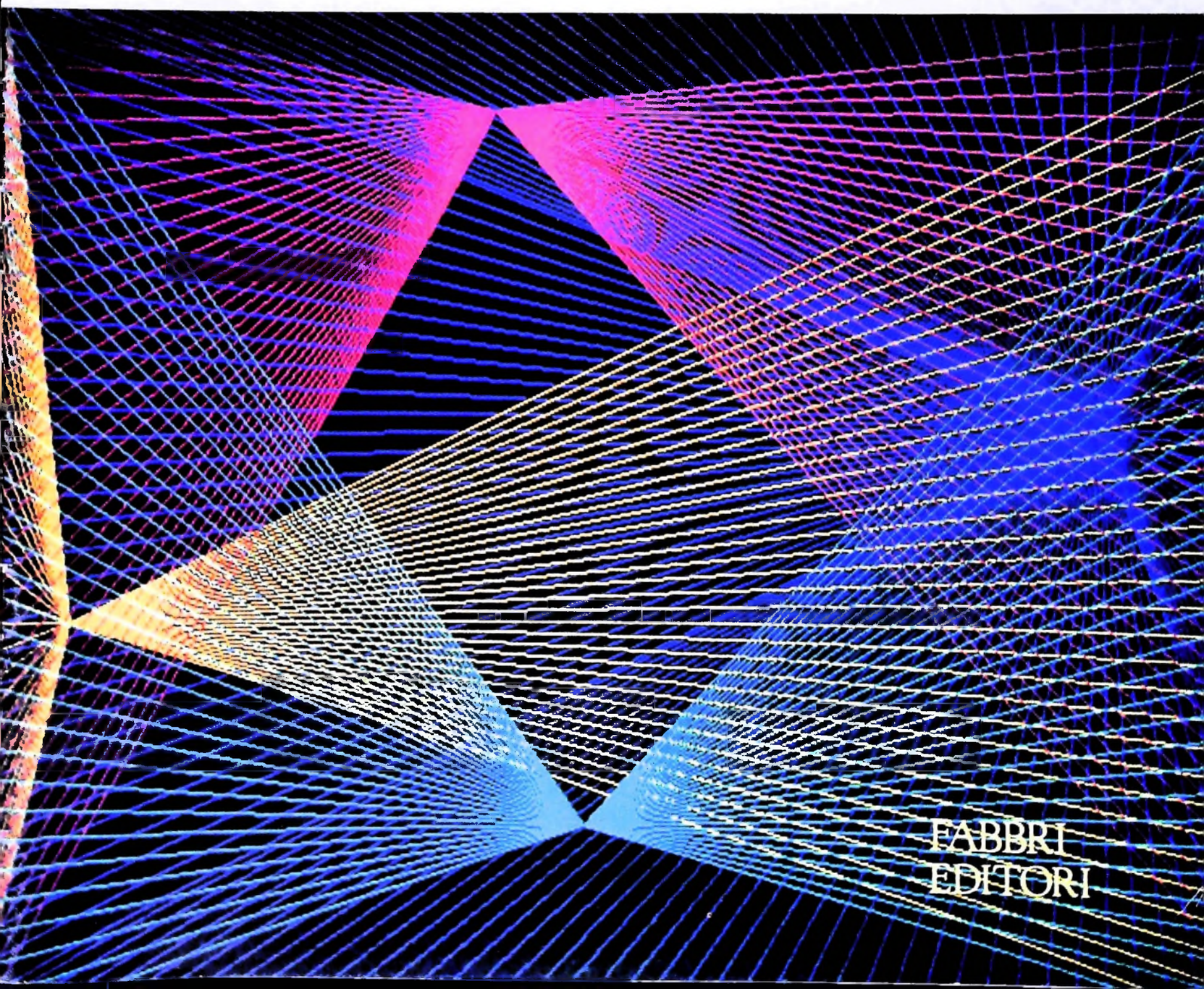
Spediz. in abbonamento postale GR II/70 L. 2 200

57 CORSO PRATICO COL COMPUTER

422097

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

diretto da GIANNI DEGLI ANTONI

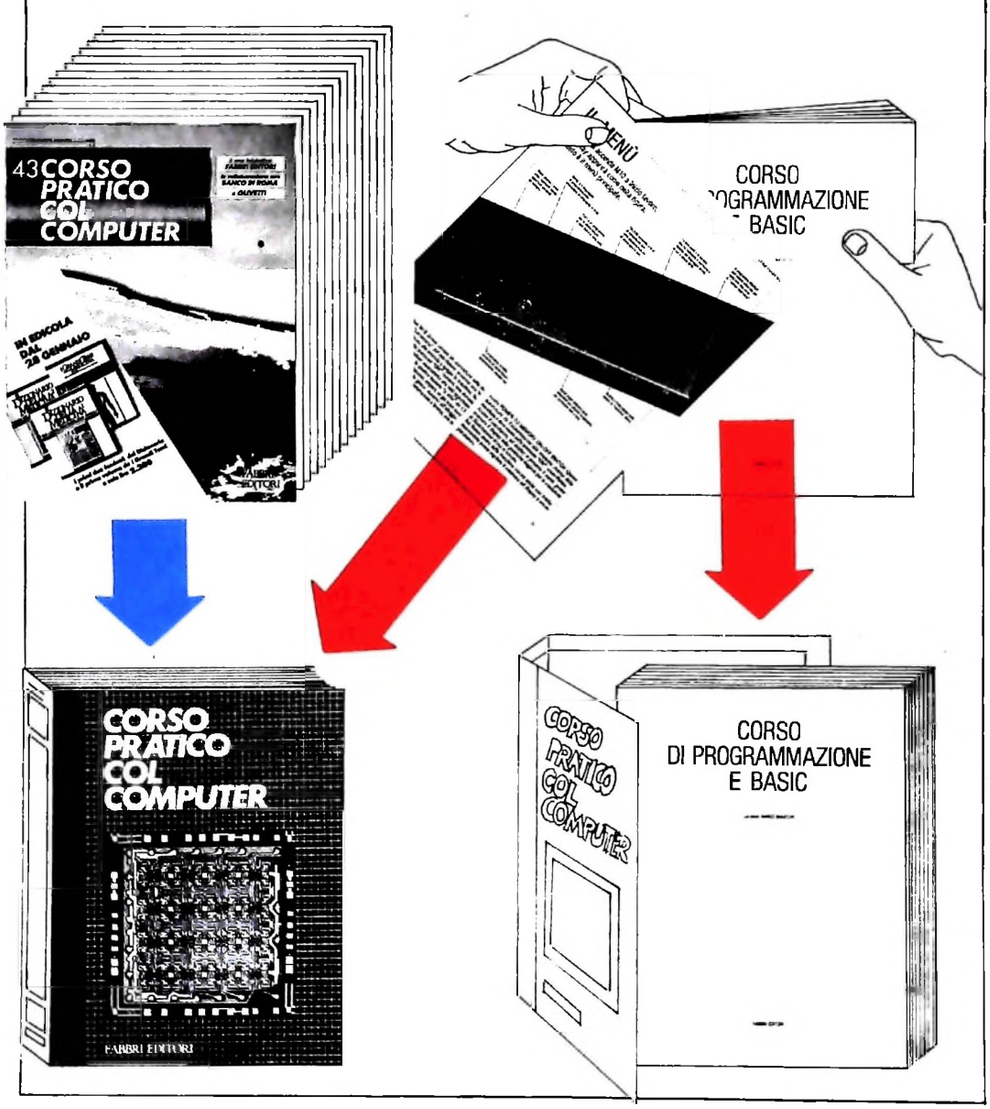


FABBRI EDITORI

AVVISO AI LETTORI

Con questo fascicolo si conclude il quarto volume del "Corso pratico col computer". Per rilegare il volume si useranno:

- i fascicoli dal n. 43 al n. 57;
- occorrerà staccare l'inserto centrale di 4 pagine relativo al "Corso di programmazione e BASIC"; tutti gli inserti contenuti nei fascicoli, fino al n. 72, dovranno essere conservati per essere rilegati nel volume "Corso di Programmazione e BASIC", la cui copertina è stata messa in vendita con il fascicolo n. 30;
- i risguardi, inseriti nella copertina del volume;
- la copertina del volume, che è stata messa in vendita con il n. 52;
- ricordiamo che il frontespizio del volume fa parte del fascicolo n. 43 e il sommario del fascicolo n. 57.



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
MARCO ANELLI, DIEGO BIASI, ANDREA GRANELLI, ALDO GRASSO, LUCA VIERI, Etnoteam (ADRIANA BICEGO)

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

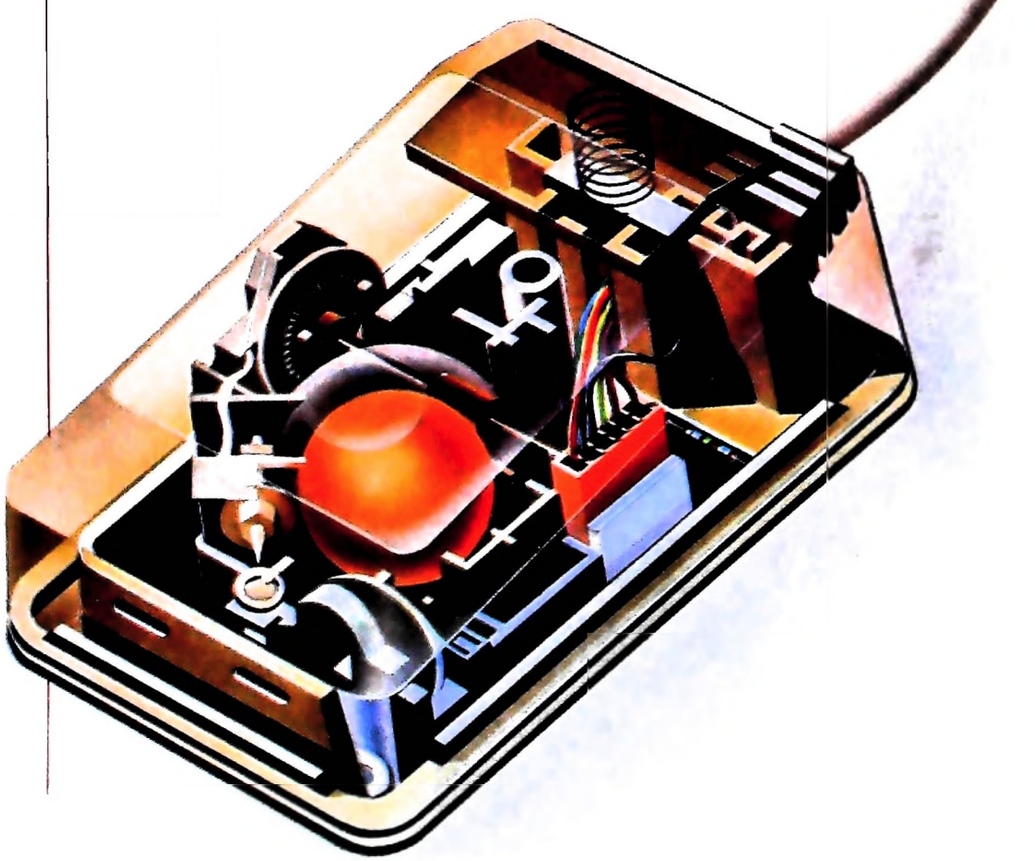
Programmazione Editoriale
ROSANNA ZERBARINI
GIOVANNA BREGGÉ

Segretarie di Redazione
RENATA FRIGOLI
LUCIA MONTANARI

JOYSTICK, PADDLE, TRACKBALL, MOUSE

Sistemi di controllo svincolati dalla tastiera.

Nel "dialogo" tra uomo e computer, come abbiamo già visto, lo strumento attualmente più impiegato per impartire ordini e per fornire dati alla macchina è senza dubbio la tastiera. Esistono però situazioni in cui le istruzioni che devono essere inviate al calcolatore sono talmente limitate da rendere la tastiera eccessivamente complessa, e quindi fonte di difficoltà di impiego e di possibili errori.
In altri casi, invece, è necessario fornire all'elaboratore un input di tipo "analogico", cioè delle grandezze variabili con apparente continuità, invece delle entità discrete e separate inseribili tramite tastiera.
Questa necessità ha spinto alla realizzazione di nuovi tipi di periferiche di input, più adatte della tastiera in un certo campo di applicazioni.
Gli esempi più diffusi di queste periferiche sono rappresentati dai joystick, dai paddle, dai trackball e dai mouse, che, oltre ai notissimi videogiochi, hanno trovato anche applicazioni molto più "serie" e utili.



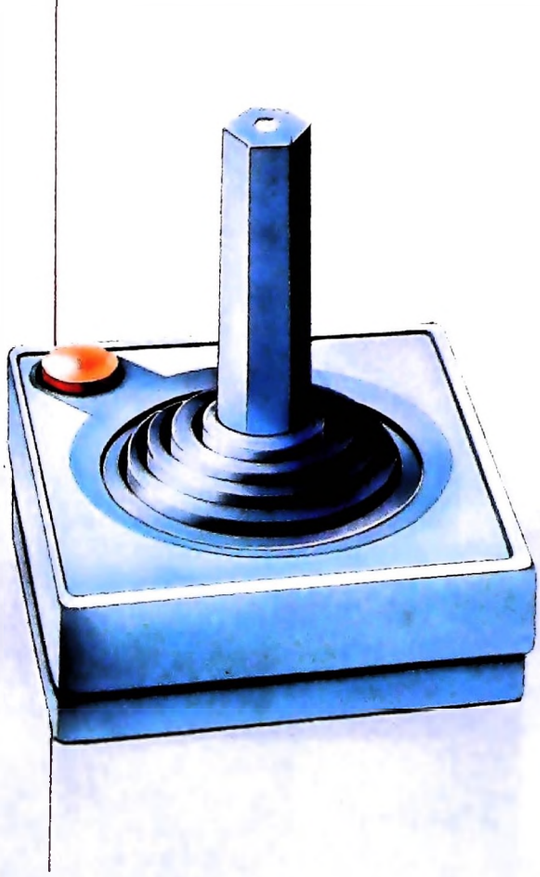
Un'alternativa alla tastiera, nel caso in cui si abbia bisogno di un tipo di periferica in grado di fornire all'elaboratore un input di tipo "analogico", è rappresentata dai joystick, dai paddle, dai trackball e dai mouse. Nella figura di questa pagina: spaccato di un mouse.

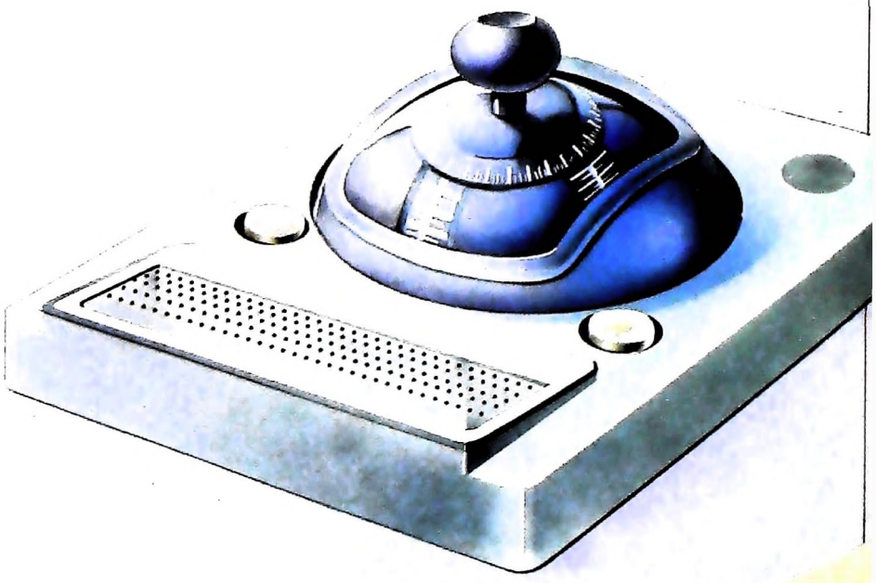
Il joystick

Tra le periferiche di input non standard il joystick rappresenta senza dubbio il modello più diffuso, in quanto la quasi totalità dei videogiochi oggi in commercio fa uso di questo strumento come controllo degli avvenimenti che si svolgono sullo schermo.
Dal punto di vista costruttivo il joystick è rappresentato da un involucro di plastica da cui spunta una specie di cloche, variamente sagomata e dotata degli stessi otto movimenti che contraddistinguono la cloche di un aereo vero. A seconda della posizione in cui si trova, la "cloche" chiude uno oppure due di un insieme di quattro interruttori posti a croce, originando, tramite uno speciale circuito, otto tipi di segnali differenti (alto, basso, destra, sinistra, alto-destra, basso-destra, alto-sinistra, basso-sinistra). Apposite molle di richiamo provvedono a riportare la cloche in posizione neutra quando nessuna forza agisce su di essa.
Un nono tipo di segnale è inviato dal "bottone di sparo" (fire button), sistemato solitamente sull'impugnatura della cloche o sulla base del joystick stesso.
Il segnale prodotto dalla periferica arriva al computer tramite un apposito cavetto per essere da questi interpretato.
In alcuni calcolatori il valore del segnale ricevuto è collocato in una apposita locazione di memoria, in altri, più sofisticati, il contenuto di questa memoria è indicato anche da una apposita variabile, chiamata solitamente JOYSTICK o JOY, seguita da Ø e 1, oppure da 1 e 2 se il sistema è in grado di gestire più di un joystick. Un programmino in Basic capace di far muovere un puntino su uno schermo di coordinate X e Y, sotto il comando di un joystick, avrebbe, in un ipotetico dialetto Basic evoluto, una struttura di questo tipo:

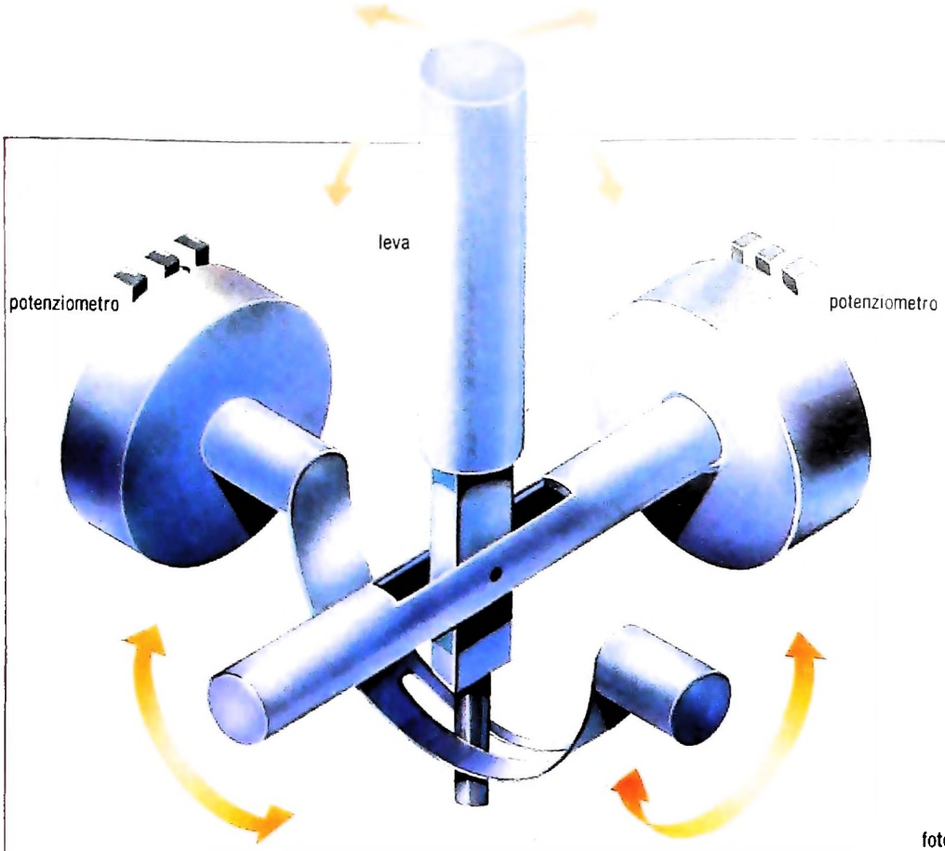
```
10 X = 10 : Y = 10
20 IF JOY(1) = 1 THEN X = 1 : Y = -1
30 IF JOY(1) = 2 THEN X = X + 1
40 IF JOY(1) = 3 THEN X = X + 1 : Y = Y + 1
50 IF JOY(1) = 4 THEN Y = Y + 1
60 IF JOY(1) = 5 THEN X = X - 1 : Y = Y + 1
70 IF JOY(1) = 6 THEN X = X - 1
80 IF JOY(1) = 7 THEN X = X - 1 : Y = Y - 1
90 IF JOY(1) = 8 THEN Y = Y - 1
100 PLOT(X,Y)
110 GOTO 20
```

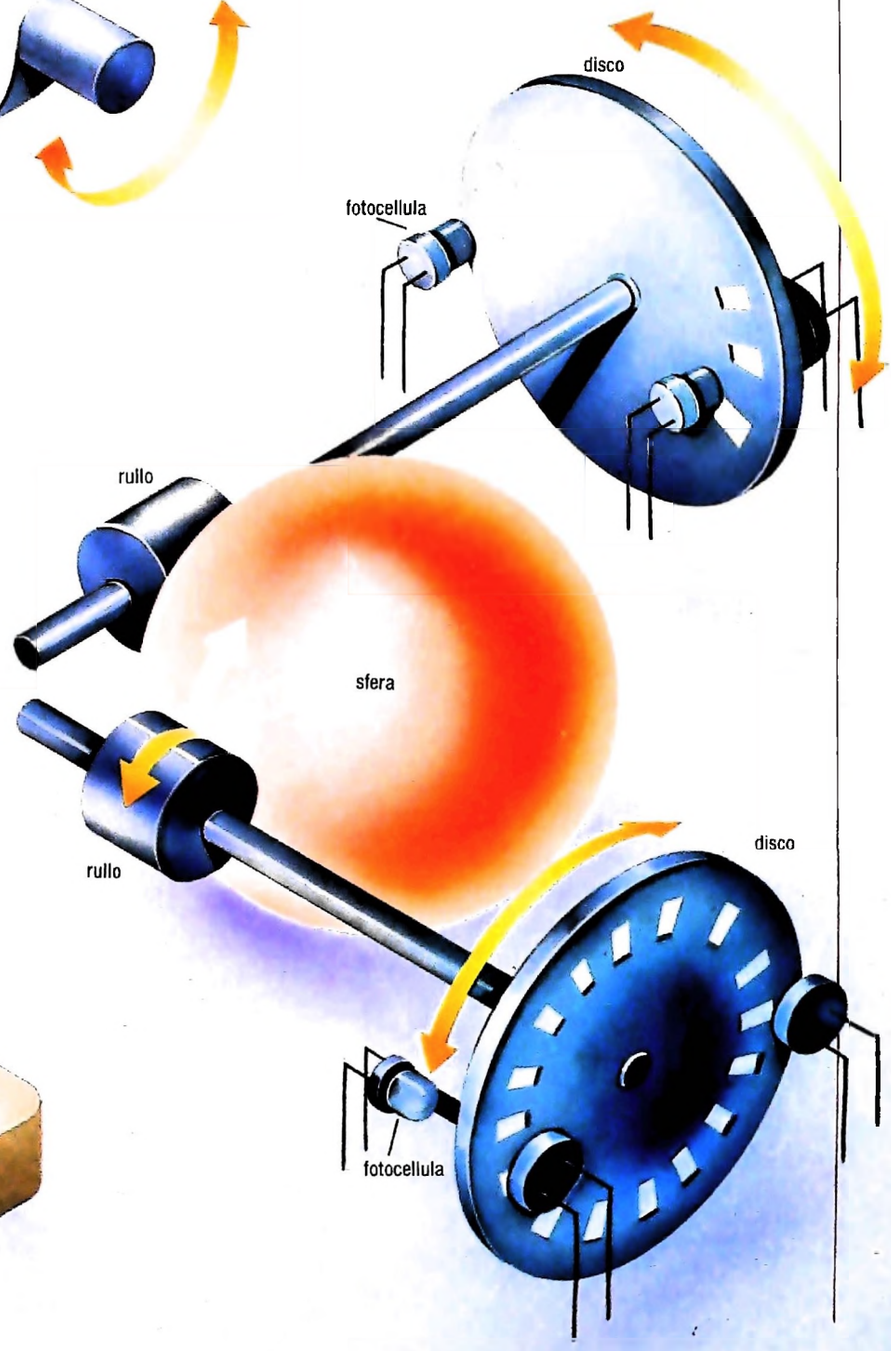
Nel caso il dialetto non preveda una variabile del tipo JOY, al posto di essa si potrà sostituire la funzione PEEK(n), dove n rappresenta la locazione di memoria in cui viene conservato, momento per momento, il valore inviato dal joystick.
Se dal punto di vista concettuale tutti i joystick funzionano nello stesso modo, all'atto pratico esistono tra i vari modelli in commercio numerose differenze, che rendono anche ragio-



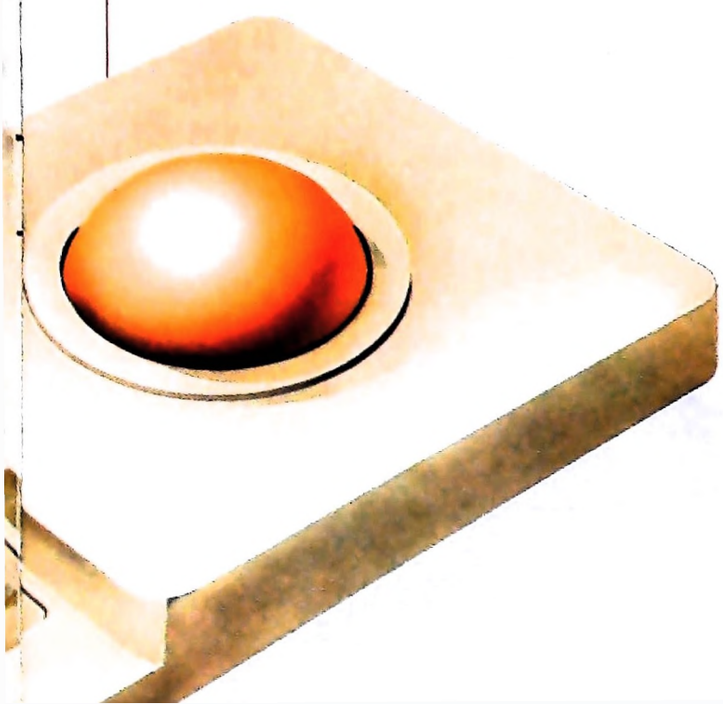


A fianco, è schematizzato il meccanismo costruttivo del joystick. Il "bastoncino della gioia" è caratterizzato da una "cloche" come quella degli aerei, sulla quale si agisce per trasmettere, tramite i potenziometri, il segnale al computer. Sotto meccanismo semplificato del mouse. Il movimento della pallina viene trasferito a due rulli collegati a due dischi muniti di una serie di fessure. Gli impulsi luminosi, provenienti da una fonte di luce costantemente diretta sul disco, vengono captati da una fotocellula e tramutati in segnali intellegibili dall'elaboratore.





In basso, da sinistra a destra: semplice joystick, joystick "professionale", trackball. Quest'ultimo è costituito da una palla libera di ruotare sul proprio supporto, i cui movimenti sono rilevati da una serie di sensori e successivamente trasformati in segnali analoghi a quelli del joystick.



ne del prezzo variabile tra le 20 e le oltre centomila lire.
Alcuni joystick da "competizione", infatti, posseggono una dolcezza di funzionamento e una precisione di controllo superiore rispetto ai joystick "casalinghi", arrivando fino ad avere appositi meccanismi di regolazione e di centraggio simili a quelli che si trovano sui radiocomandi.
Altri joystick, invece, posseggono un collegamento a infrarossi, eliminando in tal modo il problema dei fili che si aggrovigliano e che possono essere strappati da un movimento inconsulto avvenuto nel furore del gioco.
Tra gli optional possibili possiamo avere anche il "fuoco a raffica" (AUTOFIRE) oppure la possibilità di escludere i quattro movimenti diagonali nel caso si stia impiegando un gioco di "labirinto" (tipo Pac-man) in cui siano consentite solo quattro direzioni.
Una notevole differenza costruttiva esiste anche tra i joystick domestici e quelli che equipaggiano i videogiochi da bar.
Questi ultimi, infatti, fanno uso di veri e propri interruttori a foglia invece di semplici contatti metallici, soluzione certamente più costosa, ma che assicura una resistenza a tutta prova (o quasi...).
Normalmente i joystick fanno uso di una porta d'ingresso standard a 10 pin, fatto che consente di collegare alla propria console o al proprio computer uno qualsiasi dei numerosi modelli in commercio.
Alcune macchine necessitano però per alcune applicazioni di joystick speciali, dotati di tasti di funzione o addirittura di tastierino numerico. Rappresentando una via di accesso e di interfacciamento con il calcolatore estremamente comoda, il connettore dei joystick è sfruttato da numerosi fabbricanti per il collegamento di altre periferiche di tipo particolare, come le tavolette grafiche e i trackball.

Joystick anatomico, dotato di pulsante per raffiche di fuoco (fire button) sistemato sull'impugnatura della cloche.

Il paddle

Il paddle, ora piuttosto in disuso, ha rappresentato il primo tentativo di fornire un sistema di controllo per piccoli computer svincolato dalla tastiera. Questa periferica è rappresentata essenzialmente da un potenziometro, che consente il passaggio di una quantità variabile di corrente a seconda della posizione in cui si trova un particolare controllo. Il valore della corrente presente nel circuito paddle-computer è memorizzato in una apposita locazione di memoria o in una variabile, generalmente denominata PADDLE o POL.
L'impiego di un paddle per controllare il movimento di un oggetto sullo schermo risulta estremamente difficile, a meno di non fare uso di due di queste periferiche e di speciali tecniche di programmazione, situazione che ha fatto preferire il joystick al paddle nella maggior parte delle applicazioni.

Il trackball

Il trackball, abbastanza poco diffuso nel mondo dello home e del personal computing per motivi di prezzo e dimensioni, è rappresentato da una palla del diametro di una decina di centimetri libera di ruotare sul proprio supporto.
Una serie di sensori (meccanici oppure ottico-elettronici nei tipi più sofisticati) rileva gli spostamenti impartiti dall'operatore, spostamenti che saranno poi trasformati da appositi circuiti in segnali analoghi a quelli del joystick. La precisione e la dolcezza di funzionamento del trackball lo rendono il tipo di comando in assoluto più adatto per i videogiochi, ma il suo prezzo (dalle centocinquantamila lire in su) fa sì che il suo impiego sia limitato ad applicazioni professionali come il CAD e la grafica computerizzata.

Il mouse

Il mouse (in inglese "topo") è l'ultimo nato nel gruppo delle periferiche di ingresso "senza tasti", e pare destinato ad avere un notevole successo di pubblico e di vendita.
Il funzionamento del mouse può essere paragonato a quello di un trackball rovesciato. In questo caso, infatti, l'operatore non agisce sulla palla direttamente, ma fa scorrere l'intero mouse sul piano del tavolo o della scrivania, provocando in tal modo la rotazione di una pallina posta al suo interno. Il movimento della pallina viene poi trasformato in segnali intellegibili dal calcolatore.
Essendo di impiego semplice e immediato il mouse è stato adottato da alcuni sistemi improntati alla massima semplicità d'uso (Apple Macintosh e Lisa), dove, supportato da un software adatto che impiega un gran numero di menù, è in grado addirittura di sostituire la tastiera.

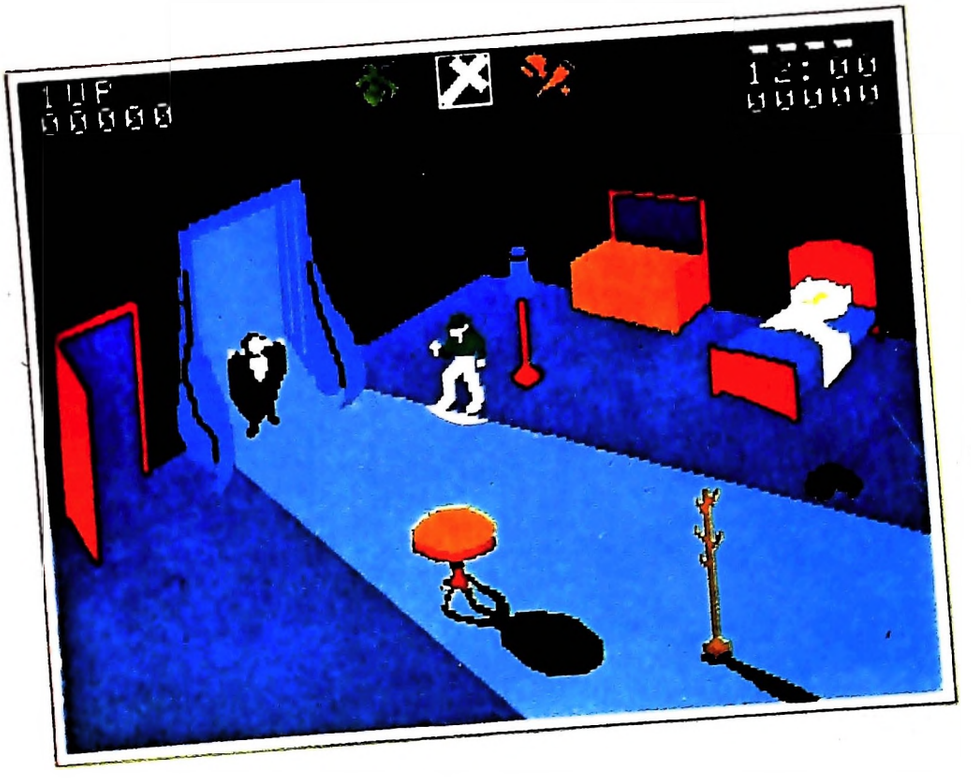
SALTA E SCAPPA

Jump: ovvero come superare pericoli di ogni genere e impadronirsi di favolosi tesori.

Se i giochi a risalita servono per sfruttare al massimo la bidimensionalità dello schermo televisivo, i giochi "jump" esaltano fino in fondo lo strumento con cui si gioca, il joystick. "Jump" è l'azione allo stato puro che riproduce i gesti essenziali di questa piccola periferica: il movimento destra/sinistra o alto/basso (impresso dal bastoncino) e il salto (o lo sparo, entrambi ottenuti premendo il bottone). Trascrivendo i giochi "jump" su tastiera (come nel nostro caso) si ottiene una sorta di formalizzazione, di razionalizzazione di quella bacchetta magica che è il joystick: svelare i misteri nuoce un po' al gioco, ma è una pratica di grande aiuto per chi comincia a digitare con il computer.

I giochi "jump" si impongono all'attenzione del pubblico grazie a "Pitfall"; si tratta di una creazione del famoso David Crane, vincitore dell'Arcade Awards 1983 (il premio Oscar dei videogame).

L'obiettivo di "Pitfall" è di guidare l'eroe del gioco attraverso un labirinto circolare di scene, ambientate in una giungla, saltando o cercando di evitare numerosi pericoli mortali e di aiutarlo ad afferrare il maggior numero possibile di tesori; pericoli mortali sono scorpioni, fuochi, serpenti a sonagli, coccodrilli, paludi, sabbie mobili e fosse di catrame; pericoli meno mortali sono invece i fossati e tronchi rotolanti. Molti ostacoli "Pitfall" li può superare — è la scena più bella — aggrappandosi a una liana. Raggiunta l'incredibile vetta dei tre milioni di cartucce vendute, "Pitfall" raddoppia, nel senso che si presenta al pubblico in una versione più sofisticata e complessa, non più ambientata nella giungla ma nelle profonde e antiche caverne dove giacciono i favolosi tesori degli Incas. Questa Lost Cavern è enorme, piena di passaggi tor-



Anche nei videogiochi Dracula è lo stesso di sempre: si tratta insomma della versione un po' horror dei giochi "jump". Il conte-vampiro si alza dalla bara al tramonto del sole, ricerca le sue vittime e... salta loro al collo, ovviamente per morderle.

tuosi e di dirupi mozzafiato: i pericoli non si contano! Il luogo dove si trova il tesoro è largo ben otto schermi e ha ventisette livelli di profondità. "Pitfall" ha così modo di esibirsi nel suo più vasto repertorio atletico: corre, salta, scala pareti, nuota e qualche volta è costretto a volare, aggrappato a un pallone volante.

Un'altra storia "jump" di figli e figliastri, fiorita all'ombra della più celebre piramide elettronica, è quella di "Q*Bert". Il successo di questo gioco ha prodotto non meno di undici game simili; e sebbene lo stesso "Q*Bert" avesse preso qualcosa in prestito dalla strategia e ingegnosità di "Pac Man", pur con una indubbia reinvenzione, nel mondo dei giochi elettronici il fenomeno di questi spinoff (prodotti che vengono catapultati fuori) è impressionante.

Q*Bert, uno strano e adorabile personaggio, salta sopra una piramide di cubi, inseguito da tutta una serie di pericolosi mostriciattoli. Il salto continuo permette a Q*Bert di destreggiarsi fra questa infida popolazione e costruire tante piramidi di diverso colore: nessuno da eliminare, niente da mangiare, soltanto cubi da saltare.

Ebbene questo semplice schema ha ispirato altri game, a volte non meno dignitosi in quanto a grafica e intreccio: "Frostbite", storia di un architetto senza lavoro, costretto a emigrare al Polo Nord e a saltare non solo di palo in frasca, ma anche di lastrone in lastrone per costruire un piccolo insediamento urbano; "Pogo Joe", ambientato nel magico mondo di "Yellow Submarine" con tanto di musica che vorrebbe evocare i Beatles; "Juice", ovvero le avventure di Edison, un androide cinetico e di bell'aspetto che deve completare il collegamento elettrico di un circuito saltando su ogni sezione.

Ma il "jump" più "narrativizzato" rimane "Dracula". L'eroe di Bram Stoker, dopo essersi travestito per anni da Bela Lugosi e da Christopher Lee, dopo aver frequentato i cartoni animati ("Mickey Gala Première" di Walt Disney, 1933) e i fumetti, è apparso anche in versione videogioco. Il giocatore deve accompagnare il conte nelle sue operazioni esistenziali: levata dalla bara al tramonto, ricerca delle vittime, rifornimento di sangue (la scena del morso sul collo, tramite saltino, è da antologia), ritorno al cimitero prima dell'alba. Vedere camminare il conte afflitto da zoppia per morso di cane vale tutto il gioco: chi va con lo zoppo impara a non saltare!

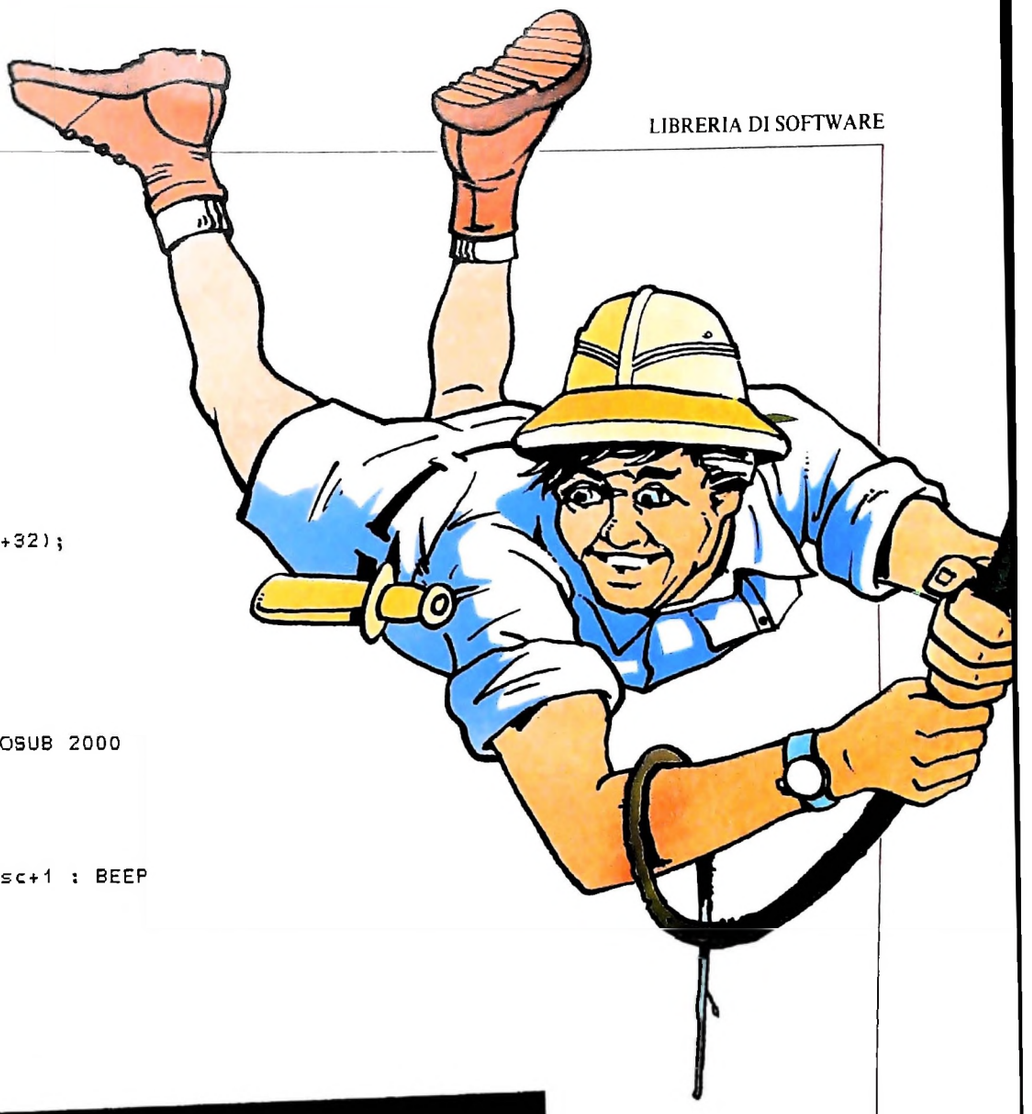
Il programma

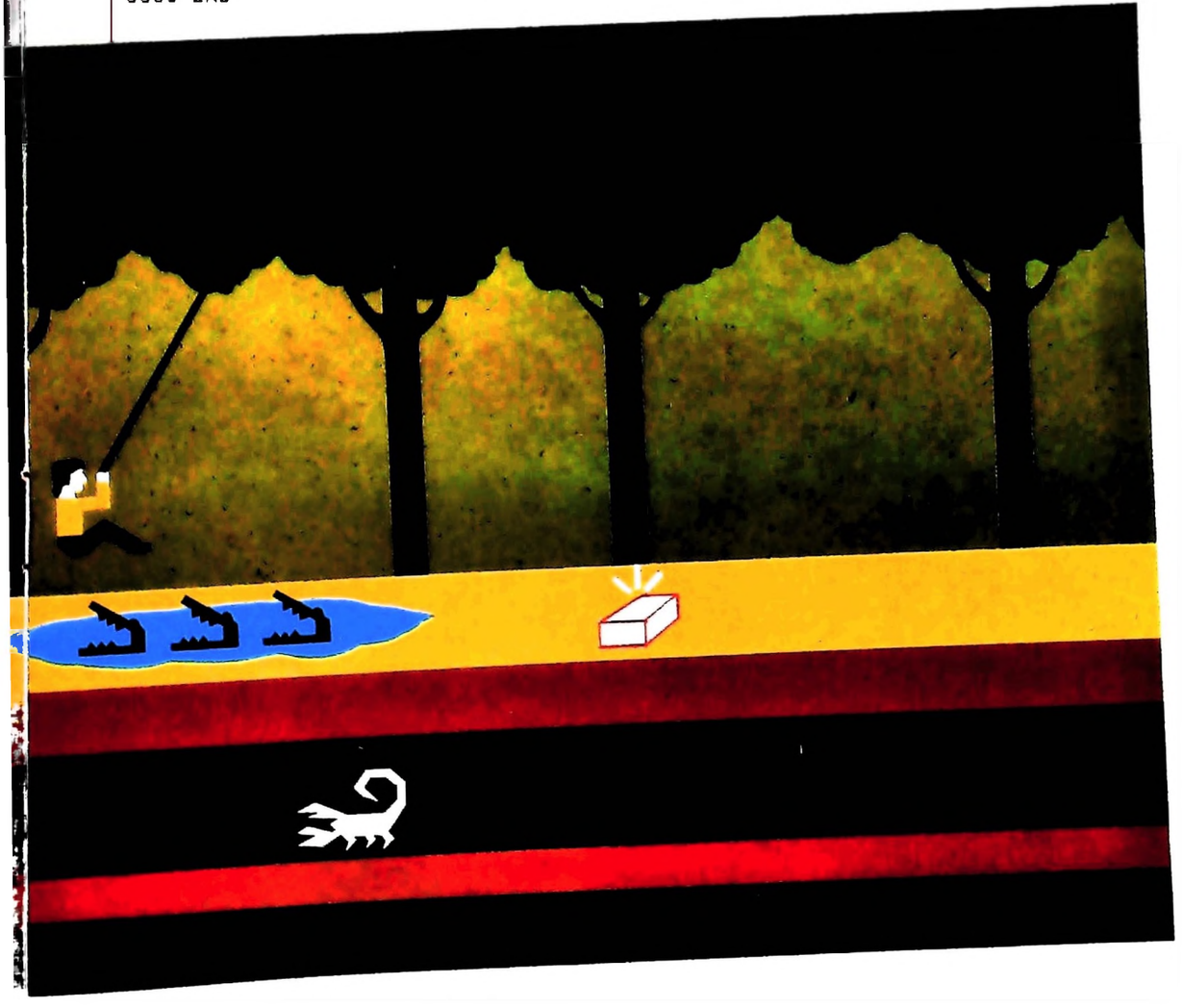
Questo programma vuole mostrare gli algoritmi sottesi a quei giochi classificabili come "JUMP". Essi generalmente mostrano degli oggetti (in questo caso sono dei palloncini) che si muovono velocemente lungo la zona alta dello schermo entrando dalla parte sinistra e uscendo dalla parte destra. Il giocatore è dotato invece di uno strumento (in questo caso un coltello) con il quale può, saltando in maniera appropriata, colpire gli oggetti in movimento. In questo programma il coltello viene mosso dai seguenti tasti:

"A": spostati a sinistra
"D": spostati a destra
" ": salta.

```
20   ESC$=CHR$(27
30   cx=20 : cy=7
50   cch$="|" : dd=-1
60   sp%=0 : si=0 : sc=0
100  DIM PA(4,40),S$(4)
105  FOR i=1 TO 4
110   FOR x=0 TO 39
120    PA(i,x)=0
130  NEXT x : NEXT i
135  FOR i=1 TO 4
140   FOR x=(i-1) TO 39 STEP 4
150    PA(i,x)=1
160  NEXT x : NEXT i
170  S$(1)="o   o   o   o   o   o   o   o   o   o   o   "
180  S$(2)=" o   o   o   o   o   o   o   o   o   o   o  "
190  S$(3)="  o   o   o   o   o   o   o   o   o   o   o "
200  S$(4)="   o   o   o   o   o   o   o   o   o   o   o"
480  CLS
490  ch$=cch$
500  x=cx : y=cy : GOSUB 8000
550  x=0  : y=0  : GOSUB 8000
600  GOSUB 1000
999  GOTO 9000
1000 REM scrive i palloncini
1010 si=si+1
1020 IF si > 4 THEN si=1
1030 x=0 : y=0 : GOSUB 7000
1040 PRINT S$(si);
1050 RETURN
2000 IF ky$="A" THEN GOTO 2100
2020 IF ky$=" " THEN GOTO 2200
2030 IF ky$="D" THEN GOTO 2300
2040 IF ky$="F" THEN GOTO 2400
2090 GOTO 2999
2100 IF sp%=1 THEN GOTO 2999
2105 ch$=" "
2107 x=cx : y=cy : GOSUB 8000
2110 cx=cx-1 : IF cx < 0 THEN cx=0
2120 ch$=cch$
2130 x=cx : y=cy : GOSUB 8000
2140 IF sp%=0 THEN px=cx
```

```
2199 GOTO 2999
2200 IF sp%=1 THEN GOTO 2999
2230 sp%=1
2280 GOTO 2999
2300 IF sp%=1 THEN GOTO 2999
2305 ch$=" "
2307 x=cx : y=cy : GOSUB 8000
2310 cx=cx+1 : IF cx>39 THEN cx=39
2320 ch$=cch$
2330 x=cx : y=cy : GOSUB 8000
2340 IF sp%=0 THEN px=cx
2350 GOTO 2999
2400 GOTO 9800
2999 RETURN
7000 IF x<0 OR x>39 THEN GOTO 7100
7010 IF y<0 OR y>7 THEN GOTO 7100
7020 PRINT ESC$+"Y"+CHR$(y+32)+CHR$(x+32);
7100 RETURN
8000 IF x<0 OR x>39 THEN GOTO 8100
8010 IF y<0 OR y>7 THEN GOTO 8100
8020 GOSUB 7000
8030 PRINT ch$;
8100 RETURN
9000 REM programma principale
9030 ky$=INKEY$
9040 IF ky$="" THEN GOSUB 1000 ELSE GOSUB 2000
9050 IF sp%=0 THEN GOTO 9070
9055 ch$=" "
9057 x=cx : y=cy : GOSUB 8000
9060 cy=cy+dd
9061 IF cy=0 THEN dd=1
9062 IF cy=0 AND PA(si,cx)=1 THEN sc=sc+1 : BEEP
9064 ch$=cch$ : y=cy : GOSUB 8000
9065 IF cy <> 7 THEN GOTO 9070
9066 sp%=0 : dd=-1
9068 ch$=cch$
9069 x=cx : y=cy : GOSUB 8000
9070 GOTO 9030
9800 CLS
9810 PRINT "punteggio = ";sc
9999 END
```





Con Harry Pitfall si va nella giungla alla ricerca di tesori fantastici, affrontando coccodrilli, sabbie mobili, passaggi sotterranei, serpenti velenosi e mille altri pericoli. Si dovrà condurre a termine questa affascinante e avventurosa caccia al tesoro entro un tempo limite di venti minuti.

PROLOG

È, al pari di Lisp, uno dei linguaggi propri dell'Intelligenza Artificiale, usato per realizzare sistemi esperti e sistemi per il trattamento del linguaggio naturale.

PROLOG è una parola composta dalle sillabe iniziali di due altre parole. In italiano possiamo dire che PROLOG sta per PROgramma LOGico.
PROLOG, come Lisp, è nato al di fuori dell'area anglosassone, ed è uno dei linguaggi propri del campo della Intelligenza Artificiale e molto diverso dai linguaggi tradizionali come Basic, Pascal o Cobol.
Dentro di noi abbiamo delle conoscenze sul mondo che ci circonda. Per esempio conosciamo un certo numero di persone e di essi sappiamo se sono sposati o se hanno dei figli. Per esempio supponiamo di sapere che un nostro vicino di casa di nome Marco ha due figli, Maria e Andrea. Inoltre sappiamo che Marco ha una madre di nome Lina e un padre di nome Gino.
Ora rileggiamo bene la frase precedente. Ciò che questo articolo vi ha detto è soltanto che Maria e Andrea sono figli di Marco e che Marco è figlio di Lina e Gino. Non vi è stato detto altro, eppure se vi chiedessi di rispondere alla domanda "Chi sono la nonna e il nonno di Maria e Andrea?" voi sapreste senza dubbio rispondere. Non solo, l'avreste fatto anche se vi avessi fornito dei nomi cinesi a voi del tutto incomprensibili. Se invece vi chiedessi "Chi sono la biznonna e il biznonno di Maria e Andrea?" non sapreste rispondere e potreste chiedermi "Cosa è il biznonno di qualcuno?"
Affrontiamo la prima questione. Che conoscenze usiamo per giungere ai nomi della nonna e del nonno di Maria e Andrea? La conoscenza fornita dall'articolo non sarebbe sufficiente di per sé, ma dentro di noi abbiamo già della conoscenza sul significato della parola NONNA e NONNO. Noi sappiamo che se Maria è figlia di Marco e Marco è a sua volta figlio di Gino allora Gino è il NONNO di Maria. Ma questa conoscenza non è specificata solo per Gino, Marco e Maria. In realtà noi sappiamo che se un tale è figlio di un tal altro e che questo tal altro stesso è figlio di un tal altro ancora allora quest'ultimo è il NONNO del primo.
Quest'ultima frase ci dice cosa è il NONNO di qualcuno. È evidente che non importano i nomi, ma il sapere se è vero che il primo è figlio del secondo e il secondo del terzo: NONNO è cioè una relazione definita sulla conoscenza circa i figli.
L'articolo ci forniva dunque la conoscenza circa i figli (e se i nonni fossero stati cinesi andava bene lo stesso), noi stessi abbiamo la conoscenza della relazione NONNO basata sulla conoscenza dei figli; queste due cose insieme ci consentono di rispondere alla domanda sui nonni.

Alla domanda sui BIZNONNI non sappiamo assolutamente rispondere. Chiaramente ci manca la definizione della relazione BIZNONNO e infatti ci sorge spontanea la domanda "Cosa è il BIZNONNO di qualcuno?". Questo conferma la nostra ipotesi circa la conoscenza invece già presente circa la relazione NONNO, e infatti basterà dare questa conoscenza perché chiunque sappia rispondere anche a tutte le domande sui BIZNONNI. Per esempio potremmo dire che il BIZNONNO di qualcuno è un altro modo di dire NONNO, ed ecco che tutti sapremmo perfettamente, magari un po' annoiati, rispondere sul BIZNONNO, che non sarebbe più indefinito (e sapremmo farlo anche su una famiglia cinese).
Ma supponiamo che io vi dica che il BIZNONNO di qualcuno è il figlio maschio del figlio maschio del nonno di quel qualcuno. Ora chi è il BIZNONNO di Andrea e chi è il BIZNONNO di Maria? Pensateci un attimo. La risposta è in entrambi i casi Andrea. Quello che ci interessa è che avete dovuto pensarci un attimo, mentre con il nonno la risposta era immediata, o almeno così sembrava. Questo pensarci un attimo non è privo di significato, ed è la conseguenza del fatto che non basta sapere tutto sui figli e sulla relazione definita da BIZNONNO, ma che occorre anche applicare la conoscenza su BIZNONNO al particolare quadro di figli fornito. Se descrivessi una numerosa famiglia cinese chiedendovi di dirmi il BIZNONNO di questo e quello, voi dovreste ogni volta applicare la definizione alla conoscenza sulla famiglia fino a calcolare il BIZNONNO.
Tutto questo è realizzabile in PROLOG. Anzi, PROLOG è fatto proprio per poter scrivere dei programmi che riflettano immediatamente il tipo di ragionamento logico usato fin qui. Riflettendo sul nostro discorso abbiamo parlato delle conoscenze sui figli fornite nel dialogo; questo tipo di conoscenze le chiameremo tutte insieme con un termine unico BASE DATI. Abbiamo parlato di NONNO e BIZNONNO come di RELAZIONI. Insieme la base dati e le relazioni forniscono una BASE DI CONOSCENZA. Infine abbiamo parlato (poco) dell'applicazione di una relazione a una base dati, cioè di come riusciamo ad applicare la definizione della relazione BIZNONNO alle conoscenze sui figli e a calcolare il BIZNONNO di qualcuno.
Mentre base dati e relazioni sono abbastanza chiare, l'applicazione delle seconde alla prima è qualcosa che sappiamo fare, ma non sappiamo come mai.
Analogamente in PROLOG un programma è fatto solo della base dati e delle relazioni. Il meccanismo tramite il quale si

Lezione 56

Ancora sulla gestione degli errori

Nella lezione precedente abbiamo imparato a usare le istruzioni ON ERROR, ERROR e RESUME. Ora le applicheremo al caso della gestione degli errori nel programma "robusto" sul programma del calcolo delle piastrelle. Il programma era composto di una sezione principale, che leggeva i dati come stringhe e che richiamava un sottoprogramma per convertirle in valore numerico; verificava la correttezza formale segnalata dal sottoprogramma e il corretto valore rispetto al problema, iterando la lettura in caso di errori; sviluppava i calcoli e visualizzava i risultati. Per modificarlo verso una migliore gestione degli errori decidiamo che:

- Il sottoprogramma segnalerà gli errori formali relativi alle stringhe, mentre il programma principale segnalerà gli errori rispetto alla natura del problema;
- useremo codici da 100 in su per gli errori di forma, e da 200 in su per gli altri;
- amplieremo i messaggi diagnostici sugli errori formali.

Allora, potremo cominciare a riscrivere il programma principale come segue:

```
1 'Gestione errori centralizzata
2 'Errori
3 '100-Segno interm. 200-Lato st.<=0
4 '101-Piu' punti    201-Lato pias.<=0
5 '102-Lettere
7 '103-Altri crt
8 ON ERROR GOTO 2000
10 LINEINPUT "LATO DELLA STANZA (in m)?";A$
12 GOSUB 1000 'Conversione
13 IF R=1 THEN 10 'Errore: ripetere
14 IF A<=0 THEN ERROR 200: GOTO 10
15 LET L=A*100 'Lato espresso in cm
20 LINEINPUT "LATO DELLA PIASTRELLA (in cm)?";A$
22 GOSUB 1000 'Conversione
24 IF R=1 THEN 20 'Errore:ripetere
25 IF A<=0 THEN ERROR 201:GOTO 20
26 LET P=A
30 LET A=L*L 'Area della stanza in cm
40 LET N=INT(L/P) 'n.piastrelle su un lato
della stanza
45 IF N*P<L THEN LET N=N+1
50 LET M=N*N 'n.piastrelle necessarie
60 PRINT "N.PIASTRELLE NECESSARIE:";M
70 LET S=M*P*P-A 'Area tot piastrelle -
area stanza
80 PRINT "SCARTO (in cmq):";S
90 PRINT "SCARTO PERCENTUALE:";INT(S/A*100+
.5)/100
100 END
```

In esso osserviamo:

- l'istruzione 6 indica che la routine di trattamento errori si trova a 2000;

- l'istruzione 13: mentre prima questa si occupava di segnalare l'errore, ora si deve occupare solo di controllare che l'errore ci sia stato (controllando la variabile R), e di iterare la richiesta di dati in caso affermativo;
- l'istruzione 14, invece, come stabilito, segnala anche l'errore;
- analoghe considerazioni valgono per le istruzioni alle linee 24 e 25.

Poiché le linee 14 e 25 hanno il GOTO dopo la segnalazione errore dobbiamo usare istruzioni RESUME NEXT nel trattamento degli errori, in modo che l'istruzione successiva al trattamento errore sia tale salto che rinvia a una nuova lettura.
Passiamo ora a esaminare quali variazioni introduciamo nel sottoprogramma:

```
1000 'Accetta una stringa in A$ come numero
1002 ' Se e' corretta, A=valore e R=0
1004 ' Altrimenti A=0 e R=1
1010 L9=LEN(A$) 'Lung. di A$ in L9
1015 A=0 'Val. iniz. del risult
1020 I9=0 'Val. iniz. parte intera
1025 D9=0 'Val. iniz. parte decim.
1030 S9=1 'Segno: inizialm. +
1031 'Scarta blank iniziali
1032 'While primo crt=" " do
1033 Z$=LEFT$(A$,1) 'Primo crt in Z$
1034 IF Z$=" " THEN A$=RIGHT$(A$,L9-1):L9=L9-1:
GOTO 1032
1035 'Endwhile
1036 'Cerca + o - iniz e registra segno
1037 IF Z$="+" THEN A$=RIGHT$(A$,L9-1):L9=L9-1:
GOTO 1040
1038 IF Z$="-" THEN S9=-1:A$=RIGHT$(A$,L9-1):L9=
L9-1 'Segno -
1040 'While ci sono crt<>"." do
1050 Z$=LEFT$(A$,1):A$=RIGHT$(A$,L9-1):L9=L9-1
1060 IF Z$=" " THEN 1120 'Elimina blank
1085 IF Z$="." THEN 1130 'A calcolo decimale
1090 IF Z$<="9" AND Z$>="0" THEN I9=I9*10+VAL(
Z$):GOTO 1120 'Costruz. valore
1100 R=1:GOSUB 1500: GOTO 1230 'Errore
1120 IF L9>0 THEN 1040
1125 'Endwhile
1130 'Calcolo parte decimale
1150 L9=LEN(A$) 'Lung. parte decim.
1155 E9=1 'Posiz.decimale
1157 'While ci sono crt do
1160 IF L9=0 THEN 1210
1170 LET Z$=LEFT$(A$,1):A$=RIGHT$(A$,L9-1):L9
=L9-1
1175 IF Z$=" " GOTO 1200 'Elimina blank
1180 IF Z$<"0" OR Z$>"9" THEN R=1:GOSUB 1500:
GOTO 1230 'Errore
1190 LET D9=D9+VAL(Z$)/10^E9:E9=E9+1 'Calcolo
e increm. posiz. decimale
1200 GOTO 1160
1205 'Endwhile
```

```
1210 LET A=S9*(I9+D9) 'Risultato finale
1220 LET R=0
1230 RETURN
```

- il ciclo da 1032 a 1035 non è stato alterato, in quanto non coinvolto;
- analogamente, non è stato alterato il controllo del segno nelle istruzioni immediatamente successive;
- le istruzioni 1060 e 1085 non possono essere legate a errori: la prima, infatti, salta, mentre la seconda viene eseguita con successo solo per il primo ".";
- qualora invece l'istruzione 1090 fallisca a causa di caratteri non ammessi, il controllo passa alla successiva; inseriremo qui il rilevamento dell'errore, e poiché abbiamo deciso di ampliarne l'analisi, invece di inserire direttamente istruzioni ERROR, provvediamo a richiamare un sottoprogramma (che collocheremo all'istruzione 1500) allo scopo; gli errori che potremo qui rilevare saranno: una lettera al posto di una cifra, un segno "+" o "-" al posto di una cifra, un altro carattere non ammesso.
- Analogamente, inseriamo il richiamo del sottoprogramma per l'analisi dell'errore nella linea 1180; il sottoprogramma può essere lo stesso, in quanto il carattere incriminato è sempre in Z$; tuttavia qui esiste un'ulteriore possibilità d'errore, che un altro punto decimale sia stato inserito (tali errori possono essere riassunti come "carattere illegale").

Possiamo ora passare alla stesura del sottoprogramma di analisi degli errori:

```
1500 'Analisi errore
1510 IF Z$="+" OR Z$="-" THEN ERROR 100:GOTO 1600
1520 IF Z$="." THEN ERROR 101:GOTO 1600
1530 IF Z$>="A" AND Z$<="Z" OR Z$>="a" AND Z$<=
"z" THEN ERROR 102:GOTO 1600
1540 ERROR 103 'Altri caratteri
1600 RETURN
```

In esso, a seconda dei vari contenuti di Z$ vengono invocati i vari errori con codici da 100 a 103. Infine, costruiamo il sottoprogramma di trattamento degli errori:

```
2000 'Gestione errori
2010 IF ERR>=200 THEN 2500
2020 'Errori di forma
2030 ON ERR-99 GOTO 2040,2050,2060,2080
2040 PRINT"Segno a meta' numero!"
2045 GOTO 2100
2050 PRINT "Punto decimale multiplo!"
2055 GOTO 2100
2060 PRINT "Lettere non ammesse!"
2065 GOTO 2100
2080 PRINT "Carattere illegale!"
2100 'Fine errori di forma
2110 GOTO 2700
2500 'Errori di valore
2510 ON ERR-199 GOTO 2520,2530
2520 PRINT "Una stanza non puo' avere lato
negativo o nullo!"
```

```
2525 GOTO 2600
2530 PRINT "Una piastrella deve avere lato
maggiore di zero!"
2600 'Fine errori di valore
2700 RESUME NEXT
```

In esso si distinguono nettamente le due parti relative al trattamento degli errori di forma e di valore, sulla base del valore assunto da ERR; in ambedue i casi un'istruzione ON... GOTO provvede allo smistamento del controllo per la visualizzazione del messaggio d'errore più appropriato; alla fine, in ogni caso, l'esecuzione del sottoprogramma termina con un'istruzione RESUME NEXT, che garantisce il ritorno al programma in cui l'errore è stato riscontrato, a partire dall'istruzione successiva a quella in cui esso è occorso, e quindi garantisce la prosecuzione dell'esecuzione.
Vediamo ora come funziona il nostro programma; inseriremo dati che scatenino i vari tipi di errore che abbiamo inserito.

• lato piastrella zero (errore 201)

```
run
LATO DELLA STANZA (in m)?quattro
Lettere non ammesse!
LATO DELLA STANZA (in m)?-1
Una stanza non puo' avere lato negativo
o nullo!
LATO DELLA PIASTRELLA (in cm)?2.$
Carattere illegale!
LATO DELLA PIASTRELLA (in cm)?0
Una piastrella deve avere lato maggiore
di zero!
LATO DELLA STANZA (in m)?4.+0
Segno a meta' numero!
LATO DELLA STANZA (in m)?4
LATO DELLA PIASTRELLA (in cm)?2.2.2
Punto decimale multiplo!
LATO DELLA PIASTRELLA (in cm)?20
N.PIASTRELLE NECESSARIE: 400
SCARTO (in cmq): 0
SCARTO PERCENTUALE: 0
Ok
```

Cosa abbiamo imparato

In questa lezione abbiamo visto:

- l'uso pratico delle istruzioni
- •ON ERROR
- •ERROR
- •RESUME.

riesce a derivare per esempio il BIZNONNO di qualcuno una volta fornita la relazione e la base dati non va specificato e "non si vede". PROLOG è in grado di capire relazioni e base dati e di applicare le une alle altre. Un programma PROLOG è un programma "logico" proprio perché bisogna fornirgli le conoscenze di base e le relazioni, ma non è necessario dire "come" usarle.
Scriviamo un programma PROLOG che calcoli NONNI e BIZNONNI.

Programmi logici

Per dire che Andrea è figlio di Marco possiamo scrivere:

```
figlio(andrea,marco).
```

Questa è una linea di programma PROLOG. Vedremo come viene usata fra poco. I nomi sono scritti minuscolo perché le parole che iniziano maiuscolo hanno per PROLOG un significato preciso che vedremo scrivendo le definizioni delle relazioni NONNO e BIZNONNO. Questa linea di programma afferma che Andrea è figlio di Marco. Qui sotto c'è la base dati del dialogo completa:

```
figlio(andrea,marco).
figlio(maria,marco).
figlio(marco,gino).
figlio(marco,lina).
```

Queste relazioni di parentela per noi sono dei fatti da cui partire per il nostro dialogo di domande e contro-domande. Anche in PROLOG questo tipo di istruzioni si chiamano FATTI. Così per dare un esempio di un altro fatto se Maria possedesse un gatto potremmo aggiungere questa conoscenza alla base dati modificandola in:

```
figlio(andrea,marco).
figlio(maria,marco).
figlio(marco,gino).
figlio(marco,lina).
possiede(maria,gatto).
```

Le parole "figlio" e "possiede" vengono chiamati PREDICATI e chiaramente hanno un significato diverso dai nomi racchiusi dentro alle parentesi che sono gli ARGOMENTI del predicato.
Anzi, il predicato ci parla dei suoi argomenti. Così gli argomenti del primo fatto, cioè "andrea" e "marco" di per se stessi non ci dicono molto di più degli argomenti dell'ultimo fatto cioè "maria" e "gatto". Sono i relativi predicati a differenziare il senso che diamo a questi due fatti.

Tornando al nostro dialogo abbiamo detto che abbiamo dentro di noi una definizione della relazione NONNO in termini di figli. A una persona con le idee confuse circa i nonni potremmo dare la regola: "se Tizio è figlio di Sempronio e Sempronio è figlio di Caio allora Caio è il nonno di Tizio". Qui "Tizio", "Caio" e "Sempronio" sono individui di comodo. Non stiamo davvero parlando di Tizio Caio e Sempronio ma ognuno di essi può essere un individuo qualsiasi, per esempio possono essere rispettivamente Andrea, Marco e Gino, ma anche Maria, Marco e Gino. Tizio Caio e Sempronio sono VARIABILI che danno un valore generale alla nostra relazione. In PROLOG NONNO è così definita:

```
nonno(Tizio,Sempronio) :-
figlio(Tizio,Caio),figlio(Caio,Sempronio).
```

Tizio Caio e Sempronio iniziano maiuscolo proprio per farsi riconoscere come variabili. Abbiamo scritto una REGOLA: Sempronio è il NONNO di Tizio solo se è vero che Tizio è figlio di Caio e Caio di Sempronio. È un po' alterato l'ordine in cui esprimiamo la relazione, ma tutti converranno che logicamente non è cambiato nulla.
A questa nuova espressione possiamo però dare un'altra interpretazione: possiamo dire che il problema di trovare il nonno di Tizio è riducibile ai due problemi di trovare prima di chi è figlio Tizio e successivamente di chi è figlio la persona trovata come padre di Tizio.
Abbiamo cioè una scomposizione di un problema in più sottoproblemi più semplici e immediatamente risolvibili a partire dalla base dati. Il programma PROLOG è terminato; chiamiamolo correttamente "base di conoscenza" e vediamo come si presenta:

```
figlio(andrea,marco).
figlio(maria,marco).
figlio(marco,gino).
figlio(marco,lina).
possiede(maria,gatto).
nonno(Tizio,Sempronio) :- figlio(Tizio,Caio),
                          figlio(Caio,Sempronio).
```

Ora vogliamo far andare questo programma. Una prima caratteristica dei programmi logici è che non esiste un unico modo per lanciare i programmi. In PROLOG possiamo partire un po' dovunque e usare il programma per scopi diversi. Per esempio prima di chiederci chi è il nonno di Andrea e Maria possiamo chiederci cosa possiede Maria. È una domanda che facciamo all'interprete di PROLOG che, come abbiamo detto, sa come utilizzare i suoi dati da solo. La domanda cosa possiede Maria va scritta così:

```
?- possiede(maria,Cosa).
```

"Cosa" naturalmente è una variabile (infatti inizia maiuscolo). Il punto di domanda iniziale segnala all'interprete PROLOG che gli stiamo facendo una domanda, e che deve utilizzare il programma. PROLOG scandisce allora l'intera base di conoscenza alla ricerca di un fatto che incominci con lo stesso predicato. Trova "possiede(maria,gatto)." Poiché hanno lo stesso predicato l'interprete passa a confrontare i vari argomenti. Questa fase è detta UNIFICAZIONE perché si tenta di ridurre un'espressione all'altra, unificandole, assegnando i valori appropriati alle variabili. Nel nostro caso il primo argomento è identico per entrambe e il secondo lo è anch'esso se assegniamo alla variabile "Cosa" il valore "gatto". L'interprete ci risponderà con l'espressione:

```
Cosa=gatto
```

In questo modo noi sappiamo che se "gatto" viene sostituito a "cosa" allora il problema "? - possiede(maria,Cosa)" possiede una soluzione nella base di conoscenze. Dunque "gatto" è proprio ciò che Maria possiede. Possiamo però utilizzare altre parti del programma logico: per esempio chiediamoci se Maria è figlia di Marco:

```
?- figlio(maria,marco).
yes
```

L'interprete riconosce il predicato e cerca fra i fatti che incominciano con il predicato "figlio" qualcuno che unifichi con la nostra domanda. Il primo non va bene perché "marco" non è una variabile e non è eguale a "maria", ma il secondo va benissimo, e l'interprete ci ritorna "yes": è vero che maria è figlia di marco. Ma se chiediamo se è vero che Maria è figlia di Lina la risposta sarà "no" perché nessun fatto è unificabile con la nostra domanda:

```
?- figlio(maria,lina).
no
```

Fino a questo punto ci siamo limitati a constatare la veridicità dei fatti, e abbiamo ottenuto delle risposte su quali oggetti soddisfano questi predicati, come per il gatto posseduto da Maria. Proviamo a utilizzare la regola. Chiediamoci chi è il nonno di Maria:

```
?- nonno(maria,Tale).
```

Stiamo chiedendoci chi è quel Tale che è nonno di maria. Avrete notato che Tale è maiuscolo e dunque è una variabile. L'interprete scandisce la base di conoscenze fino a giungere alla regola che incomincia con il predicato nonno. Abbiamo detto che questa regola scompone il problema di trovare il nonno in due sottoproblemi: è evidente che il problema da scomporre è sintetizzato dalla formula (simile a un fatto) che precede il simbolo ":<", e che i due sottoproblemi in cui viene scomposta sono le formule (anch'esse simili a un fatto) separate da virgole che seguono il simbolo. L'interprete cerca una unificazione fra la nostra domanda e la prima parte della regola (che chiameremo d'ora in avanti TESTA della regola per distinguerla dal resto che chiameremo CORPO della regola). Il primo argomento della testa della regola è la variabile "Tizio". A questa variabile viene assegnato il valore "maria". Il secondo argomento è ancora una variabile per entrambe le formule. "Sempronio" resta una variabile, ma "Tale" viene legato a "Sempronio". Questo è un legame fra variabili ancora senza valore, il senso è che qualsiasi valore venga assegnato a "Sempronio" sarà immediatamente anche assegnato a "Tale". Fatta l'unificazione l'interprete "applica" la regola, e cioè scompone il problema iniziale nei due sottoproblemi definiti dalla regola. Ma gli assegnamenti fatti modificano in parte i due sottoproblemi. L'interprete al suo interno si pone il (doppio) problema:

```
figlio(maria,Caio),figlio(Caio,Sempronio).
```

Non appare più Tizio perché alla variabile "Tizio" è stato assegnato il valore "maria". Anche se questa scritta non appare l'interprete cercherà di risolvere questi due problemi proprio come se avessimo richiesto esplicitamente:

```
?- figlio(maria,Caio),figlio(Caio,Sempronio).
```

L'interprete parte con il primo sottoproblema "figlio(maria,Caio)", scandisce la base di conoscenze e cerca delle unificazioni fra le espressioni con predicato "figlio". Il secondo fatto consente un'unificazione il cui risultato è di assegnare alla variabile Caio il valore "marco". Il primo sottoproblema è stato risolto. L'interprete passa al secondo sottoproblema che però non è più lo stesso. Infatti assegnando il valore "marco" alla variabile Caio abbiamo modificato il secondo sottoproblema che ora appare come:

```
figlio(marco,Sempronio).
```

L'interprete scandisce nuovamente la base di conoscenze alla ricerca di un fatto o una testa di regola che unifichi con il sottoproblema. Il terzo fatto è adatto e alla variabile "Sempronio" viene assegnato il valore "gino". Con questi valori assegnati alle variabili i due sottoproblemi sono verificati nella base di conoscenze.
Dunque è verificato anche il problema "nonno(maria,Sempronio)" ed è stato assegnato il valore "gino" alla variabile "Sempronio". Come detto precedentemente questo valore viene subito assegnato anche a "Tale" ed è questa risposta che ci ritorna l'interprete PROLOG. Per comodità riscriviamo anche la domanda:

```
?- nonno(maria,Tale).
Tale=gino
```

Se avessimo richiesto direttamente i due sottoproblemi avremmo avuto questa risposta:

```
?- figlio(maria,Caio),figlio(Caio,Sempronio).
Caio=marco
Sempronio=gino
```

L'interprete ci mostra come risposta solo i valori delle variabili presenti nella domanda, senza mostrarci tutte quelle coinvolte nel processo di verifica e scomposizione in sottoproblemi (PROLOG consente anche questo ma va richiesto esplicitamente con una funzione di nome TRACE).

Il backtracking

In realtà "figlio(marco,gino)." non è l'unico fatto della base dati che unifica con il secondo sottoproblema "figlio(marco,Sempronio).". Anche il fatto sottostante "figlio(marco,lina)." unifica se assegniamo alla variabile Sempronio il valore "lina" e infatti anche "lina" è una possibile soluzione di "nonno(maria,Tale)." (questo anche se il predicato "nonno" è maschile; infatti non abbiamo specificato il sesso delle possibili soluzioni). Per ottenere dall'interprete PROLOG anche questa soluzione basterà scrivere un ";" dopo la prima soluzione "Tale=gino". L'interprete (vedremo fra poco come) cerca una nuova soluzione diversa dalla precedente. In questo caso trova "Tale=lina". Se scriviamo un nuovo ";" chiedendo un ulteriore valore per il "tale" che è nonno di "maria" l'interprete, non trovandone, risponde "no":

```
?- nonno(maria,Tale).
Tale=gino;
Tale=lina;
no
```

Diamo un'occhiata in dettaglio all'importante meccanismo detto BACKTRACKING ("tornare sui propri passi") che consente all'interprete di PROLOG di trovare sempre nuove soluzioni diverse dalle precedenti, almeno finché ve ne siano. Pensiamo ai due sottoproblemi "figlio(maria,Caio), figlio (Caio,Sempronio).". Come già sappiamo il primo sottoproblema viene verificato assegnando il valore "marco" a Caio e modificando il secondo sottoproblema in "figlio(marco,Sempronio).". La verifica di quest'ultimo invece si ottiene con l'unificazione che assegna a Sempronio il valore "gino". A questo punto abbiamo una prima soluzione. Chiediamo con il ";" di trovarne un'altra. È come se fossimo insoddisfatti della prima soluzione. Chiaramente l'interprete non deve ricominciare a scandire la base di conoscenza dal principio, altrimenti ritroverà la stessa soluzione. L'interprete invece per ogni sottoproblema ricorda sempre il punto in cui ha trovato una formula, fatto o testa, che unifica. Nel cercare una nuova soluzione l'interprete agisce come se avesse sbagliato a fermarsi su quella formula e riprende a scandire la base di conoscenza a partire dal punto cui era arrivato. La ricerca di nuove soluzioni riprende dall'ultimo sottoproblema per tornare pian piano sui propri passi. Nel nostro caso l'ultimo sottoproblema era "figlio(marco,Sempronio).". L'interprete "scorda" l'ultima unificazione e libera la variabile Sempronio dal valore "gino". Ricomincia a cercarne una a partire dal punto cui era arrivato, cioè "figlio(marco,gino).". Gli basta fare un passo e trova il fatto "figlio(marco,lina)." che unifica se assegniamo a Sempronio il valore "lina". Ecco trovata una nuova soluzione. Un nuovo ";" chiede di ignorare anche questa soluzione? L'interprete butta via l'assegnamento di "lina" a Sempronio e riparte a cercare, ma non trova nulla. Ma ancora non è tutto perduto: supponiamo di avere aggiunto in precedenza sul fondo della base di conoscenza i due fatti "figlio(maria,franca)." e "figlio(franca,umberto).". La base di conoscenza si presenterà come:

```
figlio(andrea,marco).
figlio(maria,marco).
figlio(marco,gino).
figlio(marco,lina).
possiede(maria,gatto).
nonno(Tizio,Sempronio) :- figlio(Tizio,Caio),
                          figlio(Caio,Sempronio).
figlio(maria,franca).
figlio(franca,umberto).
```

Quanto detto fino adesso è ancora perfettamente vero anche su questa nuova base di conoscenza ampliata. Torniamo all'interprete. Abbiamo detto che non trova nessuna nuova formula per "figlio(marco,Sempronio)". Allora "torna sui suoi passi" e rivolge l'attenzione al sottoproblema precedente, cioè a "figlio(maria,Caio)." dove a Caio era stato assegnato il valore "marco". Vale la pena di cercare una nuova soluzione anche per questo sottoproblema, infatti una nuova soluzione significa un nuovo valore per la variabile Caio, e questo comporta una modificazione del secondo sottoproblema che contiene come primo argomento proprio la variabile Caio. Si dice che l'unificazione del primo sottoproblema determina, assegnando un valore a Caio, il CONTESTO in cui si cerca un'unificazione per il secondo sottoproblema. Riprendendo il discorso: l'interprete butta via anche l'unificazione del primo sottoproblema e libera anche la variabile Caio dall'assegnamento a "marco". Ricorda però la formula con cui aveva unificato, cioè "figlio(maria,marco)." e a partire da essa riparte a scandire la base di conoscenza. La nuova scansione incontra il nuovo fatto aggiunto "figlio(maria,franca)" che unifica e assegna a Caio il valore "franca". In questo

nuovo contesto si parte nuovamente con il secondo sottoproblema scandendo la base di conoscenza dal principio. Si riparte dal principio perché il secondo sottoproblema grazie al cambiamento di contesto è del tutto nuovo. Esso è infatti:

```
figlio(franca,Sempronio).
```

La scansione s'arresta sull'ultimo fatto della base "figlio (franca,umberto).". A Sempronio viene assegnato il valore "umberto" e viene riportata una nuova soluzione "Tale= umberto". Un ulteriore ";" metterebbe nuovamente in moto il meccanismo di backtracking senza però dare alcun frutto.
Ecco cosa vedremmo di tutto ciò che abbiamo descritto (sempre che non abbiamo usato la funzione TRACE):

```
?- nonno(maria,Tale).
Tale=gino;
Tale=lina;
Tale=umberto;
no
```

Parlando del backtracking, notiamo la funzione del simbolo "!" (detto TAGLIO): esso blocca il backtracking riservandolo solo ai sottoproblemi alla sua destra. Si tratta di collocarlo al posto giusto. Qui abbiamo introdotto nella regola per nonno il taglio in modo che si comporti come descritto:

```
nonno(Tizio,Sempronio) :-
    figlio(Tizio,Caio),!,figlio(Caio,Sempronio).
```

Va detto che se nella base dati il fatto "figlio(maria,franca)." precedesse quello "figlio(maria,marco)." allora la definizione della regola precedente cercherebbe solo i nonni materni.

I poteri di PROLOG

Una caratteristica di PROLOG è la multivalenza delle sue regole. Infatti la regola per trovare il nonno va benissimo anche per trovare i nipoti di un individuo. Se chiediamo:

```
?-nonno(Tale,gino).
```

L'interprete ci fornirà:

```
?-nonno(Tale,gino).
Tale=andrea;
Tale=maria;
no
```

Se vogliamo tutte le coppie nipote-nonno chiederemo:

```
?-nonno(Tale,Tal'altro).
Tale=andrea     Tal'altro=gino;
Tale=andrea     Tal'altro=lina;
Tale=maria      Tal'altro=gino;
Tale=maria      Tal'altro=lina;
Tale=maria      Tal'altro=umberto;
no
```

Maria ha un nonno in più perché non abbiamo specificato nella base dati che anche andrea è figlio di franca. Del resto questo può essere vero come non esserlo, e questo mette in luce come sia importante scrivere una base dati accurata. La regola usata per "nonno" è quella priva del taglio, quella con il taglio avrebbe fornito i soli nonni di andrea.
Ci rimane per completare il nostro dialogo iniziale di parlare di BIZNONNO. Questo ci dà l'occasione di mostrare come sia semplice estendere e migliorare un programma logico, davvero molto più semplice che mettere le mani in un programma scritto con linguaggi più tradizionali.
Ricordiamoci la definizione che avevamo dato di BIZNONNO: "un Tale è bisnonno di un Tal altro se è il figlio maschio del figlio maschio del nonno del Tal altro". Questa insistenza sul sesso del figlio ci costringe in primo luogo a descrivere nella nostra base dati il sesso degli individui. Introduciamo i fatti:

```
maschio(andrea).
maschio(marco).
maschio(gino).
maschio(umberto).
femmina(maria).
femmina(lina).
femmina(franca).
```

Ora possiamo definire una regola per la relazione "figlio-maschio". Diremo che X (variabile, è lo stesso che dire "Tale" ma è più conciso) è il figlio-maschio di Y se X è figlio di Y e X è maschio. Cioè:

```
figlio-maschio(X,Y) :- figlio(X,Y),maschio(X).
```

Questa regola dice che se il nonno di Z è Tale, e Tale ha un figlio maschio X che a sua volta ha un figlio maschio W, allora W è bisnonno di Z. Chiediamo:

```
?-biznonno(Tale,maria).
Tale=andrea;
no
```

IL SISTEMA OPERATIVO UCSD P-SYSTEM (II)

Vediamo quali sono le modalità con cui l'utente utilizza le varie funzionalità del sistema.

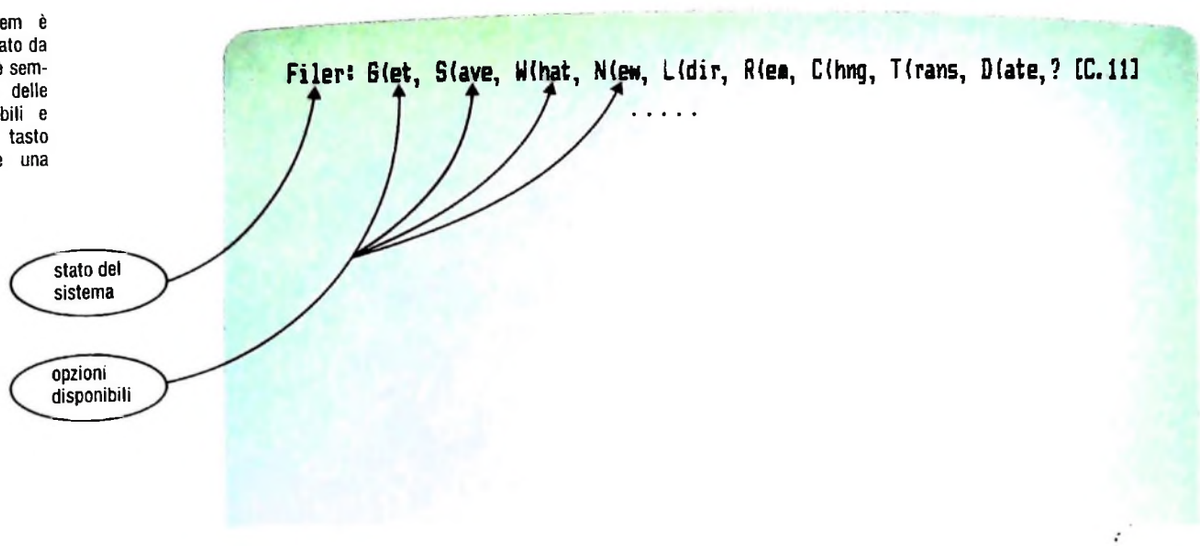
L'interfaccia utente

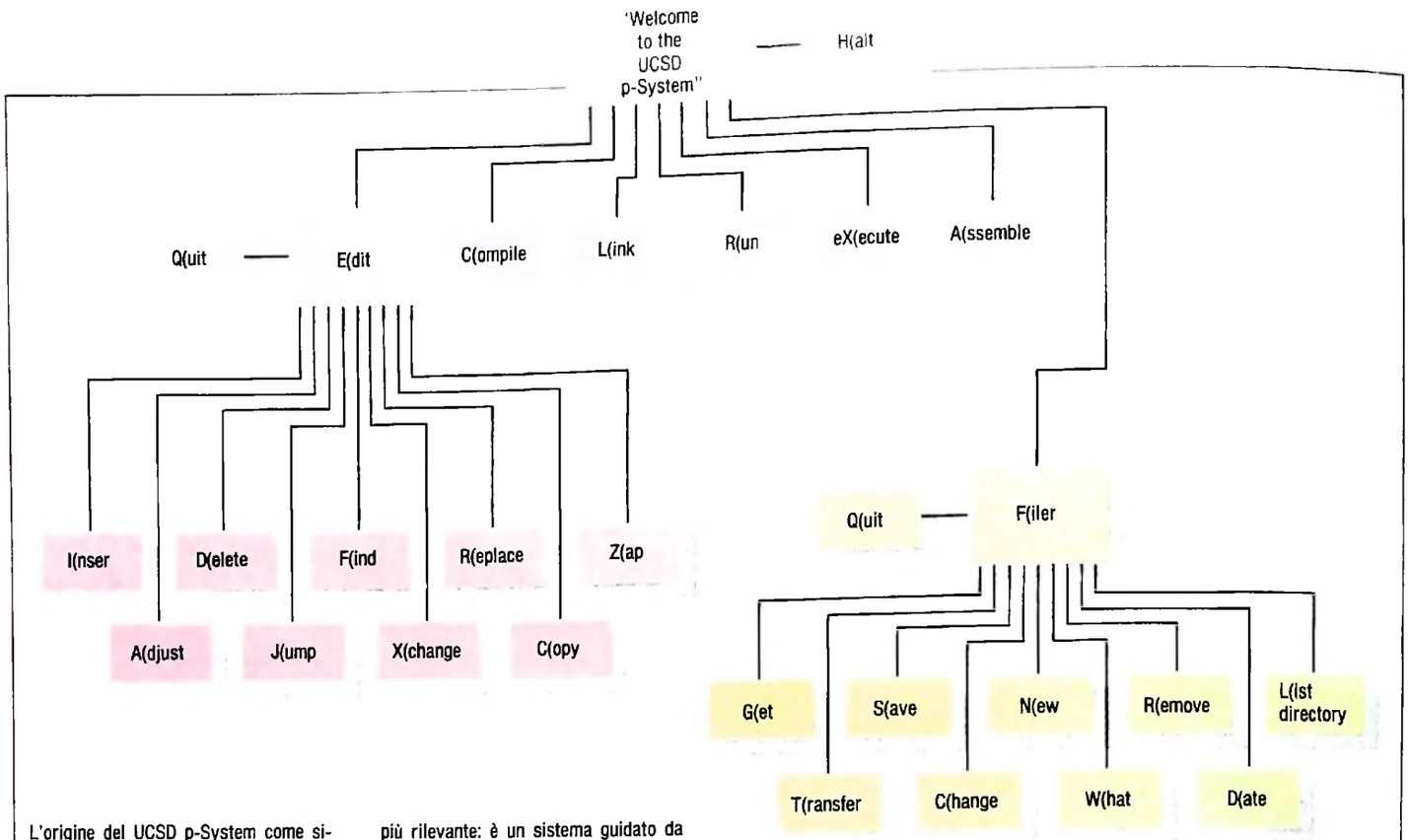
Abbiamo visto sinora le caratteristiche dell'UCSD p-System dal punto di vista della sua architettura e dei suoi componenti. Vediamo ora le caratteristiche del sistema dal punto di vista dell'interazione con l'utente.
È un sistema guidato da menù. In pratica, l'utente è costantemente informato dello stato del sistema e delle opzioni disponibili in quello stato: la prima riga dello schermo, detta *prompt-line*, elenca le opzioni disponibili, i cui nomi sono scelti in modo tale che le iniziali di ognuna siano tutte diverse; l'utente preme semplicemente il tasto corrispondente all'iniziale per richiamare una certa funzione (figura in basso).

Il menù principale

L'accesso alle varie funzioni avviene attraverso una struttura ad albero, in modo tale che operazioni logicamente correlate sono raggruppate in un unico menù. Nella figura della pagina seguente è riportata la struttura ad albero del menù principale. Vediamo i comandi più importanti.
Il comando E richiama lo *screen editor*, cioè il programma di composizione e modifica di testi normalmente utilizzato per la scrittura di programmi.
Il comando C richiama il compilatore (Pascal, Fortran o Basic, secondo la configurazione prescelta dall'utente): questo richiederà all'utente, a sua volta, le eventuali informazioni supplementari di cui ha bisogno.
Il comando L richiama il *linker*, il programma che include in un unico modulo eseguibile il programma utente, compilato, e le *unit* e le *routine* compilate separatamente.
Il comando X (execute) esegue un programma (richiama l'interprete del p-Code), il cui nome viene richiesto all'utente.
Il comando A richiama l'assemblatore, con modalità analoghe a quelle del comando C.
Il comando F richiama il *filer*, il programma di gestione dei file su disco.

L'UCSD p-System è un sistema guidato da menù. L'utente è sempre informato delle opzioni disponibili e premendo un tasto può richiamare una funzione.



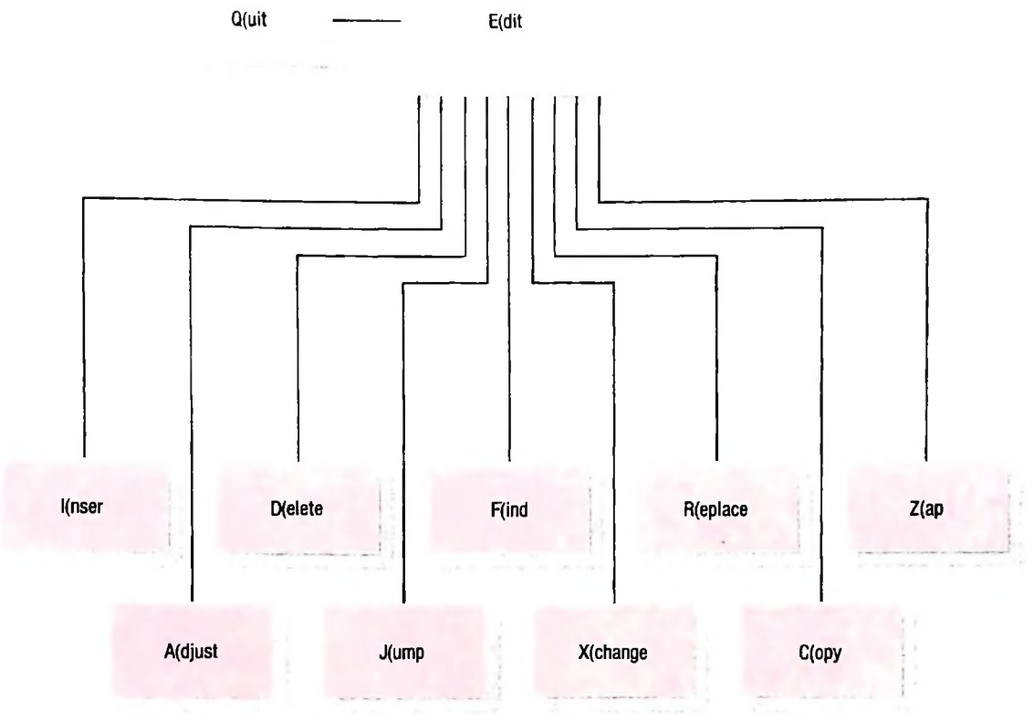


L'origine del UCSD p-System come sistema operativo orientato all'uso didattico ne ha determinato la caratteristica più rilevante: è un sistema guidato da menù. Nella figura a fianco la struttura ad albero del menù principale.

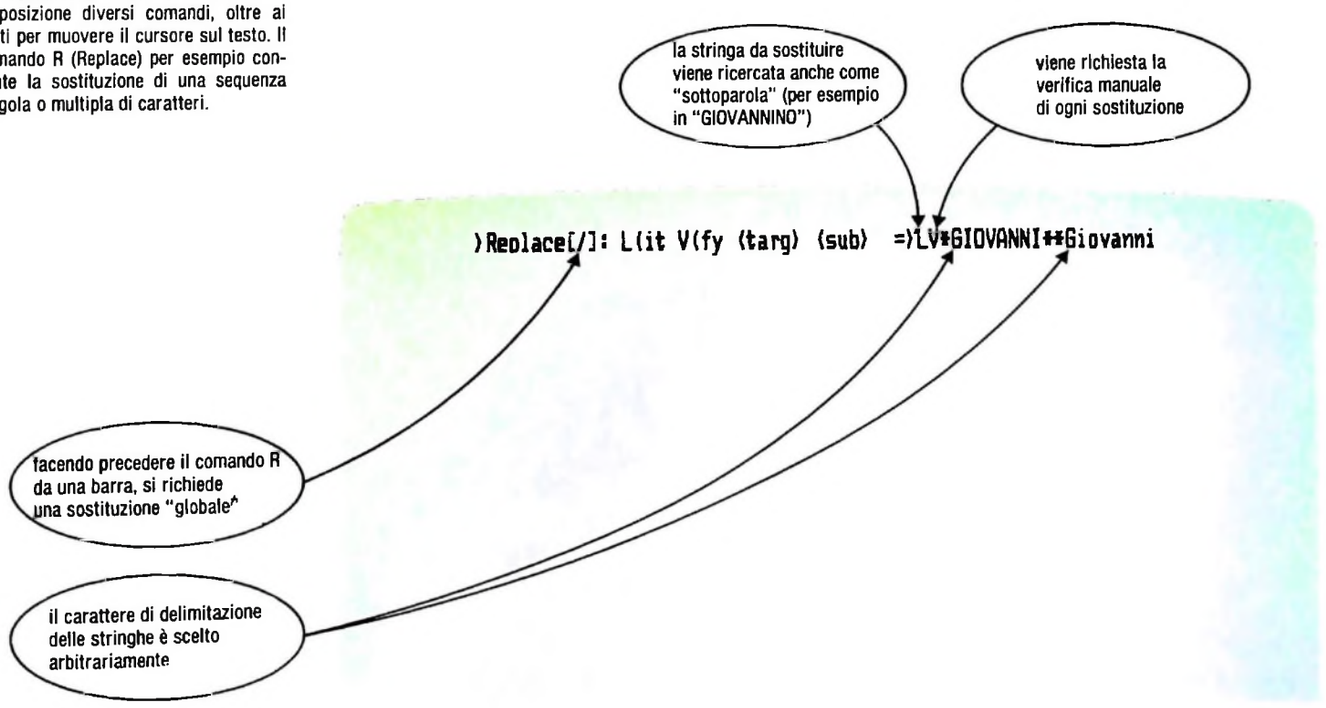
Si noti che i comandi E(dit e F(iler portano l'utente a un nuovo menù di sottocomandi.
Per non appesantire eccessivamente il menù principale, le funzioni meno frequentemente usate del sistema vengono richiamate attraverso il comando eX(ecute, cioè eseguendo un programma (per esempio il *native code generator*).

Lo screen editor

Vediamo ora il funzionamento dello *screen editor* attraverso i comandi del sottomenù ottenuto dal comando E (figura di pagina a fianco, in alto).
Oltre ai comandi elencati, l'utente ha a disposizione i quattro tasti freccia per muovere il cursore sul testo, e due tasti detti rispettivamente di *conferma* e *annullamento*. Questi tasti dipendono dalla tastiera utilizzata e possono essere definiti dall'utente in fase di configurazione del sistema.
Il comando I consente di iniziare l'immissione di testo a partire dalla posizione del cursore: l'immissione prosegue fino a che non viene premuto il tasto di conferma o quello di annullamento (nel secondo caso i caratteri inseriti vengono eliminati e il testo ritorna allo stato precedente l'immissione).
Il comando D consente di eliminare una porzione di testo (utilizzando i comandi di movimento del cursore): anche questa operazione può essere confermata o annullata.
Il comando F consente di ricercare all'interno del testo sequenze di caratteri, distinguendo eventualmente tra una parola intera e una sua parte.
Il comando R, operando in modo analogo al precedente, consente la sostituzione di una sequenza di caratteri, singola o multipla, eventualmente verificata da parte dell'utente (figura di pagina a fianco, in basso).
Il comando Z consente di cancellare grosse porzioni di testo, specificando le posizioni iniziali e finali.
Il comando A consente di allineare una o più righe di testo nella pagina video.
Il comando J consente di muovere rapidamente il cursore all'inizio o alla fine del testo o in punti "marcati" dall'utente attraverso il comando S(et.
Il comando X consente di "sovrascrivere", cioè di immettere del testo con la contemporanea cancellazione di quello precedente.
Il comando C ha una duplice funzione: esso consente infatti di recuperare porzioni di testo precedentemente cancellate (ciò può essere utile sia per rimediare a un errore, sia per spostare parti di testo) oppure di copiare un altro testo o una sua parte.
Fra gli altri comandi, non elencati in figura, citiamo il comando S(et che consente di impostare alcuni parametri di funzionamento dello screen editor (margini ecc.).
In sintesi, lo *screen editor* UCSD è un ottimo strumento per la scrittura di programmi: non è invece adeguato alla gestione di testi (non è cioè un *word processor*) in quanto mancante delle più comuni funzionalità di tale categoria (prima fra tutte quella di *word wrap*, che consiste nel fare andare a capo automaticamente una parola quando non sta nella riga, ricomponendo il testo).



Lo screen editor UCSD è un ottimo strumento per la scrittura di programmi. In alto: nella figura è schematizzato il funzionamento dello screen editor attraverso i programmi del sottomenù ottenuto dal comando E. In basso: l'utente ha a disposizione diversi comandi, oltre ai tasti per muovere il cursore sul testo. Il comando R (Replace) per esempio consente la sostituzione di una sequenza singola o multipla di caratteri.



Sommario del Quarto Volume

PERSONAL COMPUTER OLIVETTI M10

L'UFFICIO DA VIAGGIO

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 collegarsi via telefono per spedire o ricevere informazioni.

Qualunque professione sia la vostra, M10 è in grado, dovunque vi troviate, di offrirvi delle capacità di soluzione davvero molto grandi. M10: il più piccolo di una grande famiglia di personal.

olivetti

Per informazioni rivolgersi, anche con contrassegnati da "Olivetti M10 Punto di Vendita" o inviare in busta chiusa a Olivetti, Divisione Personal Computer, Via Meravigli 12, 20123 Milano.

NOME/COGNOME

VIA/N.

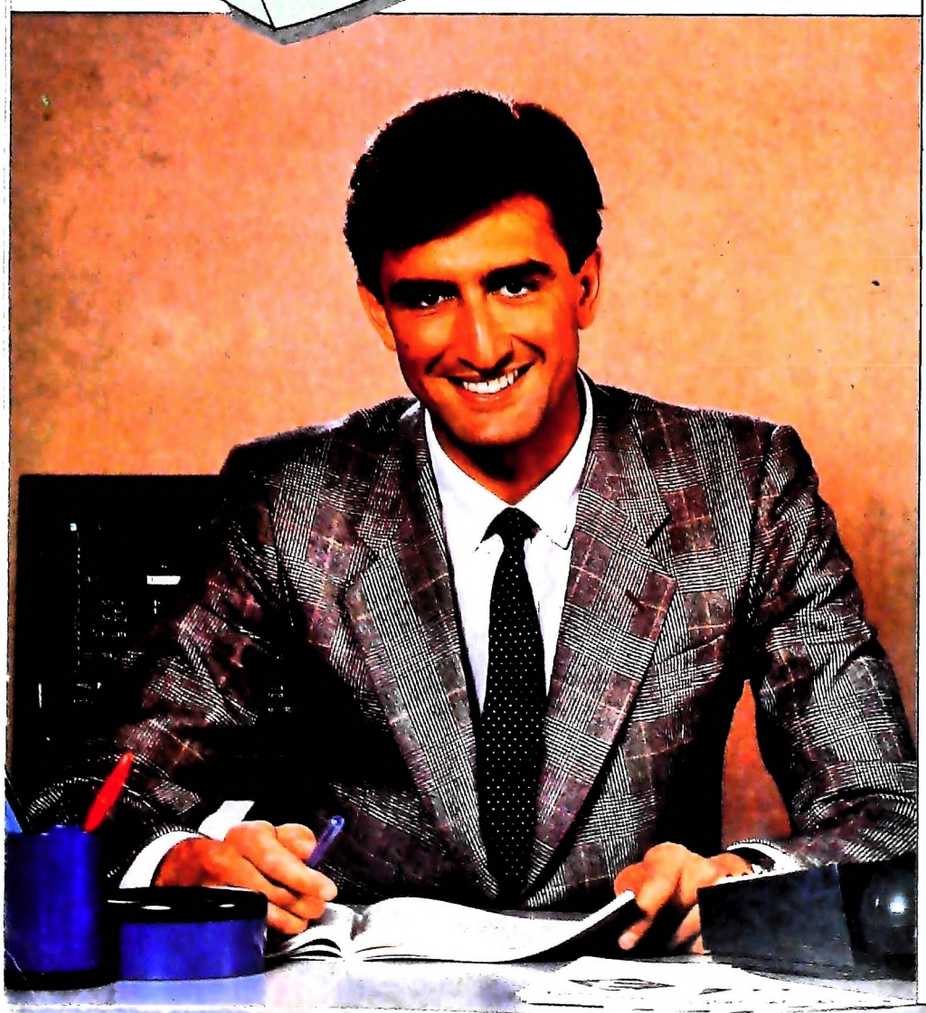
CAP/CITTÀ

TELEFONO

Anche in leasing con Olivetti Leasing.

UN NUOVO MODO DI USARE LA BANCA.

Consulenza



GLI INVESTIMENTI CON VOI E PER VOI DEL BANCO DI ROMA.

Il Banco di Roma non si limita a custodire i vostri risparmi. Vi aiuta anche a farli meglio fruttare. Come? Mettendovi a disposizione tecnici e analisti in grado di offrirvi una consulenza di prim'ordine e di consigliarvi le forme di investimento più giuste. Dai certificati di deposito ai titoli di stato, dalle obbligazioni alle azioni, il Banco di Roma vi propone professionalmente le varie opportunità del mercato finanziario. E grazie ai suoi "borsini", vi permette anche di seguire, su speciali video, l'andamento della Borsa minuto per minuto.

Se desiderate avvalervi di una gestione qualificata per investire sui più importanti mercati mobiliari del mondo, i fondi comuni del Banco di Roma, per titoli italiani ed esteri, vi garantiscono una ampia diversificazione.

Inoltre le nostre consociate Figeroma e Finroma forniscono consulenze per una gestione personalizzata del portafoglio e per ogni altra esigenza di carattere finanziario.

Veniteci a trovare, ci conosceremo meglio.

BANCO DI ROMA
CONOSCIAMOCI MEGLIO.