()</b>            | <b>tellp()</b>  |
| <b>close()</b> | <b>gcount()</b>  | <b>peek()</b>      | <b>seekg()</b>           | <b>unsetf()</b> |
| <b>eof()</b>   | <b>get()</b>     | <b>precision()</b> | <b>seekp()</b>           | <b>width()</b>  |
| <b>fail()</b>  | <b>getline()</b> | <b>put()</b>       | <b>setf()</b>            | <b>write()</b>  |
| <b>fill()</b>  | <b>good()</b>    | <b>putback()</b>   | <b>sync_with_stdio()</b> |                 |

**Manipolatori di I/O formattato (`<iomanip>`)**

| Costante               | Descrizione                                                                                                                                                                                                                      |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ios::skipws</b>     | Elimina gli spazi in testa al buffer.                                                                                                                                                                                            |
| <b>ios::left</b>       | Allinea a sinistra riempiendo i restanti caratteri a destra con il carattere di riempimento definito da <i>setfill()</i> .                                                                                                       |
| <b>ios::right</b>      | Allinea a destra (è il default) riempiendo i caratteri mancanti a sinistra con il carattere di riempimento definito da <i>setfill()</i> .                                                                                        |
| <b>ios::internal</b>   | Inserisci i caratteri di riempimento solamente dopo l'eventuale segno e prima del valore.                                                                                                                                        |
| <b>ios::dec</b>        | Rappresenta i valori numerici in base 10 (default).                                                                                                                                                                              |
| <b>ios::oct</b>        | Rappresenta i valori numerici in base 8 (ottale).                                                                                                                                                                                |
| <b>ios::hex</b>        | Rappresenta i valori numerici in base 16 (esadecimale).                                                                                                                                                                          |
| <b>ios::showbase</b>   | Antepone la base ai numeri ottali (0) ed esadecimali (0x) secondo la notazione del linguaggio C.                                                                                                                                 |
| <b>ios::showpoint</b>  | Mostra il punto decimale e i relativi zeri dei numeri in virgola mobile ( <i>float</i> e <i>double</i> ) che hanno valore intero (non contengono decimali).                                                                      |
| <b>ios::uppercase</b>  | Rappresenta le lettere dei numeri esadecimali e la lettera E (esponente) del formato scientifico in maiuscolo.                                                                                                                   |
| <b>ios::showpos</b>    | Mostra il segno anche con i numeri positivi (+).                                                                                                                                                                                 |
| <b>ios::scientific</b> | Rappresenta nel formato scientifico i numeri in virgola mobile ( <i>float</i> e <i>double</i> ).                                                                                                                                 |
| <b>ios::fixed</b>      | Rappresenta i numeri in virgola mobile ( <i>float</i> e <i>double</i> ) con un numero fisso di decimali.                                                                                                                         |
| <b>ios::unitbuf</b>    | Forza lo stream di output a scaricare tutti i caratteri in coda (come <i>flush</i> ) in ciascuna operazione di scrittura, eliminando di fatto l'accodamento nel buffer. Per default solamente <i>cerr</i> non accoda nel buffer. |

# Tag HTML

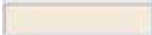
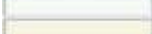
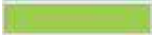
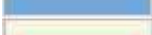
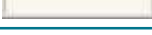
| TAG         | Funzione                                                 | Attributi           | Esempio                                                                                              | Note                                                            |
|-------------|----------------------------------------------------------|---------------------|------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|
| <a>         | Inserisce un link (àncora)                               | href, title, target | <a href="index.html">Home</a>                                                                        |                                                                 |
| <address>   | Formatta indirizzi che appaiono in corsivo               |                     | <address><a href="mailto:info@ht.com.org">Scrivici</a><br />Indirizzo: Via Roma 10, Milano</address> |                                                                 |
| <b>         | Formatta il testo in grassetto                           |                     | <b>Testo in grassetto</b>                                                                            |                                                                 |
| <body>      | Corpo del documento                                      |                     | <body>Contenuto del documento...</body>                                                              | Contenuto della pagina WEB                                      |
| <br />      | Inserisce un ritorno a capo                              |                     | Il testo contiene <br /> un a capo.                                                                  | È un tag vuoto                                                  |
| <div>       | Inserisce un contenitore generico, indica una sezione.   |                     | <div><h1>Titolo</h1><p>paragrafo</p></div>                                                           | Non presenta spaziatura. Spesso contiene altri elementi blocco. |
| <!doctype>  | Prima dichiarazione di un documento HTML5                |                     | <!doctype HTML>                                                                                      | Precede il tag <html>. Non ha una chiusura.                     |
| <em>        | Enfatizza il testo formattandolo in corsivo              |                     | <em>testo enfaticizzato</em>                                                                         |                                                                 |
| <h1> - <h6> | Inserisce un titolo                                      |                     | <h1>Titolo della pagina</h1>                                                                         | Possibili valori: h1,h2,h3,h4,h5,h6                             |
| <head>      | Testa del documento.                                     |                     | <head><title>Titolo del documento</title></head>                                                     | Contiene metatag, script, collegamenti ad altri file e <title>. |
| <hgroup>    | Raggruppa l'intestazione del documento (elementi titolo) |                     | <hgroup><h1>Benvenuti</h1><h2>Buona navigazione</h2></hgroup>                                        |                                                                 |
| <hr />      | Inserisce una barra orizzontale                          |                     | <h1>Titolo</h1><br /><p>Titolo e paragrafo sono separati da una barra</p>                            | È un tag vuoto                                                  |

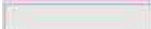
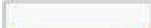
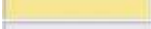
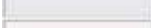
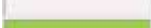
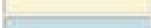
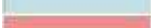
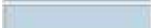
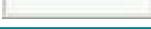


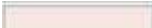
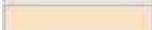
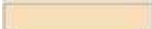
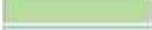
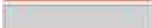
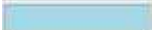
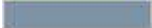
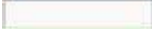
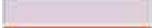
| TAG      | Funzione                                          | Attributi              | Esempio                                                                                                                                    | Note                                            |
|----------|---------------------------------------------------|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------|
| <html>   | Delimita un documento HTML                        |                        | <html lang="it">                                                                                                                           | Tag radice                                      |
| <i>      | Formatta il testo in corsivo                      |                        | <i>testo corsivo</i>                                                                                                                       |                                                 |
| <img />  | Inserisce un'immagine                             | src, alt               |                                                                                                    | Non ha tag di chiusura                          |
| <li>     | Inserisce una voce in un elenco                   |                        | <ol><br><li>Acqua</li><br><li>The</li><br><li>Latte</li><br></ol>                                                                          |                                                 |
| <link>   | Collega un foglio di stile CSS esterno            | rel, type, href, media | <link rel="stylesheet" type="text/css" href="stile.css" media="all" />                                                                     |                                                 |
| <mark>   | Evidenzia una stringa                             |                        | <p>Il sistema solare comprende il pianeta<br><mark>Terra</mark></p>                                                                        |                                                 |
| <meta /> | Veicola meta-informazioni per i motori di ricerca | charset, name, content | <meta charset="utf-8">                                                                                                                     | Contenuto all'interno di <head>                 |
| <ol>     | Inserisce un elenco ordinato                      | start, type            | <ol><br><li>Acqua</li><br><li>The</li><br><li>Latte</li><br></ol>                                                                          |                                                 |
| <p>      | Inserisce un paragrafo                            |                        | <p>Questo e' un paragrafo</p>                                                                                                              | Presenta una spaziatura prima e dopo l'elemento |
| <small>  | Ridimensiona il carattere                         |                        | <p>C'era una volta, <small>un re, diranno i miei piccoli lettori.</small> No ragazzi, avete sbagliato.</p>                                 |                                                 |
| <strong> | Dà peso al testo formattandolo in grassetto       |                        | <p>Prima del decollo<br><strong>allacciare le cinture di sicurezza</strong></p>                                                            |                                                 |
| <table>  | Inserisce una tabella                             | border                 | <table><br><tr><br><th>Gennaio</th><br><th>Febbraio</th><br></tr><br><tr><br><td>31 giorni</td><br><td>28 giorni</td><br></tr><br></table> |                                                 |

| TAG     | Funzione                                           | Attributi         | Esempio                                                                                                                                                                                                               | Note                                                        |
|---------|----------------------------------------------------|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------|
| <td>    | Inserisce una cella in una tabella                 | colspan, rowspan  | <pre>&lt;table&gt;   &lt;tr&gt;     &lt;td&gt;Cella 1&lt;/td&gt;     &lt;td&gt;Cella 2&lt;/td&gt;   &lt;/tr&gt; &lt;/table&gt;</pre>                                                                                  |                                                             |
| <th>    | Inserisce una cella di intestazione in una tabella | colspan, rowspan  | <pre>&lt;table&gt; &lt;tr&gt; &lt;th&gt;Intestazione1&lt;/th&gt; &lt;th&gt;Intestazione2&lt;/th&gt; &lt;/tr&gt; &lt;tr&gt; &lt;td&gt;Cella 1&lt;/td&gt; &lt;td&gt;Cella 2&lt;/td&gt; &lt;/tr&gt; &lt;/table&gt;</pre> |                                                             |
| <time>  | Formatta stringhe temporali                        | datetime, pubdate | <pre>&lt;p&gt;Il negozio apre alle &lt;time&gt;9:00&lt;/time&gt; tutti i giorni.&lt;/p&gt;  &lt;p&gt;Scadenza al &lt;time datetime="2020-12-31"&gt;del prestito&lt;/time&gt;&lt;/p&gt;</pre>                          | Utile per i motori di ricerca                               |
| <title> | Titolo del documento                               |                   | <pre>&lt;title&gt;Titolo del documento &lt;/title&gt;</pre>                                                                                                                                                           | Il titolo è visualizzato nella barra del titolo del browser |
| <tr>    | Inserisce una riga in una tabella                  |                   | <pre>&lt;table&gt;   &lt;tr&gt;     &lt;td&gt;Cella 1&lt;/td&gt;     &lt;td&gt;Cella 2&lt;/td&gt;   &lt;/tr&gt; &lt;/table&gt;</pre>                                                                                  |                                                             |
| <u>     | Inserisce una sottolineatura                       |                   | <pre>&lt;p&gt;&lt;u&gt;testo sottolineato &lt;/u&gt;&lt;/p&gt;</pre>                                                                                                                                                  |                                                             |
| <ul>    | Inserisce un elenco non ordinato                   |                   | <pre>&lt;ul&gt;   &lt;li&gt;Acqua&lt;/li&gt;   &lt;li&gt;The&lt;/li&gt;   &lt;li&gt;Latte&lt;/li&gt; &lt;/ul&gt;</pre>                                                                                                |                                                             |

## Tabella dei colori

| Nome simbolico |                                                                                     | Red | Green | Blue | Valore esadecimale |
|----------------|-------------------------------------------------------------------------------------|-----|-------|------|--------------------|
| aliceblue      |    | 240 | 248   | 255  | F0F8FF             |
| antiquewhite   |    | 250 | 235   | 215  | FAEBD7             |
| aqua           |    | 0   | 255   | 255  | 00FFFF             |
| aquamarine     |    | 127 | 255   | 212  | 7FFFD4             |
| azure          |    | 240 | 255   | 255  | F0FFFF             |
| beige          |    | 245 | 245   | 220  | F5F5DC             |
| bisque         |    | 255 | 228   | 196  | FFE4C4             |
| black          |    | 0   | 0     | 0    | 000000             |
| blanchedalmond |    | 255 | 235   | 205  | FFEBCD             |
| blue           |    | 0   | 0     | 255  | 0000FF             |
| blueviolet     |    | 138 | 43    | 226  | 8A2BE2             |
| brown          |    | 165 | 42    | 42   | A52A2A             |
| burlywood      |    | 222 | 184   | 135  | DEB887             |
| cadetblue      |    | 95  | 158   | 160  | 5F9EA0             |
| chartreuse     |    | 127 | 255   | 0    | 7FFF00             |
| chocolate      |    | 210 | 105   | 30   | D2691E             |
| coral          |   | 255 | 127   | 80   | FF7F50             |
| cornflowerblue |  | 100 | 149   | 237  | 6495ED             |
| cornsilk       |  | 255 | 248   | 220  | FFF8DC             |
| crimson        |  | 220 | 20    | 60   | DC143C             |
| cyan           |  | 0   | 255   | 255  | 00FFFF             |
| darkblue       |  | 0   | 0     | 139  | 00008B             |
| darkcyan       |  | 0   | 139   | 139  | 008B8B             |
| darkgoldenrod  |  | 184 | 134   | 11   | B8860B             |
| darkgray       |  | 169 | 169   | 169  | A9A9A9             |
| darkgreen      |  | 0   | 100   | 0    | 006400             |
| darkkhaki      |  | 189 | 183   | 107  | BDB76B             |
| darkmagenta    |  | 139 | 0     | 139  | 8B008B             |
| darkolivegreen |  | 85  | 107   | 47   | 55662F             |
| darkorange     |  | 255 | 140   | 0    | FF8C00             |
| darkorchid     |  | 153 | 50    | 204  | 9932CC             |
| darkred        |  | 139 | 0     | 0    | 8B0000             |
| darksalmon     |  | 233 | 150   | 122  | E9967A             |
| darkseagreen   |  | 143 | 188   | 143  | 8FBC8F             |
| darkslateblue  |  | 72  | 61    | 139  | 483D8B             |
| darkslategray  |  | 47  | 79    | 79   | 2F4F4F             |
| darkturquoise  |  | 0   | 206   | 209  | 00CED1             |
| darkviolet     |  | 148 | 0     | 211  | 9400D3             |
| deeppink       |  | 255 | 20    | 147  | FF1493             |
| deepskyblue    |  | 0   | 191   | 255  | 00BFFF             |
| dimgray        |  | 105 | 105   | 105  | 696969             |
| dodgerblue     |  | 30  | 144   | 255  | 1E90FF             |
| firebrick      |  | 178 | 34    | 34   | B22222             |
| floralwhite    |  | 255 | 250   | 240  | FFFAF0             |

| Nome simbolico       |                                                                                     | Red | Green | Blue | Valore esadecimale |
|----------------------|-------------------------------------------------------------------------------------|-----|-------|------|--------------------|
| forestgreen          |    | 34  | 139   | 34   | 228B22             |
| fuchsia              |    | 255 | 0     | 255  | FF00FF             |
| gainsboro            |    | 220 | 220   | 220  | DCDCDC             |
| ghostwhite           |    | 248 | 248   | 255  | F8F8FF             |
| gold                 |    | 255 | 215   | 0    | FFD700             |
| goldenrod            |    | 218 | 165   | 32   | DAA520             |
| gray                 |    | 128 | 128   | 128  | 808080             |
| green                |    | 0   | 255   | 0    | 00FF00             |
| greenyellow          |    | 173 | 255   | 47   | ADFF2F             |
| honeydew             |    | 240 | 255   | 240  | F0FFF0             |
| hotpink              |    | 255 | 105   | 180  | FF69B4             |
| indianred            |    | 205 | 92    | 92   | CD5C5C             |
| indigo               |    | 75  | 0     | 130  | 4B0082             |
| ivory                |    | 255 | 255   | 240  | FFFFF0             |
| khaki                |    | 240 | 230   | 140  | F0E68C             |
| lavender             |    | 230 | 230   | 250  | E6E6FA             |
| lavenderblush        |    | 255 | 240   | 245  | FFF0F5             |
| lawngreen            |    | 124 | 252   | 0    | 7CFC00             |
| lemonchiffon         |    | 255 | 250   | 205  | FFFACD             |
| lightblue            |    | 173 | 216   | 230  | ADD8E6             |
| lightcoral           |    | 240 | 128   | 128  | F08080             |
| lightcyan            |   | 224 | 255   | 255  | E0FFFF             |
| lightgoldenrodyellow |  | 250 | 250   | 210  | FAFAD2             |
| lightgreen           |  | 144 | 238   | 144  | 90EE90             |
| lightgrey            |  | 211 | 211   | 211  | D3D3D3             |
| lightpink            |  | 255 | 182   | 193  | FFB6C1             |
| lightsalmon          |  | 255 | 160   | 122  | FFA07A             |
| lightseagreen        |  | 32  | 178   | 170  | 20B2AA             |
| lightskyblue         |  | 135 | 206   | 250  | 87CEFA             |
| lightslategray       |  | 119 | 136   | 153  | 778899             |
| lightsteelblue       |  | 176 | 196   | 222  | B0C4DE             |
| lightyellow          |  | 255 | 255   | 224  | FFFFE0             |
| lime                 |  | 0   | 255   | 0    | 00FF00             |
| limegreen            |  | 50  | 205   | 50   | 32CD32             |
| linen                |  | 250 | 240   | 230  | FAF0E6             |
| magenta              |  | 255 | 0     | 255  | FF00FF             |
| maroon               |  | 128 | 0     | 0    | 800000             |
| mediumaquamarine     |  | 102 | 205   | 170  | 66CDAA             |
| mediumblue           |  | 0   | 0     | 205  | 0000CD             |
| mediumorchid         |  | 186 | 85    | 211  | BA55D3             |
| mediumpurple         |  | 147 | 112   | 219  | 9370DB             |
| mediumseagreen       |  | 60  | 179   | 113  | 3CB371             |
| mediumslateblue      |  | 123 | 104   | 238  | 7B68EE             |
| mediumspringgreen    |  | 0   | 250   | 154  | 00FA9A             |
| mediumturquoise      |  | 72  | 209   | 204  | 48D1CC             |
| mediumvioletred      |  | 199 | 21    | 133  | C71585             |
| midnightblue         |  | 25  | 25    | 112  | 191970             |
| mintcream            |  | 245 | 255   | 250  | F5FFFA             |

| Nome simbolico |                                                                                     | Red | Green | Blue | Valore esadecimale |
|----------------|-------------------------------------------------------------------------------------|-----|-------|------|--------------------|
| mistyrose      |    | 255 | 228   | 225  | FFE4E1             |
| moccasin       |    | 255 | 228   | 181  | FFE4B5             |
| navajowhite    |    | 255 | 222   | 173  | FFDEAD             |
| navy           |    | 0   | 0     | 128  | 000080             |
| oldlace        |    | 253 | 245   | 230  | FDF5E6             |
| olive          |    | 128 | 128   | 0    | 808000             |
| olivedrab      |    | 107 | 142   | 35   | 6B8E23             |
| orange         |    | 255 | 165   | 0    | FFA500             |
| orangered      |    | 255 | 69    | 0    | FF4500             |
| orchid         |    | 218 | 112   | 214  | DA70D6             |
| palegoldenrod  |    | 238 | 232   | 170  | EEE8AA             |
| palegreen      |    | 152 | 251   | 152  | 98FB98             |
| paleturquoise  |    | 175 | 238   | 238  | AFEEEE             |
| palevioletred  |    | 219 | 112   | 147  | DB7093             |
| papayawhip     |    | 255 | 239   | 213  | FFEFD5             |
| peachpuff      |    | 255 | 218   | 185  | FFDA69             |
| peru           |    | 205 | 133   | 63   | CD853F             |
| pink           |    | 255 | 192   | 203  | FFC0CB             |
| plum           |    | 221 | 160   | 221  | DDA0DD             |
| powderblue     |    | 176 | 224   | 230  | B0E0E6             |
| purple         |    | 128 | 0     | 128  | 800080             |
| red            |   | 255 | 0     | 0    | FF0000             |
| rosybrown      |  | 188 | 143   | 143  | BC8F8F             |
| royalblue      |  | 65  | 105   | 225  | 4169E1             |
| saddlebrown    |  | 139 | 69    | 19   | 8B4513             |
| salmon         |  | 250 | 128   | 114  | FA8072             |
| sandybrown     |  | 244 | 164   | 96   | F4A460             |
| seagreen       |  | 46  | 139   | 87   | 2E8B57             |
| seashell       |  | 255 | 245   | 238  | FFF5EE             |
| sienna         |  | 160 | 82    | 45   | A0522D             |
| silver         |  | 192 | 192   | 192  | C0C0C0             |
| skyblue        |  | 135 | 206   | 235  | 87CEEB             |
| slateblue      |  | 106 | 90    | 205  | 6A5ACD             |
| slategray      |  | 112 | 128   | 144  | 708090             |
| snow           |  | 255 | 250   | 250  | FFFAFA             |
| springgreen    |  | 0   | 255   | 127  | 00FF7F             |
| steelblue      |  | 70  | 130   | 180  | 4682B4             |
| tan            |  | 210 | 180   | 140  | D2B48C             |
| teal           |  | 0   | 128   | 128  | 008080             |
| thistle        |  | 216 | 191   | 216  | D8BFD8             |
| tomato         |  | 255 | 99    | 71   | 006347             |
| turquoise      |  | 64  | 224   | 208  | 40E0D0             |
| violet         |  | 238 | 130   | 238  | EE82EE             |
| wheat          |  | 245 | 222   | 179  | F5DEB3             |
| white          |  | 255 | 255   | 255  | FFFFFF             |
| whitesmoke     |  | 245 | 245   | 245  | F5F5F5             |
| yellow         |  | 255 | 255   | 0    | FFFF00             |
| yellowgreen    |  | 154 | 205   | 50   | 9ACD32             |

# Soluzioni alle domande di autoverifica

## Capitolo 1

1. c
2. a, b, c
3. c
4. b
5. b
6. c
7. b
8. b
9. a
10. V F V V
11.  $01111111 : 127 = 2^7 - 1$ ;  
 $10000000 : -128 = -2^7$
12. XOR; AND; OR
13. a) sia p che q sono veri;  
b) sia p che q sono falsi;  
c) p e q hanno valori di verità diversi;  
d) almeno uno dei due predicati è falso
14. a4, b1, c5, d2, e6, f3
15. c
16. (prezzo diverso da 80) and (peso <=50)  
(nascita diversa da Milano) or (età > 18)  
(peso diverso da 80) and (età <=50)  
(città diversa da Roma) or (lingua = italiana)  
(distanza >= 50) and (popolazione <= 20000)  
(prezzo >= 500) and (quantità >= 100)  
(distanza <= 5) or (estensione <= 190.000)  
(superficie diversa da 300) and (perimetro <= 100)  
(nascita diversa da Milano) or (statura >1.70)

## Capitolo 2

1. b, c, d, g, h
2. a
3. b
4. F V F V
5. b
6. a

7. d
8. d
9. a
10. a
11. c
12. c
13. a-4,5,9, b-2,7, c-1,6, d8, e3
14. a, c
15. d
16. c

## Capitolo 3

1. F F V V F
2. c
3. b
4. b
5. a
6. a3, b1, c8, d2, e7, f4, g6, h5
7. a) barra delle applicazioni,  
b) chiusura,  
c) cartella,  
d) guida in linea,  
e) finestra
8. F F F V F F
9. b, d
10. F V F V

## Capitolo 4

1. b
2. d
3. c
4. c
5. a) istruzioni,  
b) processo,  
c) processore
6. I, I, O, I, O, O
7. b
8. c
9. c
10. F V F F
11. 2
12. 1, 1
13. V F V F
14. No
15. b
16. d
17. a
18. V F F V

19. a) procedure,  
b) istruzione,  
c) valore,  
d) procedimento di calcolo
20. c
21. F V F V

## Capitolo 5

1. abitanti, peso specifico, iniziale di un nome
2. string, string, short, float oppure double
3. -2.147483.648,  
+2.147483.647
4. a) 32,  
b) =,  
c) const,  
d) assegnazione
5. c
6. int main, int, cout <<
7. a) 400  
b) 8  
c) 0.08
8. a) 3, -3, 6  
b) 0, 0, 0  
c) 1, 1, 2
9. a) 4  
b) 9  
c) 8
10. a2, b4, c1, d3
11. c
12. b, d, e
13. a) cin >>,  
b) cout<<,  
c) input,  
d) output
14. F V F V
15. a2, b3, c1
16. a) sorgente,  
b) eseguibile,  
c) oggetto,  
d) variabile,  
e) stringa
17. b
18. if (a != 0)
19. a) 2  
b) 0 10
20. 0, 6
21. 2, 6

22. 6, 12, 24, 48
23. 6
24. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
25. 11
26. V F F V
27. a) 1  
b) 6  
c) 10, 25
28. Il prototipo e l'implementazione della funzione hanno tipi diversi per il valore restituito (float e void).
29. F V F F
30. c
31. a2, b1, c3
32. d
33. b
34. 5, 2
35. 3, 4, 0
36. c
37. c
38. a
39. c
40. a
41. b
42. no

## Capitolo 6

1. a) oggetti,  
b) attributi,  
c) metodi
2. V V V V
3. F F V
4. a
5. a
6. V V F V V
7. b
8. a) pubblici,  
b) privati,  
c) pubblici
9. c
10. V F F
11. c
12. b
13. a3, b1, c2
14. f
15. a3, b1, c3, d2
16. F V V F F
17. V V F V
18. V F V F

## Capitolo 7

1. La funzione ChiediDimensione non restituisce alcun valore.
2. 12, 4, 32, 20, 53
3. c
4. d
5. a
6. b
7. d
8. b
9. c
10. b
11. b
12. d
13. a
14. b, e, f
15. Auto
16. Autocarro
17. a3, b5, c1, d2, e4
18. c
19. a
20. V F F F
21. b
22. A A M  
A A M  
A A A
23. b
24. 3 10 10 10 5 6 6 7 7  
3 10 5 6 7

## Capitolo 8

1. b
2. cin, cout, cerr, clog
3. b
4. c
5. testo/binario, testo, binario,  
testo, binario, testo/binario
6. F F F V
7. b
8. V V F F
9. b
10. c

## Capitolo 9

1. a
2. b
3. b
4. F V F V

5. b
6. a, c
7. a4, b1, c2, d3
8. V V F V
9. a3, b5, c4, d6, e1, f2
10. b, c
11. c
12. V F F V
13. a) browser,  
b) link,  
c) Home Page,  
d) URL,  
e) HTTP
14. c
15. d
16. d
17. c
18. a) RSS,  
b) VoIP,  
c) comunità virtuali,  
d) e-learning
19. b, c
20. c
21. a, b

## Capitolo 10

1. b, d
2. a4, b5, c3, d1, e2
3. a3, b2, c1, d5, e4
4. V V F V
5. d
6. c
7. c, e
8. c
9. F V F F
10. a4, b1, c5, d3, e2
11. b
12. V F V V
13. d
14. a
15. c
16. V F V F V
17. a4, b3, c5, d1, e2
18. d
19. b, c
20. d
21. c
22. c
23. a

# Indice analitico

## A

a href, 550  
a:link, 568  
a:visited, 568  
access point, 513  
accessibilità, 76, 586  
accesso, 442  
accesso sequenziale, 458  
account, 534  
Action, 580, 585  
Activate, 502  
Add, 491, 492  
address, 57  
ADSL, 516  
Aggiungi stampante, 113  
aggregatori, 523  
AIFF, 538  
albero, 125, 187  
alert, 598, 611  
alfanumerici, 215  
algebra booleana, 34  
algoritmo, 76, 153, 155, 186, 234, 401, 418  
Align, 556  
Al-Khuwarizmi, 153  
Alt, 557  
alternativa, 170, 234  
ALU, 54  
ambiente, 266  
amministratore del sistema, 72  
analisi, 79  
analisi del problema, 153, 320  
analogico, 18  
ancora, 550, 551  
and, 34, 159, 226  
Android, 68  
animazioni, 539  
Anteprima di stampa, 496  
antivirus, 110  
apertura, 439  
apertura esplicita dei file, 445  
apertura implicita del file, 442  
append, 444  
Application, 477  
Applicazione console, 315  
Applicazione Windows Form, 471, 476  
applicazioni multimediali, 14  
Apri, 494  
architettura, 53

archivi di dati, 434  
Area, 601  
Area di notifica, 94  
argomenti, 126, 196  
aritmetica dei puntatori, 292  
array, 279, 372, 374, 377  
array a due dimensioni, 283  
array di caratteri, 280, 460  
array paralleli, 376  
Arresta il sistema, 89  
ASCII, 31, 440  
aspetti applicativi, 14  
assegnazione, 158, 219, 329, 601  
astrazione, 320  
attributi, 322, 325  
Audio, 559  
author, 542  
automa, 49, 50  
Autoplay, 558, 559  
Autoscroll, 482  
AutoscrollMargin, 482  
AutoscrollMinSize, 482  
Autosize, 482  
AutosizeMode, 482  
AVI, 539  
avviare un programma, 97  
azione, 149, 155

## B

BackColor, 481  
background-color, 564  
backup, 65, 72, 114  
bad, 447  
badbit, 450  
banche dati, 15  
banda, 512  
banda larga, 512, 516  
Barra degli strumenti, 96, 472  
Barra dei menu, 96, 472, 498  
Barra del titolo, 96  
Barra delle applicazioni, 94  
Barre di scorrimento, 96  
Barra di stato, 96  
Barra multifunzione, 96  
basi di dati, 14  
bat, 123, 137  
batch, 123  
begin, 408, 411  
beginPath, 616

Bgcolor, 548  
binary file, 457  
bit, 21, 27, 56  
bit a bit, 226  
bitmap, 537  
blank, 551  
blocco, 63  
Blocco Note, 99  
BMP, 537  
Body, 541  
bool, 218  
bootstrap, 58, 88  
Border, 554  
BorderStyle, 481  
bps, 512  
BR, 547  
brand identity, 588  
break, 249  
breakpoint, 233, 312, 317  
browser, 519, 534  
buffer di I/O, 434, 440  
Build, 311, 312  
built-in, 267  
bus, 53  
Button, 476, 478, 482  
byte, 27, 56, 103

## C

c\_str, 453, 460  
cache, 59  
campo, 286  
canale, 18  
canvas, 613  
capacità, 62  
capacità di una memoria, 58  
carattere, 59  
caratteri jolly, 105, 125  
caratteristica, 30  
Cartella compressa, 109  
cartelle, 63, 105  
case, 249  
casella combinata, 491  
Casella degli strumenti, 473  
casella di controllo, 488  
casella di riepilogo, 480, 491  
case-sensitive, 215  
caso, 183  
casce musicali, 62  
casting, 224  
catch, 407, 447, 450



- ccTLD, 520
  - CD-R, 64
  - CD-ROM, 64
  - CD-RW, 64
  - ceil, 221
  - Cellpadding, 554
  - Cellspacing, 554
  - Cerca, 104
  - cerr, 228, 381, 407, 435
  - certificazione, 80
  - Cestino, 107
  - char, 218
  - Chdir, 127
  - CheckBox, 478, 488, 561
  - Checked, 488, 489, 561
  - CheckedChanged, 489
  - CheckedListBox, 479
  - chiamata di una funzione, 253
  - chiave, 379, 399, 414
  - chiavi USB, 65
  - chiudere una finestra, 97
  - Chiudi, 96
  - chiusura, 439
  - Chkdsk, 136
  - Church, 167
  - cin, 213, 227, 435
  - class, 325, 394, 566
  - class scope, 328
  - class template, 394
  - classe, 322, 482, 566
  - classe base, 323, 342
  - classe derivata, 342
  - clear, 405, 411, 491, 492, 493, 627, 629
  - Click, 477, 502, 503
  - client, 510
  - client wireless, 513
  - client/server, 510
  - clock, 55
  - clog, 228, 435
  - close, 442
  - closePath, 616
  - Cls, 138
  - cmath, 255, 267, 268, 269
  - cmd.exe, 123
  - coda, 385
  - CodeLite, 309
  - codice, 16, 17, 230, 472
  - codice operativo, 58
  - codici a barre, 61
  - codifica, 16, 230
  - collegamento sul desktop, 99
  - color, 564
  - ColorDialog, 495
  - Colore, 495
  - Cols, 562
  - Colspan, 555
  - com, 520
  - ComboBox, 479, 491
  - commenti, 161, 213, 219, 566, 607
  - commercio elettronico, 521
  - CommonDialog, 494
  - commutazione di pacchetto, 512
  - Compila, 317
  - compilatore, 231
  - compilazione, 77, 230
  - complemento a 2, 28
  - comprensibile, 587
  - comprensione, 322
  - compressione, 108, 534
  - computabile, 167
  - computer, 53, 101
  - comunicazione, 18
  - comunità virtuali, 524
  - Condividi con, 116
  - condivisione, 116, 117
  - congiunzione, 34
  - connettivi, 34
  - connettivi logici, 226
  - consistenza, 71
  - const\_cast, 225
  - contenitore, 401, 402, 488, 566
  - controlli, 472
  - Controlli comuni, 478
  - Controls, 558, 559
  - conversione di tipo, 223
  - conversioni esplicite, 224
  - conversioni implicite, 223
  - cookie, 626
  - Copia, 106
  - copy, 129, 424, 463
  - copyright, 525
  - cos, 269
  - costanti, 149, 215, 217
  - costanti enumerative, 444
  - costruttore, 323, 329
  - costruttore di copia, 333
  - costruttore di default, 346
  - count, 418, 491, 492
  - cout, 213, 219, 227, 381, 435
  - cpp, 213, 231, 315, 473
  - CPU, 53, 54
  - crittografia, 524
  - CSS, 563, 565
  - CSS3, 564
  - cstring, 358
  - CU, 54
  - cuffie, 62
- D**
- Data e ora, 92
  - database, 14
  - date, 135, 563, 606, 620, 632
  - DateTime, 477
  - DateTimePicker, 479
  - dati, 13, 16, 79, 149, 150, 440
  - dati di input, 157
  - dati di output, 157
  - de, 520
  - Deactivate, 502
  - debug, 311, 316, 317, 318, 477
  - debugger, 78, 233, 311
  - decodifica, 16
  - decoding, 59
  - default, 249
  - Del, 129
  - delete, 388
  - description, 542
  - desktop, 90
  - diagramma a blocchi, 160
  - diagramma della classe, 322, 347
  - DialogResult, 487, 496
  - dichiarazione, 216
  - dichiarazione della funzione, 262
  - dichiarazione di un oggetto, 326
  - digitale, 18
  - dimensione, 103, 279
  - dimensione di un'istanza, 326
  - Dir, 128
  - directory, 63, 70, 125, 497
  - directory corrente, 125
  - diretto, 442, 457
  - diritti di accesso, 72
  - diritto di autore, 525
  - dischi ottici, 14
  - Disconnetti, 89
  - disgiunzione, 35
  - disgiunzione esclusiva, 35
  - Disinstalla un programma, 94
  - Diskcopy, 135
  - dispositivi di rete, 115
  - distanza, 510
  - distruttore, 332
  - Div, 566
  - DivX, 539
  - DNS, 519
  - do, 240

DOC, 535  
 Doctype, 544  
 document, 612  
 documentazione, 80, 233  
 dominio, 519  
 doppiino telefonico, 513  
 dorsali, 513  
 double, 218  
 DragDrop, 502  
 DragOver, 502  
 drive, 63  
 driver, 66  
 DVD, 65  
 dynamic\_cast, 225

**E**

e-commerce, 521  
 e-government, 522  
 e-learning, 522  
 e-mail, 520  
 eccezione, 406, 407, 447  
 Echo, 138  
 editor, 230, 311  
 edu, 520  
 elaboratore, 53  
 elaborazione, 12, 17  
 elaborazione delle immagini, 79  
 elemento, 266  
 eliminare un file, 107  
 email, 563, 632  
 emittente, 16  
 empty, 405, 411, 415  
 Enabled, 481  
 end, 408, 411  
 Enti di certificazione, 524  
 entità, 148  
 enum, 277  
 enumerazione, 280  
 enunciato, 34  
 eof, 450  
 eofbit, 450  
 EOL, 440  
 eos, 461  
 EPS, 538  
 ePub, 536  
 Erase, 129, 415, 424  
 ereditarietà, 325, 342, 349  
 ereditarietà multipla, 347  
 ereditarietà singola, 347  
 ergonomico, 587  
 error, 231  
 errori, 446, 473  
 errori del personale, 73  
 errori di run-time, 232

errori lessicali, 77, 232  
 errori logici, 77, 233  
 errori sintattici, 77, 231  
 Errorlevel, 139  
 Errors, 311  
 esecutore, 156  
 esecuzione, 155  
 esegui, 176, 178  
 esegui finché, 180  
 Esegui istruzione, 318  
 esegui mentre, 178, 180  
 eseguibile, 77, 231  
 Esplora risorse, 101  
 Esplora soluzioni, 473  
 Espressioni di controllo, 318  
 estensione, 70, 103, 124, 347  
 EULA, 525  
 eval, 604  
 eventi, 471, 501, 502  
 Everyone, 116  
 exception, 407, 450  
 exe, 231, 312, 317, 318  
 executing, 59  
 Exit, 477  
 exp, 269  
 Extranet, 516

**F**

fabs, 267  
 fail, 446  
 failbit, 450  
 fasi, 79  
 fault tolerance, 73  
 feed reader, 523  
 feedback, 577  
 fetching, 59  
 fibra ottica, 513  
 fieldset, 580  
 FIFO, 385  
 file, 63, 439, 534  
 file batch, 137  
 file binari, 440, 441, 457, 458  
 file di log, 88  
 file di testo, 440, 441  
 file system, 70, 101  
 FileName, 497  
 fill, 614  
 fillStyle, 614  
 filmati, 538  
 Filter, 497, 498  
 find, 130, 415, 418  
 Findstr, 131  
 fine del file, 443  
 finestra, 90, 96  
 finestra del codice, 475  
 finestra della cartella, 102  
 finestra di dialogo, 487, 494, 497  
 finestra di progettazione, 475  
 firewall, 524  
 FireWire, 66  
 firma, 349  
 firma digitale, 524  
 first, 414, 464  
 Flash, 539  
 Flat, 483  
 float, 218  
 floor, 221  
 flusso, 434  
 flusso dei dati, 79  
 FLV, 539  
 focus, 493  
 fogli di stile collegati, 565  
 fogli di stile in linea, 564  
 fogli di stile incorporati, 564  
 foglie, 187  
 FolderBrowserDialog, 495  
 Font, 481, 496  
 FontDialog, 496  
 font-family, 564  
 font-size, 564, 568  
 for, 245, 580, 605  
 for\_each, 419, 464  
 ForeColor, 481  
 form, 472, 482, 560, 604  
 Format, 135  
 formati, 535  
 formattazione, 70  
 formazione, 80  
 FormBorderStyle, 482  
 fr, 520  
 frame, 544  
 freeware, 525  
 frequenza, 512  
 friend, 338, 360, 438, 439  
 friend function, 338  
 fstream, 435, 442, 443, 453, 457  
 FTP, 518, 534  
 FullHD, 537  
 function, 597, 598  
 function overloading, 272  
 funzione, 39, 79, 196, 221, 251, 252, 321  
 funzione computabile, 167  
 funzione degli stati successivi, 50  
 funzione di transizione, 50

funzione delle uscite, 50  
funzione di transizione, 165  
funzioni built-in, 251  
funzioni membro, 402  
funzioni predefinite, 267  
funzioni virtuali, 361  
furti, 74

## G

g++, 310  
GB, 103  
Gbps, 512  
gerarchia delle funzioni, 187  
gerarchia di classi, 324, 346  
gestionale, 78  
Gestione attività, 89  
gestione degli eventi, 501  
gestione della memoria, 69  
gestione delle periferiche, 70  
gestione dinamica, 388  
get, 436, 580  
getDate, 606  
getElementById, 601  
getHours, 606  
getItem, 627, 629  
getline, 436, 460  
getMinutes, 606  
getMonth, 606  
getto di inchiostro, 61  
getYear, 606  
Ghz, 55  
GIF, 538, 556  
Gigabyte, 57  
GoBack, 506  
GoForward, 506  
GoHome, 506  
good, 447  
goodbit, 450  
GotFocus, 502  
Goto, 138  
gov, 520  
GPL, 309, 525  
grado di parallelismo, 59  
grafica, 534  
grafo, 50  
GroupBox, 488  
GUI, 75, 88  
Guida e supporto tecnico, 111  
Guida in linea, 96, 314

## H

hardware, 13, 53, 66  
Hardware e suoni, 92

HD, 537  
Head, 541  
height, 557, 567  
help, 80, 111, 126  
hidden, 561  
Hn, 546  
home banking, 522  
home page, 519, 552  
HorizontalAlignment, 500  
host, 511, 519  
Hostname, 136  
HR, 548  
HTML, 519, 539, 542  
HTML4, 540  
HTML5, 540  
HTTP, 518  
HTTPS, 518, 524

## I

ICANN, 520  
icone, 60, 90  
ICT, 12  
ID, 566, 581, 601  
IDE, 230, 309, 313  
identificativo dell'utente, 72  
identificatore, 149, 160, 212, 566  
identificazione, 88  
IEEE 802.22, 513  
if, 236  
If Exist, 139  
If Not Exist, 138  
ifstream, 443, 457  
ignore, 436, 460  
IM, 523  
image, 612  
IMG, 556  
implementazione, 148, 262, 349  
Imposta pagina, 496  
incapsulamento, 323, 325  
include, 213, 267, 270  
Incolla, 106  
index.html, 552, 589  
indirizzo, 57, 257, 259, 291  
indirizzo di rientro, 382  
indirizzo IP, 517  
informatica, 12  
informatica mobile, 15  
information hiding, 271, 323  
informazione, 16  
informazioni di base, 93  
ingegneria del software, 172  
Ingrandisci, 96  
inizializzare un array, 279  
inizializzazione, 217  
inkjet, 61  
innerHTML, 601  
input, 17, 161, 560, 580, 632  
insert, 415  
installare una stampante, 113  
int, 218, 252  
int&, 259  
integrità, 71  
interattività, 80  
interfacce, 14, 65, 80, 271  
interfaccia, 18, 338  
interfaccia grafica, 75, 471  
interfaccia utente, 67, 75, 88  
Internet, 15, 88, 518  
interpretazione, 77  
interprete dei comandi, 75  
interrogazioni, 441  
intervista, 79  
Intranet, 516  
Invia a, 106  
ios, 435, 444  
iOS, 68  
ios::app, 444, 453  
ios::binary, 457  
ios::failure, 450  
ios::in, 444, 453  
ios::out, 444, 453  
iostream, 227, 267, 434, 435  
Ipconfig, 136  
ipertesto, 539, 550  
ipertestuale, 518  
IPv6, 517  
is\_open, 447  
ISDN, 515  
ISP, 515  
istanza, 322, 326  
istream\_iterator, 460, 461  
istruzioni, 155, 321  
istruzioni dichiarative, 215  
istruzioni esecutive, 216  
it, 520  
Items, 491, 492  
iteratore, 401, 411, 408, 460  
iterazione, 171

## J

JavaScript, 597  
joystick, 60  
JPEG, 538, 556  
Jump List, 105

## K

KB, 103  
Kbps, 512  
kernel, 69  
KeyDown, 502  
KeyPress, 502  
KeyUp, 502  
keywords, 542  
Kilobyte, 57

## L

Label, 476, 479, 483, 580  
LAN, 511  
larghezza di banda, 512  
laser, 61  
lato client, 511, 597  
lato server, 511, 597  
layout, 563  
LCD, 61  
Lee, 401  
left, 567  
leggi, 38  
leggi di De Morgan, 38  
lessico, 212  
lettori di CD-ROM, 64  
lettura, 439, 441, 443, 457  
lettura e scrittura, 107  
LI, 552  
libreria dei template, 401  
libreria delle classi, 482  
librerie di funzioni, 267  
librerie di I/O, 213, 434  
licenza d'uso, 525  
LIFO, 382  
linea di comando, 75, 126  
lineWidth, 614  
Lingua della tastiera, 92  
linguaggio, 18, 76, 155  
linguaggi ad oggetti, 78  
linguaggi di programmazione, 14, 18  
linguaggio di comandi, 18  
linguaggio di pseudocodifica, 159  
linguaggio HTML, 533  
linguaggio macchina, 76, 440  
link, 518, 519, 539, 550, 565, 569  
linker, 77, 230, 232  
LinkLabel, 479  
list, 410  
lista, 387, 552  
ListBox, 479, 491

ListView, 479  
Load, 477, 502, 503  
localhost, 517  
localStorage, 627, 628  
Location, 482  
log, 269  
log10, 269  
login, 88, 585  
long int, 218  
LOOP, 558, 559  
lossless, 534  
lossy, 534  
LostFocus, 502  
lpt1, 229, 451

## M

Mac OS, 68  
Macchina di Turing, 164  
Macchina di Von Neumann, 53  
macchina fotografica, 61  
mailto, 551  
main, 213, 253, 321, 473  
manipolazione, 441  
mantissa, 30  
manutenzione, 187, 251, 322  
map, 464  
mappe immagine, 599  
MaskedTextBox, 479  
massimo, 248  
masterizzatori, 64  
Math, 505, 602  
Math::Abs, 505  
Math::Pow, 505  
Math.Pow, 604  
Math.round, 605  
Math::Sqrt, 505  
Math::Truncate, 505  
matrice, 283  
matrice di transizione, 51, 165  
max\_element, 423  
Maximized, 505  
MaximumSize, 482  
Maxlength, 560  
MB, 103  
Mbps, 512  
Megabyte, 57  
member function, 326  
membri, 323  
memoria cache, 59  
memoria centrale, 56  
memoria volatile, 58  
memorie di massa, 62, 101  
memorie SD, 65  
menu, 75  
menu a tendina, 498  
MenuStrip, 499  
merge, 411, 423  
MessageBox, 487  
MessageBoxButtons, 487  
messaggio, 338, 512  
Meta, 542  
meta tag, 542  
metadati, 542  
metodi, 322, 325  
Mhz, 55  
microfono, 61  
microprocessore, 55  
MIDI, 538  
mil, 520  
min\_element, 423  
miniaturizzazione, 13  
MinimumSize, 482  
MIPS, 55  
Mkdir, 127  
modelli di classi, 394  
modelli iconici, 20  
modelli matematici, 20  
modello, 19, 148  
modem, 62, 515  
Modifica visualizzazione, 103  
moduli, 187, 321  
moduli funzionalmente indipendenti, 251  
MonthCalendar, 479  
More, 130  
motore di ricerca, 521  
mouse, 60, 93  
MouseDown, 502  
MouseLeave, 502, 503  
MouseMove, 502, 503  
MousePosition, 504  
MouseUp, 502  
MouseWheel, 502  
MOV, 539  
Move, 129  
mp3, 535, 538  
MPEG, 538  
Multiline, 483  
Multiple, 562  
multiprogrammati, 69  
multitasking, 95  
muovere una finestra, 97

## N

Name, 476, 481, 542, 560, 581  
namespace, 213, 269

nascosto, 107  
 nastri, 63  
 Navigate, 506  
 navigazione, 519, 539  
 net, 520  
 networking, 15  
 new, 388, 389  
 NIC, 514  
 nodo, 187  
 nome, 70, 124, 216  
 nome dell'utente, 88, 524  
 normalizzata, 30  
 Not, 159, 226  
 notazione scientifica, 30  
 Now, 477  
 nucleo, 69  
 Nul, 133, 281  
 number, 563, 632  
 numerici, 215  
 NumericUpDown, 479  
 Nuova cartella, 105

## O

Object, 558, 559  
 ODT, 535  
 office, 78, 541  
 office automation, 14  
 ofstream, 442, 457  
 oggetto, 77, 322, 327, 474  
 Ok, 598  
 OL, 553  
 onClick, 598, 604  
 onLoad, 606, 621, 628, 630  
 onMouseOut, 612  
 onMouseOver, 612  
 onSubmit, 631  
 OOP, 321  
 open, 445, 451  
 open source, 309, 525  
 OpenFileDialog, 494  
 operandi, 58  
 operator overloading, 357  
 operator, 357  
 operatore &, 289  
 operatore \*, 289  
 operatore ?, 238  
 operatore =, 358  
 operatore [ ], 406  
 operatore <<, 227, 436, 442  
 operatore >>, 227, 436  
 operatore ==, 360  
 operatore --, 220  
 operatore ++, 220

operatore di assegnazione, 329, 359  
 operatore di risoluzione, 328  
 operatore freccia, 293, 414  
 operatore virgola, 247  
 operatori, 215, 601  
 operatori aritmetici, 158  
 operatori booleani, 237  
 operatori di confronto, 225  
 operatori di relazione, 159  
 operatori logici, 159, 226  
 operazioni di chiusura, 88  
 operazioni di I/O, 440  
 opzioni, 126  
 or, 35, 159, 226  
 Ordinamento, 374  
 ordinare i file, 104  
 org, 520  
 organizzazione, 13  
 origine, 613  
 Orologio e opzioni internazionali, 92  
 ostream\_iterator, 460  
 out\_of\_range, 407  
 output, 17, 161, 473  
 overloading, 349, 350  
 overloading degli operatori, 402  
 overloading dell'operatore <<, 437  
 overriding, 349, 351

## P

p2p, 511  
 pacchetti, 512  
 PageSetupDialog, 496  
 pagina, 69  
 Pagina Web, 541  
 pair, 414  
 Pannello di controllo, 90  
 parallela, 65  
 parallelo, 59  
 parametri, 192, 252  
 parametri attuali, 193, 257  
 parametri di chiamata, 140  
 parametri formali, 193, 257  
 parola, 56  
 parola chiave, 126, 160  
 parole, 27  
 passaggio di array, 377  
 passaggio di parametri, 192, 257, 281  
 passaggio di parametri per indirizzo, 195, 291  
 passaggio di parametri per referenza, 257  
 passaggio di parametri per valore, 195, 257  
 passo, 181  
 password, 72, 88, 524, 560  
 path, 616  
 pathname, 71, 125  
 pathname assoluto, 125  
 pathname relativo, 125  
 Pause, 138, 140  
 PDF, 535  
 peer to peer, 511  
 per, 181  
 percepibile, 586  
 periferica, 114, 434  
 periferiche, 65  
 periferiche virtuali, 70  
 personal computer, 13  
 phishing, 524  
 PictureBox, 479  
 pila, 382  
 pin, 56  
 Ping, 136  
 pipeline, 134  
 pirata informatico, 73  
 pixel, 536  
 Placeholder, 581  
 plotter, 62  
 PNG, 538, 556  
 podcast, 523  
 Point, 504  
 polimorfismo, 325, 349  
 poly, 601  
 Pop, 382, 385  
 pop\_back, 405, 411  
 pop\_front, 411  
 porte di I/O, 65  
 position, 567  
 posta elettronica, 15  
 Poster, 558  
 pow, 221, 268, 269, 272  
 PRE, 548  
 precisione, 31  
 precisione doppia, 31  
 precisione semplice, 31  
 precisione singola, 31  
 Preload, 558, 559  
 primitive, 69  
 PrintDialog, 496  
 PrintPreviewDialog, 496  
 private, 335  
 procedura chiamante, 192  
 procedura chiamata, 192

processo, 20, 155  
processore, 20, 156  
produzione, 80  
profilo, 72  
progettazione, 79  
Progetto console, 314  
progetto, 314, 476  
programma, 66, 76, 156, 321  
programma oggetto, 231  
programma sorgente, 231  
programmatore, 156  
programmazione, 320  
programmazione generica, 401  
programmazione orientata agli oggetti, 321  
programmazione procedurale, 321  
programmazione strutturata, 168, 321  
programmazione visuale, 471  
programmi eseguibili, 440  
ProgressBar, 479  
Project, 310  
prompt, 123, 126  
Prompt dei comandi, 123  
proprietà, 148, 476, 480, 481  
protected, 350, 361  
protezione, 110  
protocollo, 17, 53, 517  
prototipo, 213, 262  
Provider, 519  
pseudocodifica, 159  
pseudoselettori, 568  
PSTN, 515  
pulsanti di comando, 482  
pulsanti di opzione, 489  
puntatore, 281, 289, 387, 402, 478  
Push, 382, 385  
push\_back, 404, 410  
push\_front, 410

## R

radice, 187  
radio, 561, 611  
RadioButton, 480, 489  
raffinamenti successivi, 186  
RAM, 58  
random, 442, 457  
range, 632  
rappresentazione esponenziale, 30  
raster, 537

rbegin, 412, 423  
read, 457  
realizzazione, 79  
record, 286  
record a lunghezza costante, 441  
record a lunghezza variabile, 441  
Ref, 195  
referenza, 195, 258, 259  
Refresh, 506  
Registrars, 520  
registri, 55  
regole di visibilità, 266  
reinterpret\_cast, 225  
release, 67, 313, 318  
REM, 141  
Remove, 491, 492  
RemoveAt, 491  
removetern, 627  
Rename, 129  
rend, 412, 423  
Replace, 130  
report, 441  
REQUIRED, 563, 580  
reset, 561  
restore, 72  
rete, 13, 15, 115, 510  
reti geografiche, 511  
reti locali, 511  
return, 252, 253  
reverse, 411, 423  
reverse\_iterator, 412, 423  
RGB, 548  
rgba, 614  
Riavvia il sistema, 89  
ricerca sequenziale, 372  
ricevente, 16  
RichTextBox, 480  
riconoscimento vocale, 76  
ricorsione, 197, 274  
ridefinizione, 347  
ridimensionare una finestra, 97  
ridirezione, 131  
ridirezione dell'output, 229, 381  
Riduci a icona, 96  
ridurre a icona, 95  
rilascio, 80  
rimuovere la periferica, 114  
rimuovere un programma, 94  
Rinomina, 106  
ripeti, 178, 180, 181  
ripeti finché, 176, 178  
ripeti mentre, 178

ripetizione, 170, 176, 234, 240, 605  
ripetizione con contatore, 181, 245  
ripetizione postcondizionale, 178  
ripetizione precondizionale, 178, 242  
Ripristina, 96  
ripristino, 114  
riscrittura, 441  
risolutore di visibilità, 213, 328  
risoluzione video, 91, 536  
risorse, 510  
riusabilità, 322  
Rmdir, 127  
robustezza, 322  
Robusto, 587  
ROM, 58  
root, 70, 125  
router, 514  
routine, 66  
Rows, 562  
Rowspan, 555, 607  
RSS, 523  
RTF, 535  
runtime errors, 77

## S

Salva con nome, 494  
SaveFileDialog, 494  
scambio di messaggi, 338  
scanner, 61  
scansione, 110  
scelta multipla, 183, 249  
scheda di rete, 514  
schede, 56  
scope resolution, 213, 269  
scope rules, 266  
screen reader, 76  
screen saver, 90, 92  
script, 137, 597  
scrittura, 439, 441, 442, 457  
se, 173  
second, 414, 464  
segmento, 69  
segnali, 18  
Select, 562  
Selected, 562  
SelectedIndex, 491, 492  
SelectedItem, 491  
selettore, 183, 249, 566  
selezionare più file, 106



- selezione, 171, 173, 234, 236, 605
  - Separator, 499
  - sequenza, 170, 234, 235
  - sequenza di escape, 228, 543
  - sequenziale, 442, 457
  - seriale, 65
  - server, 510
  - server dedicato, 511
  - servizi finanziari, 522
  - sessionStorage, 627
  - Set, 140
  - setItem, 627, 629
  - setTimeout, 619
  - settori, 63
  - setup.exe, 94
  - setw, 246
  - Sfogliare, 495
  - sfondo del desktop, 91
  - shadowing, 267
  - Shape, 601
  - shareware, 525
  - shell, 75
  - short int, 218
  - Show, 487
  - ShowDialog, 496
  - sicurezza, 71
  - simboli di input, 49, 165
  - simboli di output, 49
  - simbolo di decisione, 161
  - sin, 269
  - sintassi, 212
  - sistema, 19
  - sistema binario, 21
  - sistema di elaborazione, 19, 53
  - sistema di numerazione, 21
  - sistema esadecimale, 25
  - sistema operativo, 14, 67
  - sistema ottale, 24
  - sito Internet, 589
  - sito Web, 518, 532
  - size, 405, 410, 415, 481, 482, 560
  - sizeof, 219, 327, 457
  - sln, 314
  - SMTP, 518
  - social network, 523
  - software, 13, 53, 66
  - software applicativo, 66, 79
  - software di base, 66
  - software house, 78
  - software libero, 525
  - sola lettura, 107
  - soluzione, 314
  - sorgente, 76
  - sort, 130, 374, 411, 422, 463
  - Sorted, 491
  - Sospendi, 89
  - sottoclasse, 325, 346
  - sottodirectory, 71
  - sottomenu, 99
  - Source, 559
  - Span, 580
  - spazio di indirizzamento, 57
  - SPOOL, 70, 113
  - spostare un file, 106
  - sqrt, 221, 255, 267, 269
  - Src, 556, 558, 559, 612
  - SSH, 518
  - stack delle chiamate, 382
  - stack overflow, 382
  - Stamp, 92
  - stampa, 441, 496
  - stampa schermo, 92
  - stampante, 61, 113, 229, 451
  - stampante di rete, 118
  - stampante predefinita, 113
  - standard error, 126, 131
  - standard input, 126, 131, 214, 227
  - standard output, 126, 131, 214, 227
  - Start, 97
  - StartPosition, 482
  - statement, 215
  - stati, 49, 165
  - stati finali, 165
  - static\_cast, 224
  - stato, 164
  - stato del file, 108, 450
  - stato iniziale, 165
  - std, 213, 271
  - Stepanov, 401
  - stilo, 61
  - STL, 401
  - strcmp, 358
  - strcpy, 358
  - stream, 227, 434
  - stream iterator, 460
  - stream predefiniti, 435
  - streaming, 539
  - string, 218, 281, 402
  - string::c\_str, 403
  - string::replace, 403
  - string::size, 403
  - string::substr, 403
  - stringhe, 150, 215, 217
  - stroke, 613, 615
  - strokeStyle, 614
  - strokeText, 623
  - struct, 286
  - struttura, 171, 286
  - struttura annidata, 239, 606
  - struttura condizionale, 171
  - struttura derivata, 249
  - struttura di dati, 279
  - strutture di controllo, 171, 234, 321, 605
  - strutture di dati, 388
  - strutture dinamiche, 387
  - Style, 564
  - submit, 561
  - suoni, 538
  - superclasse, 323, 346
  - supporti, 13
  - sviluppo del software, 79
  - sviluppo top-down, 187, 251
  - SWF, 539
  - switch, 249, 514605, 608
  - Systeminfo, 136
- T**
- tabella, 378, 553
  - tabella degli stati successivi, 51
  - tabella delle uscite, 51
  - TabIndex, 481
  - Table, 553, 554
  - tag, 540
  - tag stilistici, 546
  - Taglia, 106
  - tan, 269
  - Target, 551
  - task, 95
  - tastiera, 60, 92
  - tasto destro, 93
  - tasto sinistro, 93
  - tavola di verità, 36
  - tavola grafica, 61
  - TB, 103
  - TCP/IP, 517, 518
  - TD, 553
  - tecnica, 17
  - telecamera, 61
  - template, 393, 394
  - template della classe, 394
  - template delle funzioni, 393, 409
  - tempo di accesso, 62
  - Teorema di Böhm-Jacopini, 172
  - Terabyte, 57
  - Termina attività, 89

terminale, 60  
 terminatore di stringa, 281  
 Tesi di Church-Turing, 167  
 testi, 535  
 testing, 80  
 Text, 476, 477, 481, 483, 488, 489, 560  
 TextAlign, 481, 568  
 TextArea, 561  
 TextBox, 480, 483  
 TextChanged, 501  
 TextLength, 487  
 TH, 553  
 this, 359, 604  
 throw, 447  
 TIFF, 537  
 Time, 135, 563  
 Title, 497, 542  
 TLD, 520  
 ToolTip, 480  
 ToolStripMenuItem, 499  
 top, 567  
 ToString, 477  
 touchpad, 60  
 touchscreen, 62  
 TR, 553  
 tracce, 63  
 trackball, 60  
 transizione, 79  
 Trasmissione, 17  
 Tree, 129  
 TreeView, 480  
 trovare un file, 104  
 try, 407, 447  
 Turing, 164  
 TXT, 535  
 TYPE, 129, 559, 560, 632  
 typename, 394, 409

## U

UL, 552  
 UML, 325  
 Unicode, 32, 218, 543, 544  
 uniprogrammati, 69  
 unique, 424  
 unità aritmetico-logica, 54  
 unità di controllo, 54

unità di I/O, 60, 62  
 unità di rete, 65  
 unità periferiche, 101  
 unsigned, 218  
 unzip, 108  
 upload, 533  
 URL, 518, 539, 552, 562, 632  
 usabilità, 587  
 USB, 65, 114  
 using namespace, 213, 269  
 utente, 156  
 UTF, 32, 543  
 Utilizzabile, 587  
 UTP, 513

## V

Val, 195  
 validatori, 563, 573, 587  
 validazione, 233, 562, 577, 631  
 valore, 414  
 valori, 196  
 valori di verità, 34  
 Value, 560, 608  
 variabile di ambiente, 139  
 variabili, 148, 212, 215, 216  
 variabili di lavoro, 157  
 variabili globali, 255  
 variabili locali, 255, 257  
 vcxproj, 314  
 vector, 404, 462  
 velocità, 58  
 velocità di trasferimento, 62  
 Ver, 136  
 versioni, 67  
 vettore, 279  
 vettore costante, 287  
 VGA, 536  
 video, 61, 558  
 video sensibile al tatto, 62  
 videogiochi, 79  
 virtual, 363  
 virus, 73, 524  
 Visible, 481  
 Visual C++, 313, 471  
 visualizzazione, 441  
 VLSI, 55  
 void, 252  
 VoIP, 522  
 VOL, 135  
 Volume audio, 92

## W

W3C, 540, 564, 586  
 WAI, 586  
 WAN, 511  
 warning, 231  
 WCAG, 586  
 wchar, 218  
 wchar\_t, 218  
 Web, 518  
 Web 13.0, 523  
 Web editor, 533  
 Web mail, 521  
 Web server, 518  
 Web usability, 587  
 WebBrowser, 480, 506  
 Webcam, 61  
 Webm, 558  
 Webstorage, 627  
 what, 450  
 while, 240, 242, 605  
 width, 554, 557, 567  
 Windows, 67, 88  
 Windows Phone, 68  
 WindowState, 482, 505  
 WinZip, 109  
 wireless, 513  
 WMA, 538  
 WMF, 538  
 WMV, 539  
 word, 56  
 Workspace, 310  
 write, 457  
 WWW, 518

## X

Xor, 35, 159  
 Xvid, 539

## Z

z-index, 571  
 zip, 108, 535  
 zoom, 76



La proposta editoriale mista *C++ Teoria e Ambiente di programmazione* comprende:

■ **MATERIALI A STAMPA**

- Testo base. Volume unico

■ **E-BOOK**

- Disponibile anche la versione digitale

■ **MATERIALI ON LINE**

- Approfondimenti e integrazioni dei contenuti trattati nel testo
- Materiali digitali e multimediali per la realizzazione di pagine Web e fogli di stile
- Codice eseguibile dei programmi con videate delle fasi di esecuzione
- Lezioni multimediali sull'uso operativo degli ambienti e delle tecnologie software

■ **PER IL DOCENTE**

- Materiali didattici per la programmazione e le verifiche
- CD-ROM con materiali multimediali e interattivi:
  - Presentazioni in *PowerPoint* e in *pdf* da utilizzare con la LIM
  - Codici sorgente completi dei programmi presentati nel volume
  - File completi dei progetti di pagine Web e fogli di stile
  - Ulteriore repertorio di esercizi di autoverifica per gli studenti

**C++ - Teoria e Ambiente di programmazione**