Università di Torino

QUADERNI DIDATTICI
del
Dipartimento di Matematica

M.GARETTO

Laboratorio di programmazione

a.a. 2000/2001

Quaderno # 4 – Maggio 2001

Prefazione

In questo quaderno sono state raccolte le lezioni del corso di Laboratorio di Programmazione tenuto presso l'Università di Torino nell'a.a. 2000/2001 per il Corso di Laurea in Scienza dei Materiali.

I primi due capitoli sono dedicati alla descrizione del personal computer e dei suoi componenti fondamentali: l'hardware e il software. Nel primo capitolo, riservato all'hardware, sono illustrate le parti interne del computer, dal microprocessore alla memoria, dal disco rigido alla scheda video, e le periferiche, cioè i dispositivi che permettono alla macchina di comunicare con l'esterno, dalla tastiera al mouse, dal monitor alla stampante. Nel secondo capitolo, dedicato al software, vengono descritte le fasi principali dello sviluppo di un progetto software e vengono introdotti brevemente i principali linguaggi di programmazione.

Nel terzo capitolo, interamente dedicato allo studio degli algoritmi, viene introdotta la tecnica di pseudocodifica, utilizzata per la scrittura degli algoritmi, e vengono dettagliatamente descritti numerosi algoritmi fondamentali, ampiamente utilizzati per la costruzione di procedure più complesse.

Il quarto capitolo è dedicato allo studio della rappresentazione dei numeri all'interno di un calcolatore, con particolare attenzione alla rappresentazione in floating-point e alla relativa aritmetica di macchina.

Le lezioni del corso sono accompagnate da una consistente attività di laboratorio in aula informatica, dedicata all'utilizzo del software scientifico MATLAB; nell'appendice di questo quaderno sono raccolti gli algoritmi fondamentali, descritti nel terzo capitolo, implementati in ambiente MATLAB.

Indice

1. Il Personal Computer

1.1 Introduzione

Il personal computer è ormai diventato un oggetto comune della vita quotidiana e saperlo usare non è solo un prerequisito per molte attività professionali, ma anche un buon modo per studiare, per informarsi, per comunicare e per gestire molte attività legate al tempo libero.
Nato come potente strumento di calcolo per i centri di studio e di ricerca, oggi il PC è una macchina molto versatile, con cui è possibile eseguire le operazioni più diverse: scrivere un testo o aggiornare un bilancio, disegnare, ritoccare fotografie, montare un video, sentire musica, guardare un film, giocare. Con il computer ci si può collegare a Internet per scopi diversi, fare acquisti on-line, scrivere a persone lontane, colloquiare in un ambiente virtuale con altri utenti; con il PC si può studiare, consultare un'enciclopedia, collegarsi con le più importanti biblioteche e istituzioni universitarie, imparare le lingue. I programmi che permettono di utilizzare il computer per tutti questi scopi sono diventati negli anni più semplici e amichevoli (in gergo informatico "friendly").
L'obiettivo di queste lezioni è fornire un'alfabetizzazione informatica di base che permetta di affrontare agevolmente l'utilizzo del personal computer e dei principali programmi applicativi.
Nella prima parte, riservata all'hardware, sono descritte le parti interne di un computer, microprocessore, memoria, disco rigido, scheda video, e le periferiche, cioè i dispositivi che permettono alla macchina di comunicare con l'esterno, tastiera, mouse, monitor, stampante, con l'obiettivo di fornire le informazioni necessarie per comprendere il funzionamento del computer e riconoscere i fattori che influiscono sulle sue prestazioni.
La seconda parte è dedicata a un'introduzione al sistema operativo, il programma che fa da tramite tra l'utente, i programmi in dotazione e la macchina; vengono presentate le sue caratteristiche principali, le icone, l'ambiente a finestre, e le attività essenziali ricorrenti nell'uso del computer, per esempio la creazione e la gestione di file e cartelle, i metodi per organizzare le risorse del PC.
Lo scopo che si vuole raggiungere è acquisire gli strumenti di base indispensabili per poter utilizzare i programmi applicativi più diffusi, che possono essere suddivisi in alcuni tipi fondamentali.

- **Videoscrittura**

L'elaborazione elettronica dei testi (**word processing**) è un campo in cui il Pc ha ampio utilizzo: il PC viene usato per creare, modificare e impaginare documenti e le funzionalità offerte dai programmi di videoscrittura vanno ben oltre la macchina da scrivere, permettendo di creare documenti di aspetto professionale.
I programmi di videoscrittura, partendo dalle funzionalità di base, come inserire il testo e modificarlo, consentono di scegliere carattere e stile, formato di impaginazione, bordi, sfondi, inserire tabelle, immagini e grafici, anche ottenuti con altri programmi; inoltre consentono la verifica automatica dell'ortografia e della grammatica, la creazione di indici e sommari, la realizzazione di semplici pagine Web.
I diversi prodotti per videoscrittura sono classificabili in due fondamentali categorie:
1 - quelli che prevedono l'inserimento nel testo di comandi veri e propri, visibili, che in una fase successiva sono trattati da un opportuno programma di formattazione che produce in output un file pronto per la stampa. Un prodotto di questo tipo è **TEX**.
2 - quelli che interpretano interattivamente i comandi inframmezzati al testo (non visibili) e forniscono su schermo una visualizzazione fedele di quanto sarà stampato. Questi prodotti vengono detti di tipo **WYSWYG** (**W**hat **Y**ou **S**ee Is **W**hat **Y**ou **G**et).
Un word processor di questo tipo è **WORD** per Windows.
L'area informatica dell'editoria realizzata su microcalcolatori è denotata col nome di **DTP (Desk Top Publishing)**.

- **Fogli di calcolo**

I fogli di calcolo (fogli elettronici) sono strumenti potenti e versatili, che permettono di organizzare, gestire ed elaborare in modo automatico informazioni numeriche e di testo.
Queste applicazioni sono fra le più diffuse; le funzionalità disponibili in un foglio di calcolo sono utili per creare tabelle contenenti dati e testi, effettuare calcoli di tipo diverso, gestire attività finanziarie, realizzare grafici, effettuare calcoli statistici.
Noti fogli elettronici sono **LOTUS 1-2-3**, **EXCEL** per Windows.

- **Presentazioni e grafica**

Con gli applicativi oggi disponibili è possibile realizzare documenti multimediali, in grado di integrare immagini, testi, video e audio, che possono essere utilizzati a scopo didattico, informativo, pubblicitario oppure come pagine Web. Un applicativo di questo tipo è **POWER POINT** per Windows.
Inoltre è possibile disegnare e manipolare immagini grafiche, memorizzarle, modificarle e utilizzarle in altri documenti.

- **Database**

Un database, o base di dati, è un archivio elettronico che permette di ordinare ed elaborare grandi quantità di dati. E' possibile comporre e strutturare un archivio, aggiungere nuovi dati o modificare quelli esistenti, ordinare le informazioni, collegandole fra loro in vari modi, impostare ricerche automatiche e stampare i risultati delle ricerche.
Noti data base sono **DB4, ORACLE, ACCESS** per Windows.

- **Reti**

Grande importanza hanno oggi gli strumenti per la comunicazione e l'informazione in rete. La navigazione in Internet è resa possibile dall'utilizzo di programmi detti **browser**; i più noti fra essi sono **Internet Explorer** e **Netscape**. Così anche per la comunicazione con la posta elettronica si dispone di appositi programmi, tra cui i più comuni sono **Outlook Express** e **Eudora**.

- **Software matematico**

Negli ultimi anni sono stati sviluppati numerosi packages di tipo matematico adatti per il calcolo algebrico, simbolico, numerico, statistico.
I più famosi sono: **DERIVE** (il primo uscito sul mercato e ormai superato), **MATHEMATICA**, **MAPLE**, **MATLAB.**
MAPLE è un potente sistema interattivo per effettuare calcoli simbolici e numerici, come fattorizzazione di polinomi, semplificazione di espressioni matematiche, soluzioni di sistemi di equazioni, calcolo differenziale e integrale, sviluppi in serie di potenze e di funzioni, soluzione analitica di equazioni differenziali, grafici in due e tre dimensioni, e numerose altre applicazioni.
E' importante sottolineare che i calcoli sono effettuati in modo esatto, seguendo le regole dell'algebra e non usando l'aritmetica approssimata del computer.
Dispone anche di un linguaggio di programmazione.
MATHEMATICA è di tipo simile, meno potente in alcuni tipi di problemi.
MATLAB è un sistema interattivo per risolvere molti problemi di calcolo numerico e di visualizzazione di grafici e di risultati in modo integrato, ad esempio problemi di algebra lineare, approssimazione di soluzioni di equazioni e di equazioni differenziali, problemi di interpolazione e approssimazione, problemi di integrazione numerica, di statistica, matematica finanziaria e molto altro. Anche MATLAB dispone di un liguaggio di programmazione molto potente.
Questi sistemi sono detti **general purpose systems**. Esistono anche numerosi **special purpose systems**, progettati per la soluzione di problemi in specifici settori della fisica, della matematica, della statistica.

1.2 Le informazioni nel mondo dei computer: bit e byte

Il computer è una macchina diversa da tutte le altre inventate dall'uomo: invece di lavorare su elementi fisici opera su un'entità astratta, l'**informazione**.
Più precisamente il computer è una macchina programmabile che opera mediante la memorizzazione, l'elaborazione e la trasmissione di informazioni sotto forma di impulsi elettrici: i bit.
Il computer codifica tutte le informazioni utilizzando una convenzione binaria, cioè due simboli che si riferiscono ai due stati elettrici fondamentali: l'assenza e la presenza di corrente, ossia 0 e 1.
L'unità elementare di informazione può quindi avere solo due valori (0 e 1) e viene chiamata **bit** (**BInary digiT**).
Per rappresentare lettere e numeri è necessario utilizzare gruppi di bit. Un raggruppamento di 8 bit viene chiamato **byte** ed è in grado di rappresentare 256 simboli diversi: infatti contando tutte le disposizioni possibili di 0 e 1 a gruppi di 8, per esempio:

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
.....
0 0 0 0 1 1 1 1
.....
1 1 1 1 1 1 1 1

si ottengono $2^8 = 256$ configurazioni diverse, il che permette di rappresentare i caratteri alfabetici, i numeri, i segni di punteggiatura, i caratteri accentati e speciali e i caratteri di controllo. In concreto un singolo byte può rappresentare una lettera dell'alfabeto, un numero, un segno di punteggiatura o un carattere speciale, ad esempio @.
Dato che gran parte delle informazioni elaborate da un PC sono numeri e lettere, il byte è stato usato come unità di misura della quantità di dati memorizzati sul computer e delle capacità di immagazzinamento dei dispositivi di memorizzazione.
Come avviene per le unità di misura, anche per i byte si sono definiti dei multipli:

1 **kilobyte** (KB) = 1024 byte = 2^{10} byte
1 **megabyte** (MB) = 1024 KB (circa un milione di byte)
1 **gigabyte** (GB) = 1024 MG (circa un miliardo di byte)
1 **terabyte** (TB) = 1024 GB (circa mille miliardi di byte)

Tutte queste unità di misura servono a misurare la capacità di archiviare delle informazioni; altre unità di misura servono invece a misurare quanto un computer sia veloce e quanti dati possa trasferire in un dato tempo; vi sono inoltre delle unità di misura rivolte a misurare le capacità grafiche dei monitor.

1.3 Hardware e software

La grande flessibilità del computer deriva dal fatto che coesistono due componenti, una materiale, chiamata hardware, e una logica, chiamata software.
L'**hardware** è costituito dall'insieme di parti fisiche da cui è composta la macchina.
Le istruzioni, i programmi eseguibili e i dati rappresentano il **software**, la componente non tangibile del computer.
Il **file** è la struttura logica principale con cui il PC archivia le informazioni. Un **programma** è un file che contiene le istruzioni necessarie al computer per svolgere determinate operazioni.
Possedere solo il computer in senso fisico, senza possedere il software, non serve assolutamente a nulla, in quanto inutilizzabile. Soltanto integrando le componenti hardware con le componenti software si ottiene un sistema funzionante e utile per risolvere i compiti prefissati.

1.4 L'hardware

- **Unità centrale**

Di solito i componenti del PC vengono considerati separando l'unità centrale e le periferiche.
L'**unità centrale** è una scatola (**case**) che contiene i componenti elettronici e i circuiti integrati fondamentali per il funzionamento del computer.
L'unità centrale è dotata all'esterno di un interruttore principale, che serve per l'accensione del PC.
Il tasto di ripristino (**reset**) serve per riavviare il PC se questo rimane bloccato e non risponde a nessun comando, ma va usato con cautela: quando si preme questo pulsante è possibile perdere delle informazioni, se non sono state in precedenza salvate; è quindi l'ultimo metodo cui ricorrere quando insorgono problemi.
Il **drive per dischetti** è il dispositivo per leggere e scrivere dati sui dischetti (floppy disk). Viene di solito indicato come unità A. Il drive è dotato di un motore per far girare i dischetti e di una testina di lettura e scritttura. Una spia luminosa indica se la testina è in funzione: quando la spia è accesa è importante non togliere il dischetto dal drive per evitare di danneggiarlo.
Il **lettore CD** è un dispositivo che consente di leggere dati da un CD-ROM. I lettori CD presentano solitamente l'indicazione della velocità di trasferimento dei dati, determinate secondo l'unità di misura dei primi lettori (150 KB/sec) nati per l'ascolto dei CD audio. Più il lettore è veloce meglio è, specialmente se si devono visualizzare sequenze animate, giochi o film.
Nella parte posteriore dell'unità centrale sono disposti gli ingressi (**porte**) in cui inserire i connettori dei cavi che collegano le periferiche e le prese in cui inserire i cavi di alimentazione. Sulla maggior parte dei PC porte e connettori sono contrassegnati in modo da facilitare un corretto inserimento: sulla porta della tastiera, per esempio, è riprodotto il disegno di una tastiera.
Nelle **prese di alimentazione** della corrente (In e Out) vanno inseriti rispettivamente il cavo che porta la corrente all'unità centrale e quello che trasferisce la corrente dall'unità centrale al video.
La tastiera e il mouse vengono collegati all'unità centrale con connettori rotondi che vanno collegati alle **porte per la tastiera e per il mouse**.
Nella **porta parallela** occorre inserire il connettore del cavo della stampante, che di solito viene indicata come LPT1.
I PC sono solitamente dotati di due **porte seriali**, di dimensioni diverse (a 9 e 25 pin), a cui connettere ad esempio il modem o altri dispositivi. I **pin** sono sottili spinotti metallici che servono a condurre il segnale elettrico nel connettore.
Nella **porta video**, indicata con la sigla VGA, deve essere collegato il cavo proveniente dal video.
Se il PC è collegato in rete con altri computer di uno stesso ufficio, sarà corredato di un **connettore di rete** per inserire il cavo di rete.
Se il PC è predisposto per l'audio, nella parte posteriore dell'unità centrale si trovano uno o più **ingressi e uscite audio**, collegabili a diffusori, per esempio cuffie, casse, microfono. Si osservi che senza le casse un PC non si può considerare multimediale. Per funzionare correttamente le casse hanno bisogno di un componente hardware, la scheda audio contenuta dentro l'unità centrale.
Accanto agli ingressi audio si trova la porta per il joystick, o **porta MIDI**, in cui si può inserire il connettore di una manopola per interagire con i video giochi, il joystick. La porta MIDI permette di connettere anche una periferica audio MIDI, come una tastiera per suonare.
I tipi di PC più comuni sono: **desktop**, **tower** e **portatili**. Il desktop è adatto per appoggiare l'unità centrale sulla scrivania; si parla di tower se l'unità centrale ha uno sviluppo verticale e si può appoggiare sia sulla scrivania sia sul pavimento. I PC portatili o notebook sono così chiamati perché si possono trasportare come una valigetta. Hanno prestazioni paragonabili agli altri modelli, ma costano di più per la maggior miniaturizzazione dei componenti.
Le funzionalità del PC sono garantite dai vari componenti assemblati dentro l'unità centrale: schede, circuiti elettronici e cavi.
Il componente principale posto dentro l'unità centrale è la scheda madre (**motherboard**); su di essa si innestano tutti gli altri e fa da tramite per lo scambio delle informazioni.

La scheda madre si presenta come un grosso circuito stampato di forma rettangolare che contiene: il microprocessore, la memoria RAM, i circuiti che collegano le memorie di massa (cioè il disco fisso, il floppy disk e il CD-ROM), il controller, la scheda video, la scheda audio, le unità periferiche (monitor, tastiera, mouse, joystick, stampante, modem, scanner).
Il **chip** è un circuito composto da più componenti elettronici, integrati mediante processi di miniaturizzazione in un unico involucro di dimensioni ridotte.

- **Microprocessore**

Il **microprocessore** , detto anche **CPU** (**Central Processing Unit** – unità centrale di elaborazione), è la parte più importante del PC. E' un chip integrato che dirige e controlla ogni attività del computer, situato sulla scheda madre e costituito da una piccola piastra di silicio sulla cui superficie sono stati creati milioni di transistor miniaturizzati.
L'era del personal computer è cominciata con l'avvento del microprocessore. La realizzazione di un circuito integrato di dimensioni dell'ordine di pochi millimetri, in grado di presiedere e coordinare tutta l'attività della macchina, è stato il contributo fondamentale per la miniaturizzazione dei calcolatori, il miglioramento delle loro prestazioni e, di conseguenza, l'entrata massiccia dei PC nella vita quotidiana.
La CPU svolge due funzioni fondamentali: governa tutte le operazioni richieste dal sistema operativo e dalle applicazioni, cioè genera tutti i segnali occorrenti per il funzionamento degli altri circuiti a essa collegati, ed esegue tutti i calcoli, poiché contiene al suo interno l'Unità Logico-Aritmetica, l'**ALU** (**Arithmetic-Logic Unit**).
Possiamo immaginare il microprocessore come suddiviso in due parti: l'unità di controllo e l'unità logico-aritmetica. L'unità di controllo ha il compito di controllare le informazioni e i comandi che vengono inseriti nel computer e di tradurli in un linguaggio comprensibile agli altri componenti del computer; è responsabile della memorizzazione delle informazioni e dei comandi nella memoria centrale del computer, la memoria RAM, e del loro trasferimento dalla RAM all'ALU e viceversa.
L'unità logico-aritmetica esegue tutte le operazioni logiche e aritmetiche che le vengono passate dall'unità di controllo.
Il microprocessore e gli altri componenti elettronici che si trovano sulla scheda madre comunicano tra loro per mezzo di impulsi elettrici, che viaggiano attraverso piste di rame tracciate sulla scheda stessa che, per la loro funzione di trasporto, si chiamano **bus**.
Il bus centrale, che mette in comunicazione la CPU con la RAM, si chiama **system bus** ossia bus di sistema. A esso sono connessi tutti i bus che collegano la CPU con gli altri dispositivi di ingresso e uscita, cioè tutti quei componenti che possono ricevere e inviare informazioni: drive dei dischetti, tastiera, monitor, ecc.
Il bus di sistema è definito da un valore che ne misura l'ampiezza, cioè il numero di bit di informazioni che possono essere trasferiti contemporaneamente. Questo numero è andato crescendo con il progresso tecnologico dei circuiti, passando dai 16 bit delle prime CPU ai 64 bit dei modelli oggi più comuni.
Le operazioni della CPU sono temporizzate da un cronometro (**clock**), la cui frequenza viene misurata in milioni di cicli al secondo (megahertz- MHz)
Le prime CPU lavoravano a una frequenza di 4 – 5 MHz; in seguito, con l'evoluzione dei microprocessori, la frequenza di clock è salita rapidamente a 16, 44, 66, 133, 200 MHz, fino ad arrivare ai modelli di CPU più recenti, che operano a 500, 600 MHz e oltre.
Le frequenze più elevate risultano particolarmente indicate se si vogliono utilizzare programmi di grafica, videogiochi molto recenti, film su supporto DVD, applicazioni per il montaggio video.

- **Memoria RAM**

La **memoria RAM** (**Random Access Memory**, memoria ad accesso casuale) è la memoria centrale del computer. Si tratta di un dispositivo in cui vengono caricati dati e programmi nel momento in cui devono essere elaborati.

Quando si chiede al computer di eseguire un programma, il processore ne estrae dal disco rigido una copia, la memorizza temporaneamente nella RAM e quindi la esegue.
La quantità di memoria RAM è cruciale per il buon funzionamento del PC: quanto maggiore è la RAM, tanto meno frequentemente la CPU deve rivolgersi alle memorie secondarie (disco rigido, floppy disk, CD-ROM) per lavorare.
I dati, però, restano nella RAM solo finché il computer è in funzione. Quando si spegne il computer, la RAM si svuota: in questo senso si dice che la RAM è una "memoria volatile". Il sistema operativo e tutti gli altri file verranno prelevati dal disco rigido e caricati di nuovo nella RAM alla successiva riaccensione. Un'interruzione di corrente ad esempio può causare la perdita di tutti i dati contenuti nella RAM: per questo è importante salvare il lavoro svolto con il computer a intervalli regolari e abbastanza brevi.
Il processore sfrutta la velocità della RAM per elaborare dati e informazioni nei tempi più rapidi. La RAM infatti è molto più veloce di ogni altro tipo di memoria: se per estrarre un dato da un disco rigido sono necessari alcuni millisecondi (un millisecondo = un millesimo di secondo), per compiere un'operazione analoga dalla RAM di sistema bastano alcuni nanosecondi (un nanosecondo = un miliardesimo di secondo).
Memoria ad accesso casuale non significa che all'interno della RAM i dati siano sparpagliati a caso, senza criterio; vuol dire che al processore occorre sempre lo stesso tempo per accedere a una qualsiasi, casuale, parte della memoria. Sarebbe più corretto chiamarla memoria ad accesso non sequenziale o ad accesso diretto (gli altri tipi di memoria sono in tutto o in parte ad accesso sequenziale), perché la RAM è organizzata in modo da permettere che i dati siano immagazzinati e letti direttamente dalle specifiche locazioni o celle di memoria in cui si trovano, senza passare attraverso le locazioni che precedono o seguono le locazioni richieste.

- **Cache RAM**

Nei processori più moderni alla RAM si affianca la **cache RAM**: è un tipo di RAM molto veloce che contiene i dati e i comandi utilizzati più frequentemente dal processore, in modo da poterli mettere a disposizione più rapidamente e rendere più veloci le operazioni e i calcoli del computer.
La memoria cache è organizzata in due livelli: una parte è incorporata sulla scheda madre, un'altra parte è integrata nel processore e lavora alla sua stessa frequenza ed è quindi la più veloce, ma anche la più costosa; questa parte di memoria cache non è ampliabile, perché ciò renderebbe necessaria la sostituzione del processore stesso.

- **Memoria ROM e BIOS**

Se al momento dell'accensione del computer la memoria RAM è vuota, dove sono conservate le informazioni che consentono al computer di ripartire e di eseguire i vari programmi? Le istruzioni di base che devono essere trasmesse alla CPU all'avvio del sistema sono contenute nei circuiti della **memoria ROM** (**Read Only Memory**), una memoria permanente anch'essa presente sulla scheda madre.
Come dice il nome, è una memoria di sola lettura il cui contenuto è stato registrato in fase di costruzione del computer e quindi non può essere modificato.
Ogni volta che viene acceso, il computer esegue un piccolo programma contenuto nella ROM che gli permette di:
– identificare il processore installato sulla scheda madre;
– controllare la quantità di RAM in dotazione e verificarne il funzionamento;
– esaminare il disco ed eventuali periferiche aggiuntive (ad esempio il CR-ROM);
– leggere il settore del disco rigido in cui sono contenute le istruzioni per l'avvio del sistema operativo.
In particolare la ROM che avvia il sistema è chiamata **BIOS** (**Basic Input Output System**). Il BIOS nei computer più recenti può essere aggiornato in caso di necessità, ad esempio per

correggere eventuali difetti oppure per far riconoscere alla scheda madre un nuovo microprocessore.
Il BIOS fornisce numerosi altri servizi di sistema, tra cui la gestione della data: è proprio nel corretto funzionamento del BIOS che si sono concentrate le maggiori preoccupazioni relative al millenium bug.
Il BIOS è situato su un chip di ROM, registrato in fase di costruzione; questo permette che sia sempre disponibile e non riporti conseguenze in caso di danneggiamento dell'hard disk e fa sì che il computer sia in grado di ripartire.

- **Memorie secondarie**

Poiché la memoria RAM è soltanto temporanea, dati e programmi per non essere perduti devono essere salvati su memorie permanenti, le **memorie secondarie** o **memorie di massa**.
Le più importanti e diffuse memorie di massa sono il disco rigido (hard disk), il floppy disk e i CD-ROM.
Floppy e CD-ROM sono rispettivamente i supporti più adatti per il trasporto di dati e per la lettura di software commerciali. Il disco rigido rappresenta il principale dispositivo per la memorizzazione dei dati.

- **Disco rigido**

E' un'unità molto capiente in cui dati e programmi possono essere archiviati; l'hard disk è uno dei componenti del computer che presenta componenti meccanici oltre che elettronici; è alloggiato in un lettore (drive) ed è costituito da una serie di dischi o piattelli impilati uno sull'altro, che ruotano a velocità molto elevata. Quando si accende il computer i dischi iniziano a girare, mantenendosi costantemente in moto.
La superficie dei dischi è ricoperta da particelle magnetizzate che formano delle tracce concentriche, suddivise in settori; ognuno dei dischi ha lo stesso numero di tracce e una serie di tracce corrispondenti è chiamato cilindro.
Su ogni faccia di ciascun piattello vi è una testina magnetica che legge e scrive i dati. Le testine sono tutte fissate sullo stesso supporto, quindi si muovono sempre tutte insieme in direzione radiale, e possono leggere e scrivere i dati sulle due facce di ogni piattello del disco rigido, senza toccare la superficie (data la velocità di rotazione, se la toccassero la rovinerebbero immediatamente).
Le informazioni sono memorizzate sul disco rigido per cilindri: prima è riempita una determinata traccia e settore e poi, dal momento che le testine sono posizionate sullo stesso cilindro, tutte le restanti tracce di quel cilindro. Finché il cilindro non è stato riempito completamente, le testine non si spostano su un'altra traccia e quindi su un altro cilindro.
Questo criterio semplifica e velocizza le operazioni di lettura e scrittura, perché le informazioni correlate si trovano sullo stesso cilindro o comunque su cilindri successivi.
Se si cancellano delle informazioni l'ordine di memorizzazione dei dati viene alterato perché negli spazi vuoti saranno registrate altre informazioni non collegate alle precedenti, costringendo così le testine a muoversi continuamente avanti e indietro alla ricerca dei frammenti di file durante le operazioni di lettura.
Per riordinare il disco rigido sono disponibili dei programmi appositi, chiamati programmi di **deframmentazione**, che permettono di riunire le informazioni secondo i criteri più utili per il lavoro delle testine e quindi di migliorare le prestazioni del disco rigido.
Il disco rigido è chiuso in un contenitore sottovuoto; la parte inferiore della scatola è costituita da un circuito stampato in cui sono situati i componenti elettronici che controllano il movimento dei dischi e delle testine. Il dispositivo che si occupa di posizionare la testina in modo che trovi le informazioni richieste si chiama **controller**.
Quando la CPU richiede la lettura di una determinata traccia in un certo settore e cilindro, il controller posiziona la testina e incomincia a recuperare i dati fino a riempire la memoria cache disponibile, poi si occupa di passarli alla CPU e quindi alla RAM, la memoria di lavoro del PC.

La **File Allocation Table** (**FAT**) è lo schedario che consente al controller di organizzare i dati su disco: nel caso in cui il settore su cui è registrata la FAT venga danneggiato, il controller perde tutti i riferimenti ai file registrati su disco, che diventa così inutilizzabile.
Gli hard disk, che nei primi modelli avevano una capacità di 10 MB, ora hanno capacità fino a 20 GB e oltre. La dotazione considerata minima in questo momento è di 4 – 5 GB ma, se possibile, è preferibile scegliere misure superiori (si tenga conto che da solo Windows 98 può occupare oltre 300 MB, Windows 2000 oltre 400 MB). Un disco rigido grande è anche più veloce nella lettura dei dati e quindi in grado di innalzare le prestazioni di tutto il computer, perché permette al sistema operativo di lanciare programmi, caricare e salvare documenti e risultati in modo rapido.
Sulle prestazioni dell'hard disk incidono vari elementi, quali la velocità di rotazione, il tipo di controller, la densità con cui vengono scritti i dati, il tempo medio di accesso alle informazioni.
La velocità di rotazione dei dischi nei modelli più recenti è di almeno 7200 giri al minuto, e si arriva fino a 10000 giri al minuto: maggiore è la velocità di rotazione, minore è il tempo necessario per trovare le informazioni sul disco rigido.
Il tempo medio di accesso rappresenta il tempo impiegato dall'hard disk per estrarre un dato: i dischi più veloci arrivano a circa 5 millisecondi.

- **Floppy disk**

I floppy disk sono dischi magnetici rimovibili di capacità ridotta, utilizzati per memorizzare informazioni su un supporto esterno al computer e spostare i dati da un PC all'altro, soprattutto se non si è collegati in rete.
Il tipo standard di floppy può contenere 1.44 MB.
Prima di essere utilizzati la prima volta per memorizzare informazioni, i floppy devono essere formattati, ossia predisposti dal computer per le operazioni di lettura e scrittura dei dati. La formattazione consiste nella suddivisione del rivestimento magnetico del dischetto in tracce concentriche e settori, in modo da permettere alla testina di leggere e scrivere i dati rapidamente.
Poiché sono supporti magnetici, i dischetti devono essere tenuti lontani da campi magnetici, come quelli generati da diffusori audio, televisori, calamite, e dallo stesso monitor del PC; inoltre non devono essere esposti a fonti di calore.
Per evitare che le informazioni memorizzate su un dischetto siano cancellate per errore è possibile proteggere il dischetto mettendolo in modalità di sola lettura; per questo occorre spostare il quadratino nell'angolo inferiore sinistro del dischetto; spostando di nuovo il quadratino la protezione sarà tolta.

- **CD-ROM**

I **CD-ROM** (**Compact Disk Read Only Memory**), sono supporti di memoria di sola lettura utilizzati per la memorizzazione dei dati, simili ai CD per le incisioni musicali,.
Mentre i dischi magnetici possono essere scritti e cancellati più volte, i CR-ROM, dopo essere stati registrati una prima volta, possono essere utilizzati soltanto per la lettura delle informazioni memorizzate.
Sul CD-ROM le informazioni digitali (i bit) sono codificate come incisioni (**pit**) sulla superficie del disco: queste vengono colpite da un raggio laser, che determina il valore rappresentato da ciascun pit (0 oppure 1), per poi essere decodificate e trasmesse.
L'informazione su CD-ROM viene scritta come un'unica spirale continua; i dati vengono letti a velocità costante e la lettura avviene in modo sequenziale.
I CD-ROM, che non sono sensibili ai campi magnetici, costituiscono un supporto di memorizzazione molto affidabile e di elevate capacità (circa 600 MB), che permette di distribuire grandi quantità di informazioni. Per scrivere su questi supporti sono necessarie speciali apparecchiature chiamate **masterizzatori**.
Si stanno diffondendo anche supporti ottici riscrivibili, i **CD-RW** (**Rewritable**), sui quali si può riscrivere più volte.

- **DVD-ROM**

Apparentemente solo la scritta DVD-Video distingue un **DVD** (**Digital Versatile Disk**) da un CD, ma in realtà il nuovo supporto ha una capacità molto maggiore di immagazzinare dati: può contenere l'equivalente di circa 7 CD-ROM.
Se il CD nacque principalmente come supporto per ascoltare musica in formato digitale (si definiscono digitali le informazioni che possono essere rappresentate attraverso numeri, digit, ed elaborate dal computer), il DVD deve la sua comparsa all'esigenza di riprodurre su un supporto digitale interi film. Le immagini digitali di un film su DVD sono qualitativamente migliori, per luminosità, contrasto e definizione, di quelle di una videocassetta: per questo l'industria cinematografica punta su questo supporto e le video cassette VHS probabilmente saranno destinate a scomparire quando i PC dotati di lettori DVD saranno abbastanza diffusi.

- **Scheda video**

Con l'avvento dell'interfaccia grafica, lo sviluppo di videogiochi e applicazioni multimediali sempre più sofisticate e il diffondersi del World Wide Web, la **scheda video**, il dispositivo responsabile delle immagini che appaiono sul monitor, è diventato nel giro di pochi anni uno dei componenti fondamentali del PC.
La scheda video oggi è un vero e proprio computer, dotato di processore, memoria RAM e ROM, in grado di visualizzare filmati e animazioni sempre più "reali" per definizione delle immagini e per qualità del colore.

- **Scheda audio**

Le schede audio riproducono suoni digitali, cioè suoni convertiti in file numerici, gli stessi che provengono dai CD musicali.
Allo stesso modo, quando registrano suoni o musica con un microfono o un collegamento, le schede audio convertono il suono in un file.
La tecnica che permette di trasformare i suoni in numeri, ossia in sequenze di bit, si chiama **campionamento**. La qualità dei suoni in formato digitale (e le dimensioni del file risultante) dipende dal dettaglio con cui l'onda sonora viene convertita in bit.
All'interno della scheda si trova un convertitore analogico/digitale che digitalizza i suoni in entrata; le informazioni digitali vengono opportunamente elaborate e memorizzate in un file.
Per eseguire un suono che è stato registrato, il file viene caricato dalla CPU e inviato a un convertitore digitale/analogico che trasforma i bit in segnali elettrici, che opportunamente amplificati alimentano i diffusori per produrre il suono.

Le periferiche

Le periferiche sono i dispositivi che permettono al PC di comunicare con l'esterno: possono servire per introdurre dati e programmi (dispositivi di input, ad esempio tastiera e mouse) o per comunicare all'utente i risultati di un'operazione (dispositivi di output, ad esempio video e stampante).

- **Mouse**

Il mouse è uno strumento di puntamento che serve per attivare comandi o per selezionare e trascinare oggetti agendo direttamente su ciò che compare sullo schermo.
Per utilizzare il mouse è sufficiente trascinarlo su una superficie piana; al movimento del mouse corrisponde un movimento del puntatore, cioè della freccia sullo schermo.

Sul mouse normalmente ci sono 2 pulsanti; il più usato è il sinistro, che permette di selezionare icone o bottoni ed eseguire applicazioni, mentre con il destro è possibile visualizzare le proprietà di un oggetto o attivare menu di scelta rapida.
L'uso del mouse è limitato a poche azioni fondamentali: fare clic, fare doppio clic, selezionare e trascinare.

- **Tastiera**

La tastiera è la principale interfaccia di comunicazione con il computer, il dispositivo che permette di introdurre informazioni nel computer. Grazie alla tastiera è possibile scrivere testi, introdurre dati e impartire comandi al computer.
I tasti presenti sulla tastiera sono classificati in tasti alfanumerici (lettere e numeri), tasti di punteggiatura e tasti speciali (tasti funzione, tasti di controllo e tasti freccia).
Nella parte destra della tastiera in genere è presente un tastierino numerico separato, simile a quello di una calcolatrice, per agevolare la battitura dei numeri. Non esiste un unico modello di tastiera per PC, ma vari tipi che differiscono nel numero totale di tasti e nel posizionamento di alcuni di essi; il modello più comune è la cosiddetta tastiera avanzata, che ha 101 tasti.
I tasti speciali sono tasti specifici che acquistano funzioni differenti a seconda del programma che si sta eseguendo e delle combinazioni con altri tasti con cui vengono associati.

ALT: (Alternate) è un tasto che si usa in combinazione con altri per impartire al computer diversi tipi di comandi. Il tasto Alt premuto con un altro tasto permette di variare la funzione di quest'ultimo. Ad esempio, usato in combinazione con il tasto F4 corrisponde al comando di chiusura della finestra attiva.
BACKSPACE ¬ : fa tornare indietro il cursore di una posizione, cancellando il carattere che si trova alla sua sinistra.
BARRA INVERSA (BACK SLASH) : si usa in alcuni comandi e per specificare il percorso di un file, cioè la posizione; ad esempio C:\Windows\calc.exe indica la posizione della calcolatrice di Windows.
BLOC NUM (NUM LOCK) : attiva o disattiva il tastierino numerico; quando è disattivato, i tasti numerici corrispondono ai tasti freccia per lo spostamento del cursore.
CANC : cancella il carattere a destra del cursore oppure un qualsiasi oggetto precedentemente selezionato, ad esempio una parte di testo o un file o una cartella.
CTRL : (Control) il tasto CTRL (in realtà sulla tastiera ce ne sono due, identici) viene usato in combinazione con altri per eseguire più rapidamente un comando, al posto del mouse; ad esempio per salvare un file basta premere contemporaneamente i tasti CTRL e S. Se un programma si blocca, per uscire dal programma, anziché spegnere il PC, si possono premere contemporaneamente i tasti CTRL , ALT e CANC.
ESC : (Escape) serve in genere per annullare o ignorare un comando; può anche essere usato in combinazione con altri tasti: premendo CTRL + ESC si apre il menu **Start.**
FINE : si usa per spostarsi all'interno di una finestra; ad esempio premendo CTRL + Fine si porta il cursore in fondo alla finestra che si sta visualizzando.
Tasti FRECCIA : sono 4 tasti direzionali che muovono il cursore in alto, in basso, a destra, a sinistra. Utilizzati in combinazione con i tasti ALT, CTRL, SHIFT possono svolgere diverse operazioni, a seconda del programma che si sta eseguendo.
Tasti FUNZIONE : sono tasti speciali che hanno significati diversi a seconda delle applicazioni in cui vengono utilizzati. In Windows, ad esempio, premendo il tasto F1 si apre la guida del programma che si sta utilizzando.
INVIO : è utilizzato per inviare un comando, per andare a capo quando si scrive.
MAIUSCOLE (CAPS LOCK) : è simboleggiato da un piccolo lucchetto che può essere attivato e disattivato; quando viene attivato si digitano solo lettere maiuscole; non serve per digitare i caratteri in alto nei tasti doppi.
SHIFT : come sulle macchine da scrivere, tenendo premuto il tasto SHIFT mentre si preme il tasto di una lettera si ottiene il carattere maiuscolo, premendo il tasto su cui compaiono due simboli si

produce il simbolo in alto. Come il tasto ALT, il tasto SHIFT, usato insieme ad un altro, permette di variare il significato di quest'ultimo.
Tasti Pag- e Pag¯ : permettono di scorrere velocemente le informazioni che appaiono all'interno di una finestra.
STAMP (PRINT o PRINT SCREEN) : premendo questo tasto si "fotografa" ciò che appare sullo schermo e lo si registra negli Appunti del computer, da dove può essere copiato in un altro programma. Questa procedura si chiama **Copia** e **Incolla**.
Tasto TAB : permette di creare spazi di tabulazione, utili, per esempio, per allineare parti di testo in una pagina. Il tasto TAB sposta il cursore in punti prestabiliti lungo le righe di un testo o tra le opzioni di un menu o di una finestra.
Tasti Windows : i tasti che riportano il logo di Windows aprono il menu Start; il tasto che riporta l'icona di un menu attiva un menu di scelta rapida, lo stesso che si apre quando si fa clic con il pulsante destro del mouse, e quindi la sua funzione varia a seconda di dove si trova il puntatore sullo schermo.

- **Monitor**

Il monitor è la periferica di output, che permette di visualizzare le informazioni grafiche e di testo e di vedere il risultato dell'elaborazione svolta dal computer.
Sul mercato sono presenti vari tipi di monitor. I più comuni sono quelli a tubo catodico, che si basano sulla stessa tecnica di funzionamento dei televisori. Negli ultimi anni si sono diffusi gli schermi a cristalli liquidi, utilizzati nei portatili, e quelli al plasma.
Il monitor del computer visualizza le immagini dividendo lo schermo in migliaia (o milioni) di piccoli quadratini colorati, i **pixel**, ordinati in righe e colonne. I pixel sono così vicini uno all'altro da apparire uniti. Il numero di bit utilizzati per rappresentare ogni pixel determina il numero di colori che possono essere visualizzati sullo schermo. Ad esempio se si impiegano 8 bit (ossia 1 byte) per pixel si riesce a visualizzare una gamma di 256 colori diversi.
Il numero di pixel che può essere contenuto sul monitore costituisce la **risoluzione**, definita dal numero di pixel sull'asse orizzontale e da quello sull'asse verticale.
La qualità dell'immagine che appare sullo schermo dipende dalla risoluzione e dalla quantità di bit utilizzati per rappresentare ogni pixel, quindi dal numero di colori visualizzati contemporaneamente sullo schermo, che possono essere 256, 65.536 e infine 16.700.000 .
Le risoluzioni standard per i PC sono le seguenti: per monitor da 15" o 17", che sono i più comuni, la risoluzione ideale è 800 righe × 600 colonne, oppure 1024 righe × 768 colonne; per un monitor da 21" è possibile impostare una risoluzione di 1280 righe × 1024 colonne, a 8, 16, o 24 bit.
I sistemi che utilizzano 24 bit, cioè 3 byte per pixel, si chiamano true color perché permettono di visualizzare più di 16 milioni di colori distinti, una gamma nella quale dovrebbero rientrare tutte le tonalità di colore esistenti in natura.
L'intera schermata viene ridisegnata continuamente. In termini tecnici la frequenza con cui viene ridisegnata l'immagine al secondo si chiama frequenza di refresh e può arrivare fino a 120 Hz, cioè 120 volte al secondo.

- **Stampante**

La stampante è la periferica che permette di riportare su carta ciò che appare sul monitor: testi, grafici, risultati, immagini, fotografie.
Le stampanti più diffuse sono le stampanti **a getto d'inchiostro** (dette **ink-jet**) e le stampanti **laser**.
Ogni oggetto stampato è costituito da tanti piccoli punti chiamati **dot**. Le dimensioni dei dot sono talmente ridotte che l'occhio umano ha l'impressione di continuità dell'immagine. La quantità di punti che la stampante imprime sull'unità di superficie di stampa costituisce la risoluzione, che si misura in **dpi** (**dots per inch**), cioè punti per pollice (1 pollice = 2.54 cm).

Una stampante che ha una risoluzione di 600 × 600 dpi in ogni pollice quadrato stampa fino a 360.000 (= 600 × 600) punti. In genere maggiore è il numero di dpi, più elevata è la qualità di stampa.
Le stampanti a getto d'inchiostro sono oggi le più diffuse perché hanno risoluzioni maggiori di 300 dpi, costano poco e sono disponibili a colori. Le stampanti laser arrivano a 1200 dpi nei modelli domestici, ma sono più costose.

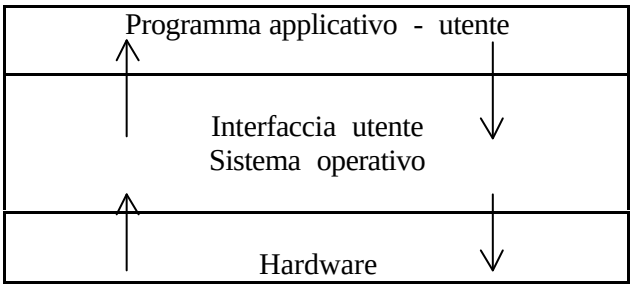
- **Modem**

E' il dispositivo utilizzato per collegare il PC a Internet o a una rete di computer aziendali.
Il termine modem è nato dalla fusione delle parole modulation e demodulation. Il modem infatti ha la funzione di trasformare i dati digitali in onde elettroacustiche per farli viaggiare sui cavi telefonici e di riconvertire le onde nel formato digitale che può essere elaborato dal computer.
L'unità di misura della velocità di trasmissione di un modem è data dal numero di bit di dati che può trasmettere in un secondo ed è chiamata *bps* (*bit per second)*.
I moderni modem garantiscono un minimo di 28800 bps, fino ad arrivare ad un massimo di 57600 bps (disponibile solo su alcuni modelli di modem ma che comunque diventerà lo standard del futuro). I più venduti attualmente sono i modem caratterizzati da una velocità di 33600 bps.

1.5 Il software

In un qualsiasi elaboratore, come già accennato, si possono individuare due componenti fondamentali: l'**hardware**, cioè l'insieme delle componenti fisiche, e il **software**, cioè l'insieme dei programmi, fisicamente non tangibile, ma fondamentale per far funzionare il calcolatore.
Con riferimento alla relazione con l'hardware, l'utente e il mondo esterno, il software può essere classificato in tre diverse categorie:

1 - software di base o di sistema, che svolge le attività primarie necessarie al funzionamento dell'elaboratore.



Il software di base è costituito da un insieme di moduli a diretta interazione con l'hardware del calcolatore:
• **sistema operativo**: è la parte più vicina all'hardware e con esso interagisce direttamente conferendogli gran parte delle sue caratteristiche operative, quali: modalità di gestione della memoria, tipi di periferiche riconosciute, numero di utenti gestibili contemporaneamente, ecc.
E' un complesso insieme di programmi con il quale l'utente non interagisce direttamente. All'accensione del calcolatore almeno una parte di esso, detta **nucleo**, viene caricata nella memoria centrale ed ivi risiede fino allo spegnimento.
• **interfaccia utente**: è un insieme di programmi che permette all'uomo di comunicare con il sistema operativo, non altrimenti accessibile.
Le modalità di interazione tra uomo e macchina cambiano a seconda della raffinatezza del programma che realizza l'interfaccia (comandi da digitare a tastiera oppure operazioni da selezionare su menu che appaiono a video).

L'utente deve imparare il linguaggio di comandi (in genere con un vocabolario ricco e complicato) o scegliere i comandi dai menu (molto più semplice e intuitivo) per poter operare con una specifica macchina.

2 - software applicativo, che utilizzando strumenti messi a disposizione dal software di base, serve per ottenere particolari elaborazioni dei dati inseriti dall'utente.
Rientrano in questa categoria tutti i programmi utilizzati dagli utenti per la soluzione di loro problemi specifici: battitura di testi, contabilità, disegno, soluzione di problemi matematici, statistici, ecc.; risiede in memoria centrale quando viene richiamato esplicitamente dall'utente.
Nella pratica l'utente interagisce più spesso con il software applicativo che con quello di base.

3 - software di comunicazione : la comunicazione fra gli elaboratori è una problematica che sta assumendo sempre maggior importanza. Esistono sistemi operativi concepiti espressamente per la comunicazione in rete, e che a questo subordinano tutte le altre funzioni, e programmi applicativi dedicati esclusivamente all'interscambio dati fra elaboratori.

1.6 Il sistema operativo

Il software di base è strettamente legato alla struttura dell'hardware, tanto che la sua principale componente, il **sistema operativo**, viene realizzata in modo sostanzialmente diverso in funzione delle caratteristiche dell'hardware che sarà destinato a gestire.
Ad ogni schema architetturale corrisponde, quindi, un sistema operativo diverso, ma, vista la complessità di realizzazione e gli enormi costi di sviluppo del software di base, i produttori di hardware limitano gli schemi architetturali utilizzati in modo da non usare molti sistemi operativi diversi.
Alcuni costruttori hanno adottato un'unica architettura hardware sull'intera gamma dei propri elaboratori, dalla stazione di lavoro individuale sino al mainframe e in tal modo sono in grado di abbattere tempi e costi di produzione di nuove macchine; all'utente inoltre risulta possibile operare su un sistema piuttosto che su un altro con maggior facilità.
Per chi scrive il software applicativo, l'operare con un unico sistema operativo significa un grosso vantaggio in termini di "portabilità dei programmi", in quanto non è necessario compiere alcuna modifica al software prodotto per consentire la migrazione su sistemi diversi da quello su cui è stato realizzato.
Nel mondo dei personal computer molti costruttori hanno adottato lo stesso sistema operativo, prima MS-DOS, poi Windows, realizzando macchine con schema architetturale praticamente identico.
Gli APPLE hanno invece un sistema operativo proprio e non se ne può usare un altro.
In un altro caso un unico sistema operativo è stato adottato da costruttori diversi su elaboratori anche di classe ed architettura diversa: è il caso del sistema operativo UNIX, scritto in modo tale che la parte di software facente riferimento alla struttura e all'architettura specifica della macchina sia ridotta al minimo essenziale. Così facendo, per utilizzare quel sistema operativo su elaboratori diversi viene modificata solo quella piccola porzione di codice.
Il sistema operativo è un insieme di numerosissimi programmi, che interagiscono fra loro dando origine a una struttura estremamente complessa.
La sua dimensione complessiva può essere anche ragguardevole, in relazione alla complessità degli eventi che è in grado di gestire.
Tutti i sistemi operativi sono costituiti da un insieme di moduli, ciascuno dei quali è deputato a svolgere una ben precisa funzione, che interagiscono fra loro secondo ben precise regole al fine di realizzare le funzionalità di base del sistema e di utilizzare al meglio le risorse.
Il sistema operativo effettua la **gestione delle risorse** del sistema informatico, risorse che sono date dai componenti principali (processore, memoria RAM, ecc.) e periferici (tastiera, video, memorie di massa, stampante, ecc.) disponibili nel sistema; ha inoltre il compito di **governare e controllare**

altri **programmi applicativi**: Questo significa che il sistema operativo attribuisce a tali programmi le risorse richieste e ne controlla l'uso.
Il controllo consiste nell'evitare che l'applicativo entri in conflitto con altri programmi o che faccia un uso indesiderato delle risorse messe a disposizione del sistema operativo; questo meccanismo è volto a ridurre ed evitare che il sistema si blocchi a causa di un conflitto tra programmi applicativi o tra programma applicativo e sistema operativo; quindi il sistema operativo ha una **funzione di supervisione sull'intero sistema**.
Inoltre il sistema operativo svolge una funzione di collegamento, o **interfaccia**, tra utente e macchina.
Tra i compiti più complessi del sistema operativo vi è la **gestione della memoria di massa**.
Come noto, i dati e i programmi sono memorizzati sotto forma di **file**, normalmente residenti su memoria di massa (dischi, nastri, ecc.).
Il termine file è molto generico: indica un contenitore di informazioni immagazzinate in un dispositivo di memorizzazione; anche un solo carattere deve essere necessariamente inserito in un file qualora si voglia memorizzarlo in modo permanente.
Sui file possono essere compiute le seguenti operazioni:
- **creazione** ;
- **apertura**, per successive operazioni sul loro contenuto;
- **chiusura**, per impedire altre operazioni sul loro contenuto finché non saranno riaperti;
- **cancellazione** ;
- **copiatura**, per la creazione di un ulteriore file identico al primo nel contenuto;
- **rinomina**;
- **visualizzazione** sullo schermo o in stampa.

I contenuti dei file possono essere manipolati da operazioni come:
- **lettura**, che comporta il passaggio del contenuto di un file a un processo;
- **scrittura**, che comporta il passaggio di dati da un processo a un file;
- **modifica**, che modifica il contenuto del file con scrittura ed eventuale cancellazione di una sua parte;
- **inserimento**, che aggiunge del contenuto al file.

La gran quantità di dati memorizzabili in un sistema di elaborazione deve essere in qualche modo organizzata al fine di consentirne un reperimento in tempi accettabili; i diversi file devono essere raggruppati e organizzati in diversi gruppi, che a loro volta sono raggruppati in altri gruppi.
Per il raggruppamento dei file si fa uso di un file particolare detto **directory**, il quale contiene sostanzialmente l'elenco dei file appartenenti a quel gruppo e il nome di eventuali sottogruppi (**sottodirectory**).
L'utente preferisce identificare i suoi programmi applicativi e i suoi archivi di dati con nomi simbolici mnemonici, senza dover ricordare tutte le altre informazioni, che sono tuttavia necessarie per rintracciare il file sul disco.
Il **file system** è la componente del sistema operativo che è adibita alla gestione dei file su memoria secondaria e che consente, di fatto, l'esecuzione delle operazioni sopra menzionate.
Compie un insieme di funzioni da "sistema di archiviazione" in modo da:
- fornire un meccanismo per identificare i file, associando un nome allo spazio fisico della memoria di massa in cui risiede il file;
- ottimizzare l'uso della memoria di massa;
- fornire opportuni metodi per accedere ai file e rendere gli accessi più veloci possibile;
- gestire in modo ordinato le variazioni di lunghezza dei file.

Nel sistema operativo è poi compreso un insieme di programmi in grado di produrre i comandi primitivi necessari per la **gestione delle periferiche** , ciascuna delle quali è controllata dal proprio insieme di comandi elementari.
Dato che un sistema di elaborazione deve funzionare per un periodo di tempo non troppo breve, deve essere garantita la possibilità di espansione per soddisfare l'eventuale insorgere di nuove esigenze. Questo obiettivo è perseguito mediante **strutture modulari**.
Lo stesso costruttore trae vantaggi non indifferenti dalla struttura modulare: a partire dagli stessi blocchi funzionali, è possibile soddisfare un ampio spettro di clienti aventi le esigenze più diverse.

D'altra parte, gli utenti trovano conveniente poter usufruire di macchine capaci di crescere insieme alle loro esigenze, conservando, se possibile, gli investimenti precedenti.
Per **configurabilità** si intende dunque la possibilità di collegare nuovi dispositivi periferici al sistema quando se ne presenti la necessità, senza dover riprogettare nulla a livello hardware o software: questa caratteristica contribuisce ad ottimizzare il parametro costo/prestazioni.
La configurabilità è consentita dal fatto che il sistema operativo gestisce i dispositivi periferici su due livelli:
- un livello generale, attivato dalle richieste dell'utente o dai moduli di servizio dello stesso sistema operativo, che controlla tutte le operazioni di input/output.
- un livello specifico, attivato quando è il caso dal modulo generale e attuato da un modulo di programma specifico per ogni unità periferica, detto **driver**.
Questo modulo interagisce con l'hardware del sistema e pilota direttamente il dispositivo periferico.
Per avere la massima operatività occorre che ogni dispositivo disponga del suo proprio driver di gestione e che questo sia inglobato, mediante un procedimento di configurazione, nel sistema operativo.
Il sistema operativo deve essere presente (almeno in parte) in memoria centrale prima che venga inviato qualsiasi comando; d'altra parte ogni volta che il computer viene spento il contenuto della sua memoria centrale viene irrimediabilmente perso.
A ogni accensione del sistema dunque il sistema operativo deve essere ricaricato. Sfortunatamente non si può semplicemente inviare un comando e lasciare che il sistema operativo si preoccupi di ricaricarsi; infatti, quando il computer viene acceso, la memoria centrale è vuota e quindi non è possibile leggere, interpretare ed eseguire nessun comando.
Tipicamente il sistema operativo è memorizzato su un disco: è sufficiente dunque disporre di un programma, da eseguire ogni volta che il sistema viene acceso, capace di copiare in memoria dal disco l'interprete dei comandi e il modulo di gestione dei dispositivi periferici. Lo speciale programma che esegue questa funzione si chiama **BOOT**.
Il programma di BOOT è registrato nei primi settori del disco e l'hardware è progettato in modo tale che, ogni volta che la macchina viene accesa, questo settore venga letto automaticamente e il programma venga eseguito; il calcolatore dispone a tal fine di una speciale memoria ROM che contiene le istruzioni capaci di prelevare dal disco il programma BOOT.
Il BOOT consta solo di poche istruzioni, sufficienti per leggere il resto del sistema operativo che deve essere caricato in memoria. A questo punto il sistema operativo è in funzione e può interpretare i comandi che gli pervengono dall'utente o da eventuali programmi applicativi.
Tra le funzioni dei sistemi operativi vi è quella di offrire differenti modalità operative, dipendenti dal sistema operativo stesso e dalla macchina usata per la gestione dei programmi in esecuzione.
Queste modalità operative si riferiscono prevalentemente alla possibilità di consentire l'accesso al calcolatore a uno o più utenti **(modalità single-user e multi-user)** oppure alla possibilità di far funzionare uno o più programmi contemporaneamente **(modalità single-tasing e multi-tasking)**.
La modalità attualmente impiegata sui PC è la modalità multi-tasking.
Riassumendo si può dire che il sistema operativo ha i seguenti compiti:
- gestione e controllo delle risorse dell'intero sistema informatico;
- supervisione di tutte le periferiche
- supervisione dei programmi in esecuzione;
- ottimizzazione delle risorse per ogni applicativo;
- semplificazione dell'uso della macchina tramite un'interfaccia verso l'utente, più adatta alle caratteristiche dell'utente stesso;
- semplificazione della gestione di operazioni ripetitive e frequenti;
- disponibilità di servizi sfruttabili direttamente dall'utente.

1.7 Interfacce a carattere e interfacce grafiche

Uno dei settori più attivi nel mondo dell'informatica riguarda la ricerca di tecniche di interazione con l'elaboratore che siano estremamente semplici e intuitive.
Incrementare la facilità di utilizzo di un calcolatore comporta vantaggi in due direzioni:
- permette l'uso del computer a persone che non hanno cultura informatica specifica;
- permette a tutti di compiere funzioni molto più sofisticate e di estendere il campo di applicazione del calcolatore nel proprio lavoro.

I calcolatori delle prime generazioni non disponevano di adeguati meccanismi di interazione con l'uomo; egli, per operarvi, doveva conoscere in modo molto approfondito gli schemi architetturali interni e doveva impartire con estremo dettaglio i comandi per svolgere le varie operazioni.
L'invenzione dei terminali video permise un passo decisivo nelle tecniche di interazione, ma solo nella seconda metà degli anni ′80 si sono diffuse le interfacce grafiche, con un conseguente salto qualitativo senza precedenti nella semplificazione del dialogo uomo-macchina.
L'**interfaccia utente** è un insieme di programmi che permette all'utente di comunicare con il sistema operativo, non altrimenti accessibile, e quindi con l'elaboratore.
Esistono vari metodi per costruire un'interfaccia utente.
La tecnica classica di interazione tra utente e macchina è basata sull'**interfaccia a carattere (CUI - Character based User Interface)**: un modulo detto **interprete di comandi** è in grado di interpretare ed eseguire dei comandi impartiti, sotto forma di successioni di caratteri, dall'utente alla tastiera.
Tali comandi, in un vocabolario più o meno limitato in rapporto alla complessità del sistema operativo, seguono una ben precisa sintassi. L'utente, per compiere qualsiasi operazione, deve prima imparare tale linguaggio ed è costretto ad acquisire anche alcune nozioni riguardanti l'organizzazione del sistema.
I manuali forniti con il software di base del calcolatore forniscono all'utente gli strumenti per lo studio e l'uso dei comandi.
Un sistema operativo con interfaccia a carattere molto noto è **MS-DOS (Microsoft Disk Operating System)**, sviluppato dalla Microsoft per i personal IBM e adottato dai costruttori dei personal computer IBM-compatibili (detti anche "cloni"); è stato il sistema più diffuso nel mondo dei personal computer, fino all'avvento delle intefacce grafiche.
La ricerca di nuove tecniche di interazione, meno ostiche dell'interfaccia a carattere, ha portato alla realizzazione dell'**interfaccia grafica (GUI - Graphical User Interface)**.
Le interfacce grafiche più diffuse sono:
- **Macintosh** della Apple; è storicamente la prima interfaccia grafica commerciale; la versione attuale, **Mac OS**, è oggi una delle più evolute e sofisticate. Anche l'hardware di Macintosh è stato progettato per ottenere una macchina perfettamente integrata.
- **Windows** per MS-DOS della Microsoft; ispirato al Macintosh, Windows costituisce un'interfaccia utente per il sistema operativo MS-DOS sul quale si basa, al contrario del sistema operativo di Apple, che è realizzato interamente in funzione dell'interfaccia grafica.

Data la grande diffusione attuale dei personal computer, Windows è l'interfaccia più diffusa.

2. La produzione del software

2.1 Introduzione

Il contributo che i calcolatori hanno apportato nei settori del calcolo scientifico e dell'automazione aziendale è stato decisivo negli ultimi decenni, grazie ad alcuni fattori di fondamentale importanza: i calcolatori sono in grado di compiere operazioni matematiche e manipolare grandi quantità di dati in tempi rapidissimi, eseguire compiti ripetitivi e prendere decisioni logiche risultanti da insiemi di istruzioni che sono in grado di riconoscere, tradurre, eseguire e memorizzare.
Il ruolo naturale del computer dell'attuale generazione è quello di semplice esecutore di un insieme ordinato e rigoroso di istruzioni che operano sui dati forniti e che portano a determinati risultati. E' sempre l'uomo a decidere ciò che l'elaboratore deve eseguire.
L'insieme ordinato di istruzioni elementari impartite a un calcolatore per compiere una certa azione, controllare lo svolgersi di un certo lavoro, eseguire una raccolta di dati secondo precisi criteri, ecc., prende il nome di **programma**; l'oggetto su cui compiere le azioni, quanto deve essere elaborato, confrontato o reperito costituisce i **dati**.
Il computer necessita di istruzioni estremamente dettagliate e inequivocabili che descrivono tutte le diverse opzioni operative che possono avere origine durante l'attività, e non è in grado di gestire una situazione imprevista e per la quale non sia stato codificato un comportamento: i programmi sono, appunto, la raccolta ordinata di operazioni che il calcolatore deve compiere in risposta a ben precisi eventi.
In base alle istruzioni contenute in un certo programma e partendo da certi dati ricevuti in input, un computer compie delle operazioni e produce altri dati in output: ogni elaborazione può essere ricondotta a questo elementare schema.
La complessità di un programma è direttamente proporzionale a quella del problema o dell'insieme di problemi che deve risolvere, ma non alla mole di dati da trattare.
Le case costruttrici di elaboratori, in genere, forniscono il software per il funzionamento "di base" degli elaboratori da loro realizzati.
Per la soluzione delle problematiche specifiche l'utente deve dotarsi di ulteriore **software**, detto **applicativo**.
Per le applicazioni più comuni esistono in commercio programmi anche molto sofisticati, detti comunemente **pacchetti (packages)** o **applicazioni (application programs)**; per esigenze più particolari e specifiche le aziende personalizzano pacchetti in commercio o ne sviluppano di interamente nuovi.
Esiste però anche la necessità di creare in proprio dei programmi che si adeguino in modo particolare al problema che si deve risolvere, e che per motivi diversi non può essere trattato con il software a disposizione.
Di qualsiasi natura sia il software (di base o applicativo), le operazioni che portano alla sua realizzazione hanno molti aspetti in comune.
La produzione di software è un'attività complessa e articolata; nel seguito si descrivono brevemente gli aspetti principali di maggior interesse in questo ambito.

2.2 Le fasi dello sviluppo di un progetto software

La completa realizzazione di un programma comporta tempi e metodiche estremamente diversi a seconda della complessità del problema o dell'insieme di problemi da risolvere; in genere essa passa attraverso le fasi di seguito elencate.

1. Analisi e definizione
Prima di iniziare a scrivere anche una sola riga di programma è necessario compiere un'analisi dettagliata del problema che si deve risolvere.

La macchina, infatti, come qualsiasi automatismo esistente, opera ed elabora solo e unicamente nel modo in cui noi abbiamo predisposto.
Le operazioni da comandare alla macchina devono quindi essere note fin dall'inizio e a un livello non vago e generico, ma molto preciso e soprattutto elementare.
La precisione nella formulazione dei problemi è una condizione indispensabile alla loro corretta risoluzione.

2. Progettazione
Una volta definito il problema da risolvere, si devono stabilire metodiche e strutture di dati per la sua soluzione.
Per le applicazioni e risoluzioni di tipo scientifico, si dovranno definire i problemi in termini di sistemi di equazioni e di formule da calcolare. Le stesse funzioni matematiche a loro volta dovranno essere poste in termini elementari, i soli comprensibili all'elaboratore.
Si passa cioè alla progettazione della soluzione, utilizzando ben precisi formalismi, ma senza ancora produrre effettive "istruzioni" di programma.
Tra i punti fondamentali vi sono:
1 - progettazione della struttura logica (individuazione dei moduli funzionali e relazioni logiche);
2 - progettazione della struttura fisica (strutture delle informazioni, metodi di accesso,...).
Nella maggior parte dei casi l'approccio utilizzato procede per livelli di dettaglio crescenti: tale metodica, comunemente detta **top-down** (dall'alto in basso) comporta la suddivisione del problema in parti separate di più semplice comprensione, fino a livelli di notevole dettaglio; ciascun problema di non immediata soluzione è ulteriormente suddivisa in problemi più semplici, fino ad identificare dei moduli elementari.

3. Codifica
Una volta definite chiaramente le metodiche da seguire per la soluzione del problema, dalla formulazione della soluzione con un certo formalismo si passa alla stesura delle istruzioni, utilizzando uno dei linguaggi di programmazione esistenti (Pascal, Cobol, C, Fortran,...).
Il risultato di questa operazione è costituito da uno o più programmi sorgente (nel caso di grossi progetti, sono anche in gran numero). Ciascun sorgente è poi tradotto in linguaggio macchina e i singoli moduli ottenuti sono fra loro collegati, per dare origine a programmi realmente eseguibili; queste operazioni vengono eseguite dal computer per mezzo di opportuni programmi detti rispettivamente **compilatore** e **linker**.

4. Correzione (debugging) e prova
Una delle attività più onerose per un programmatore è la prova di un nuovo programma.
Poiché è praticamente impossibile realizzare un intero programma senza compiere errori, è necessario predisporre una fase di correzione.
La fase di debugging (dal termine inglese "to debug", che può essere tradotto con "spulciare") porta all'eliminazione degli errori non riconducibili alla logica (gli errori di logica di chi redige il programma possono sfuggire a questa fase), ed è una fase che in realtà costa molto spesso tempo e fatica.
Le aziende produttrici di software affermano che il tempo necessario nella fase di debugging di un programma può variare da 1/3 a metà dell'intero tempo del progetto, intendendo per progetto il complesso delle operazioni di analisi del problema, stesura del programma e debugging.
La prova consente di verificare la correttezza del programma, ossia l'effettiva produzione dei risultati attesi.

5. Stesura della documentazione e della manualistica
La stesura della documentazione interna al programma deve avvenire simultaneamente alla realizzazione del codice ed ha la funzione di agevolare il compito di chi in seguito dovrà modificare il programma per adattarlo a nuove esigenze o per risolvere inconvenienti sfuggiti alla fase di prova.

La stesura della manualistica è una fase che riguarda principalmente il software prodotto dalle ditte specializzate nel settore; i manuali permettono all'utente finale l'installazione e l'uso effettivo del prodotto; di solito i pacchetti software sono anche dotati di un **help on line** (guida in linea).

6. Manutenzione e aggiornamento
Terminata la realizzazione del prodotto inizia una fase volta alla rilevazione di ulteriori imperfezioni o errori e alla risoluzione di necessità aggiuntive o modificate rispetto a quelle previste inizialmente.
Questa attività dà origine a diverse versioni (**release**) del programma, normalmente identificate da coppie di numeri. La prima versione è in genere classificata come 1.0, e ad essa seguiranno la 1.1, la 1.2,..., la 2.0, 2.1,...
Viene aumentato il primo numero quando al prodotto vengono apportate modifiche sostanziali, il secondo quando sono aggiunte nuove prestazioni marginali, e vengono risolti errori.

2.3 Gli algoritmi

La fase di progettazione, consistente nel determinare il metodo di soluzione del problema, è una delle fasi più importanti nello sviluppo di un progetto.
Il procedimento di soluzione, che non ha ancora alcun legame con quello che sarà il programma finale, è detto **algoritmo** ed è una descrizione che specifica una serie di operazioni, eseguendo le quali è possibile risolvere un determinato problema.
Le operazioni che un algoritmo può prescrivere possono essere di natura assai diversa, ma devono possedere delle caratteristiche ben precise.
Fra le molte definizioni di algoritmo possibili, particolarmente diffusa è la seguente, data da Knuth nella sua opera "*The art of computer programming*":

Definizione di algoritmo.
Un algoritmo è un insieme di regole (o istruzioni) aventi le seguenti caratteristiche:
– deve essere finito: concludersi dopo un numero finito di operazioni;
– deve essere definito e preciso: ogni istruzione deve essere definita in forma non ambigua;
– se ci sono dati in ingresso, deve essere precisato di che tipo sono, ad esempio numeri reali, interi, caratteri,...;
– deve fornire almeno un risultato (dato in uscita);
– deve essere eseguibile: tutte le operazioni devono poter essere eseguite esattamente, in un tempo finito, da un uomo che utilizzi mezzi manuali.

Va rilevato che spesso uno stesso problema può essere risolto utilizzando algoritmi diversi tra loro.
L'esecuzione di un algoritmo da parte di un esecutore, uomo o macchina che sia, si traduce in una successione di operazioni che vengono effettuate nel tempo. Con altre parole, potremo dire che l'esecuzione di un algoritmo evoca un processo sequenziale, cioè una serie di eventi che occorrono uno dopo l'altro, ognuno con un inizio e una fine ben identificabili.
Talvolta il processo evocato da un algoritmo è fisso, cioè è sempre lo stesso a ogni diversa esecuzione.
Si consideri, ad esempio, un semplice algoritmo per calcolare l'importo di una fattura:
– cerca la corretta aliquota IVA sulla tabella;
– moltiplica l'importo netto per l'aliquota trovata;
– somma il risultato all'importo netto.
Questo algoritmo è composto da tre istruzioni che devono essere eseguite in sequenza.
Ogni sua esecuzione farà sì che venga eseguita per prima l'operazione di ricerca in tabella, quindi il calcolo dell'IVA, infine il calcolo dell'importo lordo.
Le operazioni eseguite e il loro ordine, cioè, non cambiano, qualunque sia la merce da fatturare, l'aliquota da applicare o l'entità dell'importo netto.

Per descrivere in qualche modo il processo sequenziale evocato da una determinata esecuzione di un algoritmo, si elencano, una dopo l'altra, tutte le istruzioni da eseguire, nell'ordine di esecuzione. Tale lista di istruzioni verrà detta **sequenza di esecuzione** dell'algoritmo in esame.
L'esempio descrive una sola sequenza di esecuzione, ma ciò non è sempre vero; esistono casi, e sono i più frequenti, in cui uno stesso algoritmo può evocare più processi sequenziali differenti, a seconda delle condizioni iniziali.
Si osservi ad esempio la seguente versione dell'algoritmo precedente, che tiene conto della possibilità che la merce in esame non sia soggetta a IVA:

Se la merce da fatturare è soggetta a IVA
    allora
        cerca la corretta aliquota IVA sulla tabella
        moltiplica l'importo per l'aliquota trovata
        somma il risultato all'importo netto
    altrimenti
        tieni conto solo dell'importo di partenza
fine se

In questo caso il processo evocato non è fisso, ma dipende dai dati da elaborare, in particolare dal tipo di merce da fatturare. L'algoritmo infatti descrive un insieme costituito da due sequenze di esecuzione diverse.
Esistono tuttavia situazioni ancora più complesse, in cui il numero di sequenze di esecuzione descritte da uno stesso algoritmo non è noto a priori. Ad esempio, consideriamo un algoritmo per effettuare una telefonata:

Solleva il ricevitore
Componi il numero
se qualcuno risponde
    allora
        conduci la conversazione
    altrimenti
        deponi il ricevitore
ripeti l'intero procedimento
finché qualcuno risponderà

Questo algoritmo descrive due diverse sequenze di esecuzione a seconda se l'interlocutore risponde al telefono al primo tentativo o no. L'algoritmo descrive più sequenze di esecuzione, corrispondenti al fatto che la telefonata riesca al secondo, terzo, ... tentativo.
Si potrà avere un processo ciclico che avrà termine quando l'interlocutore risponderà al telefono, o potrà anche non terminare se non risponderà mai, e in questo caso l'algoritmo non risponde ai requisiti richiesti dalla nostra definizione.
Diversi sono quindi i costrutti linguistici che possono essere utilizzati allo scopo di descrivere un insieme di sequenze di esecuzione.
La formalizzazione dell'algoritmo e della struttura di un programma deve essere compiuta utilizzando opportuni standard di utilizzo comune, che ne consentono un'agevole traduzione in un programma. Gli algoritmi devono essere espressi in modo estremamente chiaro e inequivocabile. Tra i formalismi, due in particolare hanno largo utilizzo: la **pseudocodifica** e il **diagramma di flusso**.
La pseudocodifica utilizza un linguaggio il più possibile vicino a quello naturale scritto e impiega l'**indentazione**, una tecnica che prevede il rientro di gruppi di istruzioni così da evidenziare i limiti di cicli ripetitivi di istruzioni o di alternative possibili.
La pseudocodifica presenta notevoli vantaggi, tra i quali si può ricordare la possibilità di redigerla utilizzando un normale programma di videoscrittura e la facilità estrema di traduzione in un linguaggio di programmazione strutturato.

Il **diagramma di flusso** o **diagramma a blocchi** (**flow-chart**) è una rappresentazione grafica della sequenza di operazioni da compiere, cioè dell'algoritmo, per risolvere un certo problema.
Esso rappresenta in forma schematica la logica e la struttura fondamentale del programma. Poiché il diagramma serve di solito come base per la stesura del programma, deve essere sufficientemente dettagliato.
Per ottenere diagrammi di flusso si usa un numero limitato di simboli grafici, ognuno dei quali serve a distinguere specifiche operazioni diverse; questi simboli sono uniti da segmenti orientati che rappresentano la sequenza con cui devono essere eseguiti.

2.4 I linguaggi di programmazione

L'ampia gamma delle applicazioni a cui può essere dedicato un calcolatore è dovuta all'interazione tra la sua parte hardware, circuiti e dispositivi, e la sua parte software, i programmi.
La generale architettura su cui un calcolatore viene progettato e costruito e i sistemi di codifica che permettono di trattare a livello simbolico i suoi meccanismi fanno di esso uno strumento assai versatile, impiegabile nella soluzione di problemi di varia natura: calcolo scientifico, gestione di grandi banche di dati, controllo di processi.
E' comunque a carico della sua parte software che va il merito del calcolatore di prestarsi ad una grande varietà di ambiti applicativi.
Se dunque il software gioca un ruolo così importante, la possibilità di sviluppare, verificare e mantenere i programmi diventa un'attività di ordine primario: il linguaggio macchina non si presta allo sviluppo rapido e facilmente controllabile dei programmi.
Nell'evoluzione del software e degli strumenti di supporto all'attività della programmazione, sono stati messi a punto linguaggi di programmazione che si sono svincolati dalla natura hardware sottostante e che sempre di più mediano la loro struttura logico-linguistica con alcune caratteristiche del linguaggio naturale, più vicino al modo di esprimersi dell'utente.
Mentre i primi calcolatori erano difficili da usare e richiedevano l'assistenza di personale altamente specializzato, nel tempo i costruttori hanno sviluppato sistemi via via più semplici da usare anche da persone con scarsa cultura informatica.
La chiave di questa conquista è ancora il software: lo sviluppo di programmi dedicati alla gestione delle risorse della macchina ha reso l'interazione uomo/macchina molto "amichevole" (user friendly).

2.5 Il linguaggio ASSEMBLER

Per superare la difficoltà di scrivere un programma in linguaggio macchina, cioè come una sequenza di numeri, già nei primi calcolatori si ricorse all'espediente di definire in modo simbolico le istruzioni, gli indirizzi dei dati e i dati stessi.
Ebbe origine così il primo linguaggio di programmazione: l'**ASSEMBLER**.
Dato che i calcolatori differiscono sia per l'insieme delle istruzioni che per il formato dei dati, il linguaggio ASSEMBLER non è universale: ogni tipo di macchina ne ha uno proprio.
Le istruzioni elementari binarie del linguaggio macchina sono implementate nell'ASSEMBLER sotto forma di nomi simbolici:

ADD	somma
SUB	sottrai
MOVE	copia
ecc.	

Pur rappresentando un sicuro miglioramento rispetto alla codifica binaria, la programmazione in ASSEMBLER resta comunque un'impresa che richiede a chi la compie un'approfondita conoscenza dell'hardware.
Inoltre i programmi ottenuti contengono moltissime istruzioni e pertanto sono di lunga e complessa redazione, correzione e manutenzione.

Ogni microprocessore possiede il suo specifico linguaggio ASSEMBLER e risulta di fatto impraticabile la portabilità di uno stesso programma su calcolatori che utilizzano processori diversi. Oltre alla scomodità di scrittura dei programmi, si incontrano difficoltà anche nella gestione dei dispositivi di I/O e infine non è possibile definire e gestire strutture complesse di dati, quali ad esempio le matrici.
Sebbene largamente usato nel passato, l'ASSEMBLER oggi è usato solo nei pochissimi casi in cui è necessario realizzare un'interazione con l'elaboratore a livello molto stretto, ad esempio programmi controller di dispositivi di I/O.

2.6 I linguaggi di alto livello

Per superare le numerose difficoltà descritte prima, nella seconda metà degli anni '50 si è cominciato a realizzare linguaggi simbolici detti **linguaggi di alto livello**, caratterizzati da istruzioni più potenti: a ciascuna di esse possono corrispondere più istruzioni in linguaggio macchina o in ASSEMBLER.
Questi linguaggi permettono di trattare con facilità formule matematiche, strutture complesse di dati (ad esempio matrici), operazioni di input-output; inoltre mettono a disposizione strutture logiche particolari, come procedure, costrutti condizionali, ecc., le quali permettono di scrivere i programmi velocemente e con minor rischio di errori.
La codifica è abbastanza immediata grazie all'uso di termini intuitivi presi dalla lingua inglese:

READ
WRITE
IF...THEN...ELSE
FOR...DO...
.....

e alla possibilità di usare nomi significativi per i dati da manipolare (non A, B, ..., bensì SOMMA, CONTATORE,....). I programmi sono inoltre più leggibili e documentabili.
Un programma scritto con un linguaggio di alto livello, per poter essere eseguito dal calcolatore, deve essere tradotto in linguaggio macchina. Di questo compito si occupa un particolare programma, il **compilatore** oppure l'**interprete**, capace di tradurre le istruzioni complesse del linguaggio di alto livello (formule matematiche, strutture di dati, operazioni di I/O dei dati, ecc.) in sequenze di istruzioni e definizioni in linguaggio macchina.
I linguaggi di alto livello più noti sono i seguenti:

• **FORTRAN (FORmula TRANslator)**: sviluppato negli anni '50 in ambito scientifico, è particolarmente adatto ad applicazioni scientifico-matematiche. La versione standard più recente è FORTRAN 90.
Il FORTRAN è nato soprattutto per rispondere alle esigenze di ricercatori e progettisti, ed è un linguaggio molto usato negli ambienti universitari; in FORTRAN sono state sviluppate molte **librerie** di programmi standard che rendono più semplice per gli utenti la scrittura dei programmi offrendo la professionalità e la competenza di altri programmatori più esperti. Un esempio significativo è rappresentato dalla libreria **NAG (Numerical Algorithm Group)**, una raccolta molto grande e completa di **sottoprogrammi** per le applicazioni di analisi numerica. Un'altra importante raccolta di sottoprogrammi è la libreria **SPSS (Statistical Package for the Social Sciences)**, che permette agli studiosi di statistica di analizzare i dati risultanti da indagini sociologiche.

· **COBOL (Common Business Oriented Language)**: sviluppato alla fine degli anni '50, è adatto a trattare problemi commerciali e gestionali, nei quali i calcoli numerici sono scarsi e assume notevole importanza l'aspetto di presentazione dei risultati.

· **BASIC (Beginner's All-Purpose Symbolic Instruction Code)**: sviluppato nel 1963, ha dato origine a moltissimi dialetti. Fu concepito come linguaggio semplice e si è diffuso molto a livello dilettantistico.
Permette di manipolare stringhe (insiemi di caratteri posti uno dopo l'altro) e di risolvere problemi numerici usando matrici. E' un linguaggio poco adatto allo sviluppo di procedure complesse in quanto i programmi scritti in questo linguaggio sono di difficile manutenzione.

• **PASCAL**: sviluppato nel 1970 presso l'università di Zurigo, è stato il primo linguaggio a incorporare in modo coerente i concetti di programmazione strutturata.
Permette un agevole uso delle strutture di matrici e vettori e delle chiamate ricorsive.
E' ampiamente accettato come un linguaggio facile da implementare e particolarmente adatto a scopi didattici e scientifici.

• **C++** : sviluppato nei primi anni '70, è stato impiegato per la stesura del nucleo del sistema operativo UNIX. E' strutturato, potente, ricco; è uno dei linguaggi più usati al momento, benché non sia di facile impiego.

Le caratteristiche comuni dei linguaggi di programmazione di alto livello sono le seguenti:
– quasi tutti i linguaggi definiscono le operazioni da eseguire usando parole in lingua inglese;
– il problema viene definito in un modo che non richiede al programmatore una particolare esperienza sulle procedure seguite dal computer;
– l'organizzazione della memoria riservata al programma e ai dati in genere non dipende dal programmatore, che si limita a dare un nome da lui scelto a locazioni di memoria astratte che vengono automaticamente, attraverso il sistema operativo, assegnate a locazioni realmente esistenti nella memoria del computer;
– lo stesso programma può essere eseguito su diversi tipi di computer senza necessitare di cambiamenti significativi.

3. Algoritmi fondamentali

3.1 Introduzione

Prendiamo in esame in questo capitolo alcuni degli algoritmi fondamentali, che vengono utilizzati per costruire procedure di calcolo molto più vaste.
Tra questi meccanismi fondamentali uno dei più largamente usati è lo scambio dei valori associati a due variabili; in particolare questa operazione è ampiamente adoperata negli algoritmi di ordinamento.
Anche l'idea del conteggio è parte essenziale di molte procedure di calcolo più elaborate: un'estensione dell'idea del conteggio sta alla base del meccanismo molto diffuso per la sommatoria dei dati di un insieme.
Molto importanti sono anche i metodi di fattorizzazione, i più noti dei quali si basano sull'uso dei massimi comuni divisori e tecniche collegate, e gli algoritmi di ordinamento.
L'array (vettore e matrice) è uno strumento ampiamente usato in programmazione: gli array servono ad immagazzinare ed organizzare i dati nella memoria di un calcolatore: la loro importanza deriva soprattutto dal fatto che forniscono un modo molto semplice ed efficace di eseguire e fare riferimento a calcoli su insiemi di dati che condividono caratteristiche comuni.
Il trattamento degli array è semplificato dall'uso di variabili per specificare gli indici: si fa riferimento ad una particolare locazione dell'array specificando il nome dell'array e l'indice della locazione: ad esempio *a(i)* indica l'*i*-esima locazione del vettore di nome *a; A(i,j)* indica l'elemento della matrice *A* posto all'incrocio fra l'*i*-esima riga e la *j*-esima colonna.
Gli array semplificano l'implementazione di algoritmi che devono rappresentare gli stessi calcoli su insiemi di dati. Il contenuto di una locazione dell'array può essere cambiato, mediante calcolo o assegnazione diretta, ed usato in espressioni come una singola variabile.
Per descrivere gli algoritmi viene usata una **pseudocodifica** basata sulle seguenti convenzioni e strutture:
- **Tipi di dati**: la maggior parte delle variabili sono numeri interi o reali oppure array, vettori e matrici; per indicare il generico elemento di un vettore *v* o di una matrice *a* si usano le notazioni *v(i)* o *a(i,j).*
- **Struttura a blocchi**: si usa l'indentazione per mettere in evidenza i blocchi di istruzioni.
- **Istruzione di assegnazione** : per l'operatore di assegnazione si usa il simbolo = .
- **Struttura sequenziale** : una sequenza di istruzioni viene scritta su righe separate.
- **Struttura condizionale** : ha la forma

```
if < condizione > then
        < blocco istruzioni A >
else
        < blocco istruzioni B >
end if
```

Se al momento dell'esecuzione la condizione è vera si esegue la sequenza A, altrimenti si esegue la sequenza B.
L'alternativa **else** può essere omessa se è vuota; si possono usare strutture condizionali annidate.
- **Strutture ripetitive** : nel caso in cui il numero di iterazioni sia indefinito si usa il ciclo **while** .

Il ciclo **while** ha la forma

```
while < condizione > do
        < blocco istruzioni S >
end while
```

Se la condizione è vera si esegue il blocco di istruzioni S; poi si ripete il test sulla condizione: se è falsa il ciclo termina, altrimenti si ripete il blocco di istruzioni S. Il processo continua finché la condizione diventa falsa.
All'uscita da un ciclo **while** la condizione logica sarà necessariamente falsa.

Se il numero di iterazioni è noto e definito prima dell'inizio del ciclo stesso, si può usare la struttura ripetitiva **for** controllata da contatore, che ha due forme a seconda che il contatore incrementi o decrementi il suo valore:

1 – **for** $i = n$ **to** m **do**
< blocco istruzioni >
end for

2 – **for** $i = m$ **downto** n **do**
< blocco istruzioni >
end for

che equivalgono rispettivamente a:

1 – $i = n$
while $i \leq m$ **do**
< blocco istruzioni >
$i = i + 1$
end while

2 – $i = m$
while $i \geq n$ **do**
< blocco istruzioni >
$i = i - 1$
end while

- **Istruzioni di input-output (ingresso - uscita):** si sintetizzano nella forma
  Input {lista di ingresso}
  Output {lista di uscita}
- **Fine dell'algoritmo**: si indica con l'istruzione
  End

3.2 Algoritmi di base

Algoritmo 1
Scambio dei valori di due variabili

Problema
Assegnate due variabili, *a* e *b*, scambiare i loro valori.

Sviluppo dell'algoritmo
Il problema di scambiare tra loro i valori associati a due variabili coinvolge un meccanismo fondamentale che si presenta in molti algoritmi di ordinamento e di manipolazione dei dati.

Esempio
La locazione di memoria o variabile *a* contiene il numero 127 e la locazione di memoria o variabile *b* contiene il numero 364.
Configurazione iniziale:

a	*b*
127	364

Lo scopo è raggiungere la configurazione finale:

a	*b*
364	127

Per modificare il valore di una variabile si può utilizzare l'operatore di assegnazione: si potrebbero eseguire le assegnazioni seguenti:

$a = b$
$b = a$

e in tal caso si otterrebbero le configurazioni successive seguenti:

a	*b*	
127	364	configurazione iniziale
364	364	dopo l'assegnazione $a = b$
364	364	dopo l'assegnazione $b = a$

La configurazione finale non è corretta, perché è andato perso il valore 127 assunto inizialmente da *a*. Per evitare di perdere il valore iniziale di *a* si introduce una variabile temporanea *t* e si copia in *t* il valore di *a* prima di eseguire l'assegnazione $a = b$.
Le assegnazioni corrette da fare sono:

$t = a$
$a = b$
$b = t$

Le configurazioni che si ottengono sono:

a	*t*	*b*	
127		364	configurazione iniziale
127	127	364	dopo l'assegnazione $t = a$
364	127	364	dopo l'assegnazione $a = b$
364	127	127	dopo l'assegnazione $b = t$ configurazione finale

```
Input {a,b}
t = a
a = b
b = t
Output {a,b}
End
```

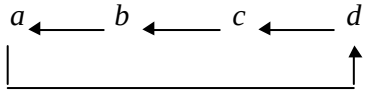
Esercizi

1. Scrivere un algoritmo per effettuare lo scambio fra le variabili *a* e *b* senza utilizzare la variabile temporanea *t*.

```
Input {a,b}
a = a + b
b = a - b
a = a - b
Output {a,b}
End
```

Questo algoritmo non è efficiente, poiché oltre alle assegnazioni richiede 3 addizioni, che non sono necessarie nell'algoritmo dell'esercizio 1.

2. Scrivere un algoritmo per eseguire lo scambio



Input {*a,b,c,d*}
t = *a*
a = *b*
b = *c*
c = *d*
d = *t*
Output {*a,b,c,d*}
End

Algoritmo 2
Conteggio

Problema
Dato l'insieme dei risultati d'esame, nell'intervallo da 0 a 30, di *n* studenti, contare il numero di studenti che hanno superato la prova; l'esame si intende superato con un voto ≥ 18.

Sviluppo dell'algoritmo
I meccanismi di conteggio sono usati molto spesso negli algoritmi per computer: di solito occorre contare il numero degli elementi di un insieme che possiedono una certa proprietà o soddisfano una o più condizioni.

Esempio
Insieme dei voti di 7 studenti

19, 15, 24, 21, 10, 28, 27.

Esaminiamo successivamente i voti, confrontandoli con 18 e modifichiamo il contatore dei promossi se il voto è ≥ 18.

	Voti	Evoluzione del contatore dei promossi	
↓	19	contatore precedente = 0	contatore corrente = 1
	15	contatore precedente = 1	contatore corrente = 1
Ordine con cui si	24	contatore precedente = 1	contatore corrente = 2
esaminano i voti	21	contatore precedente = 2	contatore corrente = 3
	10	contatore precedente = 3	contatore corrente = 3
↓	28	contatore precedente = 3	contatore corrente = 4
	27	contatore precedente = 4	contatore corrente = 5

Dopo che ogni voto è stato elaborato, il contatore corrente riproduce il numero di studenti promossi incontrati fino a quel punto nella lista dei voti.
I passi essenziali dell'algoritmo sono:
1 – leggere il prossimo voto;
2 – se il voto soddisfa la condizione di promozione, incrementare di 1 il contatore, finché ci sono voti da esaminare.
Prima che venga esaminato il primo voto il contatore deve valere 0; alla fine avrà come valore il numero dei promossi.
I voti vengono memorizzati in un vettore *v* di dimensione *n* (numero studenti).

```
Input{n, (v(i), i =1,n)}
cont = 0
i = 1
While  i ≤ n   do
                If      v(i) ≥ 18        then
                        cont = cont + 1
                end if
                i = i + 1
end while
Output{"numero promossi = ", cont}
End
```

Esercizi

1. Dato un insieme di *n* numeri reali, scrivere un algoritmo per contare il numero di quelli negativi, positivi e nulli.

Gli *n* numeri vengono memorizzati in un vettore *v* di dimensione *n*; in uscita le variabili *contneg, contpos, contzero* contengono rispettivamente il numero dei valori negativi, positivi e nulli.

```
Input {n, (v(i), i = 1,n)}
contneg = 0
contpos = 0
i = 1
While  i ≤ n   do
                If  v(i) < 0   then
                        contneg = contneg + 1
                else
                        If   v(i) > 0  then
                                contpos = contpos + 1
                        end if
                end if
                i = i + 1
end while
contzero = n - (contneg + contpos)
Output {contneg, contpos, contzero}
End
```

2. Dato un insieme di *n* numeri interi, scrivere un algoritmo per contare quanti fra essi sono divisibili per 3.

I numeri sono memorizzati in un vettore *v* di dimensione *n*.
Per stabilire se un numero *n* è divisibile per 3 si usa la funzione *mod* che restituisce il resto della divisione fra due numeri interi: se *n mod* 3 = 0 allora il numero *n* è divisibile per 3.

```
Input {n, (v(i), i = 1 ,n)}
cont = 0
For   i = 1  to  n   do
                If    v(i) mod 3 = 0    then
                        cont = cont + 1
                end if
end for
Output { cont}
End
```

Algoritmo 3
Sommatoria di un insieme di numeri

Problema

Dato un insieme di n numeri $\{a_1, a_2,..., a_n\}$, calcolare la somma $\sum_{i=1}^{n} a_i$.

Sviluppo dell'algoritmo

I valori dei numeri da sommare sono memorizzati in un vettore a di n elementi.
La somma s di n numeri viene generata iterativamente; il generico passo del procedimento è del tipo

$$s = s + a(i).$$

Il valore di s deve essere inizializzato assegnando il valore $s = 0$, prima di iniziare il ciclo. Diamo due versioni dello stesso algoritmo, la prima con l'uso della struttura ripetitiva **while,** la seconda con l'uso della forma **for** $i = 1$ **to** n **do**, che è più indicata quando la condizione da verificare per concludere il ciclo è che un certo indice raggiunga un valore fissato.

```
Input{n,(a(i), i = 1, n)}
s = 0
i = 1
While   i ≤ n   do
                s = s + a(i)
                i = i + 1
end while
Output{"somma = ", s}
End
```

```
Input{n,(a(i), i = 1, n)}
s = 0
For  i = 1  to  n  do
                s = s + a(i)
end for
Output{"somma = ", s}
End
```

Note

1 – Per sommare n numeri occorre eseguire $n - 1$ addizioni; il presente algoritmo ha invece usato n addizioni per consentire un'implementazione più semplice e pulita.
2 – Inizialmente, e ad ogni passaggio nel corso del ciclo, la variabile s rappresenta la somma dei primi i numeri. Al termine, quando $i = n$, s rappresenta la somma di tutti gli n numeri.
3 – Lo schema utilizzato non tiene conto dell'aritmetica floating-point (aritmetica finita) usata dal calcolatore e quindi non dà garanzie di accuratezza sul risultato.
L'algoritmo può produrre risultati non accurati se i numeri da sommare vengono sommati in ordine casuale o se alcuni di essi sono molto maggiori del risultato finale.
Conviene usare algoritmi più complicati, ma che danno risultati migliori: se i numeri da sommare sono positivi, prima di sommarli è necessario ordinarli in modo crescente; dovendo poi sommare numeri positivi e numeri negativi, conviene sommare separatamente tutti i positivi, dopo averli ordinati in modo crescente, e tutti i valori assoluti dei numeri negativi, anch'essi ordinati in modo crescente, e poi calcolare la differenza tra i due risultati ottenuti. Se le somme dei positivi e dei valori assoluti dei negativi sono di ordine di grandezza sensibilmente diverso, il risultato finale sarà accurato; se invece tali somme saranno all'incirca uguali, allora potrà sorgere il problema della

cancellazione numerica (perdita di cifre significative dovuta alla sottrazione fra numeri circa uguali).

Esercizi

1. Scrivere un algoritmo per calcolare la somma di n numeri con $n - 1$ addizioni (vedi nota 1, pag. 29)

```
Input{n,(a(i), i = 1, n)}
s = a(1)
For  i = 2  to  n  do
                s = s + a(i)
end for
Output{"somma = ", s}
End
```

2. Scrivere un algoritmo per calcolare la media campionaria (media aritmetica) e la varianza campionaria di n valori x_i dati

$$\text{media} = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \text{varianza} = s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

```
Input{n,(x(i), i = 1, n)}
s = 0
For  i = 1  to  n  do
                s = s + x(i)
end for
media = s/n
s = 0
For  i = 1  to  n  do
                s = s + (x(i) - media)**2
end for
varianza = s/(n - 1)
Output{"media  = ", media, "varianza  = ",varianza}
End
```

3. Scrivere un algoritmo per calcolare la media armonica di n valori x_i dati

$$h = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{x_i}$$

```
Input{n, (x(i), i = 1, n)}
s = 0
For  i = 1  to  n  do
                s = s + 1/x(i)
end for
h = s/n
Output {h}
End
```

4. Scrivere un algoritmo per il calcolo della somma dei primi n numeri interi

$$S = 1+2+3+....+n = \sum_{i=1}^{n} i$$

```
Input {n}
s = 0
For i = 1 to n do
        s = s + i
end for
Output {s}
End
```

Questo algoritmo non è efficiente, poiché richiede n operazioni di addizione; usando la formula di Gauss

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

si ottiene un algoritmo più efficiente, in quanto richiede solo 3 operazioni: 1 addizione e 2 moltiplicazioni/divisioni.

```
Input {n}
s = (n*(n+1))/2
Output {s}
End
```

5. Scrivere un algoritmo per calcolare la somma dei primi n numeri pari e dei primi n numeri dispari

$$S_p = 2+4+6+....+2n = \sum_{i=1}^{n} 2i$$

$$S_d = 1+3+5+....+(2n-1) = \sum_{i=1}^{n} (2i-1)$$

Applicando la formula di Gauss si ottiene

$$S_p = \sum_{i=1}^{n} 2i = 2\sum_{i=1}^{n} i = 2\frac{n(n+1)}{2} = n(n+1)$$

$$S_d = \sum_{i=1}^{n} (2i-1) = = \sum_{i=1}^{n} 2i - \sum_{i=1}^{n} 1 = 2\sum_{i=1}^{n} i - n = 2\frac{n(n+1)}{2} - n = n^2$$

6. Scrivere un algoritmo per il calcolo della ridotta della serie geometrica di ragione a

$$S = \sum_{i=0}^{n} a^i \qquad a \neq 1$$

Si usa la formula

$$S = \sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$$

7. Scrivere un algoritmo per il calcolo della somma degli elementi di una matrice A di dimensioni $n \times m$.

```
Input {n, m,  (A(i,j),  i=1,n  ,j=1,m)}
sum = 0
For  i = 1  to  n  do
                For  j = 1  to  m  do
                        sum = sum + A(i,j)
                end for
end for
Output {sum}
End
```

8. Scrivere un algoritmo per il calcolo del prodotto di n numeri.

I valori dei numeri da sommare sono memorizzati in un vettore a di n elementi.
Il prodotto p di n numeri viene generato iterativamente; il generico passo del procedimento è del tipo

$$p = p \times a(i).$$

Il valore di p deve essere inizializzato assegnando il valore $p = 0$, prima di iniziare il ciclo.

```
Input{n,(a(i), i = 1, n)}
p = 1
For  i = 1  to  n  do
                p = p * a(i)
end for
Output{"prodotto = " p}
End
```

Algoritmo 4
Calcolo del fattoriale

Problema
Dato il numero intero $n \geq 0$, calcolare $n!$.

Sviluppo dell'algoritmo
Definizione di $n!$:

$$n! = 1 \cdot 2 \cdot 3 \ \ldots\ldots \cdot (n-1) \cdot n \qquad n \geq 1 \qquad\qquad 0! = 1$$

$$n! = (n-1)! \cdot n \qquad\qquad n \geq 1$$

Partendo da $nfat = 1$, si può procedere iterativamente con un ciclo; all' i-esima iterazione si calcola $nfat = nfat * i$ e a tale iterazione il valore di $nfat$ è $i!$. Al termine del ciclo, quando $i = n$, $nfat$ contiene $n!$
Diamo due versioni dello stesso algoritmo, la prima con l'uso della struttura ripetitiva **while**, la seconda con l'uso della struttura **for**. In questi algoritmi non viene fatta alcuna considerazione circa la dimensione finita dei numeri che possono essere rappresentati nel computer (possibilità di overflow).

```
Input{n}
nfat = 1
While  n > 1 do
                nfat = nfat *n
                n = n - 1
end while
Output{"n fattoriale =", nfat}
End
```

```
Input{n}
nfat = 1
If      n > 1   then
                For   i = 2   to  n  do
                        nfat = nfat * i
                end for
end if
Output{"n fattoriale =", nfat}
End
```

Esercizi

1. Dato il numero intero $n \geq 0$, scrivere un algoritmo per calcolare

$$\frac{1}{n!} = \frac{1}{(n-1)!} \cdot \frac{1}{n}$$

```
Input{n}
rfat = 1
While  n > 1  do
                rfat = rfat / n
                n = n - 1
end while
Output{ rfat }
End
```

2. Scrivere un algoritmo per il calcolo della somma

$$S_n = 0! + 1! + 2! + 3! + ... + n! \qquad n \geq 0$$

```
Input {n}
s = 1
fat = 1
If    n ≥ 1   then
                For   k = 1  to  n   do
                        fat = fat * k
                        s = s + fat
                end for
end if
Output  {s}
End
```

3. Scrivere un algoritmo per il calcolo del numero e per mezzo della somma

$$e = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + ... + \frac{1}{n!} \qquad n \geq 0$$

```
Input {n}
s = 1
fat = 1
If   n ≥ 1   then
                For   k = 1  to  n   do
                        fat = fat * k
                        s = s + 1 / fat
                end for
end if
Output  {s}
End
```

Algoritmo 5
Generazione di una successione di Fibonacci

Problema

Generare e stampare i primi *n* termini della successione di Fibonacci. I primi termini sono

0, 1, 1, 2, 3, 5, 8, 13,...

Ciascun termine dopo i primi due è ottenuto sommando i suoi due predecessori più vicini.

Sviluppo dell'algoritmo

Dalla definizione abbiamo

termine corrente = somma dei due termini precedenti.

Definiamo

a, b: termini precedenti *c*: termine corrente

Per inizializzare il procedimento avremo

a = 0 1° numero di Fibonacci (per definizione)
b = 1 2° numero di Fibonacci (per definizione)
c = *a* + *b* 3° numero di Fibonacci

Ora bisogna che i due termini precedenti vengano aggiornati per generare il 4° numero, e ciò avviene con le assegnazioni successive

a = *b*
b = *c*

Dopo queste due assegnazioni, possiamo generare il 4° numero. Un modo per farlo è tornare al passo

c = *a* + *b*

Per non perdere i primi due numeri con le successive assegnazioni, bisogna stamparli prima di calcolare il terzo; dal terzo in poi i numeri vengono stampati appena calcolati.

```
Input {n}
a = 0
b = 1
Output {a,b}
For  i = 3  to  n  do
                c = a + b
                Output {c}
                a = b
                b = c
end  for
End
```

Esercizi

1. Generare la successione di Lucas: i primi numeri della successione di Lucas, che è una variazione della successione di Fibonacci, sono

1 3 4 7 11 18 29 .....

```
Input {n}
a = 1
b = 3
Output {a,b}
For i = 3 to n do
        c = a + b
        Output {c}
        a = b
        b = c
end for
End
```

2. Siano $a = 0$, $b = 1$ e $c = 1$ i primi tre numeri di una data successione. Tutti gli altri numeri della sequenza sono generati a partire dalla somma dei loro tre predecessori più recenti. Scrivere un algoritmo per generare questa successione.

```
Input {n}
a = 0
b = 1
c = 1
Output {a,b,c}
        d = a + b + c
        Output {d}
        a = b
        b = c
        c = d
end for
End
```

3. Generare la successione in cui ciascun elemento è la somma di fattoriali adiacenti:

$0! + 1!$ $1! + 2!$ $2! + 3!$ $3! + 4!$ ..... $(n - 1)! + n!$ ....

```
Input {n}
a = 1
b = 1
i = 1
While i ≤ n do
        f = a + b
        Output {f}
        i = i + 1
        a = b
        b = b * i
end while
End
```

Algoritmo 6
Il minimo divisore di un intero

Problema
Assegnato un intero *n*, trovare un algoritmo che determini il suo divisore esatto più piccolo, diverso da 1, ossia il più piccolo fattore primo di *n* (si ricordi che il numero 1 non è primo).

Sviluppo dell'algoritmo
A prima vista questo problema appare abbastanza banale. Si può dividere il numero *n* per ciascuno dei numeri

2, 3, 4, ..., *n*;

appena si incontra un numero che è divisore esatto di *n*, l'algoritmo può concludersi.
E' possibile apportare dei miglioramenti all'algoritmo.

Esempio

n = 36
divisori: 2, 3, 4, 6, 9, 12, 18 (escludendo 36)

Al divisore 3 corrisponde il divisore 12

$\frac{36}{3}=12$ e $\frac{36}{12}=3$

Nell'insieme dei divisori si hanno le coppie seguenti:

fattore più piccolo		fattore più grande
2	è associato a	18
3	è associato a	12
4	è associato a	9
6	è associato a	6

Il divisore minimo è associato al massimo, il più piccolo successivo è associato al più grande immediatamente inferiore, ecc.
Seguendo questo ragionamento concludiamo che l'algoritmo può terminare quando si è determinata la coppia di fattori *a, b* , con $a \leq b$, tali che:

a è il più grande dei fattori più piccoli
b è il più piccolo dei fattori più grandi
$a \times b = n$.

Il caso limite si ha quando *a* = *b*, ossia $a \times a = n$ (nell'esempio *a* = *b* = 6).
Ne segue che non è necessario ricercare divisori di *n* superiori alla sua radice quadrata.
Questa considerazione è particolarmente importante quando il numero *n* è un numero primo molto grande; il limite della radice quadrata ci consente di arrestare la ricerca molto prima rispetto alla richiesta di esaminare tutti i possibili numeri $\leq n$. Ad esempio per il numero primo 127, i divisori da considerare sono:

2, 3, 4, 5, 6, 7, 8, 9, 10, 11

essendo $\sqrt{127} \cong 11.3$.
Si osservi ancora che, se il numero è pari, è divisibile per 2 e in tal caso il minimo divisore cercato è 2.
Se il numero non è divisibile per 2, a maggior ragione non lo sarà per ogni altro numero pari. Perciò se il numero in esame è dispari, è sufficiente esaminare come divisori possibili i numeri dispari 3, 5, 7, ...
La struttura dell'algoritmo è la seguente:
1 – se il numero è pari, il divisore minimo è 2, altrimenti

a – si calcola la radice quadrata $\sqrt{n}$ e se ne prende la parte intera: *r = int(sqrt(n))*;
b – finché non si trova un divisore esatto minore della parte intera di r si continua a provare la divisione con il prossimo numero della successione 3, 5, 7, ...

I valori del divisore d da provare possono essere generati iniziando da $d = 3$ e poi incrementando con l'assegnazione $d = d + 2$.
Per stabilire se un numero è un divisore esatto di un altro oppure no si usa la funzione *mod*:

se $n \bmod d = 0$ allora d è un divisore esatto di n.

L'algoritmo ha termine quando sono verificate una o entrambe le condizioni

$n \bmod d = 0$ e $d \geq r$.

In chiusura occorre un controllo per verificare se n ha effettivamente un divisore, che è il valore corrente di d, oppure no, e in questo caso si pone $d = 1$; l'algoritmo restituisce quindi il valore 1 come divisore minimo, se il numero è primo.

```
Input {n}
If      n mod 2 = 0     then
                mindiv = 2
else
                r = int(sqrt(n))
                d = 3
                While  ((n mod d ≠ 0) and (d < r)) do
                        d = d + 2
                end while
                If      n mod d = 0     then
                                mindiv = d
                else
                                mindiv = 1
                end if
end if
Output {mindiv}
End
```

Algoritmo 7
Inversione dell'ordine di un vettore

Problema
Disporre gli elementi di un vettore in modo che essi risultino in ordine inverso rispetto all'ordine di partenza.

Sviluppo dell'algoritmo
Si vuole ottenere:

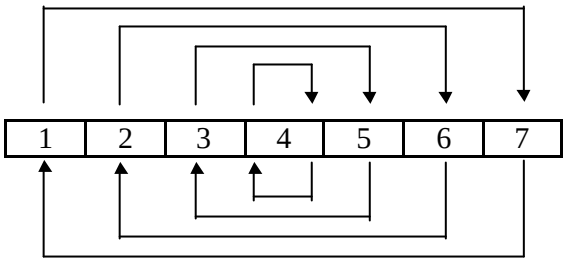
1	2	3	4	5	6	7

vettore a prima dell'inversione

7	6	5	4	3	2	1

vettore a dopo l'inversione

Il primo elemento va a finire nell'ultima posizione, il secondo nella penultima, ecc.; si arriva all'insieme di scambi:



In termini di indici gli scambi sono:

passo 1	$a(1)$	$\longleftrightarrow$	$a(7)$	
passo 2	$a(2)$	$\longleftrightarrow$	$a(6)$	
passo 3	$a(3)$	$\longleftrightarrow$	$a(5)$	
passo 4	$a(4)$	$\longleftrightarrow$	$a(4)$	nessuno scambio

dopo il passo 3 il vettore è completamente invertito.
Ogni scambio può essere eseguito con un meccanismo del tipo:

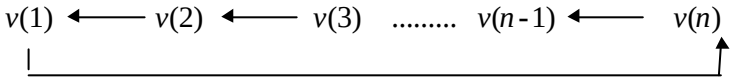
```
t = a(i)
a(i) = a(n - i + 1)
a(n - i + 1) = t
```

Dall'esempio si deduce anche che il numero r degli scambi da effettuare per permutare l'ordine del vettore è uguale alla parte intera di $n/2$ (se il procedimento di scambio continuasse per n passi, il vettore ritornerebbe nella configurazione di partenza).

```
Input {n , (a(i), i=1, n)}
r = int( n/2 )
For  i = 1  to  r  do
        t = a(i)
        a(i) = a(n - i + 1)
        a(n - i + 1) = t
end for
Output {a(i), i = 1 , n}
End
```

Esercizi

1. Scrivere un algoritmo per spostare di una posizione gli n elementi di un vettore v




```
Input {n, (v(i), i = 1, n)}
t = v(1)
For  i = 1  to  n - 1  do
        v(i) = v(i+1)
end for
v(n) = t
Output { v(i), i = 1, n}
End
```

Algoritmo 8
Ricerca del valore massimo di un insieme

Problema
Trovare il valore massimo di un insieme di n numeri.

Sviluppo dell'algoritmo
Il massimo di un insieme di numeri è quel numero che è maggiore o uguale a tutti gli altri numeri dell'insieme; questa definizione sottolinea il fatto che il massimo può non essere unico; implica inoltre che il massimo è definito solo per insiemi di uno o più elementi.
Esaminiamo la strategia che costituisce la base dell'algoritmo per il calcolatore.

Per stabilire il massimo, devono essere esaminati tutti i numeri dell'insieme. Il modo più semplice e sistematico di esaminare ogni valore di una lista è incominciare dal primo elemento della lista e analizzarla numero per numero, fino alla fine; ad ogni passo occorre operare un confronto.
Si memorizza il primo numero come se fosse il massimo in una variabile che chiamiamo *max*. Si confronta il massimo temporaneo con il secondo numero; se il secondo è minore o uguale a *max*, allora *max* non deve essere cambiato e si procede a confrontare *max* con il terzo numero; se invece il secondo numero è maggiore del massimo temporaneo, allora si deve aggiornare il massimo temporaneo, assegnando a *max* il secondo numero; quindi si procede confrontando il terzo numero con il nuovo massimo temporaneo.
L'intero processo deve continuare finché tutti i numeri dell'insieme non siano stati esaminati.
Man mano che si incontrano valori maggiori, essi assumono il ruolo di massimo temporaneo. Alla fine il massimo temporaneo è il massimo di tutto l'insieme.
Si può usare una struttura ripetitiva per esaminare tutti gli elementi dell'insieme, supponendo che si trovino in un vettore *a* di *n* elementi, con $n \geq 1$.

```
Input {n, (a(i), i = 1,n)}
max = a(1)
For  i = 2 to n do
                If      a(i) > max    then
                        max = a(i)
                end if
end for
Output{"massimo =", max}
End
```

Note
1 - Il numero di confronti che occorrono per trovare il massimo in un vettore di *n* elementi è *n* - 1;
2 - Per ogni *i* tale che $1 \leq i \leq n$, quando *i* elementi sono stati esaminati, la variabile *max* è maggiore o uguale di tutti gli elementi di indice compreso fra 1 e *i*.

Esercizi

1. Scrivere un algoritmo per trovare il minimo di un vettore *v* di *n* numeri e il numero di volte in cui tale minimo ricorre. Il vettore deve essere esaminato una sola volta.
La variabile *k* in uscita contiene il numero di volte in cui ricorre il minimo *min* .

```
Input {n, (v(i), i = 1,n)}
min = v(1)
k = 1
For  i = 2  to  n  do
                If  v(i) = min  then
                        k = k + 1
                else
                        If  v(i) < min  then
                                min = v(i)
                                k = 1
                        end if
                end if
end for
Output {min, k}
End
```

2. Scrivere un algoritmo per trovare la massima differenza in valore assoluto tra una coppia di elementi adiacenti di un vettore di *n* elementi.

```
Input{n, (v(i), i = 1,n)}
maxdif = 0
For  i = 1  to  n – 1  do
                d = abs(v(i + 1 ) – v(i))
                If  d > maxdif  then
                        maxdif = d
                end if
end for
Output{maxdif}
End
```

3.3 Ordinamento e ricerca

Le attività di **ordinamento (sorting)** e **ricerca (searching)** sono basilari in molti impieghi del computer. L'ordinamento fornisce un mezzo con cui organizzare l'informazione e facilitare il reperimento di dati; i metodi di ricerca sono progettati per trarre vantaggio dall'organizzazione dell'informazione, e pertanto ridurre lo sforzo necessario per individuare la posizione di un certo elemento o per stabilire che esso non è presente in un certo insieme di dati.
I due tipi più comuni di dati da organizzare sono i dati numerici e i dati stringa (insieme di caratteri posti uno di seguito all'altro).
Gli algoritmi di ordinamento organizzano i dati secondo una relazione d'ordine prestabilita.
La relazione d'ordine per i dati numerici è semplicemente di disporre gli elementi in sequenza dal più piccolo al più grande, cioè in ordine non decrescente.
I dati stringa sono generalmente ordinati secondo l'ordine alfabetico del dizionario.
Il problema dell'ordinamento è stato studiato estesamente dal punto di vista sia teorico che pratico.
Esistono numerosi algoritmi per riordinare le informazioni, che appartengono normalmente a una delle due classi seguenti:
1 – gli algoritmi più semplici sono caratterizzati dal richiedere circa n^2 confronti (ossia un numero di confronti che è $O(n^2)$ per ordinare *n* elementi);
2 – gli algoritmi avanzati, per ordinare *n* elementi richiedono circa $n \times log_2 n$ confronti (ossia $O(n \times log_2 n)$). Gli algoritmi di questa specie si avvicinano alla migliore prestazione possibile nell'ordinamento di dati del tutto casuali.
Confrontiamo tra loro n^2 e $n\ log_2 n$ al variare di *n* nella seguente tabella

n	n^2	$n\ log_2 n$
10	100	33.2
100	10 000	664.4
1000	1 000 000	9966
10 000	100 000 000	132 877

Chiaramente i metodi della seconda classe sono superiori agli altri, all'aumentare di *n*.
La superiorità di tali metodi è dovuta alla loro capacità di scambiare valori molto distanti fin dalle prime battute.
Si può dimostrare che per dati distribuiti casualmente, gli elementi devono essere spostati in media di una distanza *n*/3; i metodi più semplici (e meno efficienti) tendono a muovere gli elementi solo su piccole distanze, col risultato di dover eseguire molti più spostamenti prima di ottenere l'ordinamento definitivo.

Non c'è un metodo che sia il migliore per tutte le applicazioni: le prestazioni dei diversi metodi dipendono da vari parametri, come la quantità dei dati, il grado di ordine relativo già presente nei dati, la distribuzione dei valori dei dati.
Per esempio, se i dati sono già quasi ordinati, l'ordinamento a bolle (bubble sort), che è di solito il meno efficiente per dati disordinati, può dare risultati migliori di quelli di un metodo avanzato.

Algoritmo 9
Ordinamento per selezione (sort)

Problema
Dato un insieme di *n* numeri disposti casualmente, ordinarlo in modo non decrescente, utilizzando un ordinamento per selezione (sort).

Sviluppo dell'algoritmo
Consideriamo per esempio il vettore disordinato

20	35	18	8	14	41	3	39

vettore disordinato

a(1) ..... *a*(8)

Ci proponiamo di sviluppare un algoritmo che trasformi il vettore disordinato nella configurazione seguente

3	8	14	18	20	35	39	41

vettore ordinato

a(1) ..... *a*(8)

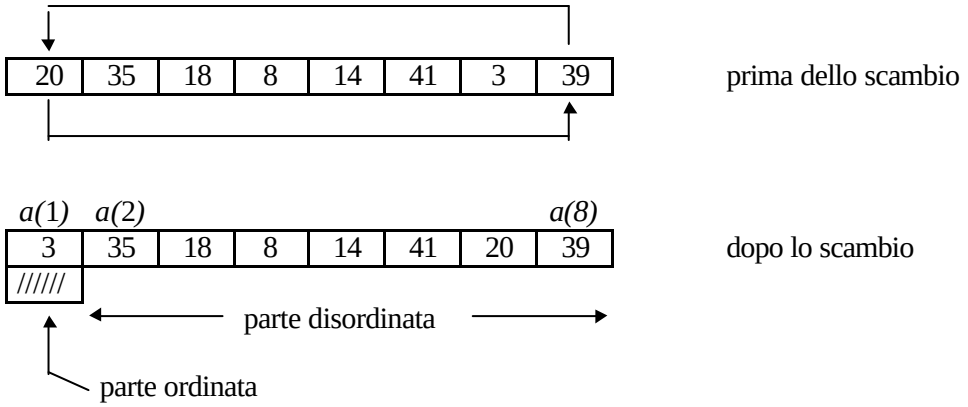
Confrontando il vettore ordinato e quello disordinato si vede che un modo per iniziare il processo di ordinamento può essere l'esecuzione dei due passi seguenti:
1 – ricercare l'elemento minimo del vettore disordinato;
2 – collocare tale elemento in *a*(1).
Per individuare l'elemento minimo si può impiegare il costrutto seguente:

```
min = a(1)
For  j = 2  to  n  do
             If      a(j) < min   then
                     min = a(j)
             end if
end for
```

Per portare il minimo in posizione 1 non basta l'assegnazione
a(1) = *min*
perché in tal modo andrebbe perso il contenuto di *a*(1), e si troverebbe la configurazione seguente (che è sbagliata):

3	35	18	8	14	41	3	39

Il contenuto di *a*(1) deve essere invece spostato nella posizione in cui stava il valore portato nella posizione 1, cioè si deve effettuare uno scambio:



Per attuare questo scambio occorre un meccanismo che non solo trovi il minimo, ma che ricordi anche la posizione del vettore in cui il minimo è collocato al momento; la variabile *p* serve a tale scopo.
Il codice precedente deve essere così modificato:

```
min = a(1)
p = 1
For  j = 2  to  n  do
                If      a(j) < min    then
                        min = a(j)
                        p = j
                end if
end for
a(p) = a(1)
a(1) = min
```

Con questo meccanismo è stato ordinato il primo elemento; la stessa strategia si può applicare per ricercare e collocare nel giusto ordine il minimo tra gli elementi rimasti; per individuarlo occorre ricominciare a scorrere il vettore dalla posizione 2.
I passi necessari sono perciò:

```
min = a(2)
p = 2
For  j = 3  to  n  do
                If      a(j) < min    then
                        min = a(j)
                        p = j
        end if
end for
a(p) = a(2)
a(2) = min
```

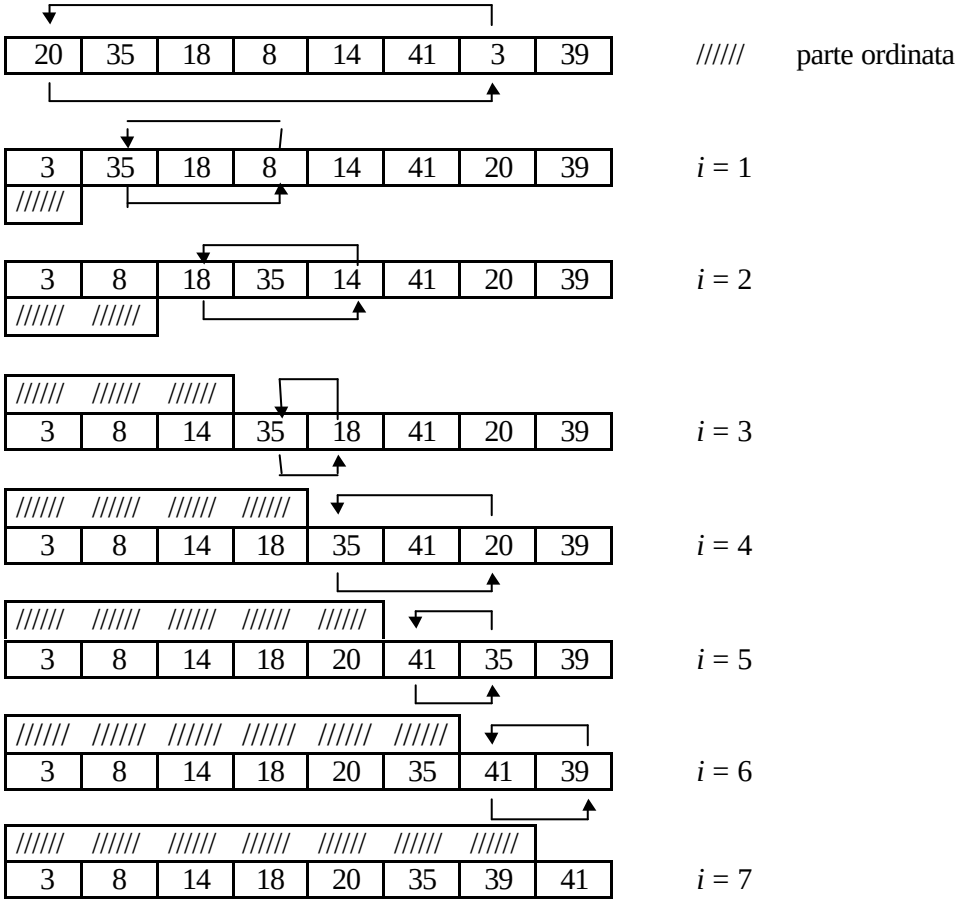
Si può estendere questo procedimento alla ricerca del più piccolo elemento dopo i primi due, dopo i primi tre, e così via.
Un modo concettualmente semplice per organizzare il procedimento consiste nel considerare il vettore suddiviso ad ogni passo in una parte ordinata e una disordinata.
Il meccanismo di selezione consiste nell'estendere ripetutamente di un elemento la parte ordinata, individuando l'elemento minimo successivo nella parte disordinata del vettore e collocandolo alla fine del vettore ordinato, ossia:

finché non sono stati ancora ordinati tutti gli elementi del vettore:

a - si individuano la posizione *p* e il valore *a(p)* dell'elemento minimo rimasto nel vettore non ordinato;

b - si scambia l'elemento *a(p)* con l'elemento in prima posizione nella parte di vettore non ordinato.

Nella figura seguente si illustra completamente l'ordinamento per l'esempio considerato.



Si osservi che dobbiamo percorrere la parte non ordinata del vettore solo $n - 1$ volte, poiché, dopo aver disposto $n - 1$ elementi in ordine, l'n-esimo deve essere per definizione il massimo e quindi è ordinato.

```
Input {n, (a(i), i = 1,n)}
For  i = 1  to  n - 1  do
                min = a(i)
                p = i
                For  j = i + 1  to  n  do
                        If      a(j) < min    then
                                min = a(j)
                                p = j
                        end if
                end for
                a(p) = a(i)
                a(i) = min
end for
Output{(a(i), i = 1,n)}
End
```

Nota
Un parametro importante è il numero di confronti effettuati; eseguendo il ciclo più interno, la prima volta sono effettuati n - 1 confronti, la seconda n - 2, la terza n - 3,..., l'ultima volta uno solo. Pertanto il numero di confronti è

$$N = (n - 1) + (n - 2) + (n - 3) + ... + 1 = \frac{n(n-1)}{2} \qquad \text{(formula di Gauss)}$$

Il numero di scambi è n -- 1, poiché è uguale al numero di volte che il ciclo più esterno è eseguito.

Algoritmo 10
Ordinamento a bolle (bubble sort)

Problema
Dato un insieme di n interi disposti a caso, ordinarlo in ordine non decrescente con un metodo di interscambio.

Sviluppo dell'algoritmo
Il metodo che consideriamo si basa su un meccanismo di interscambio di valori.

Esempio
Consideriamo ad esempio l'insieme di numeri casuali

↓	↓						
30	12	18	8	14	41	3	39

$a(1)$ ..... $a(8)$

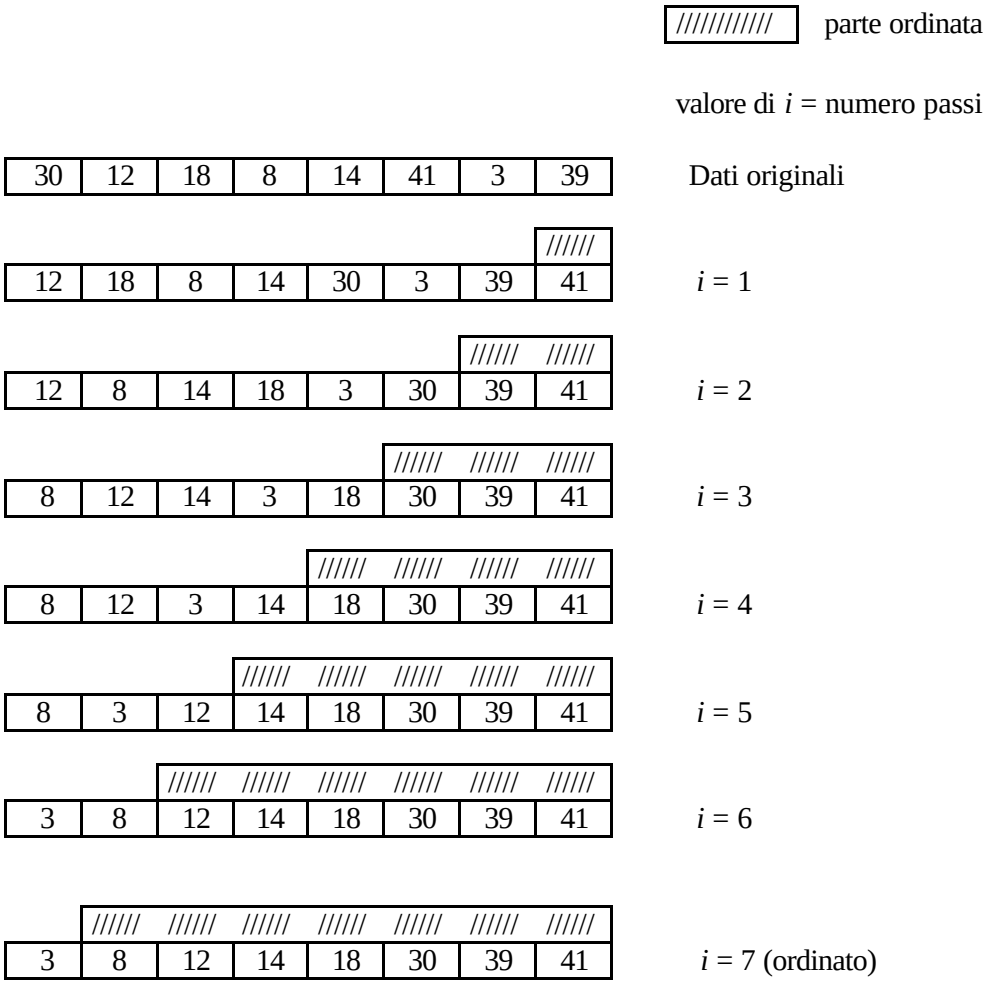
Ciò che si cerca nell'ordinamento è aumentare l'ordine dei numeri: i primi due elementi sono fuori posto, perciò scambiandoli si aumenta l'ordine

	↓	↓					
12	30	18	8	14	41	3	39

Con lo stesso ragionamento, l'ordine può essere aumentato scambiando il secondo con il terzo e così via.
Dopo aver applicato questa idea a tutte le coppie contigue del vettore otteniamo la configurazione seguente

12	18	8	14	30	3	39	41

Questo meccanismo di scambi garantisce che l'elemento massimo venga portato nell'ultima posizione del vettore, quindi l'ultimo elemento è ordinato.
Ripetendo questo procedimento su tutti gli elementi tranne l'ultimo si ottiene che il massimo fra questi venga portato nella penultima posizione.
Il metodo di scambi garantisce che a ogni passaggio si dispone in ordine un elemento in più.
Poiché gli elementi sono n, per completare l'ordinamento occorrono n - 1 passaggi, di lunghezza decrescente.
Nella figura seguente questo meccanismo è applicato in modo completo all'esempio considerato.



E' un algoritmo che si basa pesantemente sul meccanismo di scambio, perciò può essere molto costoso in termini di tempo per ordinare grossi insiemi di dati casuali.
L'algoritmo si presta a numerosi perfezionamenti. Ad esempio è possibile per certi insiemi di dati che il vettore sia completamente ordinato prima che l'indice *i* raggiunga il valore *n* - 1.

```
Input{n, (a(i), i=1,n)}
For  i = 1  to  n - 1  do
                For  j = 1  to   n - i  do
                        If       a(j) > a(j + 1)    then
                                 t = a(j)
                                 a(j) = a(j + 1)
                                 a(j + 1) = t
                        end if
                end for
end for
Output{ (a(i), i=1,n)}
End
```

4. Rappresentazione dei numeri e aritmetica di macchina

4.1 Introduzione

Un problema rilevante che si deve affrontare nei processi di elaborazione dell'informazione è la rappresentazione dell'informazione.
All'interno degli elaboratori elettronici l'informazione è rappresentata mediante fenomeni fisici relativamente semplici che possono essere ridotti a due stati (livelli di tensione o magnetizzazione di opportuni dispositivi), e quindi la rappresentazione dell'informazione sarà prodotta dall'associazione tra questi stati e i simboli di un alfabeto binario.
Le nostre informazioni saranno rappresentate da sequenze di simboli binari. Numeri, parole, più in generale oggetti, dovranno quindi essere espressi mediante codici binari.
Argomento di questo capitolo sarà la rappresentazione dell'informazione numerica: si tratteranno pertanto i **sistemi numerici**, con particolare riferimento a quello binario.

4.2 L'aritmetica elementare

L'uomo è abituato a compiere le operazioni aritmetiche usando il sistema di numerazione decimale. Si ha dimestichezza con la nozione di **posizione decimale** : si sa ad esempio che per moltiplicare un numero per 10 si sposta la virgola di una posizione verso destra o, se il numero è intero, si pone uno zero alla destra del numero, e così via.
Il sistema decimale è un sistema in **base 10**, poiché fa uso di 10 simboli diversi: si usano le cifre 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ereditate dagli Arabi nel passato.
I calcolatori digitali, d'altra parte, possiedono caratteristiche fisiche che non si prestano ad utilizzare il sistema decimale. I computer compiono tutte le operazioni utilizzando un sistema di numerazione diverso dal nostro e maggiormente rispondente alle caratteristiche dei materiali impiegati: il sistema binario (base 2), che meglio si presta a codificare situazioni come "corrente positiva" e "corrente negativa", "magnetizzazione positiva" e "magnetizzazione negativa".
Il sistema binario è d'altra parte assolutamente inadatto alla mente umana, che difficilmente manipola lunghe sequenze di 0 e 1 e inoltre la conversione dei numeri da binario a decimale e viceversa non è immediata.
In campo informatico si fa uso di due ulteriori sistemi di numerazione: l'**ottale** con 8 simboli diversi e l'**esadecimale** che ne usa 16.
Va comunque rilevato che facendo uso dei sofisticati prodotti software esistenti oggi, l'utente medio non ha mai necessità di operare con basi diverse da quella decimale.

4.3 Sistemi numerici

Un sistema numerico è determinato quando si fissano alcuni elementi che lo caratterizzano, ossia:
1 – un insieme limitato di simboli, le **cifre**, che rappresentano quantità intere prestabilite (ad esempio alcune lettere dell'alfabeto nel sistema di numerazione romano, le cifre arabe nel nostro sistema di numerazione);
2 – le regole che devono essere applicate per costruire i numeri.
Questo porta a una differenziazione fra sistemi numerici. I sistemi numerici si distinguono in:
- **non posizionali**: il valore delle cifre è indipendente dalla loro posizione all'interno del numero (ad esempio, nella numerazione romana la cifra M ha sempre il valore di mille, da sola, preceduta o seguita da altre cifre);
- **posizionali**: a ogni posizione all'interno della rappresentazione è associato un peso; tra i sistemi posizionali sono di rilevante interesse quelli a **base fissa**.

Nei sistemi numerici posizionali a base fissa r, un numero N può essere rappresentato nel seguente modo:

$$N = d_n\, d_{n-1} \cdots\cdots d_1\, d_0 \,.\, d_{-1} \cdots d_{-m}$$

Il valore di N è:

$$N = d_n\, r^n + ...+ d_1\, r^1 + d_0\, r^0 + d_{-1}\, r^{-1} + ... + d_{-m}\, r^{-m} = \sum_{i=-m}^{n} d_i\, r^i$$

dove:

d_i	rappresenta le cifre	$0 \le d_i \le r - 1 \qquad i = -m,..., n$
r	rappresenta la base	
n	rappresenta il numero di cifre della parte intera	
m	rappresenta il numero di cifre della parte decimale	
d_n	rappresenta la cifra più significativa	
d_{-m}	rappresenta la cifra meno significativa.	

In un sistema numerico a base fissa ogni numero può essere rappresentato in modo unico. I sistemi numerici più comuni sono i seguenti.

Sistema decimale

E' il sistema numerico usato nella vita quotidiana.
Il sistema numerico decimale si caratterizza nel modo seguente:

base $r = 10$ cifre $d_i \in \{0,1,2,3,4,5,6,7,8,9\}$

$$N = d_n\, 10^n + ...+ d_0\, 10^0 + d_{-1}\, 10^{-1} + ...+ d_{-m}\, 10^{-m}$$

Esempi

$$12.25_{10} = 1 \cdot 10 + 2 \cdot 10^0 + 2 \cdot 10^{-1} + 5 \cdot 10^{-2}$$
$$123.45_{10} = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 4 \cdot 10^{-1} + 5 \cdot 10^{-2}$$

Sistema binario

E' il sistema usato nei calcolatori; una cifra binaria è detta bit.
Un sistema numerico binario è definito come segue:

base $r = 2$ cifre $d_i \in \{0,1\}$

$$N = d_n\, 2^n + ...+ d_0\, 2^0 + d_{-1}\, 2^{-1} + ...+ d_{-m}\, 2^{-m}$$

Esempi

$$101_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5_{10}$$
$$101.01_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 5.25_{10}$$

Sistema ottale

Un sistema numerico ottale viene definito come segue:

base $r = 8$ cifre $d_i \in \{0,1,2,3,4,5,6,7\}$

$$N = d_n\, 8^n + ...+ d_0\, 8^0 + d_{-1}\, 8^{-1} + ...+ d_{-m}\, 8^{-m}$$

Esempi

$$57_8 = 5 \cdot 8^1 + 7 \cdot 8^0 = 47_{10}$$
$$23.4_8 = 2 \cdot 8^1 + 3 \cdot 8^0 + 4 \cdot 8^{-1} = 19.5_{10}$$

Sistema esadecimale

Un sistema numerico esadecimale (in base 16) è definito come segue:

base $r = 16$ cifre $d_i \in \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$

$$N = d_n\, 16^n + ...+ d_0\, 16^0 + d_{-1}\, 16^{-1} + \quad + d_{-m}\, 16^{-m}$$

Esempi

$$A1_{16} = A \cdot 16^1 + 1 \cdot 16^0 = 10 \cdot 16 + 1 = 161_{10}$$
$$FA3_{16} = F \cdot 16^2 + A \cdot 16^1 + 3 \cdot 16^0 = 15 \cdot 16^2 + 10 \cdot 16 + 3 = 4003_{10}$$

4.4 Conversione tra sistemi numerici

Esempi

Conversione di un numero da un sistema in base qualsiasi al sistema decimale

1.

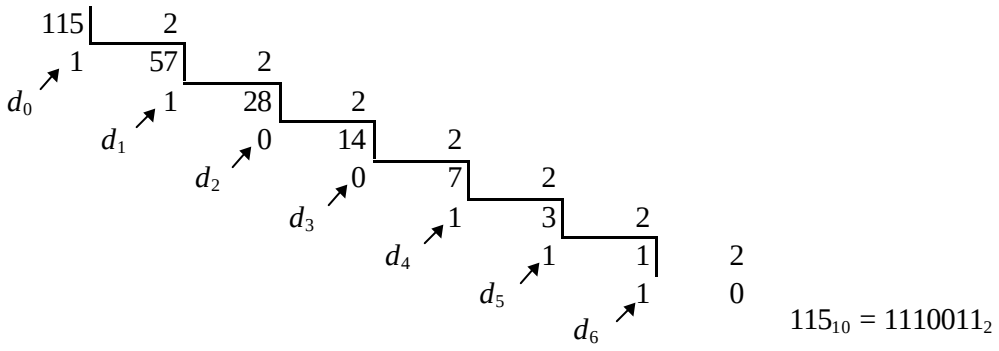
$$1010_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 8 + 2 = 10_{10}$$

$$26_8 = 2 \cdot 8^1 + 6 \cdot 8^0 = 16 + 6 = 22_{10}$$

$$0.101_2 = 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = \frac{1}{2} + \frac{1}{8} = 0.5 + 0.125 = 0.625_{10}$$

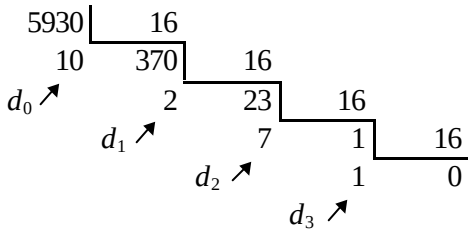
Conversione di un numero dal sistema decimale a un sistema con base qualsiasi

2. Convertire il numero 115_{10} in binario.



$115_{10} = 1110011_2$

3. Convertire 5930_{10} in base 16.



$5930_{10} = 172A_{16}$

4. Convertire il numero 0.625_{10} in binario.

$$\begin{array}{r} 0.625 \times \\ \quad 2 \\ \hline 1.250 \times \\ \quad 2 \\ \hline 0.500 \times \\ \quad 2 \\ \hline 1.000 \end{array}$$

$0.625_{10} = 0.101_2$

5. Convertire il numero 0.140625_{10} in ottale.

$$\begin{array}{r} 0.140625 \times \\ \quad 8 \\ \hline 1.125000 \times \\ \quad 8 \\ \hline 1.000000 \end{array}$$

$0.140625_{10} = 0.11_8$

Un numero reale con rappresentazione decimale non finita ha necessariamente quella binaria (o in altra base) composta da infiniti bit.
Questo fatto può accadere anche nel caso in cui il numero decimale ha un numero finito di cifre; cioè in generale non è detto che a un numero in decimale con un numero finito di cifre decimali corrisponda un numero in binario con un numero finito di cifre binarie.
L'operazione di conversione si considera terminata quando si raggiunge la precisione voluta.

6. Convertire il numero 0.5610 in binario

0. 56 ×
2
———
1. 12 ×
2
———
0. 24 ×
2
———
0 . 48 ×
2
———
0 . 96 ×
2
———
1 . 92 ×
2
———
1 . 84 ×
2
———
1 . 68 ecc.

$0.56_{10} = 0.1000111_2$
con approssimazione di $1/128 = 2^{-7}$

.

7. Convertire il numero 0.110 in binario, ottale e esadecimale.

$0.1_{10} = 0.0001100110011...._2 = 0.063146314..._8 = 0.1999..._{16}$

0.1 ×
8
———
0.8 ×
8
———
6.4 ×
8
———
3.2 ×
8
———
1.6 ×
8
———
4.8 ×
8
———
6.4 ecc.

0.1 ×
16
———
1.6 ×
16
———
9.6 ×
16
———
9.6 ecc.

8. Convertire in binario il numero 12.5_{10}
Parte intera

12	2			
0	6	2		
	0	3	2	
		1	1	2
			1	0

$12.5_{10} = 1100_{10}$

Parte frazionaria

$$\begin{array}{r} 0.5 \times \\ 2 \\ \hline 1.0 \end{array}$$

$0.5_{10} = 0.1_2$

$12.5_{10} = 1100.1_2$

4.5 Conversione dal sistema binario al sistema ottale o esadecimale

Per convertire un numero binario in ottale è sufficiente raggruppare a 3 a 3 i bit, a partire dal punto decimale verso sinistra per la parte intera e verso destra per la parte decimale, e leggere in decimale i singoli gruppi di cifre ottenute

Esempi
Numero intero: conversione da binario a ottale

$\underbrace{101}\underbrace{110}\underbrace{001}_2 = 561_8$

Numero non intero: conversione da binario ad ottale (nel caso in cui i gruppi estremi, il primo di sinistra della parte intera e l'ultimo di destra della parte frazionaria, abbiano meno di 3 bit, occorre completarli aggiungendo degli zeri)

$\underbrace{1}\underbrace{101}\underbrace{001}.\underbrace{011}_2 = 151.3_8$

$\underbrace{11}.\underbrace{101}\underbrace{1}_2 = 3.54_8$

Anche la conversione inversa è immediata: a ciascuna cifra ottale si sostituisce il corrispondente gruppo di tre bit.

Esempi

$7512_8 = \underbrace{111}\underbrace{101}\underbrace{001}\underbrace{010}_2$

$12.6 = \underbrace{1}\underbrace{010}.\underbrace{110}_2$

Analogamente, raggruppando a 4 a 4 i bit si converte un numero binario in esadecimale e viceversa.

Esempi

$\underbrace{1011}\underbrace{1001}_2 = B9_{16}$

$\underbrace{11}.\underbrace{1011}_2 = 3.B_{16}$

$\underbrace{101}\underbrace{111}_{2} = 5.E_{16}$

$EC_{16} = \underbrace{1110}\underbrace{1100}_{2}$

$6.F_{16} = \underbrace{110}.\underbrace{1111}_{2}$

$6.32_{16} = \underbrace{110}.\underbrace{0011}\underbrace{0010}_{2}$

4.6 Rappresentazione dei numeri nel calcolatore

Il sistema numerico binario è stato adottato negli elaboratori elettronici perché è quello più adatto a rappresentare stati quali assenza o presenza di tensione o di magnetizzazione.
In un calcolatore viene riservato uno **spazio finito di memoria** per la rappresentazione di ogni numero reale. Pertanto possiamo rappresentare esattamente solo quei numeri che, in base alla convenzione di rappresentazione scelta, possono essere contenuti nello spazio previsto per ognuno di essi; questi numeri sono chiamati **numeri di macchina**.
La rappresentazione che viene più convenientemente utilizzata è la rappresentazione in **virgola mobile (floating-point)**, che consente di risolvere in modo molto efficace il problema di trattare numeri con ordini di grandezza molto diversi.
Questo sistema di rappresentazione è basato sulla **notazione scientifica** o **esponenziale** comunemente usata in fisica, chimica, ingegneria, matematica.
Ogni numero reale x può essere scritto nella forma

$$x = p \times r^q$$

dove p è un numero reale, r è la base del sistema di numerazione scelto e q è un intero relativo.

Esempio
Il numero reale decimale

$$x = 23.701$$

può essere scritto come segue

$$x = 23701 \cdot 10^3 = 237.01 \cdot 10^{-1} = 0.23701 \cdot 10^2 = 0.00023701 \cdot 10^5 = \ldots.$$

Come si vede dall'esempio, la rappresentazione in forma esponenziale non è unica.
La **rappresentazione** del numero $x \neq 0$ si dice **normalizzata** quando

$$r^{-1} \leq |p| < 1$$

ossia quando la cifra della parte intera è 0 e inoltre la prima cifra di p dopo il punto decimale è diversa da zero.

Esempio
La rappresentazione normalizzata di

$x = 123.715_{10}$ è $x = 0.123715 \cdot 10^3$.

La rappresentazione normalizzata di

$x = 0.000718_{10}$ è $x = 0.718 \cdot 10^{-3}$.

Nella rappresentazione normalizzata di un numero chiamiamo p **mantissa** e q **caratteristica** o **esponente** del numero x.
Fissata la base r, ogni numero reale $x \neq 0$ è univocamente definito dalla coppia (p,q), ossia la rappresentazione normalizzata è unica. Pertanto è sufficiente memorizzare la coppia (p,q). Con questa notazione è possibile la scrittura in uno spazio limitato di numeri aventi ordini di grandezza molto diversi tra loro; viene in ogni caso evidenziata l'informazione essenziale, che è la sequenza di cifre significative nella mantissa, mentre la caratteristica fornisce l'ordine di grandezza, espresso in potenze della base.

La notazione esponenziale normalizzata è quella standard di macchina per i numeri reali. Poiché un numero reale è univocamente individuato dalla coppia di interi (p,q), per rappresentare un numero reale si deve stabilire:
– quanti byte devono essere utilizzati complessivamente per la rappresentazione di un numero reale in notazione esponenziale normalizzata;
– quanti di essi devono rappresentare la mantissa e con quali convenzioni sono utilizzati;
– quanti, analogamente, per la caratteristica.
Nella maggior parte dei moderni calcolatori, per la memorizzazione dei numeri reali in aritmetica floating-point esiste la possibilità di riservare a ciascun numero una lunghezza complessiva di 32 bit (l'**aritmetica** in questo caso viene definita **in semplice precisione**), oppure di 64 bit (**aritmetica in doppia precisione**).
Come numeri reali di macchina definiamo quei numeri le cui mantisse e caratteristiche sono rappresentabili esattamente negli spazi a loro riservati.
In particolare, se lo spazio dedicato alla rappresentazione di $|p|$ corrisponde a t cifre nel sistema di numerazione scelto, saranno rappresentabili solo quelle mantisse che non hanno più di t cifre.
La caratteristica dovrà soddisfare una disuguaglianza del tipo $m \le q \le M$, dove $m < 0$ e $M > 0$ sono interi .

Definizione dei **numeri di macchina**
Siano r, t, m, M numeri interi tali che $r \ge 2$, $t \ge 1$, $m < 0$, $M > 0$. Si definisce **insieme dei numeri di macchina floating-point**, con rappresentazione normalizzata, in base r con t cifre significative nella mantissa, l'insieme dei numeri reali definito nel modo seguente

$$\Im(r,t,m,M) = \{0\} \cup \left\{ x \in \mathrm{R} / x = sgn(x) \cdot r^q \cdot \sum_{i=1}^{t} d_i r^{-i} \right\}$$

$0 \le d_i < r$ per $i = 1,2,\ldots,t$ con $d_1 \ne 0$

$m \le q \le M$

dove $sgn(x)$ = segno del numero x.

Si osservi che l'insieme dei numeri di macchina contiene per definizione lo 0, che non è rappresentabile con una mantissa normalizzata, e quindi non è rappresentabile in modo univoco.
Di solito lo 0 è rappresentato con mantissa nulla ed esponente m.

Esempio
Scrivere tutti i numeri dell'insieme di numeri di macchina $\Im(2,3,-1,1)$ e rappresentarli sulla retta reale.
Base 2; 3 cifre per la mantissa con $d_1 \ne 0$; esponenti q compresi fra –1 e 1.
Le possibili mantisse, e i corrispondenti valori in decimale, sono:

$$.100 \rightarrow \frac{1}{2}$$

$$.101 \rightarrow \frac{1}{2}+\frac{1}{8}=\frac{5}{8}$$

$$.110 \rightarrow \frac{1}{2}+\frac{1}{4}=\frac{3}{4}$$

$$.111 \rightarrow \frac{1}{2}+\frac{1}{4}+\frac{1}{8}=\frac{7}{8}$$

I numeri di macchina positivi sono:

$0.100 \cdot 2^{-1}$ $0.100 \cdot 2^{0}$ $0.100 \cdot 2^{1}$

$0.101 \cdot 2^{-1}$ $0.101 \cdot 2^{0}$ $0.101 \cdot 2^{1}$

$0.110 \cdot 2^{-1}$ $0.110 \cdot 2^{0}$ $0.110 \cdot 2^{1}$

$0.111 \cdot 2^{-1}$ $0.111 \cdot 2^{0}$ $0.111 \cdot 2^{1}$

Il loro valore nel sistema decimale è:

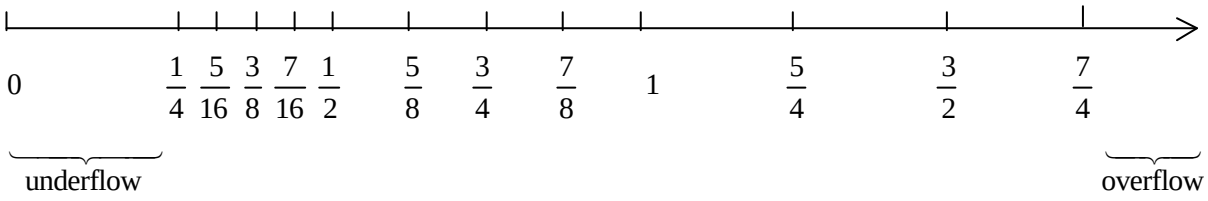
$0.100 \cdot 2^{-1} = \frac{1}{2} \cdot 2^{-1} = \frac{1}{4}$ $0.100 \cdot 2^{0} = \frac{1}{2} \cdot 2^{0} = \frac{1}{2}$ $0.100 \cdot 2^{1} = \frac{1}{2} \cdot 2^{1} = 1$

$0.101 \cdot 2^{-1} = \frac{5}{8} \cdot 2^{-1} = \frac{5}{16}$ $0.101 \cdot 2^{0} = \frac{5}{8} \cdot 2^{0} = \frac{5}{8}$ $0.101 \cdot 2^{1} = \frac{5}{8} \cdot 2^{1} = \frac{5}{4}$

$0.110 \cdot 2^{-1} = \frac{3}{4} \cdot 2^{-1} = \frac{3}{8}$ $0.110 \cdot 2^{0} = \frac{3}{4} \cdot 2^{0} = \frac{3}{4}$ $0.110 \cdot 2^{1} = \frac{3}{4} \cdot 2^{1} = \frac{3}{2}$

$0.111 \cdot 2^{-1} = \frac{7}{8} \cdot 2^{-1} = \frac{7}{16}$ $0.111 \cdot 2^{0} = \frac{7}{8} \cdot 2^{0} = \frac{7}{8}$ $0.111 \cdot 2^{1} = \frac{7}{8} \cdot 2^{1} = \frac{7}{4}$

Rappresentazione sulla retta reale dei numeri floating point positivi:



I numeri negativi sono simmetrici rispetto all'origine.

Il più piccolo numero positivo è $0.100 \cdot 2^{-1} = \frac{1}{4}$; il più grande è $0.111 \cdot 2^{1} = \frac{7}{4}$.

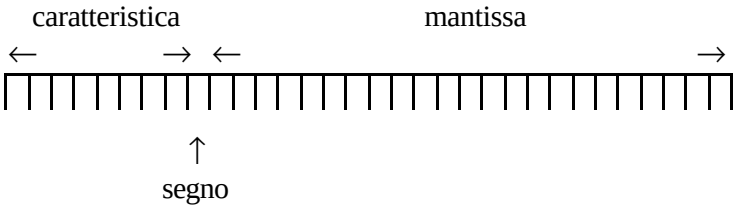
Dall'esempio possiamo osservare che i numeri di macchina non sono distribuiti in modo regolare. Si vede che i punti sono raggruppati in modo più fitto tra $\frac{1}{4}$ e $\frac{1}{2}$ che tra 1 e $\frac{7}{4}$.

Pertanto quando numeri reali x qualsiasi, compresi nell'intervallo $\left[\frac{1}{4}, \frac{7}{4}\right]$ e non coincidenti con numeri di macchina, vengono approssimati con numeri di macchina in virgola mobile, le approssimazioni che si ottengono sono più accurate per certi numeri che per altri.

Nella figura seguente si schematizza una possibile rappresentazione interna a un calcolatore di un numero di macchina su 32 bit (semplice precisione)

$$x = \pm(.d_1 d_2 \ldots d_t) \cdot r^q \in \Im(r, t, m, M)$$

nel caso $r = 2$, $t = 24$, con 8 bit per l'esponente:



Riservando 23 bit per la mantissa, si ottiene una precisione di circa 6 - 7 cifre decimali: infatti con 23 bit si rappresenta al più il numero decimale

$2^{23} - 1 = 8388607$

La rappresentazione in doppia precisione assegna in alcuni casi i 32 bit aggiuntivi tutti alla mantissa, in altri casi invece viene assegnata una parte di essi anche alla caratteristica.
In doppia precisione si hanno circa 14 cifre decimali esatte.
Per evitare di dover prevedere anche un bit per il segno dell'esponente, per convenzione anziché memorizzare q , viene memorizzata la quantità $q^* = q - m$ che risulta sempre non negativa.

4.7 Arrotondamento e troncamento

Rappresentando un numero reale positivo x (per i numeri negativi si ottengono risultati analoghi) in forma normalizzata

$$x = (d_1 r^{-1} + d_2 r^{-2} + ...) r^q \qquad \text{con } d_1 \neq 0$$

in un sistema di numeri di macchina $\Im(r,t,m,M)$ si possono presentare i seguenti casi:
1 - il numero x è tale che

$$m \leq q \leq M$$

$$d_i = 0 \quad \text{per } i > t;$$

allora x è un numero di macchina ed è rappresentato esattamente.
Se x non appartiene a $\Im(r,t,m,M)$, si pone il problema di associare in modo adeguato a x un numero di macchina $\bar{x}$.
2 - la caratteristica q non appartiene all'intervallo $[m,M]$.

Se $q < m$ si dice che si verifica un **underflow** (come ad esempio $x = \frac{1}{8}$ nell'esempio di pag. 52,53); solitamente si assume come valore approssimato del numero x il numero 0; il sistema di calcolo, in presenza di questa situazione, dà un avvertimento **(warning)**: in particolari situazioni infatti l'introduzione di questi errori può portare a situazioni inaccettabili.
Se $q > M$, si verifica un **overflow** (come ad esempio $x = 2$ nell'esempio di pag. 52,53) e di solito non si effettua alcuna approssimazione, ma il sistema di calcolo ha un comportamento più drastico, come ad esempio l'arresto del calcolo.
3 - la caratteristica q appartiene all'intervallo $[m,M]$, ma le cifre d_i per $i > t$ non sono tutte nulle. In questo caso si pone il problema di scegliere un numero $\bar{x} \in \Im$ per rappresentare x. Tale operazione viene indicata comunemente come **operazione di arrotondamento**, anche se in realtà possono essere utilizzate due tecniche di tipo diverso

Troncamento di x alla t-esima cifra	$\bar{x} = trn(x) = r^q \sum_{i=1}^{t} d_i r^{-i}$
Arrotondamento di x alla t-esima cifra	$\bar{x} = arr(x) = r^q trn\left(\sum_{i=1}^{t+1} d_i r^{-i} + \frac{1}{2} r^{-t}\right)$

Esempi

$\Im(10,3,-5,5)$ base 10

1. $x = 98273 = 0.98273 \cdot 10^5$
$trn(x) = 0.982 \cdot 10^5$
$arr(x) = 0.983 \cdot 10^5$;

infatti

$$arr(x) = 10^5 \cdot trn(0.9827 + \frac{1}{2} \cdot 10^{-3}) = 10^5 \cdot trn(0.9827 + 0.0005) =$$

$$= 10^5 \cdot trn(0.9832) = 0.983 \cdot 10^5.$$

2. $x = 98243 = 0.98243 \cdot 10^5$
$trn(x) = 0.982 \cdot 10^5$
$arr(x) = 0.982 \cdot 10^5$
in questo caso il numero ottenuto con l'arrotondamento coincide con il numero ottenuto con il troncamento.

Quindi l'arrotondamento tiene conto della $(t + 1)$-esima cifra:

se $d_{t+1} < \frac{r}{2}$ $\Rightarrow$ l'arrotondamento coincide con il troncamento,

se $d_{t+1} \geq \frac{r}{2}$ $\Rightarrow$ si aumenta di 1 la t-esima cifra e si effettua il troncamento.

Nell'effettuare l'arrotondamento di un numero può verificarsi la situazione di overflow

Esempio
$\Im(10,3,-5,5)$
$x = 99960 = 0.99960 \cdot 10^5$ $trn(x) = 0.999 \cdot 10^5$

Nell'arrotondamento si ha overflow:
$arr(x) = 0.100 \cdot 10^6$.

Definizione di precisione di macchina (epsilon di macchina)
La quantità

$$eps = \begin{cases} r^{1-t} & \text{con tecnica di troncamento} \\ \frac{1}{2} \cdot r^{1-t} & \text{con tecnica di arrotondamento} \end{cases}$$

è detta **precisione di macchina**

La precisione di macchina è una costante caratteristica di ogni aritmetica floating-point, in quanto dipende dalla base r del sistema di numerazione adottato, dal numero t di cifre usato per la mantissa e dalla tecnica di arrotondamento o troncamento scelta per approssimare i numeri.
La sua importanza numerica è data dalla seguente caratterizzazione: ***eps* è il più piccolo numero di macchina positivo tale che**

$$fl(1 + eps) > 1$$

dove $fl(1 + eps)$ è il numero di macchina che approssima la somma $1 + eps$.

Il numero *eps* rappresenta la massima precisione di calcolo raggiungibile con il calcolatore su cui tale aritmetica è implementata e non ha quindi senso cercare di determinare approssimazioni con precisione relativa inferiore alla precisione di macchina *eps*.

4.8 Aritmetica in floating point

Sull'insieme dei numeri di macchina si devono ridefinire le operazioni aritmetiche: si deve cioè introdurre una **aritmetica di macchina**.
Infatti il risultato di un'operazione aritmetica anche nel caso in cui gli operandi siano numeri di macchina, può non essere un numero di macchina per due motivi:
1 – Si ha una situazione di underflow o di overflow.

Esempio
$\Im(10,4,-2,3)$

$x = 0.9876 \cdot 10^3 \qquad y = 0.4543 \cdot 10^3$

$x + y = 0.1442 \cdot 10^4$ overflow!

$x = 0.4387 \cdot 10^{-2} \qquad y = -0.4325 \cdot 10^{-2}$

$x + y = 0.6200 \cdot 10^{-4}$ underflow!

2 – Il risultato ha un numero di cifre superiore alla precisione *t*.

Esempio

$\Im(10,4,-2,3)$

$x = 0.1123 \cdot 10^0 \qquad y = 0.1123 \cdot 10^{-2}$

$x + y = 0.113423 \cdot 10^0$.

Per il seguito consideriamo in particolare questa seconda eventualità. Poiché non è possibile realizzare esattamente le operazioni aritmetiche, occorre definire delle **operazioni di macchina**.
In ogni operazione di macchina a due numeri di macchina è associato un terzo numero di macchina, ottenuto arrotondando o troncando il risultato dell'operazione aritmetica in esame sui due operandi numeri di macchina.
Se con $\bar{x} = fl(x)$ e $\bar{y} = fl(y)$ indichiamo i due numeri di macchina in floating-point (ottenuti arrotondando o troncando x e y) e con

$\oplus \qquad \Theta \qquad \otimes \qquad \varnothing$

le operazioni di macchina corrispondenti alle quattro operazioni aritmetiche

$+ \qquad - \qquad \times \qquad /$

abbiamo

$\bar{x} \oplus \bar{y} = fl(\bar{x} + \bar{y})$
$\bar{x} \Theta \bar{y} = fl(\bar{x} - \bar{y})$
$\bar{x} \otimes \bar{y} = fl(\bar{x} \times \bar{y})$
$\bar{x} \varnothing \bar{y} = fl(\bar{x} / \bar{y})$.

Esempi

$\Im(10,4,-2,3)$ con tecnica di arrotondamento

Somma

1 - $\bar{x} = 0.1234 \cdot 10^3 \qquad \bar{y} = 0.6212 \cdot 10^2$
$\bar{x} \oplus \bar{y} = 0.1855 \cdot 10^3$

2 - $\bar{x} = 0.9876 \cdot 10^1 \qquad \bar{y} = 0.8765 \cdot 10^1$
$\bar{x} \oplus \bar{y} = 0.1864 \cdot 10^2$

3 - $\bar{x} = 0.1234 \cdot 10^2 \qquad \bar{y} = 0.4567 \cdot 10^0$
$\bar{x} \oplus \bar{y} = 0.1280 \cdot 10^2$.

4 - $\bar{x} = 0.1234 \cdot 10^3 \qquad \bar{y} = 0.2387 \cdot 10^{-1}$
$\bar{x} \oplus \bar{y} = 0.1234 \cdot 10^3$
$\bar{x} \oplus \bar{y} = \bar{x}$!! con $\bar{y} \neq 0$ (vedi proprietà 6, pag. 59)

Prodotto

5 - $\bar{x} = 0.1234 \cdot 10^1 \qquad \bar{y} = 0.2468 \cdot 10^2$
$\bar{x} \otimes \bar{y} = 0.3046 \cdot 10^2.$

Quoziente

6 - $\bar{x} = 0.5615 \cdot 10^1 \qquad \bar{y} = 0.1234 \cdot 10^2$
$\bar{x} \oslash \bar{y} = 0.4550 \cdot 10^0.$

4.9 Proprietà delle operazioni aritmetiche in floating-point

Nell'aritmetica in virgola mobile valgono solo alcune delle proprietà delle operazioni elementari. Restano valide ad esempio le proprietà:

1. Commutativa

$$\begin{cases} \bar{x} \oplus \bar{y} = \bar{y} \oplus \bar{x} \\ \bar{x} \otimes \bar{y} = \bar{y} \otimes \bar{x} \end{cases}$$

2. $\bar{x} \oplus 0 = \bar{x}$

3. $\bar{y} \otimes 1 = \bar{y}$

4. $\bar{x} \oslash 1 = \bar{x}$

5. $\bar{x} \oslash \bar{x} = 1$

6. $\bar{x} \otimes \bar{y} = 0 \Leftrightarrow \bar{x} = 0 \vee \bar{y} = 0$

7. $\bar{x} \oplus \bar{y} = 0 \Leftrightarrow \bar{x} = -\bar{y}$

Per l'aritmetica di macchina **non valgono** però le seguenti **proprietà**

1. Associativa dell'addizione

$$(\bar{x} \oplus \bar{y}) \oplus \bar{z} = \bar{x} \oplus (\bar{y} \oplus \bar{z})$$

Esempio

$\Im(10,2,m,M)$ con tecnica di arrotondamento

$\bar{x} = 0.11 \cdot 10^0 \qquad \bar{y} = 0.13 \cdot 10^{-1} \; \bar{z} = 0.14 \cdot 10^{-1}$

$$(\bar{x} \oplus \bar{y}) \oplus \bar{z} = (0.11 \cdot 10^0 \oplus 0.13 \cdot 10^{-1}) \oplus 0.14 \cdot 10^{-1} =$$
$$= 0.12 \cdot 10^0 \oplus 0.14 \cdot 10^{-1} = 0.13 \cdot 10^0$$
$$\bar{x} \oplus (\bar{y} \oplus \bar{z}) = 0.11 \cdot 10^0 \oplus (0.13 \cdot 10^{-1} \oplus 0.14 \cdot 10^{-1}) =$$
$$= 0.11 \cdot 10^0 \oplus 0.27 \cdot 10^{-1} = 0.14 \cdot 10^0$$
$$\uparrow \text{arrotondamento!}$$

2. Associativa della moltiplicazione

$$(\bar{x} \otimes \bar{y}) \otimes \bar{z} = \bar{x} \otimes (\bar{y} \otimes \bar{z})$$

Esempio

$\Im(10,2,m,M)$ con tecnica di arrotondamento

$$\bar{x} = 0.11 \cdot 10^1 \qquad \bar{y} = 0.31 \cdot 10^1 \qquad \bar{z} = 0.25 \cdot 10^1$$

$$(\bar{x} \otimes \bar{y}) \otimes \bar{z} = (0.11 \cdot 10^1 \otimes 0.31 \cdot 10^1) \otimes 0.25 \cdot 10^1 =$$

$$= 0.34 \cdot 10^1 \otimes 0.25 \cdot 10^1 = 0.85 \cdot 10^1$$

$$\bar{x} \otimes (\bar{y} \otimes \bar{z}) = 0.11 \cdot 10^1 \otimes (0.31 \cdot 10^1 \otimes 0.25 \cdot 10^1) =$$

$$= 0.11 \cdot 10^1 \otimes 0.78 \cdot 10^1 = 0.86 \cdot 10^1$$

3. Distributiva del prodotto rispetto all'addizione

$$\bar{x} \otimes (\bar{y} \oplus \bar{z}) = (\bar{x} \otimes \bar{y}) \oplus (\bar{x} \otimes \bar{z})$$

Esempio

$\Im(10,2,m,M)$ con tecnica di arrotondamento

$$\bar{x} = 0.11 \cdot 10^1 \qquad \bar{y} = 0.23 \cdot 10^1 \qquad \bar{z} = 0.24 \cdot 10^1$$

$$\bar{x} \otimes (\bar{y} \oplus \bar{z}) = 0.11 \cdot 10^1 \otimes (0.23 \cdot 10^1) \oplus 0.24 \cdot 10^1) =$$

$$= 0.11 \cdot 10^1 \otimes 0.47 \cdot 10^1 = 0.52 \cdot 10^1$$

$$(\bar{x} \otimes \bar{y}) \oplus (\bar{x} \otimes \bar{z}) = (0.11 \cdot 10^1 \otimes 0.23 \cdot 10^1) \oplus (0.11 \cdot 10^1 \otimes 0.24 \cdot 10^1) =$$

$$= 0.25 \cdot 10^1 \oplus 0.26 \cdot 10^1 = 0.51 \cdot 10^1$$

4. Legge di cancellazione

$$\bar{x} \otimes \bar{y} = \bar{z} \otimes \bar{y}, \quad \bar{y} \neq 0 \Rightarrow \bar{x} = \bar{z}$$

Esempio

$\Im(10,2,m,M)$ con tecnica di arrotondamento

$$\bar{x} = 0.51 \cdot 10^1 \qquad \bar{y} = 0.22 \cdot 10^1 \qquad \bar{z} = 0.52 \cdot 10^1$$

$$\bar{x} \otimes \bar{y} = 0.51 \cdot 10^1 \otimes 0.22 \cdot 10^1 = 0.11 \cdot 10^2$$

$$\bar{z} \otimes \bar{y} = 0.52 \cdot 10^1 \otimes 0.22 \cdot 10^1 = 0.11 \cdot 10^2$$

ma $\bar{x} \neq \bar{z}$

5. Semplificazione

$$\bar{x} \otimes (\bar{y} \oslash \bar{x}) = \bar{y}$$

Esempio

$\Im(10,2,m,M)$ con tecnica di arrotondamento

$$\bar{x} = 0.70 \cdot 10^1 \qquad \bar{y} = 0.80 \cdot 10^1$$

$$\bar{x} \otimes (\bar{y} \oslash \bar{x}) = 0.70 \cdot 10^1 \otimes (0.80 \cdot 10^1 \oslash 0.70 \cdot 10^1) =$$

$$= 0.70 \cdot 10^1 \otimes 0.11 \cdot 10^1 = 0.77 \cdot 10^1 \neq \bar{y}$$

6 - Un'ulteriore relazione anomala è la seguente

$$\bar{x} \oplus \bar{y} = \bar{x} \qquad \text{quando} \quad |\bar{y}| < \frac{eps}{r} \cdot |\bar{x}|$$

dove r è la base del sistema di numerazione.

Esempio

$\Im(10,4,m,M)$ con tecnica di arrotondamento

$$\bar{x} = 0.9998 \cdot 10^0$$

$$eps = \frac{1}{2} \cdot r^{1-t} = \frac{1}{2} \cdot 10^{-3} \qquad \text{con arrotondamento}$$

$$\frac{eps}{10} \cdot |\bar{x}| = \frac{\frac{1}{2} \cdot 10^{-3}}{10} \cdot 0.9998 \cdot 10^0 = 0.4999 \cdot 10^{-4}$$

Prendiamo ad esempio

$$\bar{y} = 0.4998 \cdot 10^{-4}$$

$$\bar{x} \oplus \bar{y} = 0.9998 \cdot 10^0 + 0.4998 \cdot 10^{-4} = 0.9998 \cdot 10^0 = \bar{x}$$

Possiamo pertanto concludere che espressioni che sono equivalenti in aritmetica esatta non risultano generalmente tali nell'aritmetica di macchina.
Malgrado ciò, due espressioni (non nulle) saranno definite "equivalenti" dal punto di vista del calcolo numerico quando, valutate in un calcolatore, forniscono risultati che differiscono per una tolleranza relativa dell'ordine della precisione di macchina.

Appendice

Nella presente appendice sono raccolti gli algoritmi fondamentali descritti nel terzo capitolo, implementati in ambiente MATLAB.
Una delle caratteristiche principali del software scientifico MATLAB è data dal fatto che l'elemento base è l'array, uni o multidimensionale; da questo deriva la capacità di gestire insiemi di numeri, vettori o matrici, come se fossero una singola variabile. Ad esempio è possibile calcolare il prodotto di due matrici A e B semplicemente con il comando **`C = A*B`**; in molti altri linguaggi di programmazione questa operazione richiede più di un comando.
Questa speciale capacità di gestire gli array ha come conseguenza che i programmi di MATLAB possono essere molto brevi, quindi più facili da scrivere, leggere e documentare, rispetto ad altri linguaggi di programmazione: infatti in molti casi è possibile realizzare un'implementazione di tipo completamente vettoriale, senza utilizzare cicli **`for`** o **`while`**, il che rende anche il calcolo più veloce.
Inoltre in MATLAB sono disponibili molti comandi per il calcolo di funzioni elementari, per lo studio di insiemi di dati, ad esempio per il calcolo di media, varianza e deviazione standard di un insieme di dati sperimentali, per la ricerca del massimo e del minimo, per l'ordinamento dei dati.
Per questo motivo per alcuni degli algoritmi fondamentali si presentano più implementazioni, che differiscono tra loro per l'utilizzo dei cicli o della struttura vettoriale o per l'uso di comandi già disponibili.

Algoritmo 2
Conteggio

Dato un insieme di *n* studenti, contare il numero di studenti che hanno superato la prova; l'esame si intende superato con un voto ≥ 18.

```
%   conteggio
v = input('introdurre il vettore dei voti ');
n = length(v);
cont = 0;
i = 1;
while i<=n
    if v(i)>=18
         cont = cont+1;
    end
    i = i+1;
end
disp(['numero promossi = ',int2str(cont)])
```

```
%   conteggio
v = input('introdurre il vettore dei voti ');
n = length(v);
np = find(v>=18);
cont = length(np);
disp(['numero promossi = ',int2str(cont)])
```

Esercizio 1
Dato un insieme di *n* numeri reali, contare il numero di quelli negativi, positivi e nulli.

```
%   conteggio
v = input('introdurre il vettore dei numeri ');
n = length(v);
contneg = 0; contpos = 0;
i = 0;
while (i<n)
    i = i+1;
    if (v(i)<0)
         contneg = contneg+1;
    elseif (v(i)>0)
         contpos = contpos+1;
    end
end
contzero = n-(contneg+contpos);
disp(['numero negativi = ',int2str(contneg)])
disp(['numero positivi = ',int2str(contpos)])
disp(['numero nulli    = ',int2str(contzero)])
```

```
%   conteggio
v = input('introdurre il vettore dei numeri ');
n = length(v);
nneg = find(v<0);
contneg = length(nneg);
npos = find(v>0);
contpos = length(npos);
contzero = n-(contneg+contpos);
disp(['numero negativi = ',int2str(contneg)])
disp(['numero positivi = ',int2str(contpos)])
disp(['numero nulli    = ',int2str(contzero)])
```

Esercizio 2
Dato un insieme di *n* numeri interi, scrivere un algoritmo per contare quanti fra essi sono divisibili per 3.

```
%   conteggio
v = input('introdurre il vettore dei numeri ');
n = length(v);  cont=0;
for i=1:n
   if rem(v(i),3)==0
      cont=cont+1;
   end
end
disp(['i numeri divisibili per 3 sono ',int2str(cont)])
```

```
%   conteggio
v = input('introdurre il vettore dei numeri ');
n = find(rem(v,3)==0);
cont = length(n);
disp(['i numeri divisibili per 3 sono ',int2str(cont)])
```

Algoritmo 3
Sommatoria di un insieme di numeri

Dato un insieme di n numeri $\{a_1, a_2,..., a_n\}$, calcolare la somma $\sum_{i=1}^{n} a_i$.

```
%    somma
a = input('introdurre il vettore dei numeri ');
n = length(a);
s = 0;
i = 1;
while i<=n
   s = s+a(i);
   i = i+1;
end
disp(['somma = ',int2str(s)])
```

```
%    somma
a = input('introdurre il vettore dei numeri ');
n = length(a);
s = 0;
for i=1:n
   s = s+a(i);
end
disp(['somma = ',int2str(s)])
```

Esercizi 1-6
La somma e il prodotto di n numeri, memorizzati nel vettore a, si calcolano in MATLAB con i comandi

```
s = sum(a)
p = prod(a)
```

La media e la varianza di n numeri, memorizzati nel vettore x, si calcolano in MATLAB con i comandi

```
m = mean(x)
v = var(x)
```

La media armonica $h = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{x_i}$ di n numeri, memorizzati nel vettore x, si calcola in MATLAB con il comando

```
h=length(a)/sum(1./a)
```

La somma dei primi n numeri interi si calcola con il comando

```
s = sum(1:n)
```

La somma dei primi n numeri dispari si calcola con il comando

```
s = sum(1:2:2*n-1)
```

La somma dei primi n numeri pari si calcola con il comando

```
s = sum(2:2:2*n)
```

Esercizio 7
Calcolo della somma degli elementi di una matrice A di dimensioni $n \times m$.

```
%      somma degli elementi di una matrice
A = input('introdurre la matrice A ');
[n,m]=size(A);
S = 0;
for i=1:n
   for j=1:m
      S = S+A(i,j);
   end
end
disp(['somma = ',int2str(S)])
```

Il calcolo può essere fatto più semplicemente con il comando
`S = sum(sum(A))`

Algoritmo 4
Calcolo del fattoriale
Dato il numero intero $n \geq 0$, calcolare $n!$, convenendo che $0! = 1$.

```
%   calcolo di n!   per n >= 0
n = input('introdurre il valore di n ');
N = n;
nfat = 1;
while n>1
   nfat = nfat*n;
   n = n-1;
end
disp(['  ',num2str(N),'! = ',int2str(nfat)])
```

```
%   calcolo di n!   per n >= 0
n = input('introdurre il valore di n ');
nfat = 1;
if n>1
   for i=2:n
      nfat = nfat*i;
   end
end
disp(['  ',num2str(n),'! = ',int2str(nfat)])
```

L'algoritmo può essere scritto come M-file di tipo funzione, introducendo un controllo per evitare valori negativi di n e usando la funzione **`prod`**

```
function nfat=fatt(n)
%   calcolo di n!
if n<0
   error('l''argomento deve essere positivo o nullo')
end
if n>1
   nfat = prod(2:n);
else
   nfat = 1;
end
```

In MATLAB è possibile implementare algoritmi ricorsivi, cioè scrivere funzioni che richiamino se stesse. L'algoritmo per il calcolo del fattoriale può essere scritto nella seguente forma ricorsiva

```
function nfat = fatt(n)
%   calcolo di n!
if n<0
   error('l''argomento deve essere positivo o nullo')
end
if n==0
   nfat = 1;
else
   nfat = n*fatt(n-1);
end
```

Esercizio 2
Generare la successione in cui ciascun elemento è la somma di fattoriali adiacenti

$$0! + 1! \quad 1! + 2! \quad 2! + 3! \quad 3! + 4! \quad ..... \quad (n-1)! + n! ....$$

```
%   calcolo degli elementi di una successione
%   n-esimo elemento:   (n-1)!+n!
n = input('numero elementi della successione (n>=1) ');
a = 1; b = 1; i = 1;
while i<=n
   f = a + b;
   disp(f)
   i = i+1;
   a = b; b = b*i;
end
```

Esercizio 3
Calcolo del numero e per mezzo della somma

$$e = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + ... + \frac{1}{n!} \qquad n \geq 0$$

```
function e=esp(n)
%   calcolo del numero e
if n<0
   error('l''argomento deve essere positivo o nullo')
end
e = 1;
fatt = 1;
if n>=1
   for k=1:n
      fatt = fatt*k;
      e = e+1/fatt;
   end
end
```

Algoritmo 5
Generazione di una successione di Fibonacci

Generare e stampare i primi n termini della successione di Fibonacci

0, 1, 1, 2, 3, 5, 8, 13,...

```
%    numeri di Fibonacci
n = input('numero di termini della successione (n>=2) ');
a = 0;
b = 1;
disp(a), disp(b)
for i=3:n
   c = a+b;
   disp(c)
   a = b;
   b = c;
end
```

```
%    numeri di Fibonacci
n = input('numero di termini della successione (n>=2) ');
a = 0; b = 1; i = 2;
while (i<n)
    disp(a), disp(b)
    a = a+b;  b = a+b;  i = i+2;
end
if i==n
    disp(a), disp(b)
else
    disp(a)
end
```

```
function f=fibo(n)
% numeri di Fibonacci  (n>=2)
f = [0 1];  i = 1;
while i<n-1
   f(i+2)=f(i+1)+f(i);
   i = i+1;
end
```

Algoritmo 6
Il minimo divisore di un intero

Assegnato un intero *n*, determinare il suo divisore esatto più piccolo, diverso da 1.

```
% minimo divisore di un intero
n = input('introdurre un numero intero >0 ');
if rem(n,2)==0
   mindiv = 2;
else
   r = fix(sqrt(n));   d = 3;
   while (rem(n,d)~=0)&(d<r)
      d = d+2;
   end
   if rem(n,d)==0
      mindiv = d;
   else
      mindiv = 1;
   end
end
disp(['minimo divisore di ',int2str(n),' e'' ',int2str(mindiv)])
```

Algoritmo 7
Inversione dell'ordine di un vettore
Disporre gli elementi di un vettore in modo che essi risultino in ordine inverso rispetto all'ordine di partenza.

```
%   inversione dell'ordine di un vettore
a = input('introdurre il vettore dei numeri ');
n = length(a);
r = fix(n/2);
for i=1:r
   t = a(i); a(i) = a(n-i+1); a(n-i+1) = t;
end
disp(a)
```

L'inversione dell'ordine di un vettore *a* può essere ottenuta con il comando
`a=a(length(a):-1:1)`

Algoritmo 8
Ricerca del valore massimo di un insieme
Trovare il valore massimo in un insieme di *n* numeri.

```
%   massimo di in insieme di numeri
a = input('introdurre il vettore dei numeri ');
n = length(a);
massimo = a(1);
for i=2:n
   if a(i)> massimo
      massimo = a(i);
   end
end
disp(['massimo = ',int2str(massimo)])
```

Il massimo e il minimo di un insieme di *n* numeri, memorizzati nel vettore *v*, si calcolano in MATLAB con i comandi
`massimo = max(a)`
`minimo = min(a)`

Esercizio 1
Trovare il minimo fra gli elementi di un vettore *v* e il numero di volte in cui tale minimo ricorre.

```
%   minimo di un insieme
v = input('introdurre il vettore dei numeri ');
n = length(v);
minimo = min(v);
r = find(v==minimo);
nv = length(r);
disp(['minimo = ',int2str(minimo)])
disp(['il minimo compare ',int2str(nv),' volte'])
```

Esercizio 2
Trovare la massima differenza in valore assoluto tra una coppia di elementi adiacenti di un vettore di *n* elementi.

```
%   massima differenza
v = input('introdurre gli elementi del vettore ');
n = length(v);
maxdiff = 0;
for i=1:n-1
   d = abs(v(i+1)-v(i));
   if d>maxdiff
      maxdiff = d;
   end
end
disp(['massima differenza = ',int2str(maxdiff)])
```

Algoritmo 9
Ordinamento per selezione (sort)
Ordinare in modo non decrescente un vettore *v* di *n* numeri.

```
%   ordinamento degli elementi di un vettore (sort)
v = input('introdurre gli elementi del vettore ');
n = length(v);
for i = 1:n-1
   minimo = v(i);
   p = i;
   for j = i+1:n
      if v(j) < minimo
         minimo = v(j);
         p = j;
      end
   end
   v(p) = v(i);
   v(i) = minimo;
end
disp('vettore ordinato '), disp(v)
```

Algoritmo 10
Ordinamento a bolle (bubble sort)
Ordinare in modo non decrescente un vettore di *n* numeri (ordinamento a bolle, bubble sor).

```
%   ordinamento degli elementi di un vettore (bubble sort)
v = input('introdurre gli elementi del vettore ');
n = length(v);
for i = 1:n-1
   for j = 1:n-i
      if v(j) > v(j+1)
         t = v(j);
         v(j) = v(j+1);
         v(j+1) = t;
      end
   end
end
disp('vettore ordinato'), disp(v)
```

L'ordinamento di un vettore può essere ottenuto con il comando **`sort(v)`**.

Algoritmo 11
Calcolo dell'epsilon di macchina
Calcolare l'epsilon di macchina in ambiente MATLAB; *eps* è il più piccolo numero di macchina positivo tale che

$$fl(1+eps) > 1$$

dove $fl(1 + eps)$ è il numero di macchina che approssima la somma $1 + eps$

```
%   calcolo dell'epsilon di macchina
epsilon = 1;
while (1+epsilon)>1
      epsilon = epsilon/2;
end
epsilon = epsilon*2
```

```
%   calcolo dell'epsilon di macchina
epsilon = 1;
for num = 1:1000
      epsilon = epsilon/2;
      if (1+epsilon)<=1
           epsilon = epsilon*2
           break
      end
end
```

Bibliografia

1. R. G. Dromey, *Algoritmi fondamentali*, Gr. Ed. Jackson, Milano
2. E. Piccolo, P. Demichelis, *Introduzione all'informatica*, McGraw-Hill Libri Italia
3. J. H. Mathews, *Numerical Methods for Mathematics, Science and Engineering*, 2ª ed., Prentice Hall
4. J. S. Vandergraft, *Introduction to Numerical Computation*, Academic Press