Spediz. in abbonamento postale GR. II/70 L. 2.000
(...)

39 CORSO PRATICO COL COMPUTER

421917

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

diretto da GIANNI DEGLI ANTONI

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
ADRIANO DE LUCA (Professore di Architettura dei Calcolatori all'Università Autonoma Metropolitana di Città del Messico), GOFFREDO HAUS, MARCO MAIOCCHI, DANIELE MARINI, GIANCARLO MAURI, CLAUDIO PARMELLI, ENNIO PROVERA

Testi
GIANCARLO MAURI, CLAUDIO PARMELLI, Eidos (Tiziano Brugnetti, Bruno Motta, Carmine Stragafede)
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 BREGGÈ

Segretarie di Redazione
RENATA FRIGOLI
LUCIA MONTANARI

AVVISO AI LETTORI

Con il n. 40 sarà in edicola la copertina per rilegare il terzo volume del "Corso pratico col computer". Prenotatela dal vostro edicolante

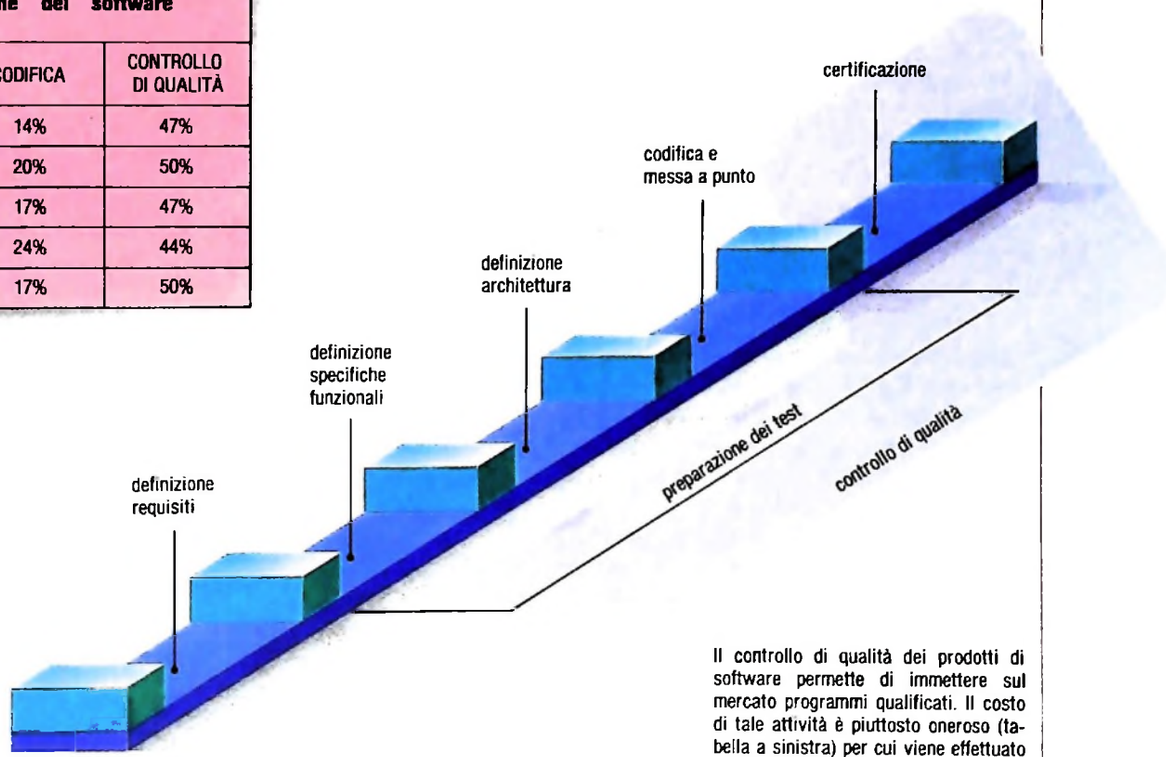
IL CONTROLLO DI QUALITÀ DEI PROGRAMMI

Una delle attività più rilevanti per il buon successo di un programma.

L'attività di controllo di qualità dei prodotti software è una delle più rilevanti per il buon successo di un programma: essa infatti garantisce al costruttore di dover effettuare successivamente pochi interventi sul prodotto, per correggere errori, e di dare una buona immagine di sé presso l'acquirente, grazie al fatto che quest'ultimo non avrà fastidi a causa di malfunzionamenti.
Nonostante ciò, un controllo di qualità sistematico e adeguato viene attualmente fatto solo dalle grandi case di sviluppo di software. La maggior parte dei prodotti, in particolare di tipo gestionale, che è possibile reperire sul mercato, risente molto delle carenze di controllo; ciò è dovuto essenzialmente all'elevato costo di questa attività, che ripaga in tempi lunghi, e i cui benefici economici sono riscontrabili solo a fronte di un'adeguata misurazione di costi lungo tutta la vita del prodotto. È noto infatti che il costo di qualificazione di un prodotto sofisticato arriva a eguagliare tutti gli altri costi di analisi e sviluppo (tabella a sinistra), ma è altresì noto che il costo di correzione di un errore, dopo che un prodotto è stato rilasciato arriva a essere decine di volte maggiore del costo di correzione dello stesso errore prima del rilascio.
L'attività di controllo di qualità, spesso indicato anche gene-

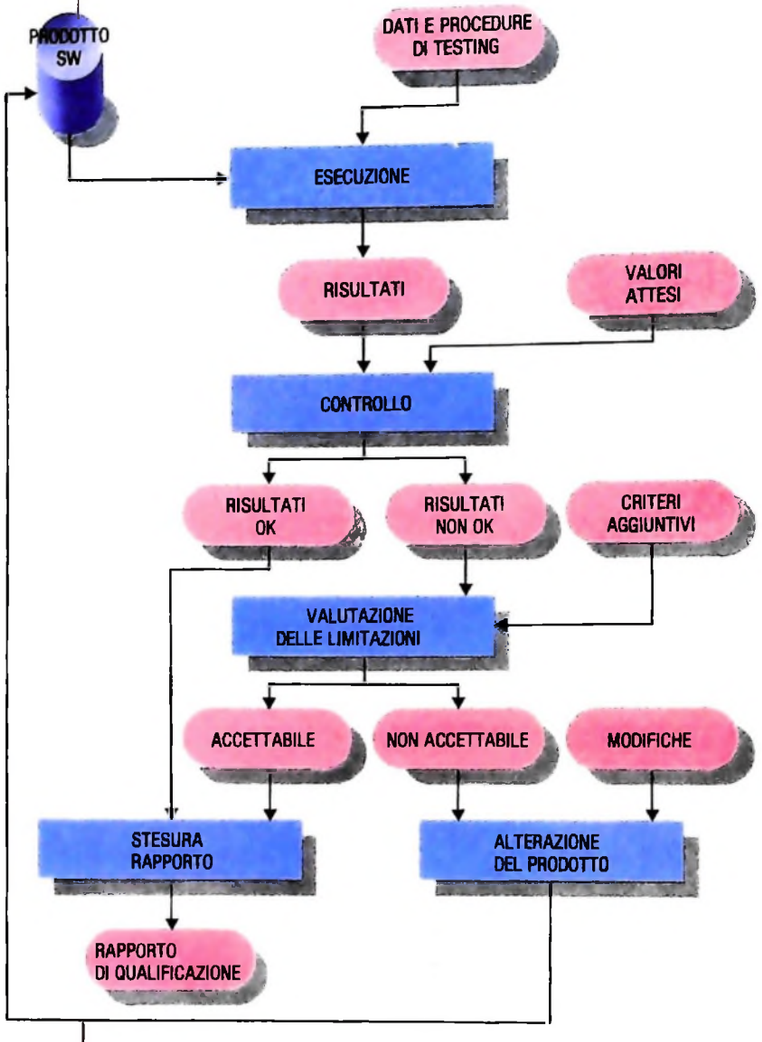
Costo della qualificazione del software

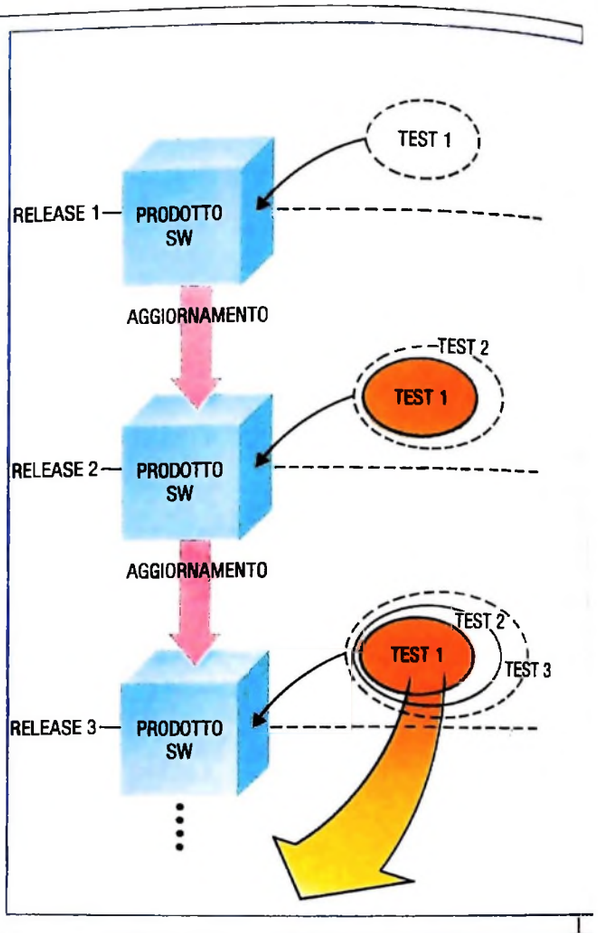
PROGETTI	ANALISI E DISEGNO	CODIFICA	CONTROLLO DI QUALITÀ
SAGE	39%	14%	47%
NTDS	30%	20%	50%
GEMINI	36%	17%	47%
SATURN V	32%	24%	44%
OS/360	33%	17%	50%

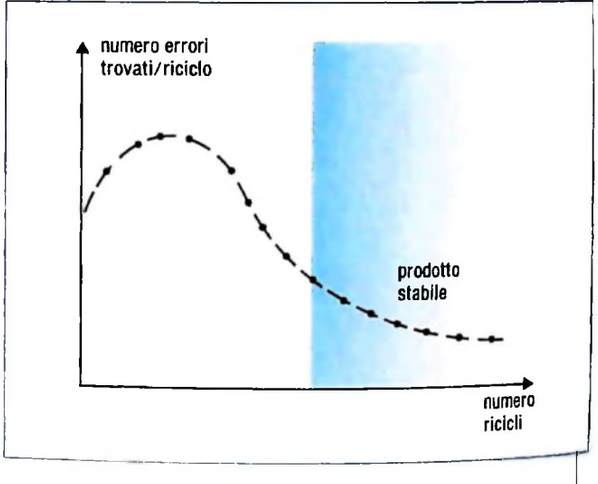


Ciclo di sviluppo del software

Il controllo di qualità dei prodotti di software permette di immettere sul mercato programmi qualificati. Il costo di tale attività è piuttosto oneroso (tabella a sinistra) per cui viene effettuato solo dalle grandi aziende. Sopra, come viene condotta l'operazione di controllo o "testing".





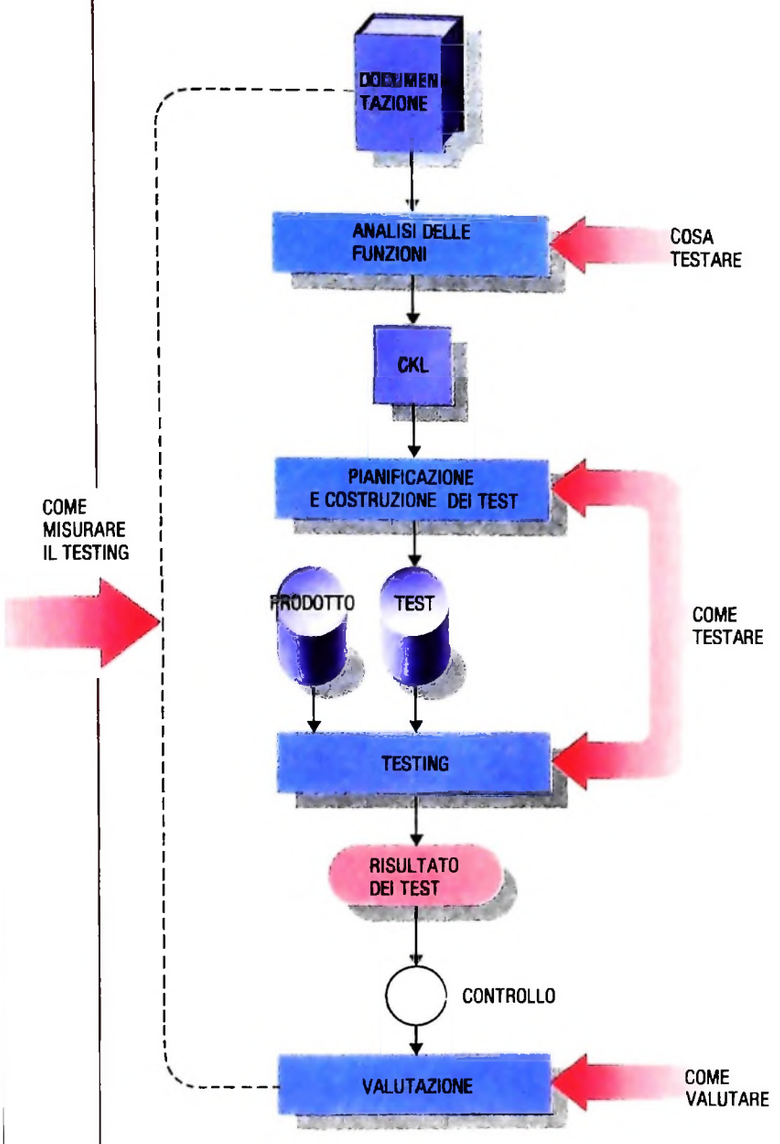


I risultati che si ottengono dai test sono suddivisi in "corretti" o "errati". Su quelli errati si effettua un ulteriore esame per verificare se il prodotto può essere ugualmente distribuito senza le funzioni errate. Se ciò non fosse possibile è necessaria la correzione del programma e la riesecuzione di tutti i test.

Nel caso si debbano aggiungere nuove funzioni o correggere errori ecc., la nuova versione del prodotto deve essere certificata con la riesecuzione di tutti i test precedenti (in alto a destra). Un prodotto si considera assestato quando la curva di reperimento degli errori tende ad appiattirsi (a lato).

ricamente come "testing", si articola in due fasi ben distinte: una prima è legata all'analisi di che cosa controllare e della preparazione dei test, cioè degli strumenti di controllo; una seconda è invece legata all'uso dei test per la certificazione vera e propria del prodotto (figura a pagina precedente).
Il modo di effettuare la certificazione finale è riportato in figura qui sopra. I test sono in genere costituiti da dati da fornire al programma da controllare, scelti in modo tale che i risultati siano noti a priori. L'esecuzione del prodotto da controllare con tali test permette di ottenere dei risultati che vengono sottoposti a una verifica e che vengono suddivisi in "corretti" ed "errati". Quelli errati vengono ulteriormente esaminati per verificare se è accettabile che il prodotto venga ugualmente distribuito agli utenti senza le funzioni errate (cioè dando un prodotto che ha delle limitazioni essendo state eliminate nelle funzioni), o se è inevitabile la loro correzione. In questo secondo caso si rende necessaria la correzione dell'intero programma e la completa riesecuzione di tutti i test (infatti, in generale, la modifica di un programma in un certo punto può provocare effetti collaterali in altri punti, introducendo errori nuovi: l'unico modo di garantirsi da tali fenomeni è la completa riesecuzione di tutte le prove). Quanto tutti gli errori inaccettabili sono stati eliminati, viene steso un rapporto di qualificazione a testimonianza delle prove ef-

Problemi dell'attività di testing



La lista delle funzioni da esercitare "checklist" è un metodo di pianificazione dei test da eseguire facendo in modo che ogni funzione venga esercitata almeno una volta da almeno un test.

fettuate e del relativo esito, e il prodotto viene finalmente rilasciato all'utente.
Il fatto di dover rieseguire tutti i test a fronte di una modifica del programma impone che i test siano mantenuti in vita insieme al prodotto stesso: sappiamo infatti che i costi del software impongono un suo continuo aggiornamento, per aggiungere nuove funzioni, per correggere errori e così via. Ad ogni nuova versione del prodotto sono quindi necessarie la riesecuzione di tutti i test precedenti, per evidenziare che le vecchie funzioni siano ancora fornite correttamente, e l'esecuzione di nuovi test, per certificare le parti aggiunte (figura a destra della pagina accanto). La certificazione deve cioè verificare che nel prodotto non sia presente una regressione della qualità precedentemente raggiunta.
Durante tutta l'attività di certificazione e di correzione degli errori, viene tenuta traccia del numero di malfunzionamenti trovati a ogni nuovo riciclo dei test: l'andamento di tale numero è riportato nel grafico della pagina accanto, e permette di evidenziare l'andamento della qualità del prodotto. È illusorio, per prodotti industriali di dimensioni non banali, pensare di poter eliminare tutti gli errori presenti; in generale un prodotto si considera assestato quando la curva di reperimento degli errori raggiunge valori sufficientemente piccoli e tende ad "appiattirsi".
Il fatto di dover certificare un prodotto con la tecnica descritta pone subito in evidenza un certo insieme di problemi (figura a sinistra): innanzitutto è necessario avere a disposizione documenti di specifiche funzionali ben dettagliati, per poter individuare che cosa deve essere controllato. Da queste è possibile costruire una lista delle funzioni da esercitare, detta in gergo "checklist"; la checklist permette di pianificare i test da effettuare, facendo in modo che ogni funzione sia esercitata almeno una volta da almeno un test (si parla in questo caso di *copertura funzionale* del prodotto); quindi si procede all'esecuzione dei test (spesso questa sola fase viene chiamata con il nome di "testing") e alla verifica dei risultati. È inoltre opportuno che l'intera attività sia esaminata per "misurare" quanto testing è stato effettuato (sarebbe infatti poco significativo sapere che non si sono trovati errori se si sono eseguite poche prove).
La preparazione della checklist prevede la necessità di verificare due tipi distinti di funzionamento corretto:

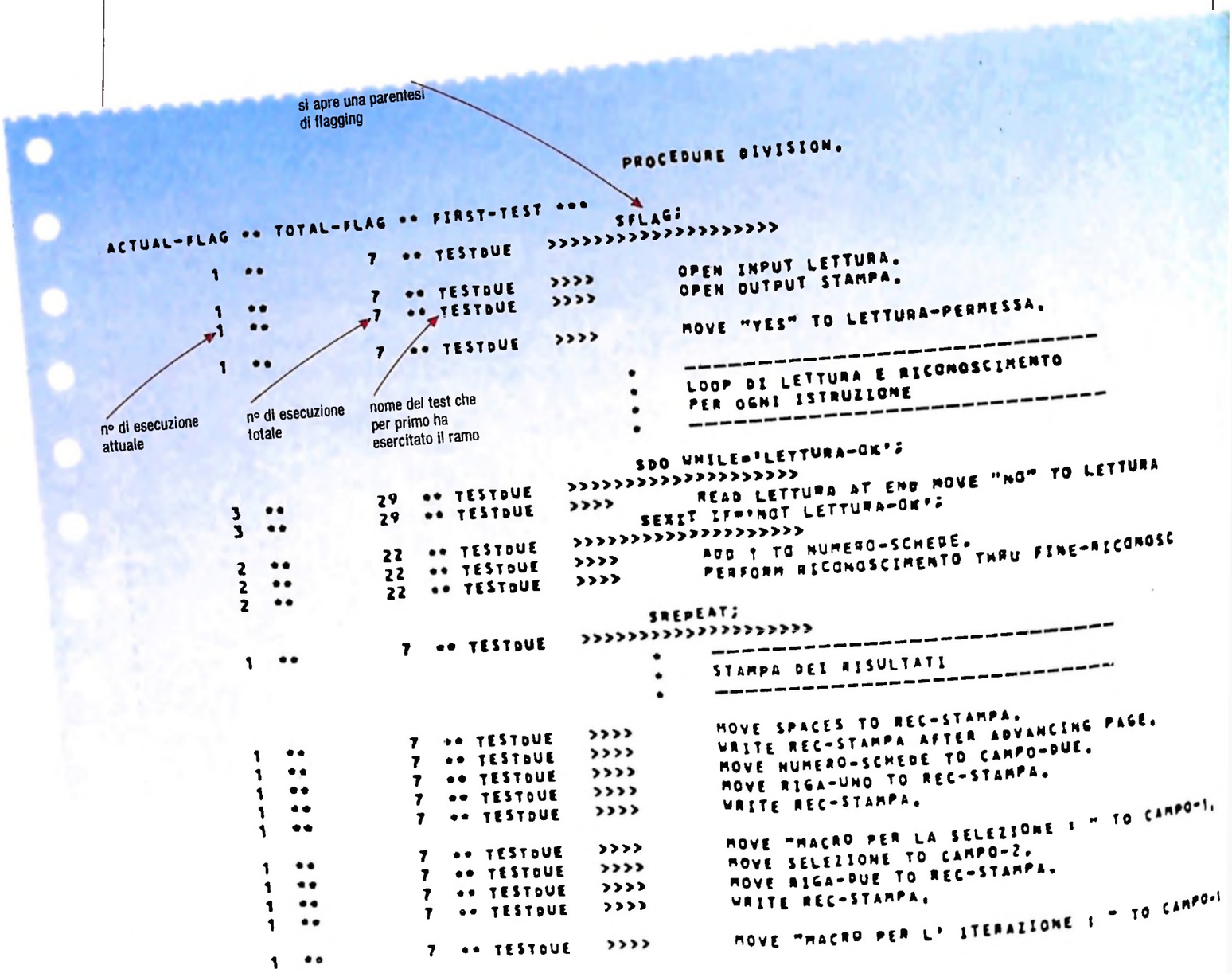
- le *funzionalità positive*, ovvero la verifica che il programma funzioni correttamente a fronte di un suo uso;
- le *funzionalità negative*, ovvero la verifica che il programma funzioni correttamente anche a fronte di un suo uso scorretto da parte dell'utente.

La ripetuta esecuzione dei test, sia durante il processo di certificazione di una singola versione del prodotto, sia nelle varie versioni durante tutta la vita del prodotto, impone la necessità di ridurre lo sforzo del controllo dei risultati: tale attività, infatti, è non solo molto poco motivante e noiosa, ma rischia anche di richiedere molto tempo, e quindi di avere costi elevati (si pensi per esempio al controllo di un tabulato di qualche centinaio di pagine pieno di registrazioni contabili); si tende allora, ove possibile, a rendere *automatico* il controllo dei risultati. Si tratta di costruire dei test che non sono semplicemente l'insieme dei dati di prova, ma ciascuno dei quali sia costituito da:
- dati campione d'ingresso;
- risultati attesi;
- programma che verifica che i risultati ottenuti sono uguali a quelli attesi.

In tal modo l'esercizio dei test si riduce al loro lancio e al controllo di semplici tabulati che, per ogni test, riportano l'informazione di funzionamento corretto, ovvero di reperi-

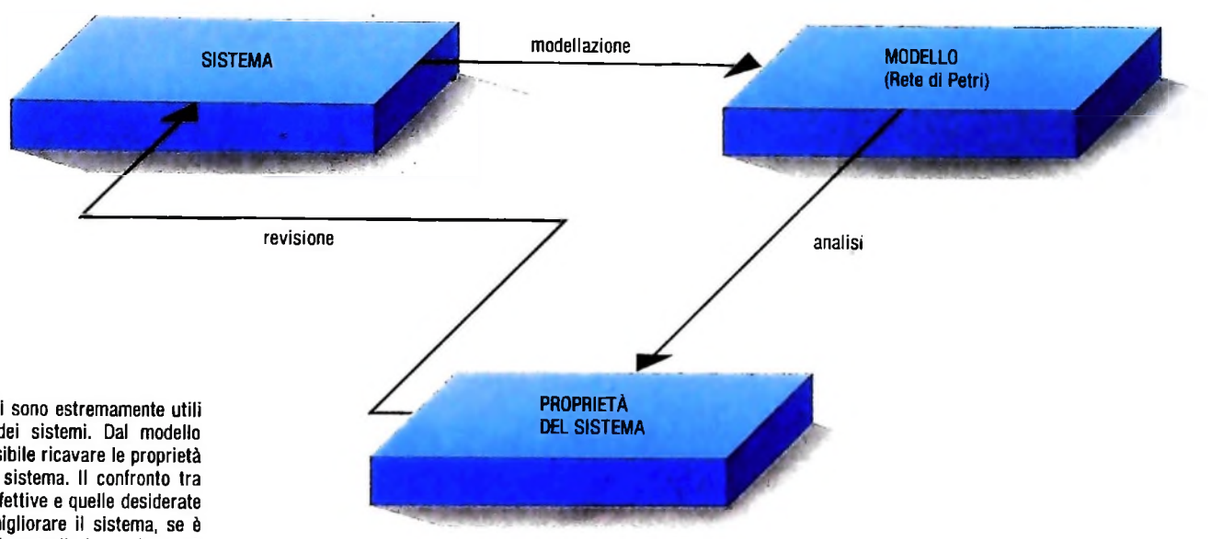
mento d'errore (con informazioni sull'errore). Purtroppo il controllo automatico non è sempre applicabile (ciò è infatti possibile solo ove i risultati siano disponibili su un supporto magnetico leggibile dal programma di controllo) e una parte dei test deve essere soggetta ancora a controllo *visivo*.
Infine, si tratta di misurare la quantità di testing effettuata: una tecnica particolarmente interessante è quella dei programmi Flagger.
Un Flagger è un programma che, attraverso un'opportuna manipolazione del programma, sottoposto al controllo, permette di ottenere l'informazione di quali istruzioni siano state realmente eseguite durante l'esercizio dei test: in tal modo è possibile seguire direttamente sulla lista del programma quali parti di esso siano state eseguite e quali no, e la misura del testing (si parla di *copertura topologica*) può essere fornita come percentuale delle istruzioni eseguite almeno una volta.
La figura in basso illustra un frammento di programma sottoposto a un Flagger per una versione del linguaggio COBOL estesa con strutture di controllo della programmazione strutturata; in essa compare, oltre al numero di volte che ciascuna istruzione è stata eseguita durante l'ultimo test, anche il numero cumulativo di volte che ogni istruzione è stata eseguita da tutti i test e il nome del test che per primo ha esercitato ogni istruzione.

Un frammento di programma in COBOL, sottoposto a test con un programma di tipo Flagger: si possono vedere i risultati del test.



LE RETI DI PETRI

Uno strumento per la descrizione di sistemi complessi.



Le reti di Petri sono estremamente utili per l'analisi dei sistemi. Dal modello formale è possibile ricavare le proprietà essenziali del sistema. Il confronto tra le proprietà effettive e quelle desiderate consente di migliorare il sistema, se è artificiale, o di controllarlo meglio, se è naturale.

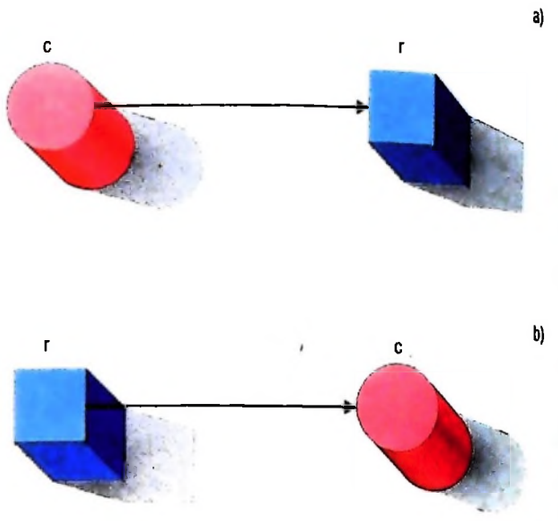
Reti e sistemi

Un sistema di calcolo di una certa dimensione può essere costituito da una o più unità centrali, eventualmente collegate in rete, che controllano una o più memorie e un certo numero di unità periferiche, attraverso sistemi operativi che gestiscono un complesso flusso di informazioni e comandi; vi è così la possibilità di eseguire contemporaneamente più processi di calcolo.
Dare una descrizione dell'organizzazione e del funzionamento di un sistema del genere, che consenta di individuare e quindi eliminare eventuali errori di funzionamento e di evidenziare i punti in cui l'organizzazione del sistema può essere modificata in modo da renderlo più efficiente, non è un problema di facile soluzione.
Le reti di Petri, introdotte nel 1962 da Carl Adam Petri, si propongono come base concettuale e teorica per la costruzione di modelli formali di sistemi complessi, non solo sistemi di calcolo, ma anche sistemi fisici, socio-economici e così via.
Una volta costruita la rete di Petri che modella un sistema assegnato, si possono analizzare le proprietà matematiche della rete, che corrispondono a proprietà fisiche del sistema. Nel caso di sistemi artificiali, qualsiasi problema incontrato nel corso dell'analisi evidenzia un errore nel progetto del sistema, e quindi ne impone la revisione. Il sistema modificato è a sua volta modellato, analizzato ed eventualmente modificato e così via, finché non si ottiene un sistema correttamente impostato (figura sopra).
Prima di definire le reti di Petri, cerchiamo di capire quali sono gli elementi essenziali di un sistema che un modello formale deve riuscire a descrivere.
In termini generali, un sistema complesso è costituito da un insieme di componenti ognuna delle quali esegue alcune azioni all'interno del sistema, determinate dal suo stato corrente e dall'interazione con altre componenti.
L'interazione tra due diverse componenti di un sistema può essere essenzialmente di tre tipi:
1) *di concorrenza*: non c'è alcuna interazione, nel senso che le due componenti eseguono le proprie azioni in modo del tutto indipendente l'una dall'altra. Per esempio, due automobili che viaggiano dal centro di Milano rispettivamente verso la periferia sud e verso la periferia nord sono un esempio di componenti concorrenti del "sistema Milano";
2) *di dipendenza causale*: una delle due componenti può eseguire la propria azione solo dopo che l'altra ha eseguito la propria: se due automobili sono in coda a un semaforo, la seconda può partire solo dopo che è partita la prima;
3) *di conflitto*: le due componenti devono usare la stessa ri-

sorsa per eseguire la propria azione. In caso di richiesta contemporanea, è necessario stabilire delle regole di precedenza. Un esempio di conflitto si ha quando due automobili arrivano contemporaneamente a un incrocio da direzioni perpendicolari. In assenza di un semaforo o di regole di precedenza, è abbastanza difficile evitare incidenti.
Le reti di Petri hanno come obiettivo principale quello di consentire una descrizione delle interazioni tra le componenti del sistema da modellare, che sia in grado di evidenziare chiaramente il verificarsi di una delle tre situazioni precedenti, e inoltre di fornire strumenti per l'analisi delle principali proprietà del sistema.

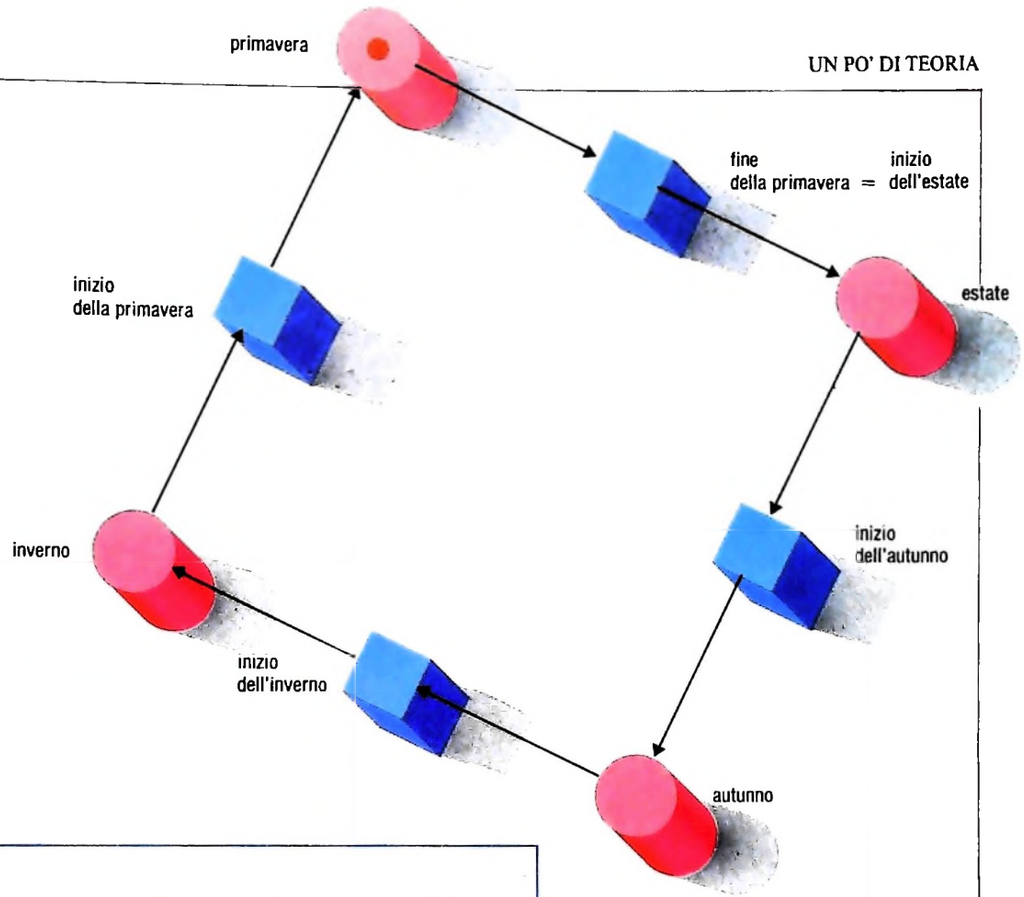
Condizioni ed eventi

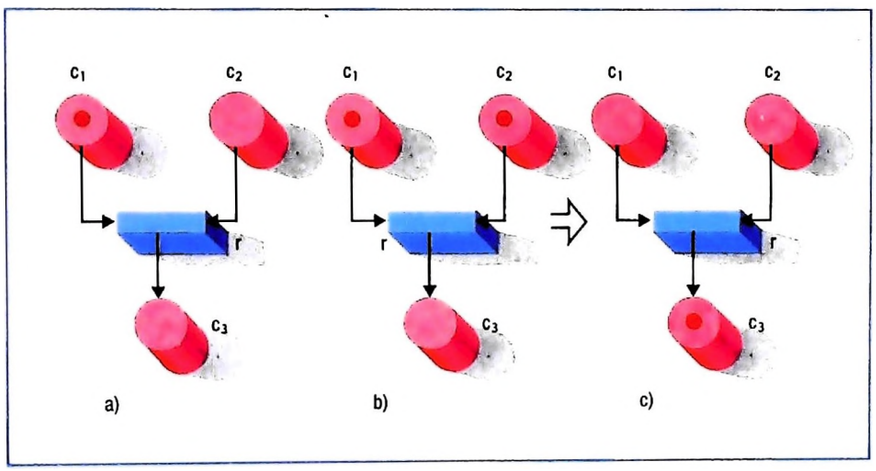
L'idea che sta alla base delle reti di Petri è che nel funzionamento di un sistema si possono distinguere due elementi fondamentali. Anzitutto le *condizioni*, cioè proprietà o relazioni tra diverse componenti del sistema che in ogni istante possono valere o non valere, e che quindi caratterizzano lo stato del sistema a ogni istante. Vi sono poi gli *eventi*, elementi dinamici del sistema che ne modificano lo stato. Un evento può aver luogo solo se vale un certo insieme di condizioni, dette *precondizioni*. L'occorrenza di un evento rende false le precondizioni, mentre fa diventare vere altre condizioni, le *postcondizioni*, che in precedenza non lo erano.
Una rete di Petri è un oggetto grafico e matematico che consente di descrivere tale situazione utilizzando un insieme C di cilindri (per rappresentare le condizioni) e un insieme R di cubi (per gli eventi) collegati tra loro da frecce che rappresentano la *relazione di flusso* $F \subseteq C \times R \cup R \times C$ tra elementi di tipo diverso. Più precisamente, avremo una freccia da un cilindro c a un cubo r se la condizione corrispondente a c è una precondizione per l'evento r, mentre avremo una freccia da r a c se l'evento r rende vera la condizione c. La figura in alto mostra i due tipi di connessione fondamentali nelle reti di Petri.
In precedenza, si è detto che lo stato di un sistema in un certo istante è caratterizzato dall'insieme delle condizioni che valgono in quell'istante. Chiameremo *caso* l'insieme delle condizioni verificate in un certo istante, e rappresenteremo graficamente un caso S contrassegnando con un punto nero (detto *marca*) ogni cilindro appartenente a S.
In termini formali, diremo che una *marcatura* è una funzione $M: C \to \{0,1\}$, con $M(c)=0$ se la condizione c non vale e $M(c)=1$ se vale.
Un semplicissimo esempio di rete di Petri è il modello del ciclo delle stagioni (figura in alto della pagina accanto). In questo caso si hanno quattro condizioni ("è primavera", "è estate" ecc.) e quattro eventi (i cambi di stagione: fine della primavera e inizio dell'estate, fine dell'estate ecc.). Per indicare che ci si trova in primavera, si mette una marca nel cilindro corrispondente e così via.
Evidentemente, in questa rappresentazione l'evento "fine della primavera e inizio dell'estate", che può aver luogo solo se siamo in primavera, corrisponde allo spostamento della marca dal cilindro "primavera" al cilindro "estate".



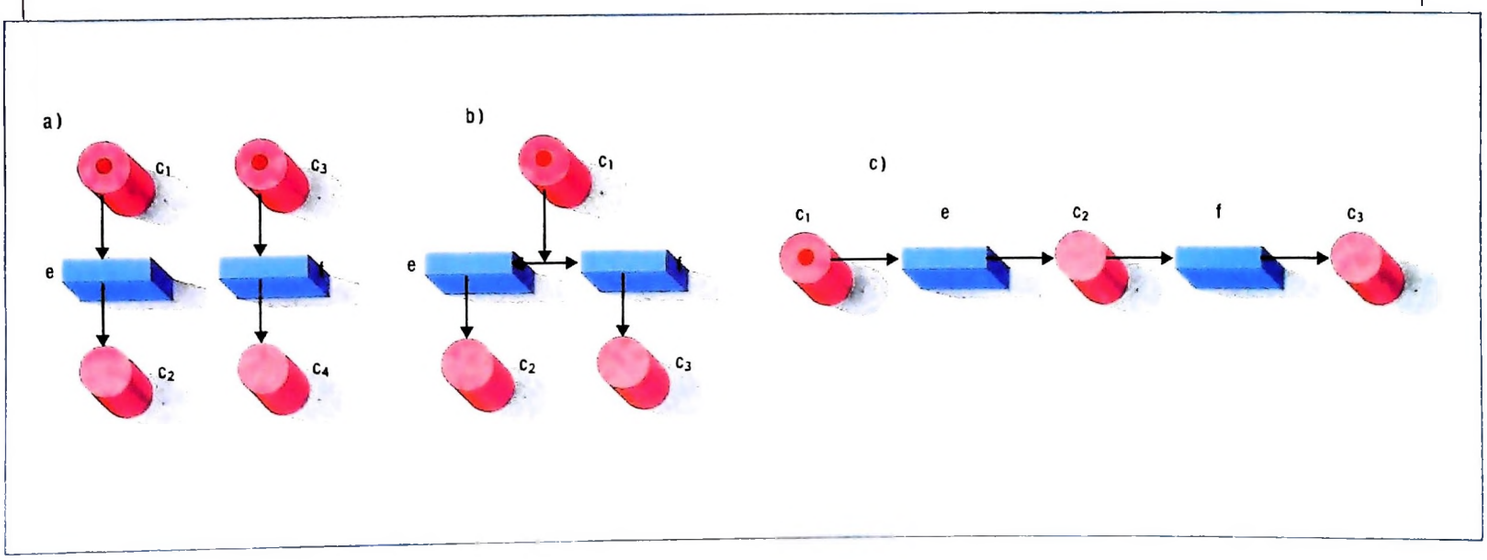
Possiamo generalizzare questo semplicissimo esempio, enunciando la *regola di scatto* per le reti di Petri:
a) un evento e può aver luogo ("ha concessione") in un caso S solo se tutte le sue precondizioni e nessuna della sue postcondizioni appartengono a S;
b) l'occorrenza di evento e che ha concessione nel caso S comporta l'eliminazione di tutte le marche nelle precondizioni di e e l'inserimento di una marca in ognuna delle sue postcondizioni. Si ottiene così un nuovo caso S'.
Questa regola è alla base di un "gioco delle marche" che consente di visualizzare la dinamica del sistema attraverso gli spostamenti delle marche nella rete di Petri che lo modella. Nella figura al centro (pagina accanto) sono rappresentati un evento che non può accadere, un evento che può accadere e il cambiamento di stato prodotto da tale evento.
La caratteristica più interessante delle reti di Petri è la naturalezza con cui vengono visualizzate le tre situazioni di concorrenza, dipendenza causale e conflitto (figura in basso). Infatti, due eventi e e f risultano essere:
a) concorrenti, se non hanno precondizioni né postcondizioni in comune (a);
b) in conflitto, se hanno in comune almeno una pre o una postcondizione (b);
c) infine, e è causalmente dipendente da f se almeno una delle precondizioni di e è postcondizione di f (c).
Un'osservazione di estrema importanza va fatta a riguardo degli eventi concorrenti. Se due eventi e e f sono concorrenti, non vi è alcuna ragione interna al sistema che imponga che e deve aver luogo prima di f o viceversa: i due eventi possono aver luogo in qualsiasi ordine, o anche contemporaneamente. A differenza di tutti i modelli di tipo sequenziale, che presuppongono l'esistenza di un "orologio universale" che controlla la sequenza degli eventi, le reti di Petri sono inerentemente asincrone, e quindi sono particolarmente adatte per modellare sistemi con componenti di diversa velocità e con

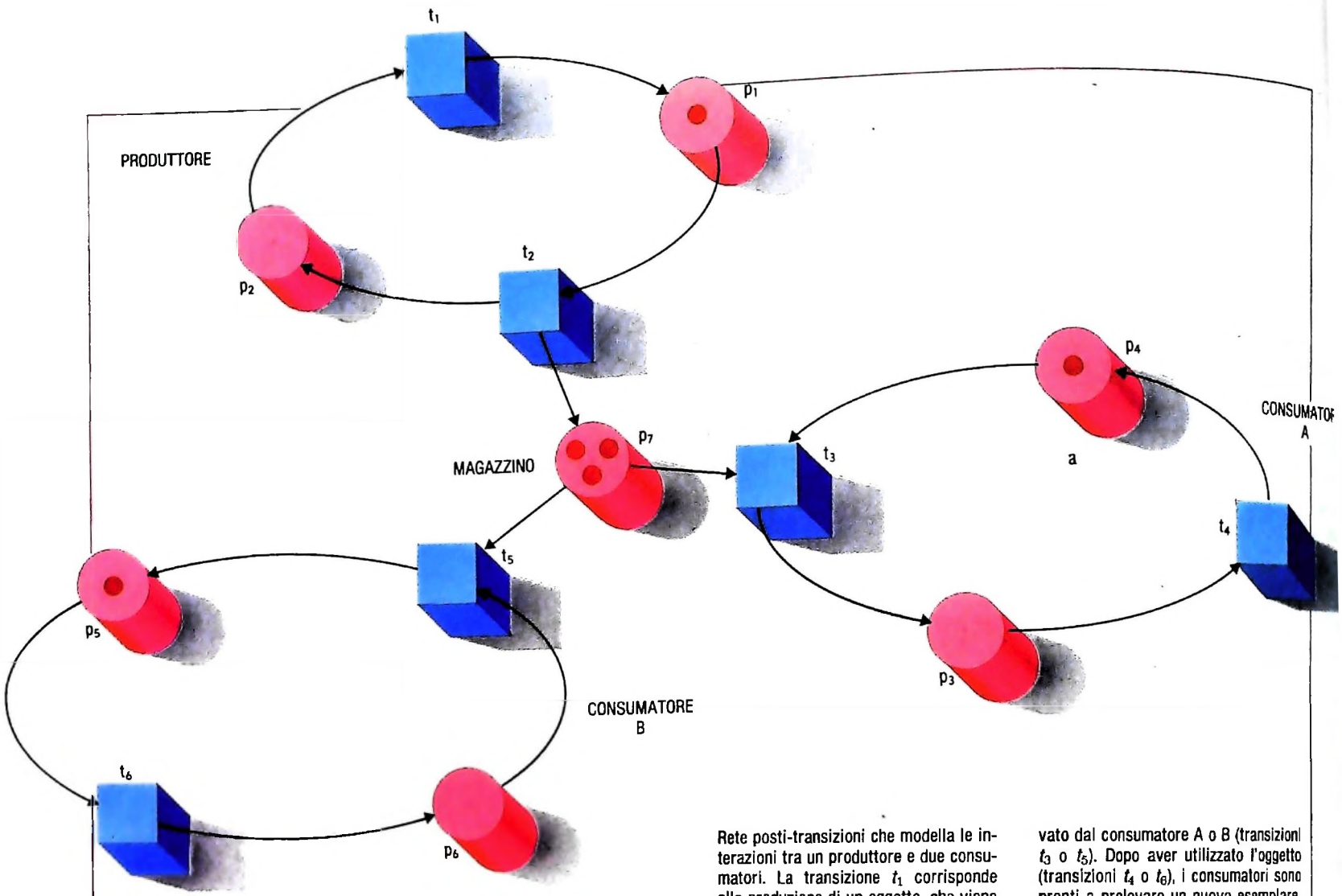
A sinistra, le connessioni fondamentali tra gli elementi di una rete di Petri; *c* rappresenta una condizione che deve essere vera perché possa accadere l'evento rappresentato da *r* (a). In b) *c* rappresenta una condizione che comincia a valere dopo che si è verificato l'evento modellato da *r*. A destra, una rete di Petri che rappresenta il ciclo delle stagioni. La marca indica che siamo in primavera. Sotto, la regola di scatto per le reti di Petri; in a), l'evento *r* non può aver luogo perché una delle sue precondizioni, c_2, non vale. In b), *r* può aver luogo: il verificarsi di *r* porta nella situazione modellata da c)





Sotto, in a) gli eventi *e* e *f* sono concorrenti, cioè non s'influenzano a vicenda e possono aver luogo in un ordine qualsiasi. In b) *e* e *f* sono in conflitto, poiché se si verifica uno dei due, l'altro perde la possibilità di verificarsi. In c), *f* può verificarsi solo dopo che *e* ha reso vera la condizione c_2, e quindi dipende causalmente da *e*.





Rete posti-transizioni che modella le interazioni tra un produttore e due consumatori. La transizione t_1 corrisponde alla produzione di un oggetto, che viene poi immagazzinato (transizione t_2) nel magazzino p_7, da cui può essere prelevato dal consumatore A o B (transizioni t_3 o t_5). Dopo aver utilizzato l'oggetto (transizioni t_4 o t_6), i consumatori sono pronti a prelevare un nuovo esemplare, e quindi collocano una marca nel posto p_4 o p_6, rispettivamente.

controllo distribuito. Naturalmente, resta valido l'ordinamento "logico" determinato dalla dipendenza causale: un evento e che dipenda casualmente da f deve necessariamente avvenire dopo f.

Posti e transizioni

Consideriamo ora una situazione diversa, che è quella rappresentata dalla figura in alto. In questo caso, abbiamo una fabbrica che produce merci, per esempio salami, e li immagazzina in un deposito da cui due consumatori possono prelevarli. Si potrebbe modellare il sistema descrivendo eventi del tipo "si produce il salame numero 1" ... "si produce il salame numero N" ... e condizioni del tipo "nel magazzino c'è il salame numero N", ma in realtà l'informazione significativa non è "quali" salami ci sono nel deposito, ma "quanti" ce ne sono. Possiamo allora evitare di costruire una rete che segue individualmente ogni salame, modificando leggermente il significato dei cilindri e dei cubi e la regola di scatto.
In questa nuova interpretazione, più sintetica, i cilindri rappresentano dei *posti* che possono contenere più di una marca, e i cubi rappresentano *transizioni* che modificano il numero di marche contenute in alcuni posti.
Nell'esempio, abbiamo un posto che rappresenta il magazzino, contenente tante marche quanti sono i salami; l'evento "deposito di un salame" (che può aver luogo solo dopo che il salame è stato prodotto, e cioè in presenza di una marca nel posto "salame prodotto") aggiungerà una marca a questo posto. L'evento "consumo di un salame da parte del consumatore A (oppure B)" potrà aver luogo solo se nel magazzino vi è almeno un salame e comporterà l'eliminazione di una marca dal posto corrispondente.
Lo stato di una rete posti-transizioni di questo tipo è descritto da una funzione M dall'insieme dei cilindri all'insieme dei numeri interi non negativi, che ad ogni cilindro associa il numero di marche che esso contiene, detta *marcatura.*
Una transizione può scattare solo se è abilitata, cioè se ogni posto d'ingresso a essa contiene almeno una marca (si osservi che non si pone più il vincolo che i posti di uscita, cioè le postcondizioni, non contengano marche). Lo scatto di una transizione t trasforma la marcatura corrente M in una nuova marcatura M', ottenuta da M togliendo una marca da ognuno dei posti d'ingresso a t e aggiungendo una marca ad ognuno dei posti di uscita da t e lasciando invariate le marche in tutti gli altri posti.
Benché siano state definite classi più generali di reti (reti colorate, predicati-transizioni e così via), le reti posti-transizioni sono le più semplici e usate. In futuro ci riferiremo quindi sempre a esse.

Lezione 38

Accesso ai dati di un file: una "lavagna" su file

Abbiamo visto fin qui esempi di programmi che provvedevano sia al caricamento sia alla lettura di un archivio. I file generati con tali programmi sono tutti di tipo dati, come evidenzia il suffisso ".do" con il quale compaiono nel menù principale di M10.
Trattandosi di file dati è possibile modificarli nello stesso modo con cui possiamo modificare i file generati tramite la funzione di TEXT EDITING di M10. Di più possiamo pensare alla possibilità di costruire "manualmente" un file tramite tale funzione e di disporre quindi di un programma che tratti opportunamente i dati in esso contenuti.
Questa funzionalità può essere di elevato interesse nel caso in cui, per esempio, si voglia offrire la possibilità all'utente di modificare "direttamente" le informazioni di un archivio costruito via programma.
Ciò ha anche il vantaggio di evitare la necessità di costruire un "editor", cioè un programma ad hoc per la gestione delle informazioni in archivio, che è in generale un programma di non limitata complessità.
Vediamo dunque un esempio di programma che legga dati da un archivio generato manualmente.
Pensiamo di simulare con un file una "lavagna" di quelle di uso domestico, su cui segniamo senza particolare ordine le spese effettuate per esempio nel corso di una settimana.
Supponiamo di usare al posto di carta e matita, o di lavagna e pennarello, un file sul quale scriviamo i costi, senza per altro preoccuparci di seguire un ordine e un formato specifici. Allora potremo avere un file come il seguente:

```
 1000
2000 5500 6000
200,500


7500     8000,10000
```

nel quale i valori sono scritti senza ordine e distinti tra loro usando i più comuni separatori: spazi e virgole.
Inoltre non vi è alcuna attenzione a lasciare righe vuote, né d'altro lato è posta alcuna attenzione al numero di informazioni che sono scritte su una stessa riga.
Costruiamo adesso un programma che legga tale file e ne stampi il contenuto in modo ordinato, eventualmente con l'obiettivo di calcolare il totale:

```
10 CLS
20 PRINT "   STAMPA COSTI SETTIMANALI"
30 PRINT
40 OPEN "RAM:COSTI" FOR INPUT AS #1
50 LET T=0 ' Azzera totale
60 ' While not eof(COSTI) do
65 IF EOF(1) THEN GOTO 200
```

```
70 INPUT #1,C
80 LET T=T+C
90 PRINT TAB(10) C
100 GOTO 60
200 ' Endwhile
210 PRINT "----------------"
220 PRINT "TOTALE" TAB(10) T
230 CLOSE #1
240 END
```

Eseguiamo il programma: vedremo scorrere le cifre sul display di M10 e alla fine vedremo il totale come di seguito:

```
              .
              .
              .
              0
              7500
              8000
              10000
----------------
TOTALE        40700
Ok
```

Come si può facilmente notare, nonostante la totale assenza di struttura del file, il programma è risultato molto semplice, grazie al fatto che l'istruzione di acquisizione dei dati da file e cioè la INPUT riconosce, come separatori tra valori numerici, esattamente quelli presenti sul file.
Si noti che in corrispondenza delle righe vuote è stata effettuata una lettura "nulla" a cui corrisponde il valore 0 nella variabile C usata.
In altre parole abbiamo usato i seguenti separatori:

- virgola
- ritorno a capo
- spazio tipografico

per riconoscere l'inizio e la fine di un dato di tipo numerico su un file, appoggiandoci alle caratteristiche dell'istruzione INPUT.

Alcune osservazioni sul formato di stampa

Nel programma precedente è stata usata la funzione TAB per ottenere una stampa secondo un formato prestabilito: tale funzione infatti consente di specificare la posizione di inizio stampa. Grazie all'uso di tale funzione abbiamo ottenuto una stampa allineata dei valori e del totale. Il risultato però non è molto soddisfacente, poiché come si nota dal frammento di stampa riportato in figura, i valori sono tutti allineati a sinistra, il che risulta poco naturale in particolare poiché i valori sono allineati come in una operazione aritmetica. Vediamo allora come possiamo operare per ottenere un allineamento a destra.
Esiste un particolare formato dell'istruzione PRINT che risolve il nostro problema. Alteriamo le due istruzioni di stampa nel modo seguente:

```
90 PRINT TAB(10) USING "######";C
220 PRINT "TOTALE" TAB(10); USING "######";T
```

e otterremo il risultato sperato:

```
                .
                .
                .
              0
           7500
           8000
          10000
----------------
TOTALE    40700
Ok
```

La clausola USING nell'istruzione PRINT consente di specificare il formato con cui vengono visualizzati i dati. Ne vedremo nel seguito l'uso in modo più ampio. Nel nostro caso la stringa racchiusa tra apici dopo la USING specifica che intendiamo visualizzare un dato numerico, di sei cifre. Ogni cifra è rappresentata dal carattere "£". In questo modo, qualunque sia il numero di cifre del valore stampato, verranno riservati sei caratteri per ogni stampa e l'allineamento sarà a destra.

Attenzione: nelle istruzioni compare il carattere "#" al posto del carattere "£": ciò è dovuto alle caratteristiche della stampante impiegata. Nella costruzione del programma digitiamo però il carattere "£".

Una lavagna "promemoria"

Pensiamo adesso ancora a una lavagna realizzata con un file, sulla quale però registriamo alcune note, che servano per esempio come promemoria delle cose da fare.
Come nel caso precedente l'idea è quella di usare il file senza rispettare un formato particolare, ma semplicemente usando i caratteri che sono riconosciuti come separatori, in questo caso di dati di tipo stringa.
Vediamo subito allora come è possibile distinguere una stringa da un'altra su un file:

- una stringa può essere racchiusa tra doppi apici. In tal caso l'inizio e la fine della stringa stessa sono evidenziati da tali caratteri;
- se una stringa non è compresa tra doppi apici si intende che possa essere lunga fino a 255 caratteri. Pertanto, se leggiamo una stringa da un file, tutti i caratteri incontrati verranno interpretati come facenti parte della stringa stessa fino a quando si è raggiunto il limite dei 255 caratteri o si è incontrato uno dei due seguenti separatori:

- virgola
- ritorno a capo

Pensiamo allora di avere segnato sulla "lavagna" le seguenti note:

```
"lunedi': danza""martedi': lezione d'inglese"
venerdi' sera: cena con Antonio,sabato mattina partenza per la
```

```
montagna"ricordarsi di ritirare gli sci da Filippo"
domenica rientro per l'ora di cena
lunedi': di nuovo al lavoro
```

Come si vede le note sono riportate senza cura, proprio come faremmo su una lavagna o su un'agenda.
Costruiamo adesso, come per il caso precedente, un programma che stampi il contenuto del file in modo ordinato:

```
10 CLS
20 PRINT "STAMPA NOTE"
30 PRINT
40 OPEN "RAM:NOTE" FOR INPUT AS #1
50   While not eof(NOTE) do
60 IF EOF(1) THEN GOTO 200
70 INPUT #1,A$
80 PRINT A$
90 GOTO 50
200 ' Endwhile
210 PRINT
220 PRINT "      FINE NOTE"
230 CLOSE #1
240 END
```

Se lo eseguiamo otteniamo il seguente risultato:

```
                :
ricordarsi di ritirare gli sci da Filipp
o"
domenica rientro per l'ora di cena
lunedi': di nuovo al lavoro

          FINE NOTE
Ok
```

Cosa abbiamo imparato

In questa lezione abbiamo visto:

- l'uso dell'istruzione INPUT per fornire dati a un programma prelevandoli da un file di tipo dati;
- le modalità di riconoscimento dell'inizio e della fine di un'informazione numerica e di tipo stringa registrate su un file;
- come si può definire un formato di stampa con l'uso della funzione TAB e della clausola USING nell'istruzione PRINT.

```
montagna"ricordarsi di ritirare gli sci da Filippo"
domenica rientro per l'ora di cena
lunedi': di nuovo al lavoro
```

Come si vede le note sono riportate senza cura, proprio come faremmo su una lavagna o su un'agenda.
Costruiamo adesso, come per il caso precedente, un programma che stampi il contenuto del file in modo ordinato:

```
10 CLS
20 PRINT "STAMPA NOTE"
30 PRINT
40 OPEN "RAM:NOTE" FOR INPUT AS #1
50   While not eof(NOTE) do
60 IF EOF(1) THEN GOTO 200
70 INPUT #1,A$
80 PRINT A$
90 GOTO 50
200 ' Endwhile
210 PRINT
220 PRINT "      FINE NOTE"
230 CLOSE #1
240 END
```

Se lo eseguiamo otteniamo il seguente risultato:

```
                :
                :
ricordarsi di ritirare gli sci da Filipp
o"
domenica rientro per l'ora di cena
lunedi': di nuovo al lavoro

         FINE NOTE
Ok
```

Cosa abbiamo imparato

In questa lezione abbiamo visto:

- l'uso dell'istruzione INPUT per fornire dati a un programma prelevandoli da un file di tipo dati;
- le modalità di riconoscimento dell'inizio e della fine di un'informazione numerica e di tipo stringa registrate su un file;
- come si può definire un formato di stampa con l'uso della funzione TAB e della clausola USING nell'istruzione PRINT.

BASIC-LISP: UN INTERPRETE PER M10 (I)

Un modo per rendere accessibile un linguaggio che si diversifica nettamente dai linguaggi più tradizionali per sintassi e struttura dei dati.

Sono parecchi, al giorno d'oggi, i ricercatori di matematica simbolica, logica e intelligenza artificiale che utilizzano il LISP o linguaggi ad alto livello basati sul LISP.
Naturalmente lo scrivere un programma in LISP non ne garantisce l'intelligenza così come il disporre di materiale edile in un magazzino non garantisce la capacità di costruire una casa. Abbiamo a disposizione gli strumenti, ma occorre molto lavoro per riuscire ad assemblarli in un programma che possa "esibire" un comportamento intelligente.
Per la sua associazione con "cose" astratte, come l'intelligenza artificiale, il LISP gode della fama di linguaggio "complesso": le persone che hanno avuto l'occasione di vedere un programma in LISP senza comprenderlo ne ricordano solo la sintassi apparentemente contorta e le interminabili parentesi. Queste cose tendono a confondere il principiante, mentre nelle mani di un programmatore esperto divengono potenti oggetti di elaborazione.

Comprendere il LISP

Il LISP fu inventato da John McCarthy per facilitare la manipolazione di espressioni simboliche (come descritto nel suo articolo pubblicato sul n. 4 del 1960 di *Communications of ACM*), ed è particolarmente adatto all'elaborazione di dati strutturati a lista e ad albero. Per questa ragione è stato utilizzato per scrivere compilatori, per programmare robot e per manipolare formule matematiche. Le sue principali caratteristiche sono: mobilità dei dati (capacità di spostare dei dati con il minimo sforzo), modularità delle funzioni (non esiste un programma principale con diverse subroutine, ma il programma è esso stesso un insieme di funzioni), programmazione dichiarativa (un programma LISP è una collezione di "fatti" piuttosto che di equazioni), e metalinguistica (LISP è un linguaggio che ha la capacità di parlare di linguaggio).
Laddove molti linguaggi lavorano con numeri, LISP lavora con oggetti come matita e mina; le relazioni tra gli oggetti sono rappresentate come "liste", perciò è un processore di liste. Un esempio di interrelazione tra una matita e una mina può essere mostrato come: (la mina della matita).

Queste parole od oggetti sono chiamati "atomi". I numeri sono anch'essi atomi. Atomi simbolici, comunque, non possono cominciare con un numero ma possono contenerlo. Cioè FACT, ARG1, UNO, 12,-3.14159 sono tutti atomi; FACT, ARG1 e UNO sono atomi simbolici; 12 e -3.14159 sono numeri.
In ogni sistema LISP vi sono due atomi speciali predefiniti; l'atomo T e NIL, che sono usualmente utilizzati per rappresentare, rispettivamente, il "vero" e il "falso" logico.
Le liste sono costruite con una parentesi sinistra per segnalarne l'inizio e una destra per segnalarne la fine. (A B C), (MUL 6 3), (A(B(CD)E)FG), e () sono tutti esempi di liste.
L'atomo NIL, oltre alla funzione già vista, viene utilizzato per rappresentare la lista vuota. NIL e () sono equivalenti a tutti gli effetti.
LISP lavora con simbolici o s-espressioni composte da atomi e liste; cioè tutto ciò che è atomo o lista è anche un s-espressione. Agli occhi di un interprete LISP, i programmi e i dati sono quasi identici: questo contribuisce enormemente alla "potenza" del LISP, in quanto permette a un programma di scriverne un altro e di farlo poi eseguire.

Utilizzazione dell'interprete BASIC-LISP

Per caricare l'interprete, è sufficiente digitare il programma nell'M10; operando opportune modifiche per quanto riguarda la gestione del video e dei file, è possibile adattarlo anche ad altri computer (M20, TANDY, MACINTOSH).
Potrebbe sembrare contraddittorio scrivere un interprete per un simile linguaggio simbolico, ma questo può servire a rendere accessibile il LISP al maggior numero di persone possibile. Vale la pena di ricordare che BASIC-LISP è solo un sottoinsieme di un sistema LISP ben più sviluppato: tutti gli esempi presentati in questa serie sono garantiti per il programma in oggetto, e le differenze più importanti tra esso e le versioni più standardizzate verranno di volta in volta puntualizzate.
Fornire un'espressione all'interprete è semplice: dopo aver digitato uno statement, tipo (ADD 11), non occorre fornire

l'ENTER; non appena tutte le parentesi aperte vengono chiuse, l'espressione viene elaborata e la risposta fornita. In questo caso, il computer visualizzerà il 2.
Una cosa importante da ricordare è che gli atomi debbono essere separati con uno spazio o ritorno carrello; infatti (ADD11) non è la stessa cosa di (ADD 1 1).
Se vengono commessi errori in fase di digitazione, occorre chiudere immediatamente tutte le parentesi aperte e quando riappare il prompt (identificato dal simbolo $) digitare (%) e l'errore viene così cancellato. È bene ricordare che in questa fase il BACKSPACE non viene abilitato e quindi la tecnica suesposta è l'unica valida per evitare che la memoria dell'interprete risulti inquinata da errori (l'interprete memorizza nella lista OB tutte le espressioni che vengono digitate e di conseguenza anche quelle errate).
La tabella qui sotto mostra un esempio di routine completa.

Correzione degli errori di digitazione

```
$(MUL2)
:MUL2 NOME FUNZIONE NON VALIDO

$(%)
:MUL2 CANCELLATO DALLA LISTA OB

$(MUL 2 6)
12
```

Le operazioni compiute dall'interprete sono semplici: legge, valuta una s-espressione e stampa il risultato (anch'esso una s-espressione). Quando una lista (ADD 2 2) viene elaborata, il primo atomo è considerato come funzione mentre gli altri elementi costituenti la lista sono considerati come argomenti della funzione. Questa è conosciuta come notazione a prefisso e, sebbene scomoda all'inizio, dopo un po' di pratica risulta facile da leggersi.
La tabella in basso contiene alcuni esempi di funzioni aritmetiche nel BASIC-LISP. Dopo averle utilizzate alcune volte, si provi a eseguire calcoli più complessi per meglio comprendere la notazione e i meccanismi di funzionamento.

Funzioni aritmetiche nel BASIC-LISP

```
$(ADD 2 4.76-1)
5.76

$(SUB 13 8)
5

$(MUL 6 5 4 3 2 1)
520

$(DIV 34 7)
4.857142

$(POWER 2 4)
16

$(MUL(ADD 4 5)(POWER 2.5))
12.72792
```

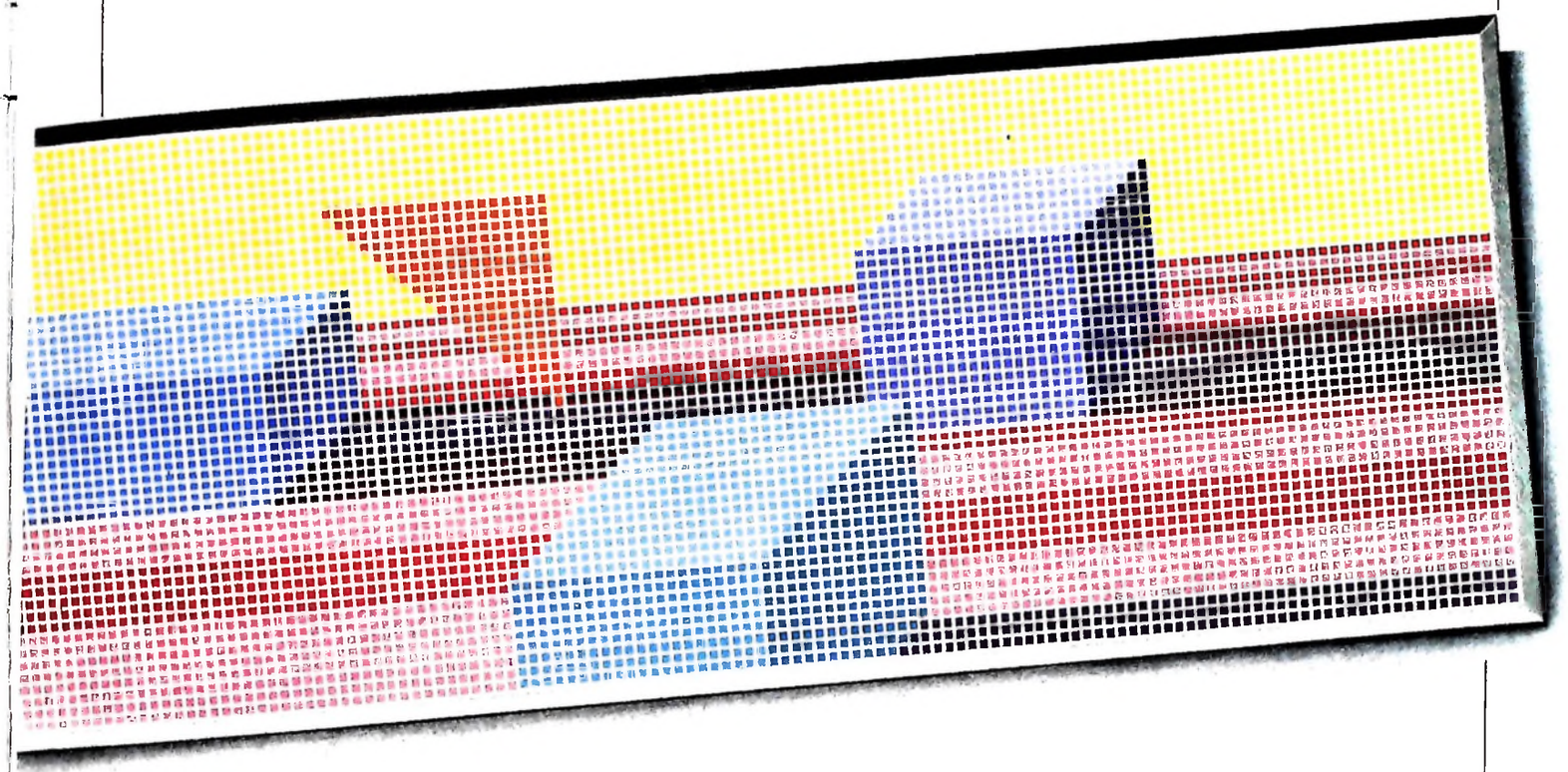
```
5'*****************************************************************
6'*                          BASIC-LISP                           *
7'*                          by C.V.P.                            *
9'*****************************************************************
15 CLS
  :CLEAR325
  :DEFINTA-E,G-V,X-Z
  :DEFSTR0
  :DIMLM(1100),PL(1100),0B(90),PT
  (90),ST(350),FP(50),T1(15),X1(15)
  :N=3000
22 PRINT@14,"BASIC-LISP"
  :PRINT@85,"attendi
   inizializzazione"
  :PRINT
24 FOR J=0 TO 48
  :READ 0B(J),PT(J)
  :NEXT
  :PE=48
  :FE=1
  :0B(46)=CHR$(13)
  :FP(1)=FRE(0)
26 FOR J=1 TO 1099
  :PL(J)=J+1
  :NEXT
  :PL(1100)=N
  :AS=1
28 T=3001
  :LP=3043
  :RP=3044
  :CC=33
  :N1=58
  :N2=44
  :LB=3031
  :QU=3030
  :NB=3032
29 CLS
30 A=0
  :QT=0
  :J=0
  :PRINT""
  :PRINT"$ ";
  :ON ERROR GOTO 26000
  :GOSUB 50
  :GOSUB 265
  :GOSUB 210
  :GOTO 30
50 J1=0
  :PRINTCHR$(14);
  :GOSUB90
55 GOSUB 100
  :IF X<>LP THEN RETURN
60 J1=J1+1
  :X1(J1)=AS
  :T1(J1)=AS
  :LM(T1(J1))=0
  :AS=PL(AS)
  :IF Q THEN RETURN
65 GOSUB 55
  :IF X=RP THEN 80
70 IF LM(T1(J1))<>0 THEN PL(T1(J1))
   =AS
  :T1(J1)=AS
  :AS=PL(AS)
75 LM(T1(J1))=X
  :IF Q THEN RETURN ELSE 65
80 PL(T1(J1))=N
  :X=X1(J1)
  :IF LM(X)=0 AND PL(X)=N THEN
   PL=(X)=AS
  :AS=X :X=N
85 J1=J1-1
  :RETURN
90 A$=INKEY$
  :IF A$=""THEN 90 ELSE PRINTA$;
  :KK=ASC(A$)
  :RETURN
100 IF KK=40 THEN X=LP
   :GOTO 200
105 IF KK=41 THEN X=RP
   :IF J1=1 OR J1=2 AND QT THEN
    RETURN ELSE 200
110 IF KK=39 THEN Q=-1
   :QT=QT+1
   :GOSUB60
   :LM(T1(J1))=QU
   :Q=0
   :GOSUB 90
   :GOSUB 55
   :Q=-1
   :GOSUB 70
   :Q=0
   :GOSUB 80
   :QT=QT-1
   :RETURN
115 IF KK<CC THEN GOSUB 90
   :GOTO 100 ELSE 125
120 IF KK<CC OR KK=40 OR KK=41 OR
    KK=39 THEN 130
125 I$=I$+A$
   :GOSUB 90
   :GOTO 120
130 IF ASC(I$)<N1 AND ASC(I$)>N2
    THEN 150
135 FOR J=0 TO PE
   : IF 0B(J)=I$ THEN X=J+N
   :I$=""
   :J=0
```

```
    :RETURN ELSE NEXT
145 J=0
    :PE=PE+1
    :OB(PE)=I$
    :X=PE+N
    :I$=""
    :RETURN
150 WW=VAL(I$)
    :GOSUB 10000
    :I$=""
    :RETURN
200 GOSUB 90
    :RETURN
210 IF A$()CHR$(13) THEN PRINT""
215 J1=1
    :X1(J1)=X
    :GOSUB 225
    :PRINT""
    :RETURN
225 IF X>5000 THEN PRINT" codice
    macchina non visualizzabile";
    :RETURN ELSE IF X>4000 THEN
    PRINT FP(X-4000);CHR$(24);
    :RETURN
230 IF X>=N THEN PRINT OB(X-N);
    :RETURN
235 IF X=0 THEN RETURN
237 IF LM(X)=QU THEN PRINT"'";
    :X=LM(PL(X))
    :GOSUB 225
    :RETURN
    :X1(J1)=X
    :PRINT"(";
245 X=X1(J1)
    :X=LM(X)
    :GOSUB 225
250 X=X1(J1)
    :J1=J1-1
    :X=PL(X)
    :IFX=N THEN PRINT")";
    :RETURN ELSE IF X>N THEN
    PRINT" . ";
    :GOSUB 225
    :PRINT")";
    :RETURN ELSE IF X=0 THEN X=1/0
255 J1=J1+1
    :X1(J1)=X
    :PRINT" ";
    :GOTO 245
265 FP(1)=FRE(0)
    :IFX>4000 AND X<5001 OR X=N OR
    X=T THEN RETURN
270 IF X>N THEN V=X
    :X=PT(X-N)
    :IF X=0 AND A=0 THEN ER=6
    :GOTO25000 ELSE RETURN
275 ST(A+1)=TT
    :ST(A+2)=AL
    :ST(A+3)=C
    :ST(A+4)=E
    :A=A+4
280 AL=PL(X)
    :E=X
    :X=LM(X)
    :GOSUB 265
285 IF X>=N AND X<4001 THEN ER=1
    :GOTO25000
290 IF X>6000 THEN 320 ELSE IF
    X>5000 THEN 315 ELSE IF LM(X)
    =LB THEN 335 ELSE IF LM(X)=NB
    THEN 337 ELSE ER=1
    :GOTO 25000
315 TT=X
    :GOSUB500
    :ON TT-5000 GOSUB 4000,4010,
    4025,4035,4060,4070,4295,4290,
    4085,4095,4130,4170,4200,4220,
    4230,4245,4255,4300,4315,4310,
    4450
    :GOTO330
320 R=X
    :X=AL
    :ONR-6000 GOSUB4050,50,4120,
    4150,4190,4285,4265,4275,4399,
    4500,4600,4650,4700,4750
330 E=ST(A)
    :C=ST(A-1)
    :AL=ST(A-2)
    :TT=ST(A-3)
    :A=A-4
    :RETURN
335 TT=AL
    :E=PL(X)
    :AL=LM(E)
    :GOSUB500
    :AL=TT
    :GOSUB500
    :C=LM(E)
    :A=A-ST(A)
    :GOTO340
337 TT=AL
    :E=PL(X)
    :AL=LM(E)
    :GOSUB500
```

▶ segue a pagina 626

ANCORA SULLA GRAFICA INTERATTIVA

Riprendiamo e ampliamo il discorso sulla grafica interattiva.



È importante notare che uno dei maggiori benefici che si possono trarre dall'utilizzazione di un programma interattivo per la creazione di figure sta nella possibilità di far "ricordare" in maniera permanente al calcolatore ciò che è stato creato, al fine di poter utilizzare la figura introdotta per gli usi più disparati.
Il programma diviene così un mezzo atto a creare forme semplici di vario tipo, tali da poter essere poi utilizzate da altri programmi, completamente indipendenti, che utilizzano il disegno così creato leggendolo da una memoria di massa (cassetta, disco ecc.) come se fosse solo una successione di numeri (le coordinate dei punti da unire).
A questo proposito si osserva che la creazione di una serie di dati da leggere mediante un altro programma impone una compatibilità tra i formati di scrittura e di lettura dei due programmi (si devono in pratica scrivere e leggere i dati nello stesso modo in entrambi i programmi).

PEN2: programma esteso di grafica interattiva

Il programma è un'effettiva estensione del programma PEN precedentemente illustrato. Esso si propone infatti di offrire la possibilità di memorizzare un disegno creato e poi di rivederlo; è ovvio che il disegno memorizzato verrà visualizzato in maniera più veloce poiché di esso vengono ricordate solo le coordinate dei punti estremi di ogni segmento.

Commento al programma

PEN2 si basa sulla possibilità fornita da M10 di utilizzare la memoria operativa per la memorizzazione stabile non solo di programmi ma anche di dati creati per mezzo di essi.
Le istruzioni BASIC qui ampiamente utilizzate, che

permettono di inserire un gruppo di dati in una memoria permanente creando il nome di *data file* (archivio di dati su memoria di massa), sono le seguenti:

OPEN "dove:nomefile" FOR azione AS numerofile

dove = unità fisica per l'apertura del *data file* (RAM: memoria del calcolatore, CAS: cassetta)

azione = operazione da effettuare sul file aperto:

1) Input = per leggere dati da un file
2) Output = per scrivere dati su un file cancellando quelli già eventualmente presenti
3) Append = per scrivere dati aggiungendoli alla fine di quelli già introdotti

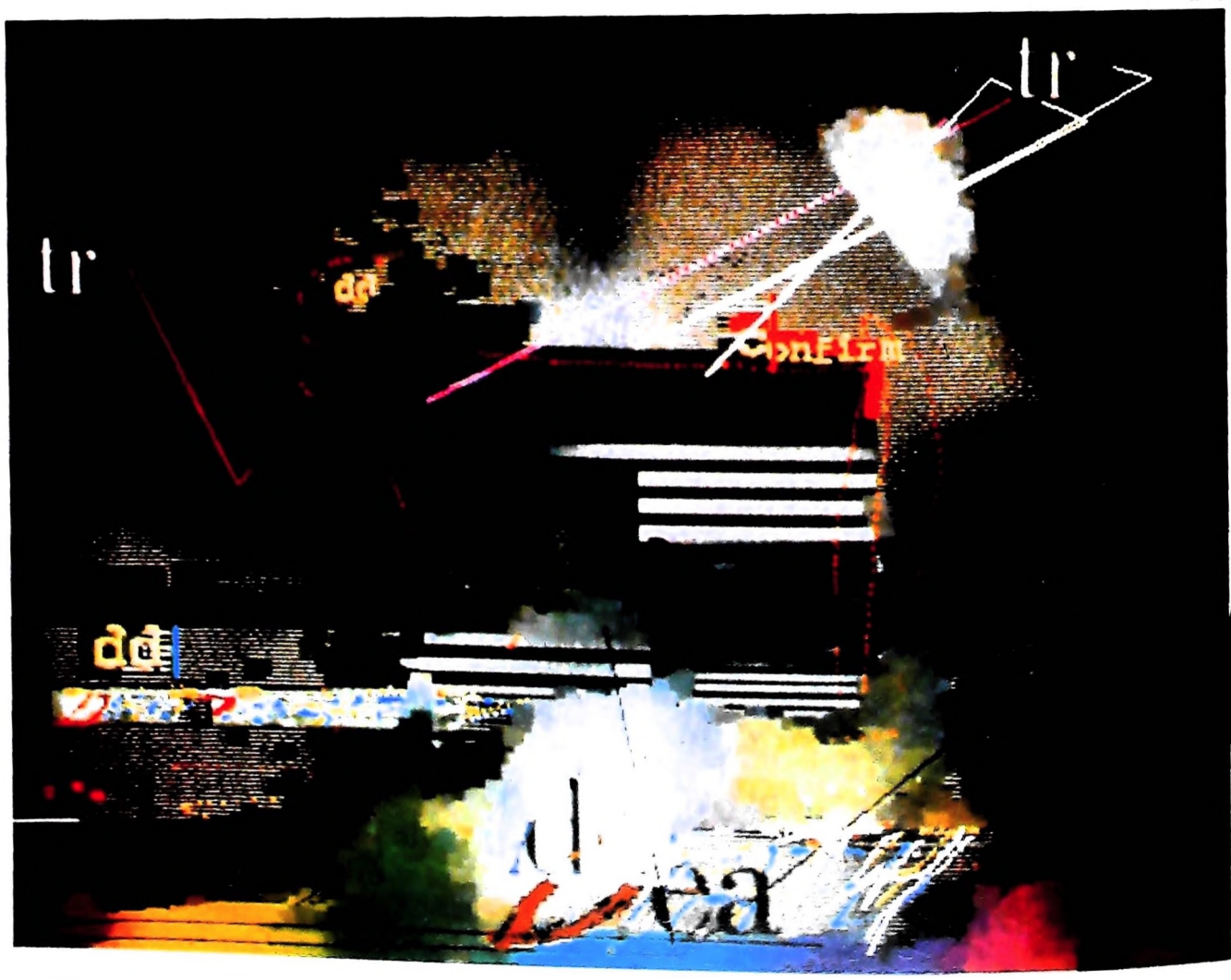
numerofile = il numero che indica il riferimento interno del file

CLOSE numerofile

Computer-art

Fra le applicazioni della Computergrafica sta avendo sempre più diffusione e seguito quella che va sotto il nome di "computer-art".
La possibilità di realizzare immagini, abbinamenti cromatici e sintesi di scene offre anche alla creatività dell'artista la possibilità di utilizzare un nuovo strumento per comporre: il computer.
Come per ogni strumento creativo, una buona conoscenza delle possibilità offerte dall'elaboratore grafico consente all'artista di forgiarsi uno stile personale, proprio come i diversi utilizzi del pennello e della tavolozza dei colori differenziano fra loro i pittori. In proposito, esistono tuttora dei validi esempi di opere di "computer-art" che simulano molto bene l'immagine pittorica, dallo stile impressionista fino all'iperrealista e al surrealista.

Attualmente però si sta formando una generazione di giovani artisti per i quali il computer è in grado di dare origine a uno stile proprio, che ambisce cioè a sciogliere quei legami psicologici che hanno, fino a poco tempo fa, legato il suo impiego all' "imitazione" degli stili tradizionali.
Lungi dall'essere una limitazione per la fantasia e la creatività dell'uomo, il computer sta avviandosi a essere un ulteriore ed importante ponte di collegamento fra la cultura tecnologica e quella artistica. Il successo e l'affermarsi della "computer-art" rappresenterebbe senza dubbio un grosso risultato della capacità di sintesi dell'uomo, che da un sofisticato strumento tecnologico, generato dalle sue capacità logico-razionali, riesce a ottenere dei validi risultati artistici, legati invece alla sua creatività logico-intuitiva.



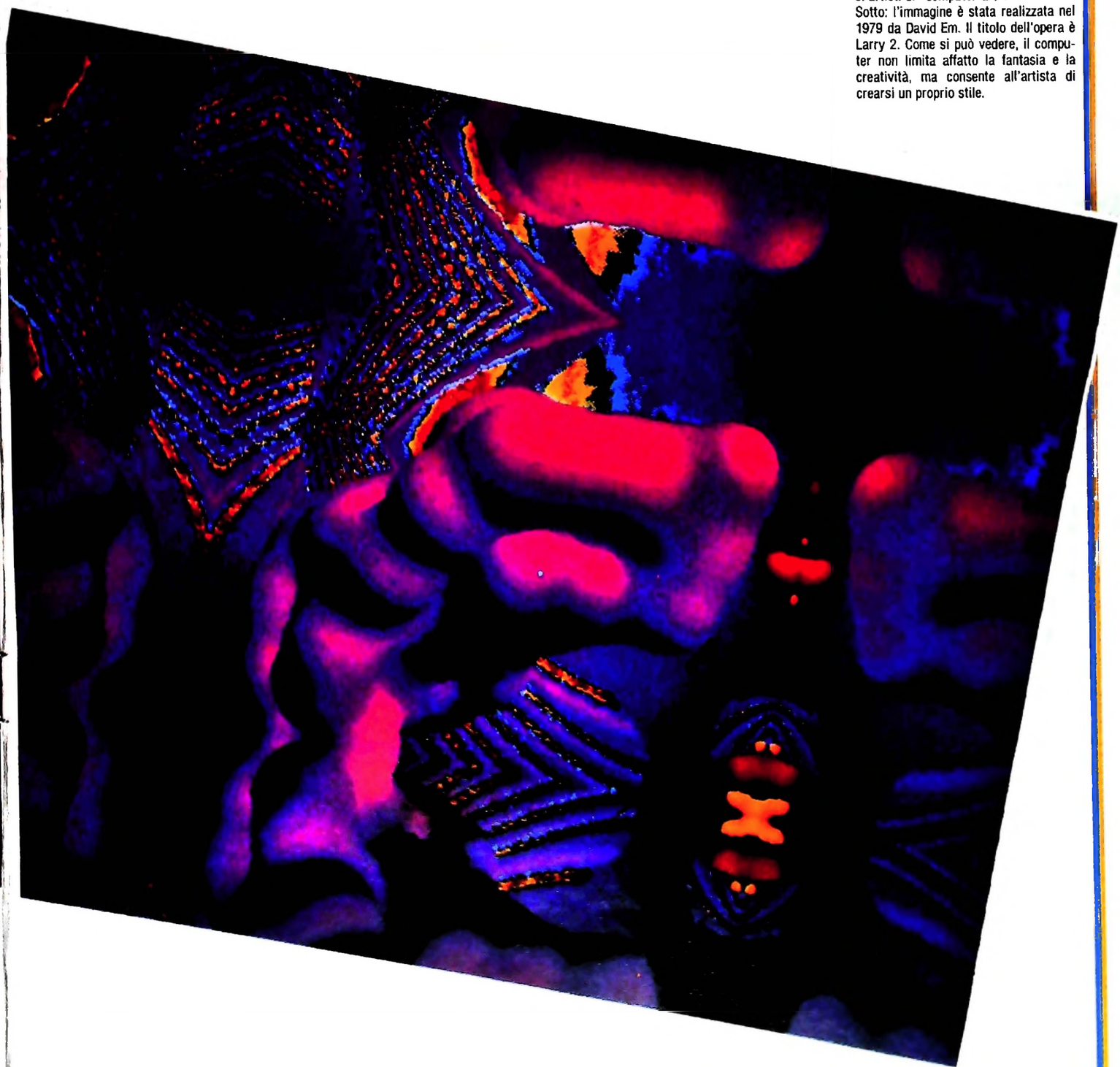
numerofile= è il numero con il quale il file è stato aperto.

Questo numero si rende necessario perché il data file non è riconosciuto dal programma con il suo nome, il quale è unicamente usato perché costituisce una semplice "etichetta" di identificazione, ma viene riconosciuto con il numero che lo rappresenta (non si dice infatti CLOSE nomefile, ma CLOSE numerofile ecc...)
È così possibile, con un opportuno utilizzo delle istruzioni ora richiamate, scrivere e leggere in maniera rapida e agevole su un'unità di memoria indicata.

Come si utilizza

Digitato il programma e il conseguente comando di RUN (oppure F4) viene presentato un menù che indica le operazioni disponibili. L'unica operazione nuova rispetto

A lato: un'opera di Joel Slayton, del Massachusetts Institute of Technology, realizzata nel 1980. Il titolo è JSDD♯2. Con David Em, Slayton è fra i più famosi artisti di "computer-art".
Sotto: l'immagine è stata realizzata nel 1979 da David Em. Il titolo dell'opera è Larry 2. Come si può vedere, il computer non limita affatto la fantasia e la creatività, ma consente all'artista di crearsi un proprio stile.



alla precedente realizzazione è F>ile.
F>ile. La pressione del tasto f minuscolo permette l'ingresso in un "sottomenù" che ha lo scopo di indicare le possibili operazioni di gestione del data file:
M>emo: permette di memorizzare il disegno che verrà creato con le quattro freccette atte al posizionamento del cursore grafico in un data file di nome DRAW.DO che, se inesistente, verrà creato, se già presente verrà ripulito del suo contenuto e aperto per scrittura.
Questo comando visualizza sullo schermo il cursore grafico permettendone lo spostamento per creare un disegno e inoltre una linea di messaggi.
C>ancella file: con la pressione del carattere c minuscolo si ottiene la cancellazione dello schermo e del data file in via di creazione.
S>top: permette l'uscita dal programma e la memorizzazione stabile del data file.
D>isegna: se una precedente sessione di lavoro si era chiusa con la creazione di un disegno memorizzato esso verrà ora visualizzato sullo schermo.

Il listato

Si notino le linee 130-142 che hanno lo scopo di controllare la presenza del data filè da leggere; si utilizza l'istruzione APPEND per aprire il file per un controllo evitando di variare in alcun modo i dati in esso presenti.
La linea 160 inizializza un ciclo di lettura con un valore di controllo sovrabbondante; il ciclo viene interrotto quando si raggiunge la fine del data file, mediante la linea 170.
Il sistema di memorizzazione dei dati si basa essenzialmente sul codice numerico ASCII dei caratteri di movimento del cursore grafico. Viene infatti memorizzata la coppia di coordinate numeriche che si rilevano a ogni cambiamento di direzione del cursore.
È quindi possibile memorizzare un poligono qualsiasi registrandone solo i vertici (dopo un vertice si deve necessariamente cambiare direzione con il cursore per continuare il disegno). Tale operazione di riconoscimento dei tasti premuti con conseguente analisi è compiuta dalle linee 300-350.
Si noti che alla fine del programma (linee dalla 1000 in poi) viene memorizzata, se si è in modalità M>emo, l'ultima coordinata rilevabile; questo perché l'ultimo tasto premuto non è una freccetta (con cambiamento di direzione) ma il tasto s che provocherebbe, secondo la logica finora seguita, la perdita dell'ultima coppia di coordinate.

Ampliamenti

Fra i numerosi ampliamenti possibili è consigliabile una analisi approfondita del listato allo scopo di estrarre la routine di visualizzazione del data file in modo tale da poter utilizzare il programma presentato per creare disegni da inserire in altri programmi che dovranno avere all'interno la sola procedura di lettura. Si tratta in pratica di trasformare PEN2 in quello che si indica solitamente con il nome di *EDITOR GRAFICO* (programma con cui creare in modo interattivo immagini utilizzabili da altri programmi).
A tale scopo si consiglia di rendere possibile la correzione e l'aggiunta di altri elementi nel data file, nonché la possibilità di creare diversi file con nome diverso.

```
5  REM**INIZIALIZZAZIONE E MENU**
8  CLS
10 PRINT@0,"*PEN2*M>ATITA,C>LS,S>TOP,F>ILE+,-.MOV.
11 MM=0:X=2:Y=10
12 CL=1:LL$=A$:A$=""
13 A=X:B=Y
14 REM**LETTURE DA TASTIERA
15 A$=INKEY$:IF A$=""THEN 15
20 REM
25 IFASC(A$)=28THENX=X+1ELSEIFASC(A$)=29THENX=X-1ELSEIFASC
(A$)=30THENY=Y-1ELSEIFASC(A$)=31THENY=Y+1ELSEIFASC(A$)=102
THENCL=0ELSEIFASC(A$)=109THENCL=1ELSEIFASC(A$)=115THEN1000
ELSEIFASC(A$)=99THEN800ELSE1
26 REM
28 REM**CONTROLLO SULLE COORDINATE**
30 IF X<3 THEN X=3
31 IF X>236 THEN X=236
32 IF Y<10 THEN Y=10
33 IF Y>60 THEN Y=60
45 IF CL=0THEN 60
46 REM
48 REM**DISEGNO E MOVIMENTO MATITA**
50 LINE(A-1,B-1)-(A+1,B+1),0,B F
52 LINE(X-2,Y-2)-(X+1,Y+1),1
53 IF MM=1 THEN 300
54 PSET(X,Y,1)
55 GOTO 12
58 REM
60 REM**GESTIONE FILE SUPPORTO**
100 PRINT@ 0,"GESTIONE FILE M>EMO,D>ISEGNA
110 RP$=INKEY$:IF RP$=""THEN 110
120 IF RP$="M" THEN 250 ELSE IF RP$<>"D" THEN 100
125 '***LETTURA DISEGNO DA FILE CREATO*
127 '*CONTROLLO SE ESISTE IL FILE******
130 OPEN "RAM:DRAW"FOR APPEND AS 1
142 CLOSE 1
146 OPEN "RAM:DRAW" FOR INPUT AS 1
148 IF EOF(1) THEN PRINT@200,"ATTENZIONE, DATI INESISTENTI
":CLOSE 1:GOTO 10
149 '**IL CONTENUTO VIENE DISEGNATO****
150 INPUT# 1,X1,Y1
160 FOR CI=1 TO 2000
170 IF EOF(1) THEN CLOSE 1:GOTO 10
180 INPUT#1,X2,Y2
190 LINE(X1,Y1)-(X2,Y2)
200 X1=X2:Y1=Y2
210 NEXT CI
220 CLOSE 1
250 '***GESTIONE MEMO DEI DATI*********
260 PRINT@0,"MEMO COORDINATE***C>ANCELLA FILE,S>TOP"
270 MM=1
275 OPEN "RAM:DRAW" FOR OUTPUT AS 1
280 GOTO 48
300 IF ASC(A$)>27 AND ASC(A$)<32 THEN 330 ELSE 12
330 '*********MEMORIZZA I DATI*********
340 IF A$<>LL$ THEN PRINT# 1,A,B
350 GOTO 12
800 'CANCELLA IL FILE CHE SI STA CREANDO
805 IF MM=1 THEN CLOSE 1:OPEN"RAM:DRAW"FOR OUTPUT AS
1:CLOSE 1
810 GOTO 8
1000 '***FINE DEL PROGRAMMA***********
1010 IF MM=1 THEN PRINT# 1,X,Y
1020 CLOSE 1
1030 END
```

Anche in leasing con Olivetti Leasing

LA FAMIGLIA DEI PERSONAL COMPUTER OLIVETTI

FRIENDLY & COMPATIBLE

Questa famiglia di personal compatibili tra loro e con i più diffusi standard internazionali, non ha rivali per espandibilità e flessibilità. Prestazioni che su altri diventano opzionali, sui personal computer Olivetti sono di serie. Per esempio M24 offre uno schermo ad alta definizione grafica, ricco di 16 toni o di 16 colori e con una risoluzione di 600x400 pixel; mentre la sua unità base dispone di 7 slots di espansione, fatto questo che gli consente di accettare schede di espansione standard anche se utilizza un microprocessore a 16 bit reali (INTEL 8086).
Ma ricchi vantaggi offrono anche tutti gli altri modelli.

Basti pensare che tutte le unità base includono sia l'interfaccia seriale che quella parallela. Oppure basti pensare all'ampia gamma di supporti magnetici: floppy da 360 a 720 KB o un'unità hard disk (incorporata o esterna) da 10 MB.
La loro compatibilità, inoltre, fa sì che si possa far uso di una grande varietà di software disponibile sul mercato. Come, ad esempio, la libreria PCOS utilizzabile anche su M24. Come le librerie MS-DOS®, CP/M-86® e UCSD-P System®, utilizzabili sia da M20 che da M21 e M24.

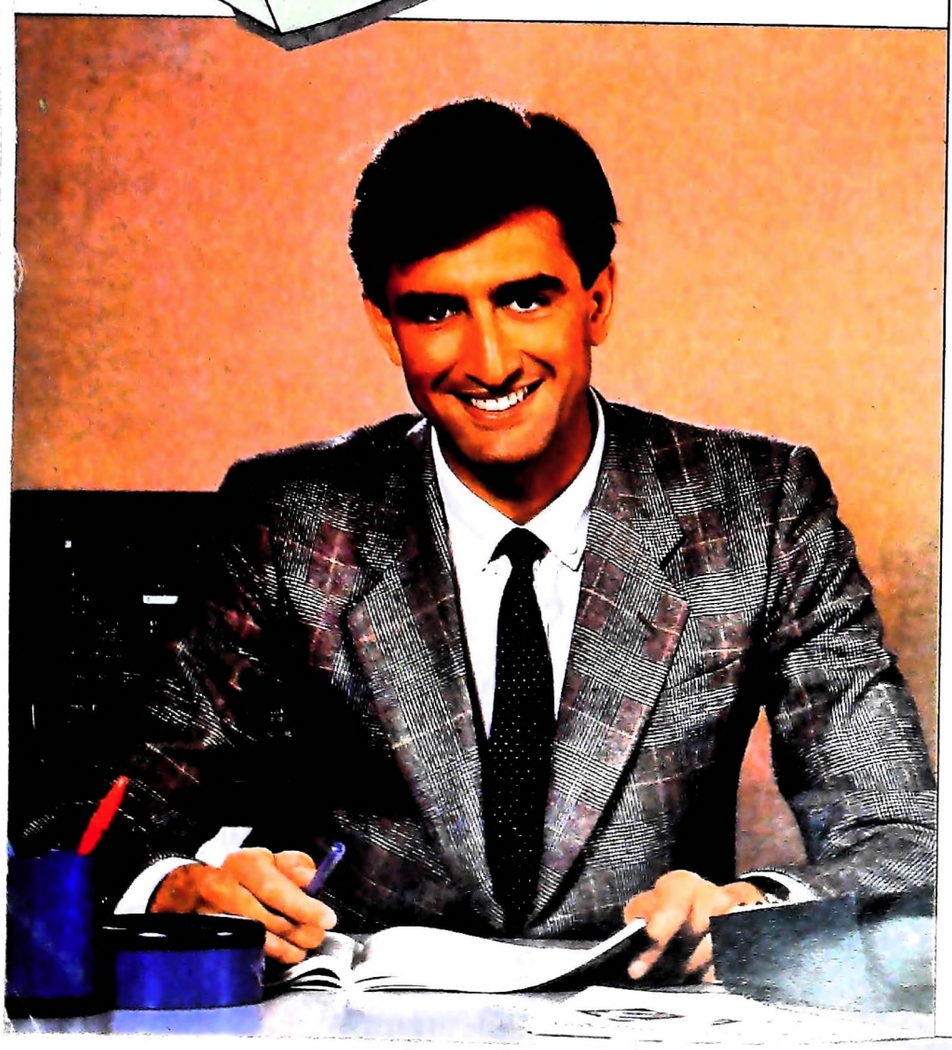
MS-DOS è un marchio Microsoft Corporation
CP/M-86 è un marchio Digital Research Inc.
UCSD-P System è un marchio
Regents of the University of California

olivetti

inviare il coupon a Olivetti
Personal Computer Via Meravigli 12, 20123 Milano.
NOME
INDIRIZZO
CITTÀ
TELEFONO

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.