Spediz. in abbonamento postale GR II/70 L. 2.200

# 56 CORSO PRATICO COL COMPUTER

422089

è una iniziativa
FABBRI EDITORI
in collaborazione con
BANCO DI ROMA
e OLIVETTI

direttore GIANNI DEGLI ANTONI

FABBRI EDITORI

Direttore dell'opera
GIANNI DEGLI ANTONI

Comitato Scientifico
GIANNI DEGLI ANTONI
Docente di Teoria dell'Informazione. Direttore dell'Istituto di Cibernetica dell'Università degli Studi di Milano

UMBERTO ECO
Ordinario di Semiotica presso l'Università di Bologna

MARIO ITALIANI
Ordinario di Teoria e Applicazione delle Macchine Calcolatrici presso l'Istituto di Cibernetica dell'Università degli Studi di Milano

MARCO MAIOCCHI
Professore Incaricato di Teoria e Applicazione delle Macchine Calcolatrici presso l'Istituto di Cibernetica dell'Università degli Studi di Milano

DANIELE MARINI
Ricercatore universitario presso l'Istituto di Cibernetica dell'Università degli Studi di Milano

Curatori di rubriche
MARCO ANELLI, DIEGO BIASI, ANDREA GRANELLI, ALDO GRASSO, MARCO MAIOCCHI, DANIELE MARINI, GIANCARLO MAURI, CLAUDIO PARMELLI

Testi
GIANCARLO MAURI, ANDREA GRANELLI, ALDO GRASSO, Etnoteam (ADRIANA BICEGO), Eidos (TIZIANO BRUGNETTI, BRUNO MOTTA, CARMINE STRAGAPEDE)

Tavole
Logical Studio Communication
Il Corso di Programmazione e BASIC è stato realizzato da Etnoteam S.p.A., Milano
Computergrafica è stato realizzato da Eidos, S.c.r.l., Milano
Usare il Computer è stato realizzato in collaborazione con PARSEC S.N.C. - Milano

Direttore Editoriale
ORSOLA FENGHI

Redazione
CARLA VERGANI
LOGICAL STUDIO COMMUNICATION

Art Director
CESARE BARONI

Impaginazione
BRUNO DE CHECCHI
PAOLA ROZZA

Programmazione Editoriale
ROSANNA ZERBARINI
GIOVANNA BREGGÉ

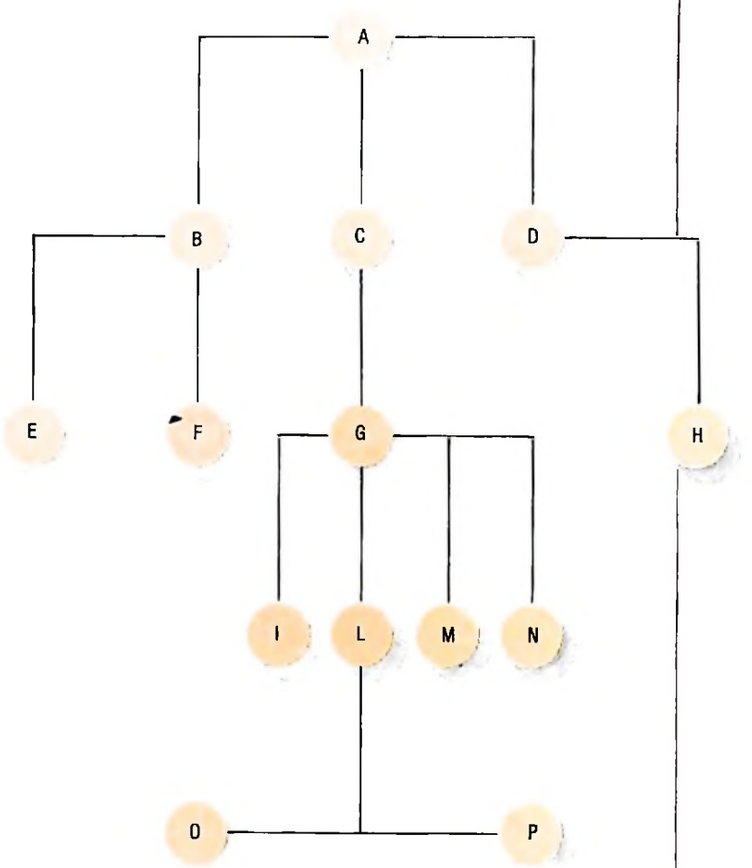
Segretaria di Redazione
RENATA FRIGOLI
LUCIA MONTANARI

# GLI ALBERI

## Un'importante classe di grafi, utile in diverse applicazioni.

### La struttura di albero

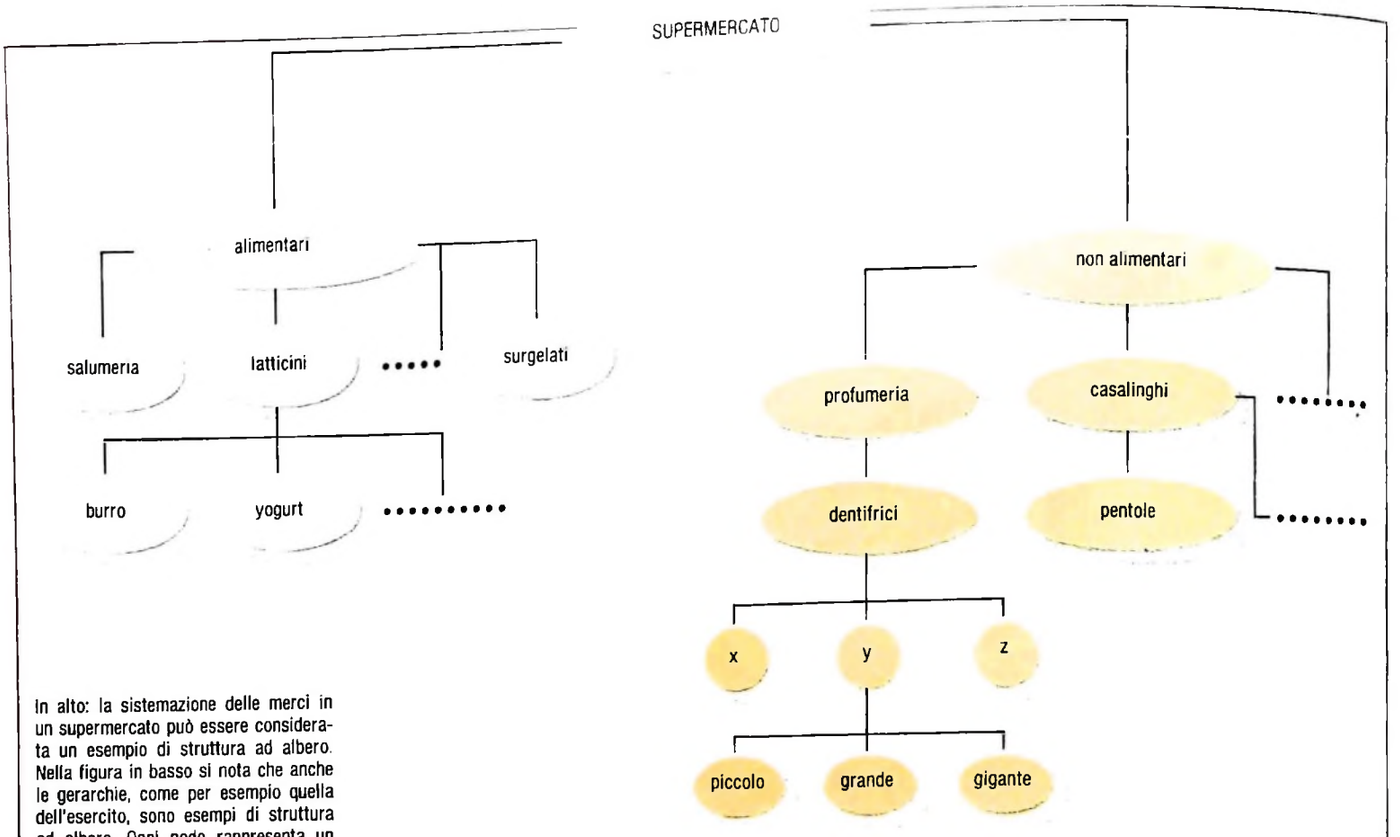
Un'importante classe di grafi orientati è quella degli alberi. Un *albero* può essere definito come un grafo orientato in cui:
a) esiste uno e un solo nodo n, detto *radice* dell'albero, privo di archi entranti;
b) in ogni nodo diverso dalla radice entra uno e un solo arco;
c) per ogni nodo n esiste un cammino dalla radice al nodo n;
usando la condizione b), si vede facilmente che tale cammino è unico.
La condizione c) ci consente di definire la *profondità* di un albero come la lunghezza del più lungo cammino che congiunge la radice a un qualsiasi nodo.
Di solito, gli alberi vengono rappresentati graficamente collocando la radice in alto e quindi man mano, a strati o livelli successivi sotto di essa, i nodi raggiungibili dalla radice con un cammino di lunghezza 1 (cioè costituiti da un solo arco), 2, 3 e così via. Alternativamente, si può collocare la radice in basso e gli altri nodi a livelli sopra di essa. Ovviamente, il numero di livelli è pari alla profondità dell'albero più uno (il livello zero, che contiene solo la radice). Il grafo nella figura a fianco è un albero.
I termini albero e radice derivano evidentemente dal fatto che un grafo come quelli che stiamo definendo somiglia, sia pure in forma stilizzata, a un albero reale, con la radice e il tronco da cui escono i rami; con la stessa analogia, i nodi da cui non esce nessun arco si dicono *foglie* dell'albero.
Nell'albero della figura a lato, la radice è il nodo A, le foglie sono i nodi E, F, H, I, M, N, O, P. Gli altri nodi si dicono *nodi interni*.
Poiché la struttura ad albero più antica è l'albero genealogico, altri termini sono derivati da questo uso. Così, se esiste un arco da X a Y, X si dice *padre* di Y e Y *figlio* di X; se esiste un cammino da X a Y, X si dice *antenato* di Y e Y *discendente* di X. Nell'albero dell'esempio, C è padre di G, H è figlio di D, A è antenato di L, P è discendente di G.
Dalla definizione data è facile ricavare che un albero è un grafo connesso senza cicli. Infatti, poiché ogni nodo è collegato con la radice, da un nodo V possiamo raggiungere qualunque altro nodo Z risalendo fino alla radice e quindi raggiungendo Z; inoltre, non possono esservi cicli poiché ciò implicherebbe la presenza di nodi con due archi entranti. Viceversa, ogni grafo connesso senza cicli in cui si indichi un nodo come radice può essere visto come albero. Possiamo così dare una definizione alternativa: possiamo cioè dire che un albero è un grafo connesso senza cicli con un nodo speciale detto radice.




Esempio di grafo orientato sotto forma di albero di profondità. Si tratta della rappresentazione classica, con la radice in alto e i nodi posti in livelli successivi sotto di essa.

### Strutture organizzate ad albero: esempi

Gli alberi sono uno dei più importanti strumenti utilizzati per rappresentare strutture di dati, cioè insiemi di informazioni collegate tra di loro secondo un preciso schema logico, in diversi settori, di cui possiamo ricordare i principali.
Il primo esempio è costituito dagli schemi di classificazione. Le più note classificazioni si trovano nelle scienze naturali, ma la classificazione e il ritrovamento di oggetti o informazioni sulla base di una struttura ad albero si ritrovano in numerose situazioni quotidiane. Per esempio, la disposizione delle merci in un supermercato è organizzata ad albero, i cui rami sono i vari reparti e sottoreparti, fino ad arrivare alle foglie che sono i singoli articoli. Avremo quindi un reparto




In alto: la sistemazione delle merci in un supermercato può essere considerata un esempio di struttura ad albero. Nella figura in basso si nota che anche le gerarchie, come per esempio quella dell'esercito, sono esempi di struttura ad albero. Ogni nodo rappresenta un singolo individuo, ed è gerarchicamente superiore ai nodi figli.

alimentari e un reparto non alimentari; all'interno di questo avremo un reparto profumeria, organizzato a scaffali tra cui quello dei dentifrici, a loro volta suddivisi per marca e per formato (figura in alto).

Un secondo esempio è costituito dalle strutture gerarchiche, in cui ogni nodo rappresenta un singolo individuo, ed è gerarchicamente superiore ai nodi figli. La figura a destra rappresenta la struttura gerarchica dell'esercito italiano, compreso il vertice politico. Struttura analoga hanno gli organigrammi aziendali.

Anche le espressioni algebriche possono essere rappresentate sotto forma di albero, evidenziando l'ordine di esecuzione delle operazioni che in esse compaiono; questa è proprio la forma in cui vengono idealmente rappresentate all'interno di un elaboratore.

Supponiamo di voler calcolare il valore assunto dall'espressione algebrica

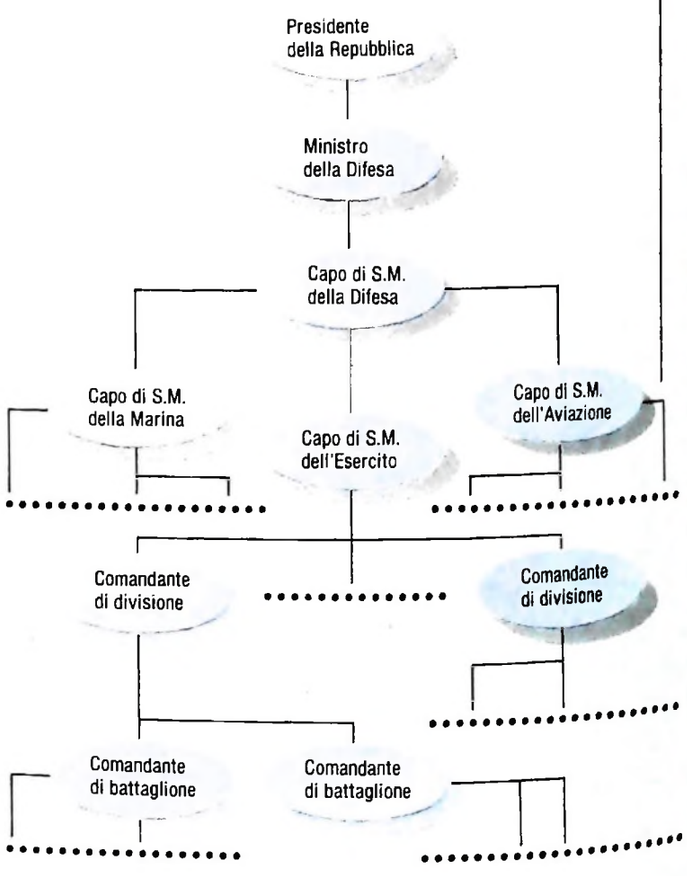
$$(x+y*z) /[w + (x + z) * (y-z)]$$

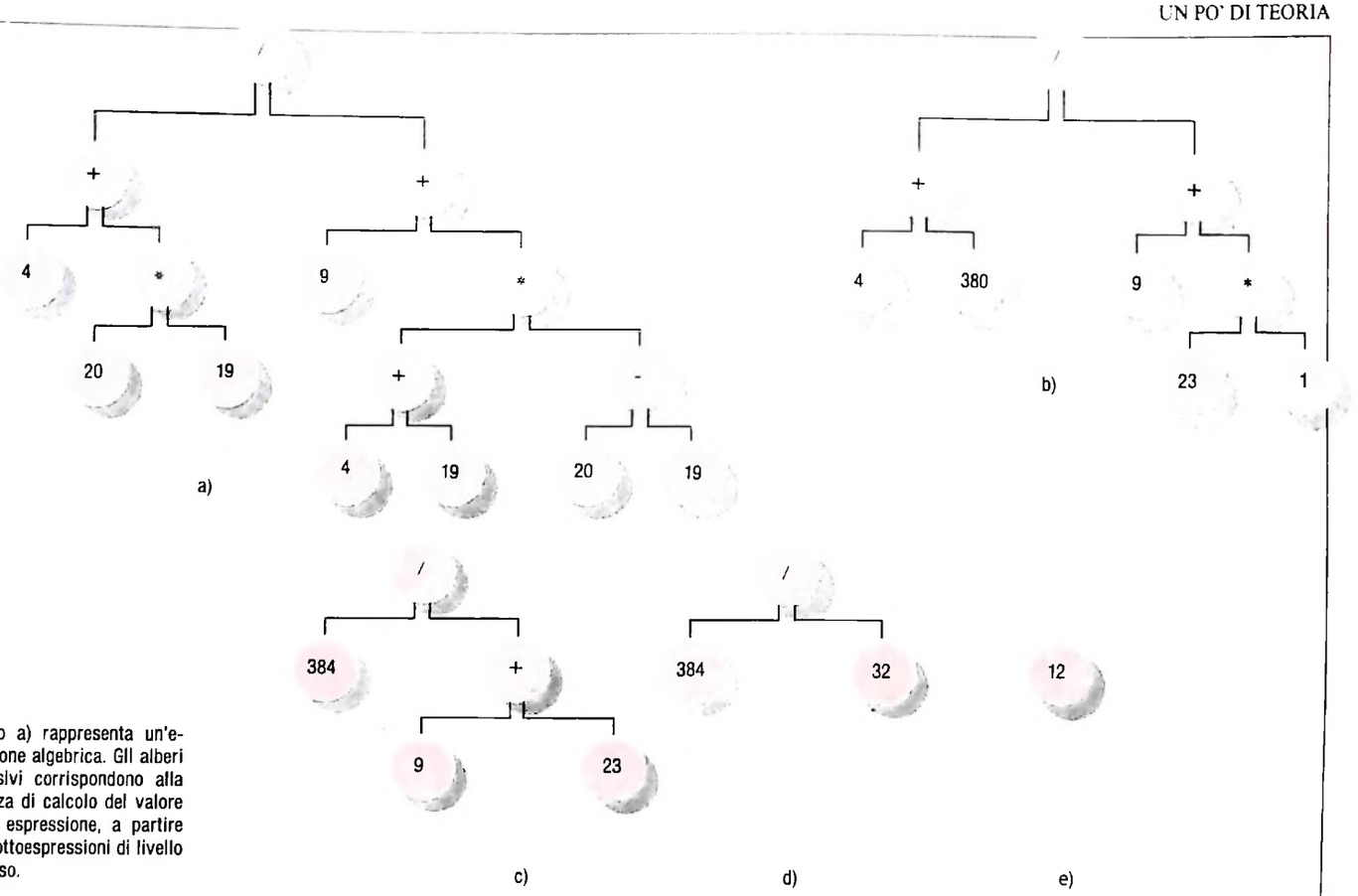
quando si assegnino ai simboli letterali dei valori numerici come per esempio: x=4, y=20, z=19, w=9.

Il calcolo viene eseguito partendo dalle sottoespressioni più interne, cioè y*z, x+z e y-z, dopo aver assegnato i valori indicati ai simboli letterali, e utilizzando poi i risultati parziali ottenuti per eseguire le operazioni più esterne:

$(4+20*19)/[9+(4+19)*(20-19)] = (4+380)/[9+23*1] = 384/32 = 12$

Il risultato, nel nostro caso, viene raggiunto in tre passaggi. L'ordine in cui devono essere eseguite le operazioni, che è






L'albero a) rappresenta un'espressione algebrica. Gli alberi successivi corrispondono alla sequenza di calcolo del valore di tale espressione, a partire dalle sottoespressioni di livello più basso.

stabilito dalle parentesi e dalle regole di precedenza ben note a chiunque conosca l'aritmetica, non risulta ovvio per un calcolatore. Un modo per stabilire l'ordine di esecuzione senza alcuna ambiguità, adottato praticamente da tutti i compilatori nel corso della traduzione di una espressione algebrica in un programma eseguibile dalla macchina, consiste nel rappresentare l'espressione da calcolare sotto forma di albero. Ogni nodo interno corrisponde a una operazione, e i suoi figli agli operandi. Più precisamente, ciascun figlio può essere una foglia, nel qual caso ha associato direttamente un numero, oppure un altro nodo interno, che identifica una sottoespressione.

L'espressione viene calcolata a partire dalle foglie; a ogni passo, si eseguono le operazioni al livello più basso, e il valore calcolato per ognuna viene sostituito al nodo interno corrispondente, che diventa una foglia; in tal modo a ogni passata si riduce di uno la profondità dell'albero, fino a raggiungere la radice, che corrisponde all'operazione più esterna, l'ultima a essere eseguita. Nella figura in alto è rappresentata la sequenza di alberi associata al calcolo dell'espressione data come esempio.

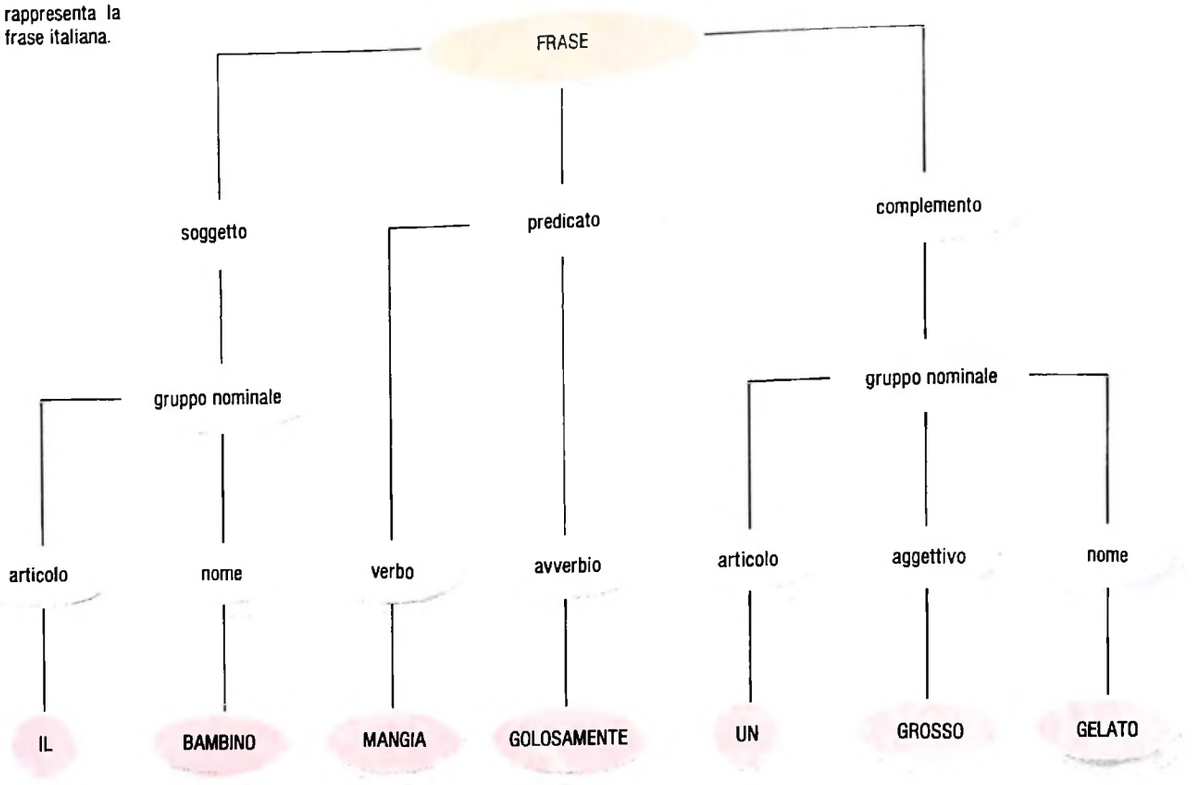
Questo modo di rappresentare le espressioni aritmetiche non è che un caso particolare di uso della struttura ad albero per rappresentare frasi o formule corrette in un linguaggio (in questo caso il linguaggio dell'aritmetica). L'idea di rappresentare la struttura delle frasi attraverso alberi è dovuta ai linguisti, nel tentativo di trovare uno strumento che consentisse di esprimere in modo sintetico e rigoroso le regole sintattiche e grammaticali delle lingue cosiddette naturali, cioè quelle che si parlano comunemente nei diversi paesi del mondo.

Consideriamo per esempio una frase italiana come "il bambino mangia golosamente un grosso gelato". Possiamo dire che questa frase è costituita da un soggetto, da un predicato e da un complemento oggetto. A loro volta, il soggetto e il complemento oggetto sono costituiti da un articolo, eventualmente seguito da un aggettivo, e da un nome, mentre il predicato è un verbo seguito da un avverbio. Possiamo allora fornire le seguenti regole, ovviamente incomplete:

a) una frase italiana è costituita da un soggetto, un predicato e un complemento;
b) il soggetto e il complemento sono gruppi nominali;
c) un gruppo nominale è costituito da un nome, oppure un articolo seguito da un nome, oppure un articolo seguito da un aggettivo seguito da un nome;
d) gli articoli possibili sono: il, lo, la, un, uno, ...;
e) i nomi possibili sono: bambino, gatto, casa, gelato, ...,
f) gli aggettivi possibili sono: bello, grosso, noioso, ...;
g) un predicato è costituito da un verbo oppure un verbo seguito da un avverbio;
h) i verbi possibili sono: mangia, costruisce, costruivo, costruiremo, ...;
i) gli avverbi possibili sono: golosamente, ...

Come si può vedere, si parte dall'oggetto più complesso, la frase, e si dice da quali oggetti più semplici è costituito, fino ad arrivare alle singole parole. Questo procedimento è rap-

Un albero che rappresenta la struttura di una frase italiana.




presentato molto chiaramente da un albero in cui la radice rappresenta la frase, e i figli di ogni nodo sono gli oggetti più semplici in cui esso è decomponibile. Nella figura in alto si può vedere l'albero che rappresenta la struttura della frase che abbiamo usato come esempio.

Benché questo tipo di approccio si sia rivelato inadeguato per trattare in modo completo le lingue naturali, esso funziona perfettamente per i cosiddetti linguaggi artificiali, tra cui vi sono i linguaggi di programmazione. In effetti, sia la definizione sia il processo di traduzione in linguaggio macchina dei linguaggi di programmazione ad alto livello sono basati essenzialmente sulla rappresentazione ad albero della struttura dei programmi e delle istruzioni.

## Alberi supporto per grafi

Vediamo ora un importante problema che può essere riformulato in termini di grafi e alberi.

Si supponga di voler collegare un certo numero di città con delle linee telefoniche in modo che da ogni città si possa telefonare in ogni altra, e che la lunghezza complessiva delle linee sia minima.

Per affrontare il problema, possiamo cominciare a costruire un modello in cui ogni città è rappresentata con un nodo in un grafo, e tutti i possibili collegamenti sono rappresentati con archi, a ognuno dei quali è associato un peso, che indica la lunghezza, o il costo, della linea telefonica corrispondente. Un grafo di questo tipo, in cui ogni nodo è collegato direttamente con ogni altro, si dice *completo*.

Si tratta ora di eliminare un certo numero di archi, fino a ottenere un grafo in cui sia mantenuto il collegamento tra due vertici qualsiasi, anche se non più necessariamente diretto, e inoltre la somma dei pesi degli archi rimasti sia minima. Tale grafo deve essere connesso, per soddisfare la prima di queste richieste, e non può contenere cicli, poiché se ne contenesse uno il costo complessivo non sarebbe minimo: sarebbe infatti possibile eliminare uno degli archi del ciclo senza isolare nessuna città, ottenendo così una rete di collegamenti di costo totale più basso.

Poiché un grafo connesso senza cicli è un albero, possiamo riformulare il problema dicendo che, dato un grafo pesato non orientato e connesso G, vogliamo trovare un albero A i cui nodi siano esattamente i nodi di G e i cui archi siano un sottoinsieme di peso totale minimo degli archi di G. Un albero con queste proprietà si dice *albero supporto minimo* per il grafo G. Nella pagina a lato sono mostrati un grafo (a) e due alberi supporto per esso (b, c); il grafo (d) non è invece un albero supporto per G, poiché non è connesso.

Per costruire un albero supporto minimo per un grafo connesso assegnato G, si può procedere come segue.

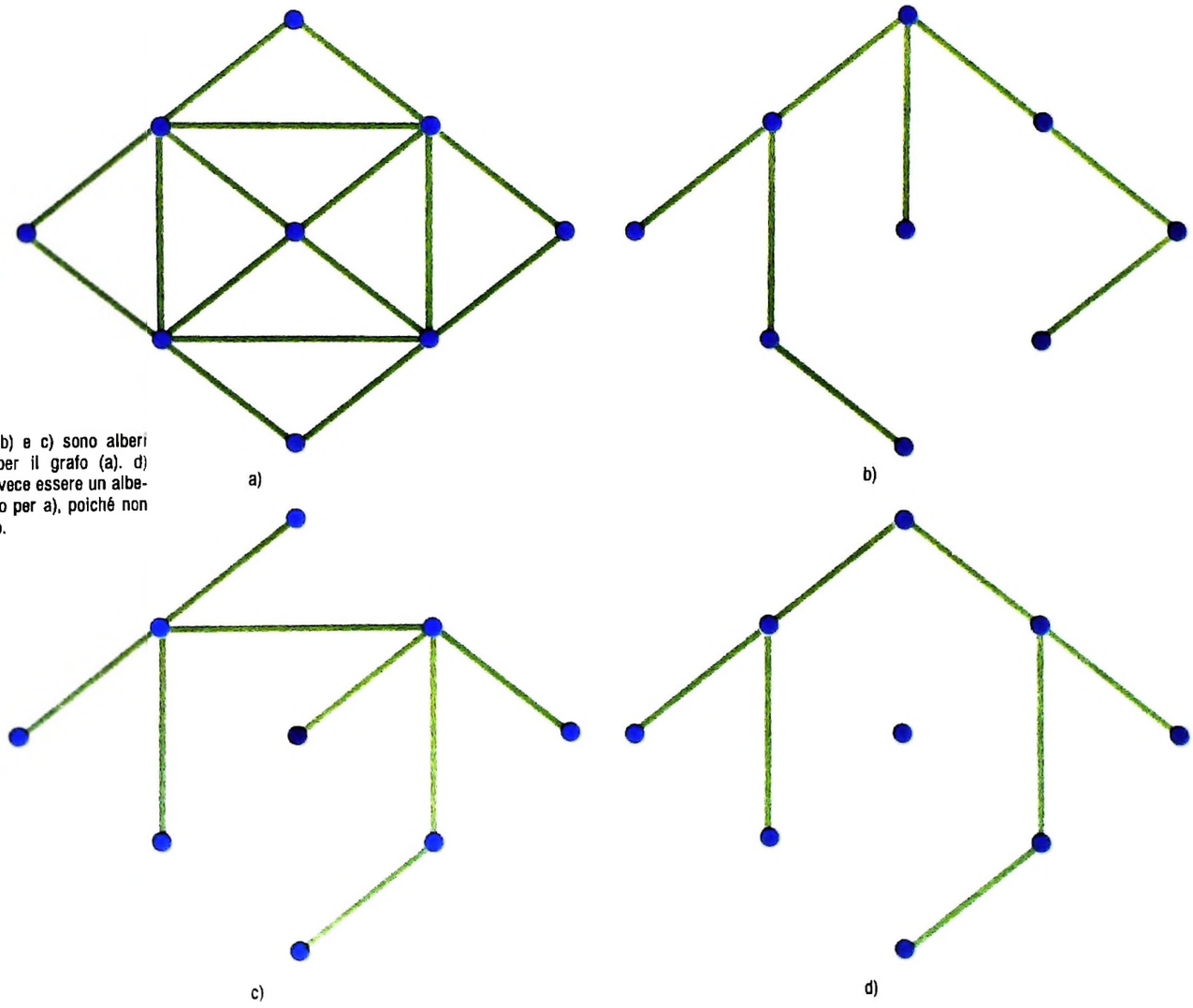
Anzitutto si sceglie un arco tra quelli (che possono essere più di uno) di peso minimo. Si procede poi scegliendo a ogni passo uno degli archi di peso minimo tra quelli che non in-

troducono cicli nel grafo costituito dagli archi scelti in precedenza. Ci si arresta quando tutti i vertici sono stati collegati. Poiché il sottografo A così costruito è connesso, non contiene cicli e contiene tutti i nodi di G, esso è un albero supporto per G.

Ci resta da dimostrare che nessun altro tra gli alberi supporto di G, che sono in generale più di uno, può avere un peso totale inferiore.

Supponiamo infatti che esista un albero supporto minimo A' diverso da A. Allora esiste almeno un arco a che sta in A ma non in A'. Se aggiungiamo tale arco ad A' otteniamo un ciclo, poiché i nodi V e Z che esso collega direttamente erano già raggiungibili l'uno dall'altro con un cammino indiretto; di conseguenza, questo ciclo deve contenere almeno un arco b che non appartiene all'albero A, che possiamo togliere, ottenendo un nuovo albero A''. Ora, non è possibile che il costo di b sia inferiore al costo di a, poiché altrimenti l'albero A'' ottenuto sostituendo a con b avrebbe costo inferiore ad A', contro l'ipotesi che A' sia minimo. D'altra parte, poiché a appartiene ad A, che è l'albero costruito prendendo a ogni passo l'arco di costo minimo che non introduce cicli, il costo di a non può essere superiore a quello di b, per cui i loro costi devono coincidere. Quindi la sostituzione di a con b non cambia il costo complessivo dell'albero. A questo punto, se il nuovo albero A'' contiene archi non contenuti in A possiamo ripetere il ragionamento, ottenendo un nuovo albero dello stesso costo. Possiamo così procedere fino a eliminare tutti gli archi non appartenenti ad A senza alterare il costo degli alberi ottenuti, per cui A è effettivamente un albero di costo minimo.

Problemi di questo tipo, legati per esempio alla collocazione ottimale delle autoambulanze in una città per ridurre i tempi medi di percorrenza, alla ottimizzazione delle linee di comunicazione e così via si ritrovano frequentemente nel campo della ricerca operativa, e vengono spesso risolti costruendo modelli a grafo o ad albero e progettando quindi algoritmi di soluzione efficienti. È questo quindi un altro importante ambito di uso dei grafi e degli alberi.




Gli alberi b) e c) sono alberi supporto per il grafo (a). d) non può invece essere un albero supporto per a), poiché non è connesso.

# IL SISTEMA OPERATIVO UCSD P-SYSTEM (I)

## Di piccole dimensioni, veloce e semplice da usare, risponde alle esigenze di un'utenza non specialistica.

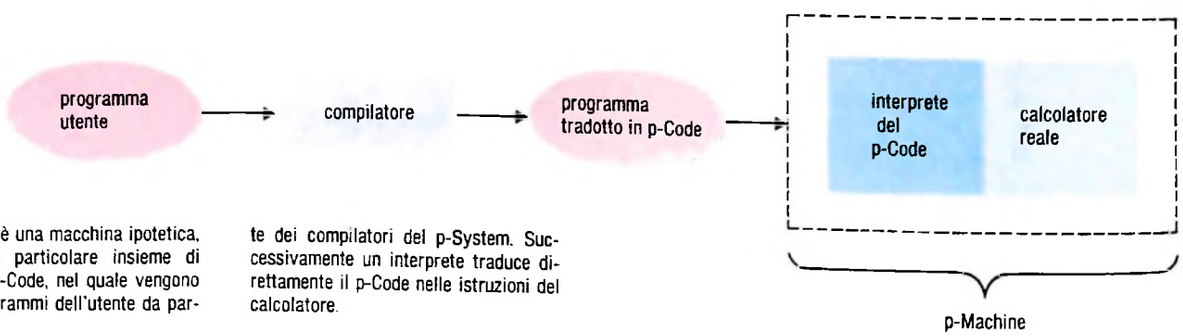
Con la diffusione dei piccoli calcolatori, negli ultimi dieci anni, si è venuta facendo sempre di più sentire la necessità di sistemi operativi orientati alla produzione e all'esecuzione di programmi per questa classe di macchine. Ciò che caratterizza un tale ambiente è in primo luogo una limitata quantità di memoria centrale e una bassa velocità di calcolo della CPU, e inoltre un'utenza generalmente non specialistica (sensibile quindi al problema della facilità di uso), che utilizza il calcolatore come sistema dedicato (monoutente) eseguendo un solo programma alla volta (*single-task*). In questo contesto si colloca il sistema operativo UCSD p-System.

### La storia di UCSD p-System.

Il sistema è stato sviluppato all'Università di San Diego in California (da cui il nome del sistema: University of California, San Diego) sotto la direzione del Professor Kenneth Bowles, a partire dal 1974, da un gruppo di studenti fra i quali Mark Overgaard ha svolto il principale ruolo tecnico.
Lo scopo originale del sistema era quello di mettere a disposizione degli studenti del corso fondamentale di *computer science* un ambiente di sperimentazione adeguato. Da qui la scelta di utilizzare piccoli calcolatori a uso individuale, piuttosto che grossi calcolatori multi-utente, e la scelta del linguaggio Pascal sia come linguaggio utilizzato dagli studenti per le sperimentazioni, sia come linguaggio per realizzare le applicazioni base del sistema operativo stesso. Questo sistema doveva quindi necessariamente essere semplice da usare da parte di studenti alle prime armi, ma anche adeguato all'utilizzo da parte di esperti.
Tutte queste caratteristiche hanno determinato sia la nascita sia la successiva evoluzione di UCSD p-System.

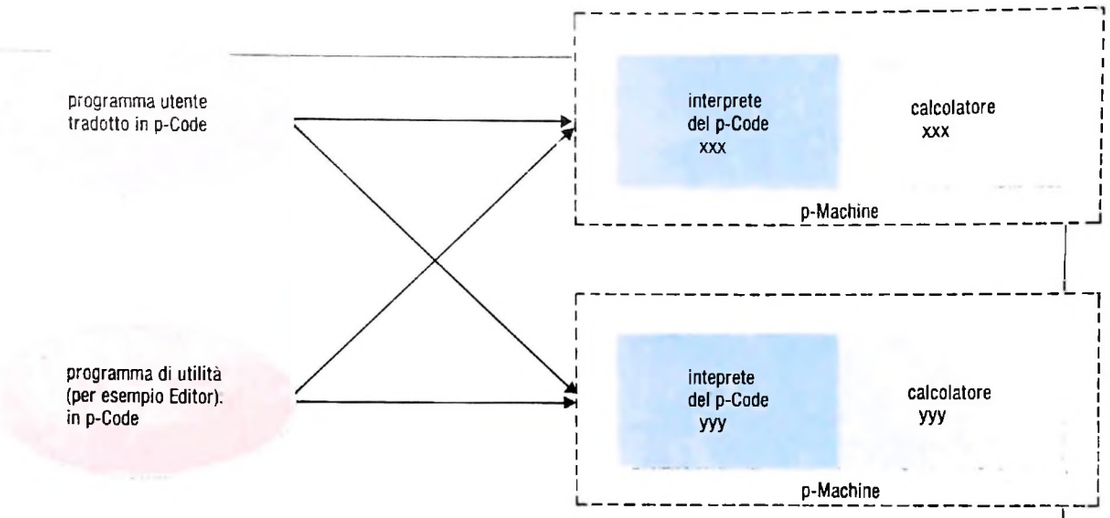
### La p-Machine e la portabilità

L'idea base dell'architettura di UCSD p-System è il concetto di *p-Machine*. La p-Machine è una macchina ipotetica, dotata di un particolare insieme di istruzioni, detto *p-Code*: i compilatori del p-System traducono i programmi dell'utente in p-Code. In fase di esecuzione, un interprete simula il comportamento della p-Machine, traducendo direttamente il p-Code nelle istruzioni del calcolatore reale (figura in basso).
I vantaggi di tale architettura sono fondamentalmente due: il primo consiste in una riduzione di complessità (e quindi di dimensione e di occupazione di memoria) dei compilatori, dal momento che è più semplice tradurre un programma in p-Code che nelle istruzioni base della macchina; il secondo sta nella possibilità di trasportare praticamente l'intero p-System e le applicazioni utente su un altro calcolatore, semplicemente riscrivendo l'interprete del p-Code (figura della pagina a fianco, in alto). Questa caratteristica del sistema operativo prende il nome di portabilità.




La p-Machine è una macchina ipotetica, dotata di un particolare insieme di istruzioni, il p-Code, nel quale vengono tradotti i programmi dell'utente da parte dei compilatori del p-System. Successivamente un interprete traduce direttamente il p-Code nelle istruzioni del calcolatore.

La caratteristica fondamentale dell'UCSD p-System consiste nella portabilità. Infatti, semplicemente riscrivendo l'interprete del p-Code, è possibile trasportare l'intero p-System su un altro calcolatore.




## I linguaggi di programmazione

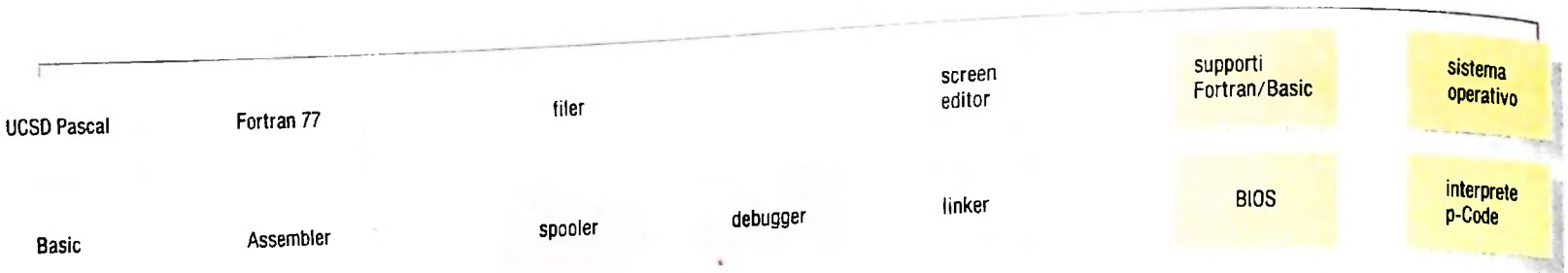
Utilizzando il p-System, l'utente ha a disposizione quattro linguaggi di programmazione: UCSD-Pascal, Fortran-77, Basic e Assembler (quest'ultimo dipende ovviamente dal calcolatore reale utilizzato).
Il più importante di questi è certamente UCSD-Pascal, anche perché gran parte del sistema operativo stesso è stato scritto in questo linguaggio.
UCSD-Pascal è un'estensione del linguaggio Pascal, creato da Niklaus Wirth. Esso include tutte le caratteristiche più salienti del Pascal, come la strutturazione dei dati, le strutture di controllo (WHILE... DO, REPEAT... UNTIL, ecc.) e dispone, inoltre, di primitive supplementari per la manipolazione delle stringhe di caratteri, per la gestione delle periferiche e la simulazione di processi concorrenti. Sono possibili anche compilazioni condizionali e compilazioni separate.
Il linguaggio Pascal è stato infatti criticato per la mancanza di uno strumento che permetta la compilazione separata di porzioni di programma: linguaggi più recenti, come Ada, lo consentono in quanto esigenza fondamentale per la progettazione di programmi di grandi dimensioni. UCSD-Pascal supera questo limite del Pascal standard mediante lo strumento delle *UNIT*.
Una *unit* UCSD consiste di due parti principali: *interface* e *implementation*. Nella parte *interface* è definito un insieme di servizi che la *unit* mette a disposizione di qualsiasi programma che la utilizzi; nella parte *implementation* viene descritta la effettiva realizzazione di tali servizi. I programmi che utilizzano una *unit* hanno pieno accesso alla parte *interface*, ma non alla parte *implementation*. In questo modo i servizi forniti da una *unit* possono essere migliorati o messi a punto senza dover toccare alcuno dei programmi che li utilizzano: ciascuna *unit* può essere indipendentemente sviluppata, compilata e provata e solo alla fine incorporata nel sistema.
Una tecnica analoga viene utilizzata per i programmi scritti in FORTRAN-77, in modo tale che programmi Pascal possono utilizzare i servizi offerti da *unit* FORTRAN e viceversa, ed è possibile utilizzare *routine* assembler con entrambi i linguaggi.

## I calcolatori

Il p-System richiede, nella sua configurazione standard, almeno 64K di memoria centrale (sebbene ne siano stati fatti adattamenti per macchine più piccole, come il Radio Shack TRS-80) e una unità floppy disk. La sua architettura basata sulla p-Machine ne ha consentito la realizzazione su numerosi calcolatori per lo più basati su processor a 8 e 16 bit fra cui Apple II, Osborne I, IBM PC, Commodore, TRS 80, Ohio Scientific, Xerox, Hewlett Packard, Philips, Texas Instruments, Zenith, DEC Professional, Olivetti M20.

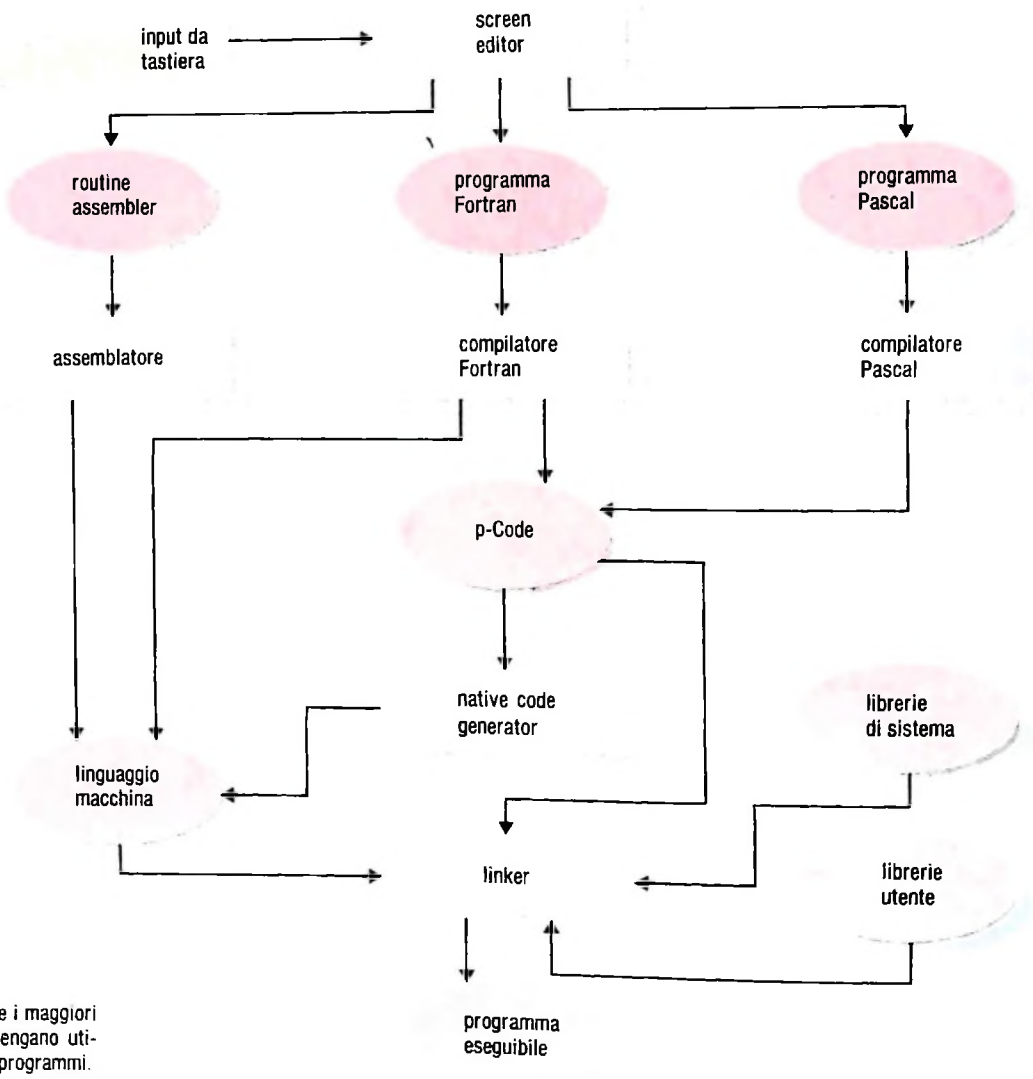
## I componenti del p-System

Il p-System comprende, oltre al sistema operativo vero e proprio, un gran numero di servizi (figura di pagina seguente, in alto) molti dei quali richiamabili, come vedremo, premendo un singolo tasto.
Vi sono quattro compilatori (Pascal, Fortran, Basic più l'assemblatore) che traducono i programmi scritti dall'utente nei rispettivi linguaggi in p-Code e/o direttamente in linguaggio macchina; numerosi servizi di utilità (*tool*) tra cui il *filer*, che consente la gestione dei file su disco; lo *screen editor*, un programma per la composizione e la modifica di testi (orientato al trattamento di programmi) dotato di vari comandi di ricerca, sostituzione, eccetera; lo *spooler*, che consente di effettuare la stampa di un testo parallelamente alla sessione di lavoro; il *debugger*, che consente raffinate prove di esecuzione per la messa a punto di programmi; il *native-code-generator*, che effettua la traduzione diretta del p-Code in codice macchina, allo scopo di renderne più veloce la successiva esecuzione; diversi altri servizi, contenuti nelle librerie del sistema sotto forma di *unit*, richiamabili dai programmi applicativi (per esempio servizi grafici); il *linker*, che consente di includere in un programma compilato le *unit* e le *routine* di cui necessita.
Vi sono infine, i supporti *run-time*, vale a dire quei componenti del sistema che, pur non direttamente richiamabili dall'utente, consentono il funzionamento di tutti i servizi e lo




In figura sono mostrati i servizi che il p-System comprende: quattro compilatori, un gran numero di servizi di utilità (tool) e i supporti run-time, non richiamabili direttamente dall'utente.

sfruttamento delle risorse del calcolatore: i supporti FORTRAN e BASIC consistono in un insieme di *routine*, invisibili al programmatore, utilizzate in fase di esecuzione di programmi scritti in questi linguaggi; il sistema operativo vero e proprio, che realizza numerose ed essenziali funzionalità invisibili all'utente (ad esempio la gestione della memoria); l'interprete del p-Code (detto anche *p-Machine emulator*) che come abbiamo visto è l'effettivo esecutore dei programmi tradotti in p-Code; infine il BIOS (Basic Input Output System): una collezione di *routine* per la gestione dell'input-output su tutte le periferiche (video, tastiera, stampante, unità floppy-disk ecc.). Nella figura in basso vediamo come i maggiori componenti del p-System vengono utilizzati nella preparazione di programmi. Il compilatore Fortran produce p-Code e codice macchina mescolati: questo non costituisce un problema in quanto l'interprete del p-Code è in grado di riconoscere il codice macchina e di farlo eseguire direttamente senza interpretarlo.




Nella figura è mostrato come i maggiori componenti del p-System vengano utilizzati nella preparazione di programmi.

*Lezione 55*

## La gestione degli errori

Nelle ultime lezioni abbiamo imparato a costruire programmi "robusti" attraverso la lettura dei dati numerici forniti in ingresso sotto forma di stringhe di caratteri, e il relativo controllo di correttezza della forma sintattica o dei valori, segnalando e gestendo poi eventuali errori. Esistono nel BASIC dell'M10 un certo insieme di potenti istruzioni per centralizzare tale gestione.
Esaminiamo ciò che accade quando noi costruiamo un programma sbagliato. Esso viene normalmente eseguito secondo le indicazioni delle istruzioni di cui è composto, ma quando viene incontrato un errore non è più seguita la normale sequenza, il controllo viene "strappato" al programma e ceduto a chi lo sta interpretando (cioè, in definitiva, il programma che realizza la capacità di interpretare i programmi scritti in BASIC, sempre presente nell'M10), che provvede a segnalare l'errore e a interrompere l'esecuzione.
Anche a noi potrebbe fare comodo una simile tecnica: se in un programma come quello "robusto" che abbiamo costruito nelle precedenti lezioni potessimo disporre di un'istruzione che ci indica l'occorrenza di un errore, potremmo connotare diversamente i vari errori e demandare a una particolare routine la loro segnalazione.
Nel nostro BASIC ciò è possibile con la coppia di istruzioni ON e ERROR:
• l'istruzione ERROR, infatti, segnala l'occorrenza di un errore, di cui fornisce un numero identificativo:

ERROR 117

Sta a indicare qualcosa del tipo "dichiaro che è occorso un errore, in quanto i valori che ho ottenuto durante l'elaborazione non sono congruenti con quelli che mi aspettavo; identifico tale errore con il numero 117".
Tale istruzione ha come unico effetto quello di "far avvertire" al BASIC la presenza di un errore, di assegnare il valore del numero che segue la parola ERROR a una variabile predefinita di nome ERR e di segnalarne l'occorrenza mediante la stampa di un opportuno messaggio.
Così, per esempio, l'istruzione

```
10 ERROR 100
```

provoca la visualizzazione del messaggio

```
run
?UE Error in 10
Ok
```

che sta a indicare "Unprintable Error" (Errore non stampabile in chiaro, in quanto non previsto dal BASIC) incontrato alla linea 10.
Il meccanismo seguito è il medesimo che il BASIC usa per le proprie segnalazioni,

otteniamo il messaggio

```
10 ERROR 1
```

tanto che, se eseguiamo l'istruzione

```
run
?NF Error in 10
Ok
```

che è l'indicazione di un "NEXT senza FOR", emesso non perché la condizione sia occorsa, ma perché il codice d'errore 1 è usato dal BASIC per tale errore.
I codici d'errore usati dal BASIC nell'M10 sono elencati nella seguente tabella:

**CODICI D'ERRORE**

| CODICE | SIGLA | SIGNIFICATO |
|---|---|---|
| 1 | NF | NEXT senza FOR |
| 2 | SN | Errore sintattico |
| 3 | RG | RETURN senza GOSUB |
| 4 | OD | Dati insufficienti |
| 5 | FC | Chiamata funzione illegale |
| 6 | OV | Overflow |
| 7 | OM | Memoria insufficiente |
| 8 | UL | Linea non definita |
| 9 | BS | Indice fuori dai limiti |
| 10 | DD | Vettore ridimensionato |
| 11 | /0 | Divisione per zero |
| 12 | ID | Comando immediato illegale |
| 13 | TM | Tipo di variabile errato |
| 14 | OS | Spazio per stringhe insufficiente |
| 15 | LS | Stringa troppo lunga |
| 16 | ST | Espressione stringa troppo complessa |
| 17 | CN | Non possibile continuare |
| 19 | NR | Manca l'istruzione RESUME |
| 20 | RW | RESUME senza routine d'errore |
| 21 | UE | Errore non stampabile |
| 22 | MO | Operando mancante |
| 23 | IO | Errore di input/output |
| 24-49 | UE | Errore non stampabile |
| 50 | IE | Errore interno |
| 51 | BN | Errato numero di file |
| 52 | FF | File non trovato |
| 53 | AO | File già aperto |
| 54 | EF | Input oltre la fine del file |
| 55 | NM | Nome file errato |
| 56 | DS | Istruzione immediata in un file |
| 57 | FL | Troppi file |
| 58 | CF | File non aperto |
| 59-255 | UE | Errore non stampabile |

Da essa vediamo come il programmatore sia libero di adottare per i propri scopi i codici 21, da 24 a 49 e da 59 a 255.
L'altra istruzione, ON ERROR, è il naturale complemento alla ERROR; una volta che essa viene eseguita, indica l'istruzione a cui trasferire il controllo nel caso in cui

un errore venga incontrato; così. un programma costruito come segue:

```
10 ON ERROR GOTO 1000
.....
50 IF... THEN ERROR 100
.....
1000 IF ERR=100 THEN PRINT "Errore"
.....
```

si comporta come segue:

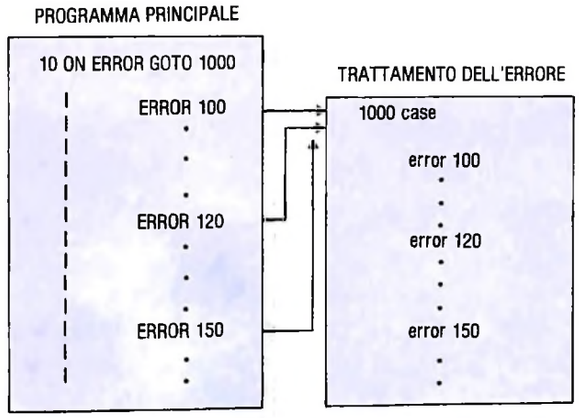
- all'istruzione 10 dichiara che, a fronte di eventuali istruzioni ERROR eseguite, il controllo dovrà essere trasferito all'istruzione 1000;
- all'istruzione 50 dichiara che, se vale una certa condizione, si deve considerare occorso un errore, identificato dal valore 100;
- all'istruzione 1000, cui viene trasferito il controllo senza alcuna segnalazione, si verifica il codice d'errore, e in caso che sia 100 viene stampato un messaggio deciso dal programmatore.

Perché fare tutta questa fatica? Forse non erano sufficienti gli strumenti che avevamo a disposizione precedentemente? Tutto contato, nella costruzione del nostro programma "robusto" ce la siamo cavata semplicemente con qualche istruzione IF e un po' di messaggi di stampa!

È vero, ma l'uso di ON ERROR e di ERROR permette una maggiore pulizia formale. Infatti, nel trattare gli errori è importante distinguere tre elementi:

- quali errori vengono presi in considerazione;
- dove essi vengono rilevati;
- dove essi vengono trattati.

Nel nostro programma "robusto" i tre aspetti non erano così ben comprensibili e localizzati, e trattamenti e rilevamenti erano mescolati al testo del programma. Con l'uso delle istruzioni indicate, invece, abbiamo una situazione come quella illustrata in figura:



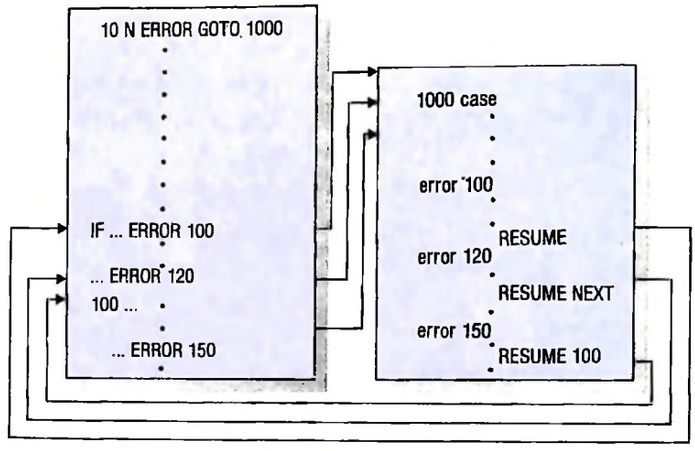

- il trattamento è concentrato nella parte dalla linea 1000 in poi;
- la rilevazione è evidenziata dalle istruzioni ERROR;
- la definizione di quali errori sono trattati è parzialmente identificata dai codici numerici usati per gli errori e deve essere completata con adeguati commenti.

Unico problema: che cosa accade dopo che il controllo dell'errore è stato effettuato e che l'eventuale messaggio d'errore è stato visualizzato?

Nel nostro programma "robusto", a fronte dell'individuazione di un errore era necessario rilevarlo, segnalarlo, ma anche proseguire con il programma per permetterne la correzione. A questo scopo è disponibile l'istruzione RESUME; dopo che è stata eseguita una ON ERROR all'inizio del programma e che l'occorrenza di un'istruzione ERROR ha "strappato" il controllo al programma per cederlo alla routine di trattamento dell'errore;

- l'istruzione RESUME fa riprendere l'esecuzione a partire dall'istruzione in cui la ERROR era stata incontrata;
- l'istruzione RESUME NEXT fa riprendere l'esecuzione dall'istruzione successiva a quella in cui la ERROR è stata incontrata;
- l'istruzione RESUME n fa riprendere l'esecuzione a partire dall'istruzione con numero di linea n.

Ecco quindi come si presenta la nuova situazione, con l'uso di RESUME:




Nella lezione successiva altereremo il programma "robusto" congruentemente all'uso delle istruzioni viste.

Il primo caso può essere adottato per un errore che la routine giudica "correggibile" direttamente; supponiamo di incontrare in un programma una divisione per zero; tale errore può causare l'interruzione del programma, che potrebbe in realtà avere una parte dei risultati ugualmente valida; perché allora non segnalare che una parte dei risultati è senza dubbio scorretta, ma permettere la prosecuzione dell'esecuzione? Semplice: basta porre a 1 la variabile che conteneva il divisore a zero, segnalare tale fatto, e usare RESUME.

Il secondo caso è più tipico: si tratta di effettuare nella routine l'operazione fallita, e quindi di proseguire come se nulla fosse accaduto.

Il terzo caso è analogo al secondo; si tratta però di rinviare, con l'istruzione RESUME, direttamente all'istruzione che richiede nuovamente la riesecuzione dell'operazione fallita, piuttosto che non delegare tale compito all'istruzione di salto successiva alla ERROR. Il lettore attento avrà già capito che questo modo di usare RESUME ci convince meno degli altri, in quanto non si propone come un modo "strutturato" di programmare, ma permette di saltare da una parte di trattamento degli errori (che nulla dovrebbe sapere sul programma che l'ha invocata) a una specifica istruzione del programma principale.

## Cosa abbiamo imparato

In questa lezione abbiamo visto:

- la gestione degli errori, con attenzione a:
  — loro identificazione
  — loro rilevamento
  — loro trattamento;
- l'uso dell'istruzione ON ERROR GOTO ... per definire una vera e propria routine di trattamento dell'errore;
- l'uso dell'istruzione ERROR... per indicare l'occorrenza di un errore;
- l'uso dell'istruzione RESUME... per proseguire dopo che un errore è stato incontrato e trattato;
- la lista di tutti i codici d'errore già usati dal BASIC e il relativo significato.

# UN PACCHETTO GRAFICO PER M10

## Un semplice programma per la modellazione di oggetti tridimensionali.

Nelle ultime lezioni abbiamo rapidamente analizzato le problematiche inerenti la rappresentazione e la modellazione di un oggetto tridimensionale in un sistema bidimensionale quale il monitor di un calcolatore. Abbiamo cioè appreso come sia possibile rappresentare un solido su un piano e quindi operare su di esso con operazioni tali da modificare le sue caratteristiche originali.
Scopo della presente lezione è quello di fornire le indicazioni indispensabili allo studio e quindi alla realizzazione di un software in grado di fornire un utile supporto per la modellazione di oggetti tridimensionali. Si cercherà quindi di mettere in risalto la suddivisione del problema in blocchi principali, andando poi ad approfondire singolarmente le parti fondamentali per la stesura di un semplice programma per elaborazioni grafiche che includa tutte le operazioni trattate nelle lezioni precedenti.
Per creare un modello semplice, ma reale, del problema in questione, supponiamo di voler visualizzare dei generici solidi geometrici, quali, per esempio, una piramide, un prisma, un parallelepipedo ecc. Per raggiungere lo scopo è necessario seguire scrupolosamente diverse fasi esecutive; in prima analisi è opportuno operare, per esempio, una suddivisione del seguente tipo:
1) rappresentazione del solido: introduzione della figura all'interno del calcolatore;
2) operazioni disponibili: definizione delle funzioni necessarie allo studio dell'oggetto;
3) limitazioni della macchina: definizione degli eventuali problemi che possono insorgere durante la rappresentazione e la modellazione dell'oggetto a causa di caratteristiche proprie della macchina utilizzata.

### Rappresentazione del solido

Esistono diverse metodologie per l'introduzione di una figura nella memoria di un calcolatore; alcune in particolare richiedono l'ausilio di sofisticati sistemi di lettura delle coordinate che definiscono un oggetto tridimensionale, ma che noi non tratteremo; per il nostro scopo è in realtà sufficiente effettuare l'introduzione dell'oggetto utilizzando le coordinate cartesiane dei vertici che lo definiscono. A tal fine è necessario fissare un sistema di riferimento xyz e definire ogni oggetto che dobbiamo elaborare all'interno di questo riferimento. Effettuata questa operazione, che è purtroppo manuale, disponiamo delle coordinate dei vertici che definiscono l'oggetto. Potremo a questo punto affidare l'input delle coordinate a un semplice programma (editor), il quale non farà altro che permettere l'introduzione delle coordinate dei vertici, in un qualunque ordine definito dall'utente.
La riproduzione della figura è immediata: sarà infatti sufficiente congiungere la sequenza di punti con dei segmenti nello stesso ordine con cui sono stati introdotti.

### Operazioni disponibili

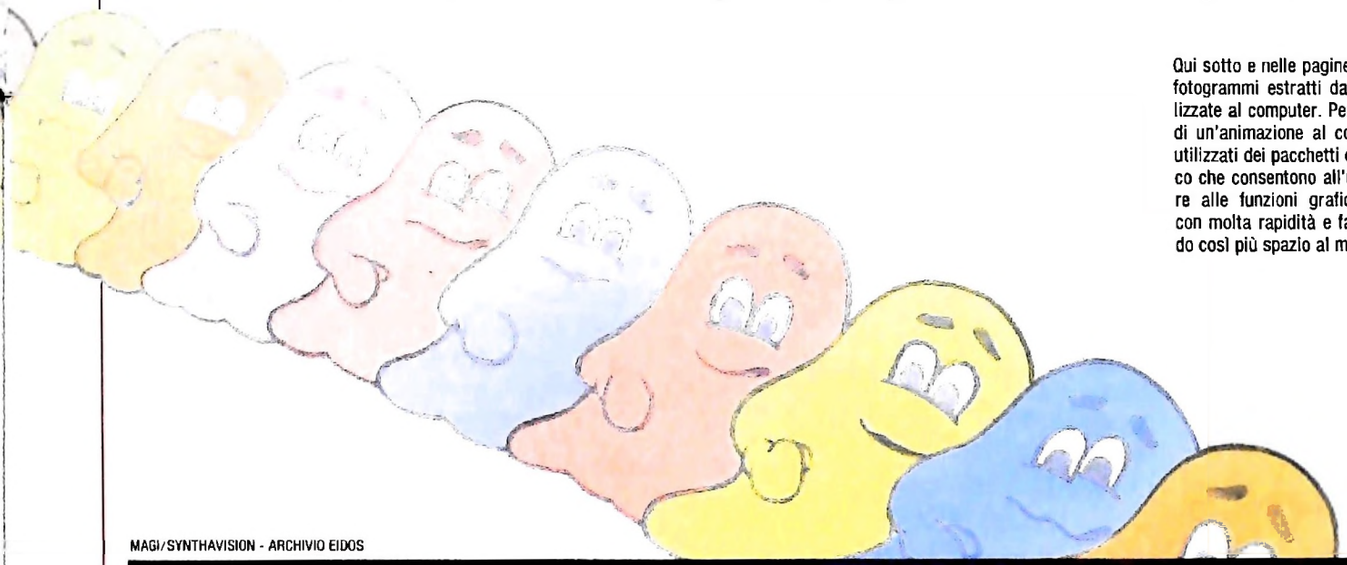
In questa sezione è richiesta l'analisi delle funzioni necessarie per compiere le elaborazioni volute. Allo scopo è quindi indispensabile definire cosa realmente si vuole ottenere dal programma; l'esempio proposto fornisce alcune tra le più importanti operazioni richieste in applicazioni grafiche. Si sono cioè rese operative diverse trasformazioni:
1) Rotazione intorno l'asse x
2) Rotazione intorno l'asse y
3) Rotazione intorno l'asse z
4) Zoom IN
5) Zoom OUT
6) Variazione in verticale del punto di osservazione (alto e basso)
7) Variazione in orizzontale del punto di osservazione (destra e sinistra)
Non esistono in realtà limiti sul numero di funzioni che possono essere rese disponibili; a seconda delle necessità è possibile per esempio inserire procedure aggiuntive che compiano altre trasformazioni specifiche. Prendendo spunto dalle lezioni precedenti, potrebbe essere interessante rendere disponibili per esempio anche le trasformazioni di scala, traslazioni, ecc.
Ogni trasformazione, come ormai sappiamo, si ottiene con l'ausilio della relativa matrice di trasformazione. Analizzando attentamente il listato del programma allegato, si può notare che tutto il modulo di calcolo si basa su un'unica matrice (matrice generale di trasformazione) che include tutti i passi di calcolo condensati in una struttura omogenea.
L'utilizzo di calcoli matriciali risulta quindi opportuno, vista

```
10 PRINT 3
20 CLS:PRINT@135,"ATTENDERE"
68 REM***Inizializzazione variabili**********
70 PIE=3.1415                '****PI greca
75 DIM MA(4,4),O1(100,4),O2(1,4),F1(100,
4),F2(1,4)
80 DE=-1200  '**distanza punto di vista
90 SLE=2     '******** fattore di scala
100 AGE=(PIE/12)  '***incremento angolo
110 AXE=0 '*********angoli di rotazione
120 AYE=0
130 AZE=0
140 TXE =0   '***fattori di traslazione
150 TYE =-200
160 TZE = 800
170 IRE = 100      '***incr. prospettico
180 XIE = 150  '***incr. di traslazione
190 YIE = 150
200 ZIE = 150
210 LX=30    '***coordinate window***********
220 RX=209
230 LY=5
240 HY=58
350 REM*************************************
370 GOSUB 950 '*********** lettura dati
410 '**Init matrice di trasformazione**
420 C1E=COS(AZE)
430 C2E=COS(AYE)
440 C3E=COS(AXE)
450 Z1E=SIN(AZE)
460 Z2E=SIN(AYE)
470 Z3E=SIN(AXE)
480 MA(1,2)=(-Z1E*C3E-C1E*Z2E*Z3E)*SLE
490 MA(1,4)=-(Z1E*Z3E-C1E*Z2E*C3E)*SLE/D
E
500 MA(2,1)=(C2E*Z1E)*SLE
510 MA(2,4)=-(-Z3E*C1E-Z1E*Z2E*C3E)*SLE/
DE
520 MA(1,1)=SLE*C1E*C2E
530 MA(2,2)=SLE*(C1E*C3E-Z1E*Z2E*Z3E)
540 MA(3,4)=-SLE*C2E*C3E/DE
550 MA(4,4)=1E-TZE/DE
560 MA(3,3)=0
570 MA(3,1)=Z2E*SLE
580 MA(4,1)=TXE
590 MA(3,2)=C2E*Z3E*SLE
600 MA(1,3)=0
610 MA(4,2)=TYE
620 MA(4,3)=0
630 MA(2,3)=0
700 REM***********MAIN*********************
720 CLS
740 REM***Disegna la window*****************
750 LINE(LX,LY)-(RX,HY),1,B
770 FOR K%=1 TO OJ%
780 GOSUB 1070 '********trasforma***********
785 GOSUB 1240 '********windowing***********
790 GOSUB 3000 '********clipping**********
795 GOSUB 2500 '********display***********
800 NEXT K%
860 GOSUB 2230 '**display linea comandi
870 GOSUB 1300 'modifica trasformazioni
880 IF FG$="0" GOTO 860
900 GOTO 720
910 REM********* END MAIN ******************
950 REM*Apertura e lettura Data file********
952 F$="":CLS
954 INPUT"Nome File (max 6 lettere) ";F$
:IF LEN(F$)>6 THEN 952
958 OPEN F$ FOR INPUT AS 1
972 FOR I%=1 TO 10000
975 IF EOF(1) GOTO 1020
980 INPUTE 1,O1(I%,1),O1(I%,2),O1(I%,3),
F1(I%,1),F1(I%,2),F1(I%,3)
990 O1(I%,4)=1E
1000 F1(I%,4)=1E
1010 NEXT I%
1020 OJ%=I%-1
1025 CLOSE 1
1030 RETURN
1070 REM*********Trasforma*******************
1080 FOR I%=1 TO 4
1090 O2(1,I%)=0E
1100 F2(1,I%)=0E
1110 FOR J%= 1 TO 4
1120 O2(1,I%)=O2(1,I%)+O1(K%,J%)*MAT(J%,
I%)
1130 F2(1,I%)=F2(1,I%)+F1(K%,J%)*MAT(J%,
I%)
1140 NEXT J%
1150 NEXT I%
1160 X1%=O2(1,1)/O2(1,4)
1170 X2%=F2(1,1)/F2(1,4)
1180 Y1%=O2(1,2)/O2(1,4)
1190 Y2%=F2(1,2)/F2(1,4)
1200 RETURN
1240 REM****Windowing**********************
1250 X1%=INT(X1%*.0317+119.5):Y1%=INT(Y1
%*.021+31.5)
1260 X2%=INT(X2%*.0317+119.5):Y2%=INT(Y2
%*.021+31.5)
1280 RETURN
1300 REM*Comando fine programma**************
1315 IF C$="q" THEN CLS:GOTO 2400
1330 REM**********Rotazioni*****************
1340 FG$="1"
1350 IFLEFT$(C$,1)="-"THEN AGE=-AGE:C$=R
IGHT$(C$,1)
1360 IFC$="x"THEN AXE=AXE+AGE:GOTO 1620
1370 IFC$="y"THEN AYE=AYE+AGE:GOTO 1620
1380 IFC$="z"THEN AZE=AZE+AGE:GOTO 1620
1420 REM******* Traslazioni ****************
1430 IFC$="d"THEN TYE=TYE+YIE:GOTO 1620
1440 IFC$="u"THEN TYE=TYE-YIE:GOTO 1620
1450 IFC$="r"THEN TXE=TXE-XIE:GOTO 1620
1460 IFC$="l"THEN TXE=TXE+XIE:GOTO 1620
1470 IFC$="i"THEN TZE=TZE-ZIE:GOTO 1620
1480 IFC$="o"THEN TZE=TZE+ZIE:GOTO 1620
1590 IF C$="-" THEN 1610
1600 REM**** Errore in Input **************
```

la notevole organicità che si ottiene nel modulo di calcolo; si tenga presente che, in realtà, ogni operazione può essere interpretata in modo autonomo, rendendo così possibile la stesura di funzioni completamente indipendenti.

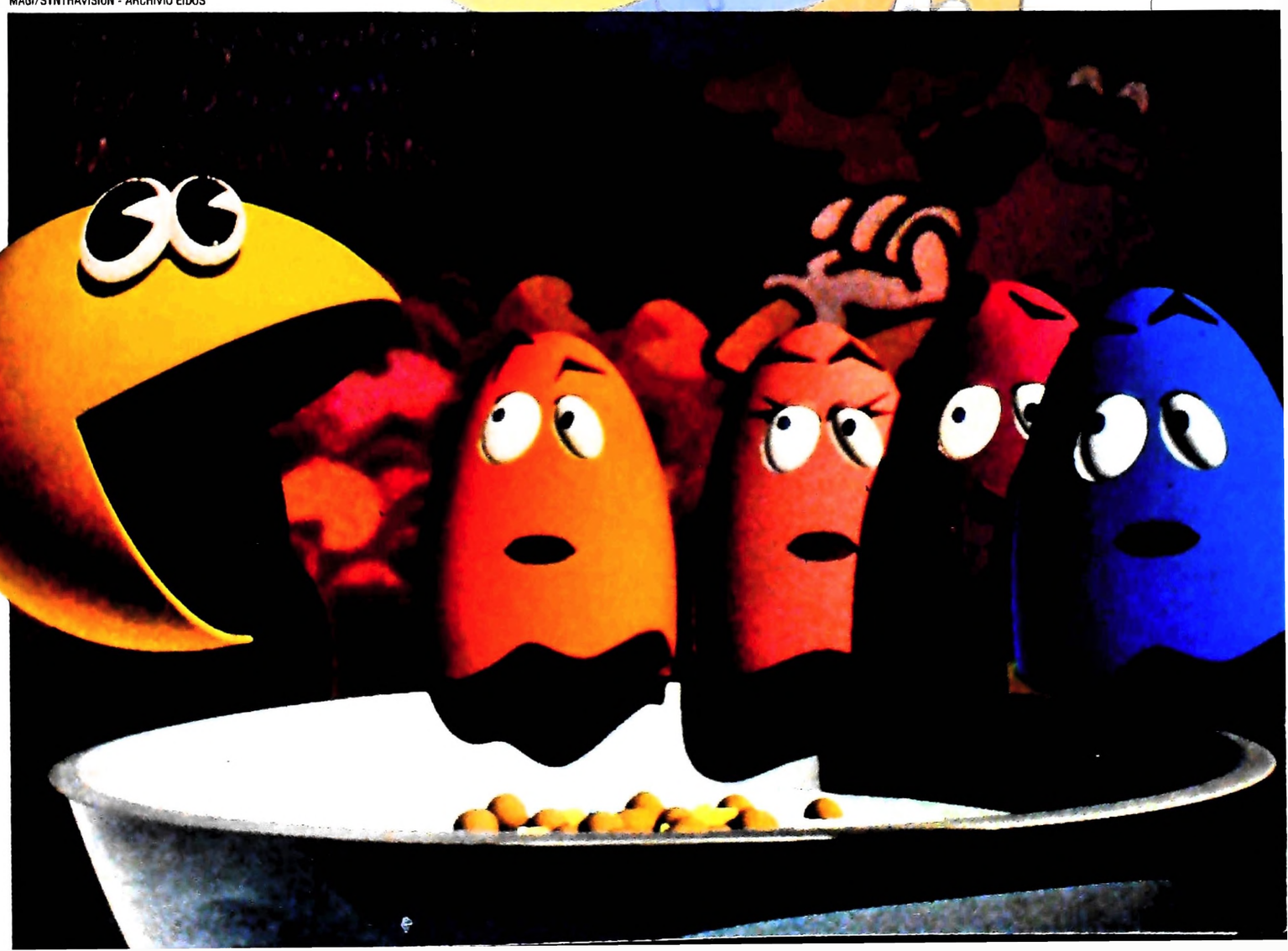
## Limitazioni della macchina

È necessario in proposito ricordare che le applicazioni grafiche richiedono computi numerici relativamente complessi; spesso è quindi possibile incorrere in errori di overflow o di underflow, se la macchina che supporta il pacchetto grafico non è adeguata agli scopi. Questi tipi di errori vengono difficilmente controllati, visto che si verificano in modo casuale durante l'esecuzione del programma; è quindi utile rendersi ben conto delle potenzialità possedute dalla macchina sotto il punto di vista puramente aritmetico, per evitare di realizzare funzioni che poi si possano dimostrare di scarsa utilità per mancanza della necessaria potenza di calcolo.
Altro punto, che deve essere obbligatoriamente considerato



Qui sotto e nelle pagine successive: due fotogrammi estratti da animazioni realizzate al computer. Per la realizzazione di un'animazione al computer vengono utilizzati dei pacchetti di software grafico che consentono all'utente di accedere alle funzioni grafiche fondamentali con molta rapidità e facilità, concedendo così più spazio al momento creativo.

MAGI/SYNTHAVISION - ARCHIVIO EIDOS

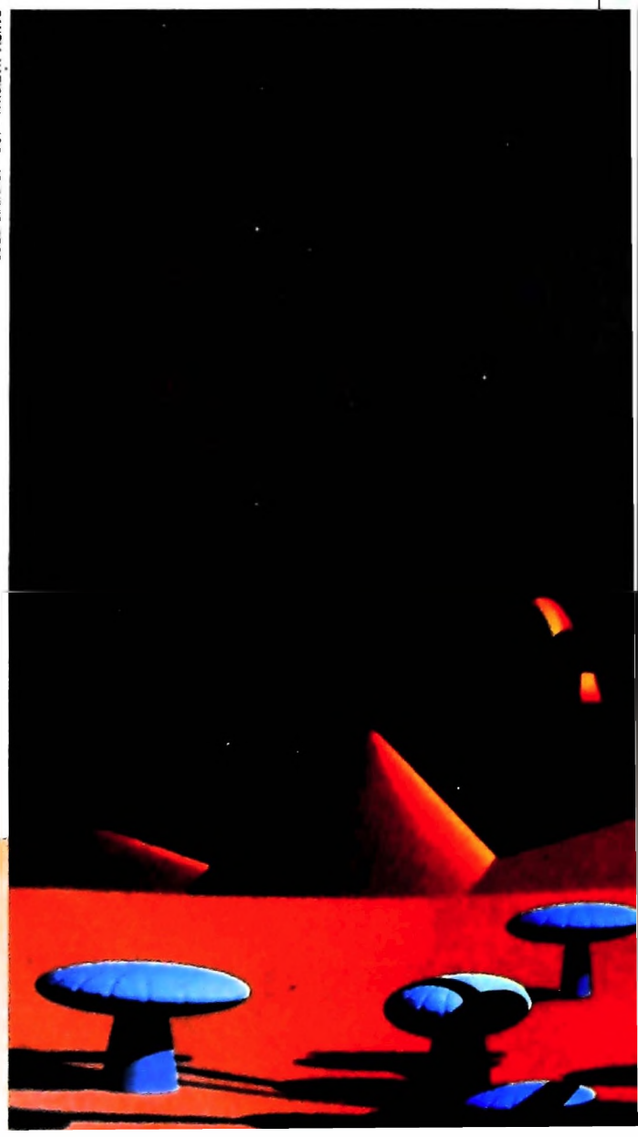


```
1605 BEEP:BEEP
1610 FG$="0":GOTO 880
1620 GOSUB 410
1630 RETURN
2230 REM****Display linea comandi**************
2235 C$="":PL=0
2240 PRINT$0,"X,Y,Z,Up,Down,Left,Right,I
n,Out,Quit:? "
2245 C$=INKEY$:IF C$=""THEN 2240
2247 PRINT$38,C$
2250 RETURN
2400 REM********Fine programma****************
2410 END
2500 REM***Disegna una linea******************
2510 IF PL=0 THEN RETURN
2520 LINE(X1%,63-Y1%)-(X2%,63-Y2%)
2530 RETURN
3000 REM********Clipping**********************
3005 PL=1
3010 GOSUB 3500
3020 IF W1$="" AND W2$="" THEN 3220
3030 IF LEFT$(W1$,1)=LEFT$(W2$,1) OR RIG
HT$(W1$,1)=RIGHT$(W2$,1) THEN PL=0:GOTO
3220
3040 W$=W1$
3050 IF W$="" THEN W$=W2$
3060 IF LEFT$(W$,1)="S" THEN 3100
3062 IF LEFT$(W$,1)="D" THEN 3130
3064 IF RIGHT$(W$,1)="B" THEN 3160
3066 IF RIGHT$(W$,1)="A" THEN 3190
3070 IF W$=W1$ THEN X1%=INT(XX):Y1%=INT(
YY)
3074 IF W$<>W1$ THEN X2%=INT(XX):Y2%=INT
(YY)
3080 GOSUB 3500
3090 GOTO 3020
3100 YY=Y1%+(Y2%-Y1%)*(LX-X1%)/(X2%-X1%)
3110 XX=LX
3120 GOTO 3070
3130 YY=Y1%+(Y2%-Y1%)*(RX-X1%)/(X2%-X1%)
3140 XX=RX
3150 GOTO 3070
3160 XX=X1%+(X2%-X1%)*(LY-Y1%)/(Y2%-Y1%)
3170 YY=LY
3180 GOTO 3070
3190 XX=X1%+(X2%-X1%)*(HY-Y1%)/(Y2%-Y1%)
3200 YY=HY
3210 GOTO 3070
3220 RETURN
3500 W1$="":W2$=""
3510 IF X1%<LX THEN W1$="S"
3515 IF X1%>RX THEN W1$="D"
3520 IF Y1%<LY THEN W1$=W1$+"B"
3525 IF Y1%>HY THEN W1$=W1$+"A"
3530 IF X2%<LX THEN W2$="S"
3535 IF X2%>RX THEN W2$="D"
3540 IF Y2%<LY THEN W2$=W2$+"B"
3545 IF Y2%>HY THEN W2$=W2$+"A"
3560 RETURN
3600 REM**************************************
```

in una applicazione di carattere grafico, è quello del clipping, cioè dell'eliminazione delle linee che fuoriescono dallo schermo video. Anche se questo problema può apparentemente risultare non connesso con la macchina, in realtà la mancanza di una simile procedura può portare a stranissimi errori in fase di rappresentazione e disegno di oggetti; ciò è dovuto al fatto che ogni macchina ha una propria gestione della memoria video e che, in alcuni casi, il non rispettarne le prefissate dimensioni può portare a risultati indesiderati. Nel programma proposto il clipping viene fornito in una routine autonoma che è già stata oggetto di discussione in altre lezioni sulla grafica computerizzata.

## Commento al programma

Con il presente programma si intende dare un primo concreto approccio a quello che può essere l'utilizzo di un pacchetto grafico. A tale scopo la parte di introduzione dei dati non prevede uno specifico programma di editing. Si utilizzerà quindi la modalità TEXT dell'M10, che permette direttamente di introdurre una sequenza di numeri (nel nostro caso le coordinate dei vertici dell'oggetto) su un file che potrà quindi essere letto dal programma di modellazione.
Per introdurre i valori delle coordinate si deve dunque procedere come segue:



SANDIA NATIONAL LABS - ARCHIVIO BIDDS

- entrare in modalità TEXT di M10;
- scrivere il nome del Data File;
- inserire le coordinate come se si stesse scrivendo un normale testo. Le coordinate vanno inserite seguendo la seguente modalità:

  numero "spazio" numero "spazio" numero ENTER

  dove la prima riga è quella delle coordinate del primo estremo e quella successiva è relativa alle coordinate del secondo estremo di ciascun segmento.

Effettuata l'introduzione delle coordinate si può procedere alla visualizzazione del Data File come segue:

- entrare nel programma di modellazione (precedentemente introdotto usufruendo del listato allegato);
- alla prima richiesta del programma rispondere con il nome del Data File;
- appare quindi il disegno seguito dalla lista delle operazioni disponibili:

  X Y Z Up Down Left Right In Out Quit?

X: effettua la rotazione dell'oggetto rappresentato intorno all'asse x di un angolo di +15 gradi; l'angolo di rotazione sarà invece di -15 gradi nel caso si sia premuto il – prima dell'x.

Y: effettua l'operazione precedente in riferimento all'asse y.

Z: effettua l'operazione precedente in riferimento all'asse z.

UP: questa opzione permette di vedere l'oggetto da una altezza differente da quella iniziale, come salendo con un ascensore; si avrà cioè l'impressione di muoversi lungo una direzione verticale in direzione positiva in modo tale che l'oggetto si trovi sempre più in basso rispetto al nostro punto di osservazione.

DOWN: è l'operazione opposta a quella precedente; la direzione ipotetica lungo la quale si muove il punto di osservazione è la stessa, però viene percorsa nel senso opposto.
L'effetto che si ottiene è quello di osservare l'oggetto volta per volta da un punto sempre più basso, fino a far sì che esso ci appaia al di sopra.

LEFT: permette di spostare il punto di osservazione lungo una direzione orizzontale, rendendo così possibile la visione dell'oggetto dalla parte sinistra relativamente alla posizione iniziale.

RIGHT: è la funzione precedente riferita a destra.

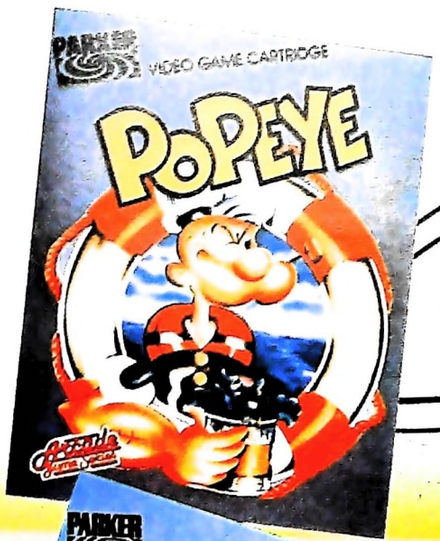
La scelta di una funzione si effettua semplicemente premendo il tasto della lettera iniziale. Il comando di Quit serve per abbandonare la sessione operativa.

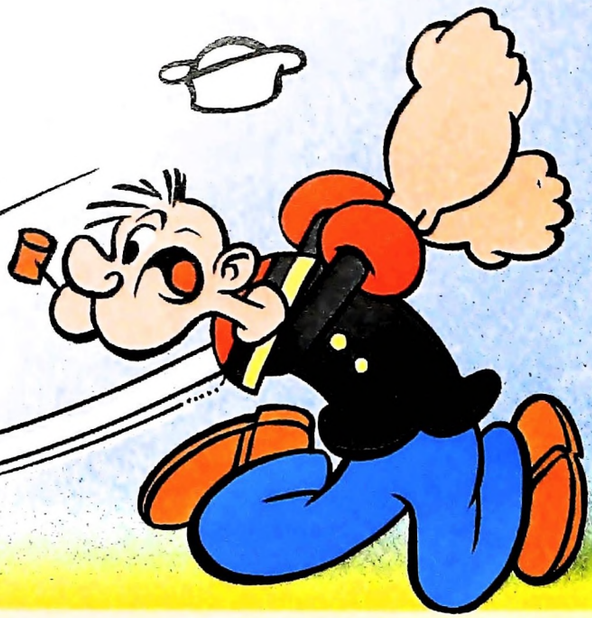
Viene infine fornito un file di prova che rappresenta una griglia tridimensionale; dopo aver introdotto le coordinate nel modo sopraindicato sarà possibile operare con il programma di trasformazioni per la manipolazione della figura stessa; risulteranno particolarmente efficaci operazioni tipo IN, OUT che daranno proprio l'impressione, ad esempio, di avvicinarsi o di allontanarsi dall'oggetto come nella realtà.

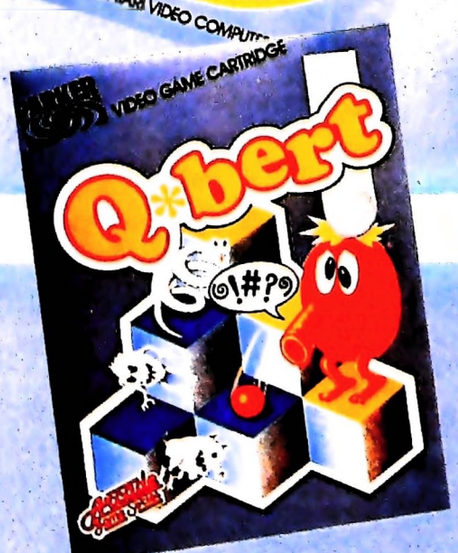
# I GIOCHI DI RISALITA

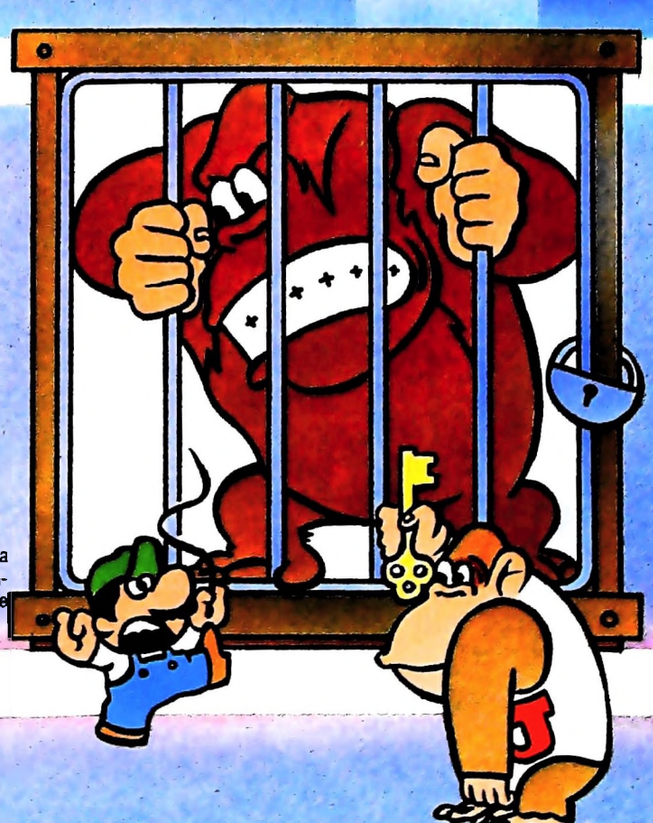
## La vita è fatta a scale: è una lenta risalita, è un conquistarsi la parte alta dello schermo.

Come un tempo, in ambito cinematografico, si facevano tentativi empirici di classificazione — tipo *slapstick* o *hard boiled*, definizioni che non appartengono a nessuna retorica se non a quella dell'uso — così nell'universo dei giochi elettronici circolano definizioni vivacemente pragmatiche, non solo: il successo del capostipite crea un filone e i generi, alla fine, risultano essere nient'altro che i filoni più fortunati.
E infatti uno dei più strepitosi successi delle Arcade è stato "Donkey Kong" della giapponese Nintendo, presto tradotto in versione casalinga in quasi tutti i sistemi che il mercato offre. Rapidamente la trama del game: Donkey (=stupido) Kong ha rapito la fidanzata di Mario e l'ha portata in cima a un palazzo in costruzione, fatto di putrelle che si elevano di tre piani in tre piani. Mario, logicamente, tenta di riprendersi la fidanzata, ostacolato però in questo suo generoso tentativo dallo scimmione rapitore. Al primo tentativo Mario deve salire su delle scale, al secondo togliere tutti i bulloni delle putrelle del grattacielo, al terzo gettarsi prontamente su dei montacarichi (dai quali bisogna scendere sempre al volo). Per aiutare Mario a riavere con sé la sua bella rapita è necessario diventare giocatori molto esperti. "Donkey Kong" è stato oggetto di molte imitazioni, scopiazzature, duplicati: si pensi a "Crazy Cong", "Horrible Hank", "King Kong" (che in realtà dovrebbe essere l'originale perché questo è il titolo del celeberrimo film del 1933 di Shoedsack e Cooper, con lo scimmione King Kong che portava sull'Empire State Building la bella Fay Wray). Insomma si è creduto che con una scimmia, una ragazza rapita, un edificio e delle scale da salire fosse possibile sempre e comunque ricavare un gioco di successo. Ma evidentemente non è così; basta paragonare la freschezza, la ricchezza di dettagli, la risoluzione grafica della versione "originale" con le altre per rendersene conto. Va invece sottolineato il fatto che gli eroi di "Donkey Kong" hanno dato vita a una serie di altre avventure, secondo una caratteristica tipica dei fumetti, dei cartoni e di alcuni telefilm: Mario e il fratello Luigi appartengono ormai allo *star system* del mondo dei videogiochi. Eroi di numerose avventure hanno conosciuto sia il successo pubblico sia quello privatissimo. Proprietari di una piccola fabbrica di legnami e affini sono assediati da orride bestioline che li disturbano sul lavoro e distruggono il legname.
A proposito di eroi dei *cartoons*, la Nintendo ha proposto un altro celebre personaggio, Popeye (Braccio di ferro), incontrando naturalmente il favore del pubblico; anche in questo caso, lo schema a risalita serve per sfruttare a pieno le capacità bidimensionali dello schermo: infatti Popeye deve correre su e giù per le scale per cercare di afferrare gli oggetti che cadono lentamente dall'alto (sono cuoricini, note musicali e lettere come H.E.L.P.). La situazione è piuttosto semplice: Bruto ha rapito Olivia, Popeye può contare sulle miracolose scatole di spinaci.
"Congo Bongo" della Sega è stato presentato come una variante di "Donkey Kong" realizzata con lo stesso splendido effetto tridimensionale di "Zaxxon" (un gioco di guerre stellari). Lo slogan è efficace perché serve a spiegare l'effetto, nei tradizionali giochi a risalita, di alcune commistioni di genere: in altre parole, sullo schema di base "ladder" vengono introdotte altre situazioni o altre varianti caratteristiche di altri filoni. Così abbiamo la risalita nello spazio, la risalita attraverso mezzi elettromeccanici, come l'ascensore, la risalita nei



Videogiochi come "Frogger", "Q. Bert", "Popeye" sfruttano il fortunato schema di gioco a "risalita". In particolare con "Popeye" si è riusciti a combinare la semplicità della storia e il tema della violenza "non cattiva" di uno dei più famosi cartoon con una grafica lineare e un gioco eccitante.

castelli abitati da fantasmi, secondo la più bella tradizione del romanzo gotico. Quanto a "Congo Bongo", il gioco è articolato in due fasi separate, con due immagini video differenti, e consiste nell'abilità di condurre il cacciatore in stretti passaggi, ripide scalinate, gradoni, scivoli, cascate e fiumiciattoli in un avvicendarsi di imprevisti e di situazioni particolari.
Curiosa la variante della "discesa", tipo "H.E.R.O." (Activision), con intenzionalità quasi sempre volte al positivo: si salvano dei minatori in pericolo, si aiuta il prossimo, si diventa crocerossa spaziale.

## Il programma

Questo programma spiega la costruzione e l'animazione di una semplice figura (in questo caso un piccolo aeroplano che decolla). La difficoltà principale sta nel fatto che il computer M10 non è stato pensato per questo tipo di applicazioni; è infatti dotato di un video "molto lento"; ciò nonostante è possibile creare degli esempi che spieghino come poter affrontare il problema.
Per animare una figura si devono costruire due procedure:
— una che disegni l'oggetto in un certo punto dello schermo;
— una che cancelli l'oggetto.
Ovviamente entrambe queste procedure devono essere espresse in termini di coordinate relative

```
line (x,y) - (x+3,y+2),1
```

e non di coordinate assolute

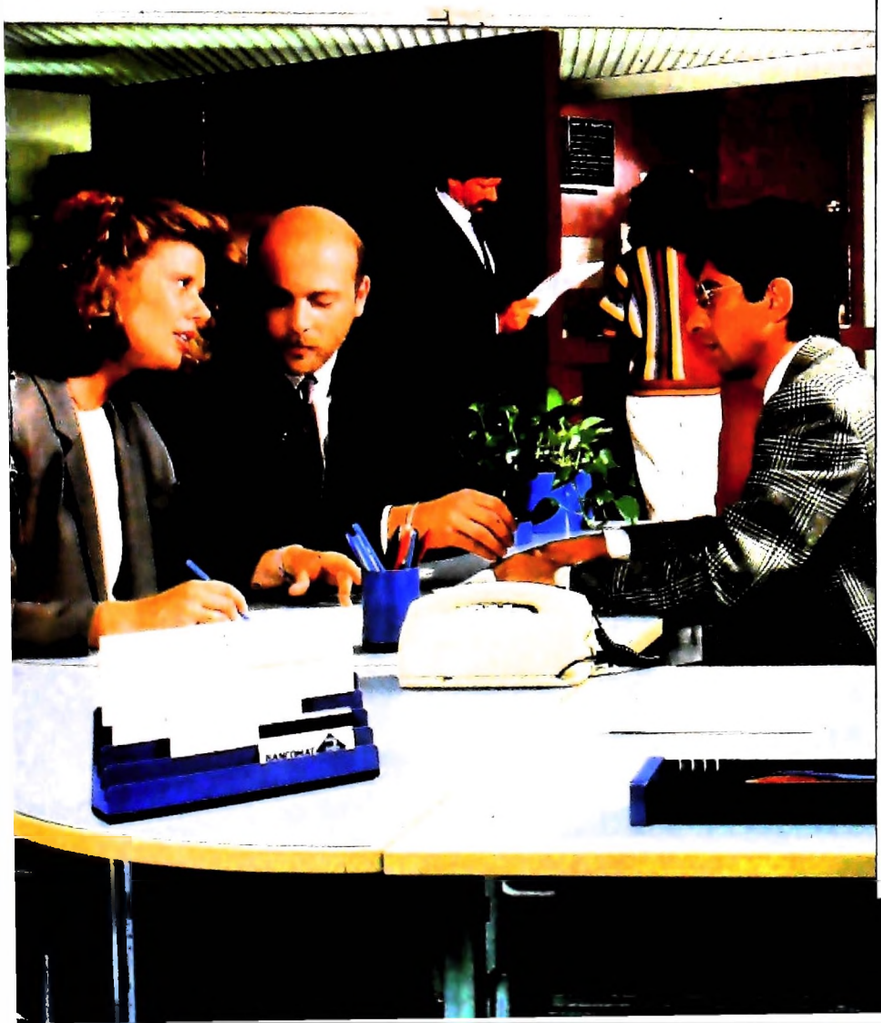
```
line (10,10) - (13,12),1
```

inoltre la procedura di cancellazione può essere costruita in due modi diversi:
— tracciando un box nero che contenga tutto l'oggetto;
— ritracciando l'oggetto in complemento (detto anche reverse mode).
La scelta di quale dei due dipende generalmente da considerazioni di velocità, anche se solo il secondo ha la proprietà di "salvare" lo sfondo.

```
5     CLS
10    xs=0   : ys=-1
20    xb=16  : yb=8
25    x=0    : y=60
30    GOSUB 1000
35    GOSUB 700
40    FOR i=1 TO 51
50     GOSUB 500
60     IF i <> 30 THEN GOTO 70
65      xs=3 : ys=0
70     IF i>34 THEN xs=xs+1
100    x=x+xs
110    y=y+ys
120    GOSUB 1000
130   NEXT i
150   GOSUB 500
400   END
500   REM cancella l'aereo
520   line (x,y)-(x+xb+4,y-yb),0,B F
550   RETURN
700   REM traccia il paesaggio
710   line (30,63)-(238,58),1,B F
720   line (40,58)-(60,35),1,B
750   RETURN
1000  REM traccia l'aereo
1010  line (x,y)-(x+xb,y),1
1020  line (x,y)-(x,y-yb),1
1030  line (x,y-yb)-(x+3,y-yb),1
1040  line (x,y-4)-(x+xb,y-4),1
1050  line (x+xb,y)-(x+xb,y-4),1
1060  line (x+xb,y-2)-(x+xb+4,y-2),1
1070  Line (x+3,y-yb)-(x+5,y-4),1
1080  line (x+9,y-4)-(x+10,y-6),1
1090  line (x+10,y-6)-(x+13,y-6),1
1100  line (x+13,y-6)-(x+14,y-4),1
1110  line (x+7,y-2)-(x+15,y-2),1
1200  RETURN
```

# UN NUOVO MODO DI USARE LA BANCA.

## Conto corrente più



## TANTI PENSIERI IN MENO CON IL CONTO CORRENTE "PIU'" DEL BANCO DI ROMA.

Essere cliente del Banco di Roma vuol dire anche essere titolari del conto corrente "più". Un conto corrente più rapido: perché già nella maggior parte delle nostre filiali trovate gli operatori di sportello che vi evitano le doppie file.

Più comodo, perché potete delegare a noi tutti i vostri pagamenti ricorrenti: dai mutui all'affitto, dalle utenze alle imposte.

Più pratico, perché consente l'utilizzo del sistema di prelievo automatico Bancomat e l'ottenimento della carta di credito.

Inoltre un servizio utilissimo, soprattutto per imprenditori e commercianti denominato "esito incassi", consente di avere comunicazione dell'eventuale insolvenza entro solo cinque giorni dalla scadenza. Una opportunità veramente speciale.

Più sicuro, perché con una minima spesa potrete assicurarvi contro furti e scippi mentre vi recate in banca o ne uscite.

Veniteci a trovare, ci conosceremo meglio.

**BANCO DI ROMA**
**CONOSCIAMOCI MEGLIO.**

Olivetti M10 vuol dire disporre del proprio ufficio in una ventiquattrore. Perché M10 non solo produce, elabora, stampa e memorizza dati, testi e disegni, ma è anche capace di comunicare via telefono per spedire e ricevere informazioni. In grado di funzionare a batteria oppure collegato all'impianto elettrico, M10 mette ovunque a disposizione la sua potenza di memoria, il suo display orientabile a cristalli liquidi capace anche di elaborazioni grafiche, la sua tastiera professionale arricchita da 16 tasti funzione.

Ma M10 può utilizzare piccole periferiche portatili che ne ampliano ancora le capacità, come il microplotter per scrivere e disegnare a 4 colori, o il registratore a cassette per registrare dati e testi, o il lettore di codici a barre. E in ufficio può essere collegato con macchine per scrivere elettroniche, con computer, con stampanti. Qualunque professione sia la vostra, M10 è in grado, dovunque vi troviate, di offrirvi delle capacità di soluzione che sono davvero molto grandi. M10: il più piccolo di una grande famiglia di personal.

## PERSONAL COMPUTER OLIVETTI M10

# L'UFFICIO DA VIAGGIO


Anche in leasing con Olivetti Leasing.

olivetti

Per informazioni rivolgersi ai negozi contrassegnati da "Olivetti M10 Punto di Vendita" o inviate il coupon a Olivetti Divisione Personal Computer, Via Meravigli 12, 20123 Milano

NOME/COGNOME

VIA N.

CAP/CITTÀ

TELEFONO