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

36 CORSO PRATICO COL COMPUTER

421883

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

diretto da GIANNI DEGLI ANTONI

FABBRI EDITORI

Direttore dell'opera
GIANNI DEGLI ANTONI

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

UMBERTO ECO
Ordinario di Semiotica presso l'Università di Bologna

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

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

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

Curatori di rubriche
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
ETTORE DECIO, LUCA SPAMPINATO, GIANCARLO MAURI, ADRIANO DE LUCA 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

L'USO DELLO STACK (I)

Una tecnica efficiente per la soluzione di problemi software, utilizzata da moltissimi linguaggi e sistemi operativi.

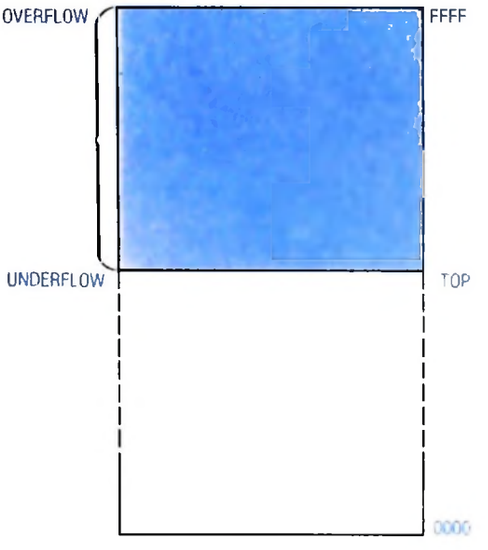
Abbiamo visto nel capitolo precedente l'uso dello STACK per il controllo dei sottoprogrammi: possiamo adesso completare il quadro prendendo in considerazione altri usi di questa tecnica. È necessario tuttavia precisare che quando si definisce una zona della memoria come zona STACK, questa diventa non accessibile alla scrittura da parte di programmi. Inoltre è possibile, quando non si effettuano i necessari controlli, superare i limiti sia superiore che inferiore dello STACK. Alcuni microprocessori sono in grado di rilevare questo errore (figura in basso a sinistra) di *overflow* o *underflow* (traboccamento superiore e inferiore) e di segnalarlo, mentre per altri bisogna provvedere per via software.
Come abbiamo detto, l'uso dello STACK non si limita solo alla gestione dei sottoprogrammi. La sua efficienza nella risoluzione di problemi software è ben nota e oggi ci sono moltissimi linguaggi e sistemi operativi che utilizzano la tecnica dello STACK. Per illustrare questa possibilità prendiamo, per esempio, il calcolo di una espressione aritmetica con la forma *polacca inversa*, comunemente chiamata anche notazione *postfissa* usando lo STACK e le istruzioni:
PUSH: deposita il contenuto dell'accumulatore A nel TOP dello STACK, incrementando il valore di TOP;
POP: deposita il contenuto del (TOP - 1) nel registro ausiliario A.R. decrementando il valore di TOP.
Ovviamente per questo tipo di uso, onde evitare interferenze con l'utilizzazione dello STACK per la gestione dei sottoprogrammi, è opportuno creare un secondo STACK in un'altra parte della memoria.
L'espressione aritmetica che consideriamo è:

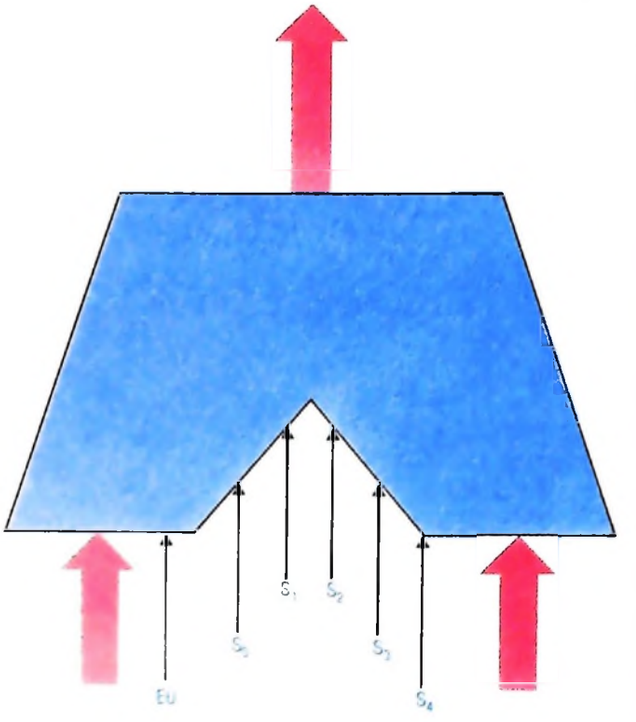
(H + L) * C - (D/E)

che trasformata in notazione post-fissa diventa:

""HL + C * DE / -""

La regola per calcolare il valore di un'espressione in notazione postfissa è molto semplice: si comincia sempre da sinistra e si procede verso destra indicando prima le variabili poi le operazioni, ricordando che un operatore aritmetico viene ap-

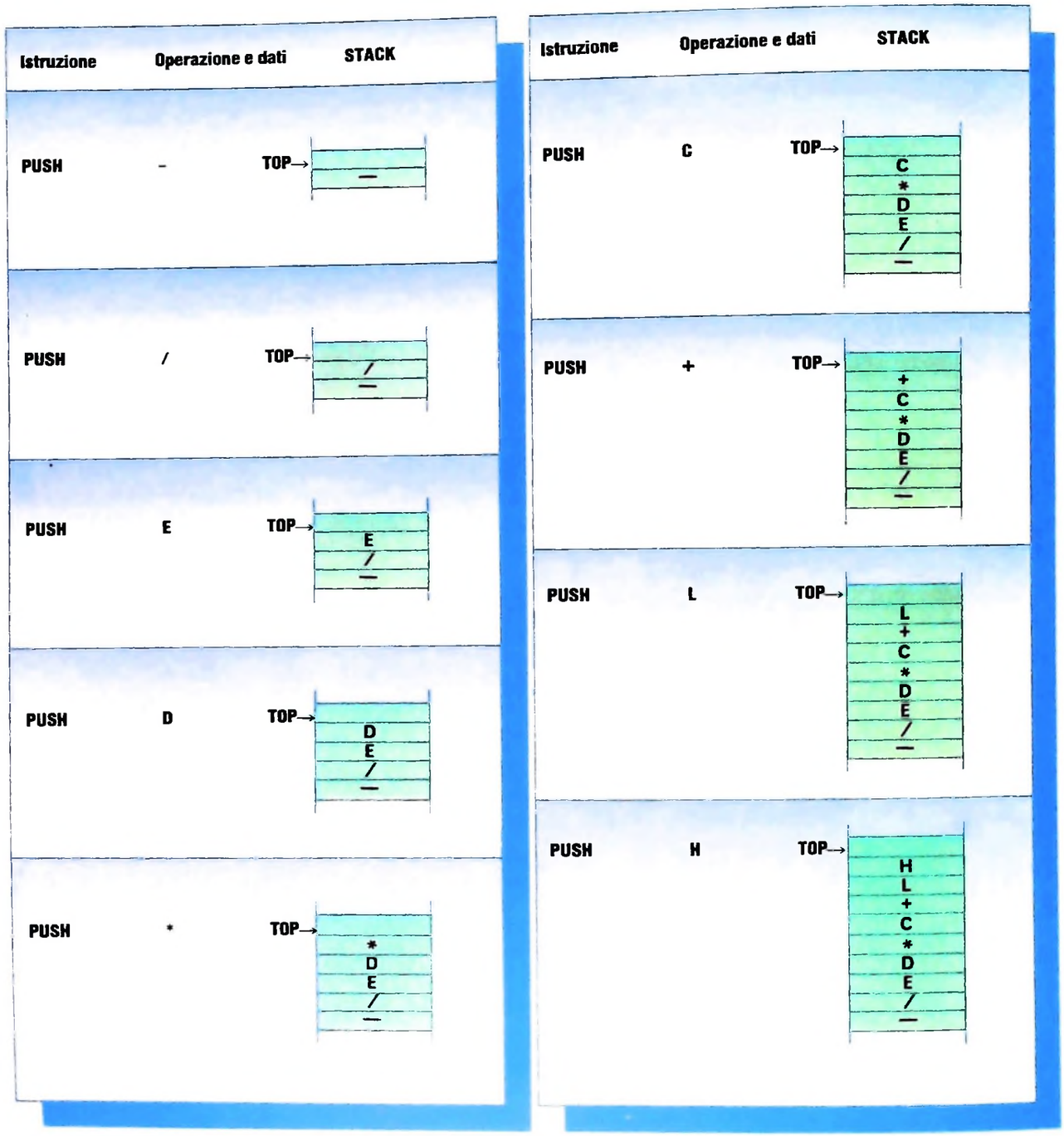


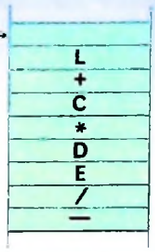
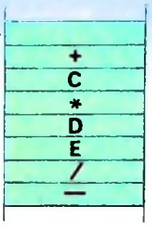
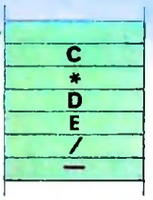
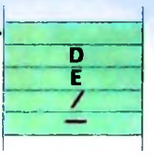
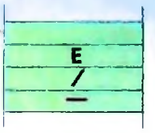
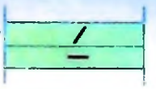
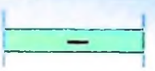


Schematizzazione del modo con cui alcuni microprocessori rilevano l'errore di overflow o di underflow. A lato, il registro ALU, con i due BUS di entrata provenienti dagli accumulatori, la sua uscita verso il BUS interno e i comandi di funzioni.

plicato ai due operandi che lo precedono e che la terna così considerata deve essere quindi sostituita con il risultato dell'operazione.
Ma vediamo come vengono usati lo STACK e le sue istruzioni nell'esempio proposto qui sotto.
In fase di programmazione carichiamo l'espressione già trasformata nello STACK nella forma indicata nell'illustrazione in questa pagina:

Fatto questo, incominciamo il processo inverso, cioè l'esecuzione del programma e quindi il calcolo dell'espressione nell'accumulatore A.
È bene notare che in questo esempio stiamo mettendo in risalto solo l'uso dello STACK e le sue istruzioni, tralasciando la spiegazione di tutto il supporto software necessario per realizzare correttamente le operazioni (istruzioni della pagina accanto).



Istruzione	Operazioni e dati	STACK
POP	**A.R. ← (TOP — 1);** "analizziamo il suo contenuto per vedere se è un dato o una operazione. Trattandosi di un dato lo depositiamo in A" **A ← (A.R.);** **A = H;**	TOP→ 
POP	**A.R. ← (TOP — 1);** **B ← (A.R.);** **B = L;**	TOP→ 
POP	**A.R. ← (TOP — 1);** "trattandosi di una operazione di somma questa si effettua sugli accumulatori e il risultato si deposita in A" **A = (H + L);**	TOP→ 
POP	**A.R. ← (TOP — 1);** **B ← (A.R.);** **B = C;**	TOP→ 
POP	**A.R. ← (TOP — 1);** **A ← (A * B);** **A = (H + L)*C;**	TOP→ 
POP	**A.R. ← (TOP — 1);** **B ← (A.R.);** **B = D;**	TOP→ 
POP	**A.R. ← (TOP — 1);** "poiché i due accumulatori sono già occupati e il contenuto di A.R. è un dato, allora si sposta il contenuto di A in una località di memoria con indirizzo XXXX;	TOP→ 
STA XXXX	**(XXXX) A;** spostiamo il contenuto di **B in A;** **A ← B;** depositiamo il contenuto di **A.R. in B; B (A.R.);**	
POP	**A.R. ← (TOP — 1); A ← (A/D); A = D/E;**	TOP→ 
POP	**A.R. ← (TOP — 1);** "essendo anche questa una operazione come la precedente allora **B ← A; B = D/E;**	TOP→ 
LDA (XXX)	**A ← (XXXX); A = (H + L) *C;** **A ← (A — B); A = (H + L) *C — D/E;** come volevasi dimostrare"	

Alu

Il registro ALU (figura a pag. 557) ha le due entrate degli accumulatori A e B, l'uscita a tre-stati che va al BUS interno e il segnale EU che abilita il registro, mentre i segnali S_0, S_1, S_2, S_3, S_4, che in totale darebbero 32 operazioni, effettuano le 25 operazioni elencate nella tabella riportata qui sotto. Le funzioni svolte dalle prime di queste operazioni sono facilmente comprensibili, mentre le funzioni svolte dalle operazioni di rotazione verranno spiegate assieme all'analisi del registro STATUS FLAG (registro di stato) che vedremo nel prossimo articolo.

S_0	S_1	S_2	S_3	S_4	
0	0	0	0	0	NESSUNA OPERAZIONE
0	0	0	0	1	A + B
0	0	0	1	0	A - B
0	0	0	1	1	B - A
0	0	1	0	0	NEGATIVO DI A
0	0	1	0	1	NEGATIVO DI B
0	0	1	1	0	Ā
0	0	1	1	1	B̄
0	1	0	0	0	A AND B
0	1	0	0	1	A OR B
0	1	0	1	0	A ⊙ B
0	1	0	1	1	ROL A
0	1	1	0	0	ROL B
0	1	1	0	1	ROR A
0	1	1	1	0	ROR B
0	1	1	1	1	ASL A
1	0	0	0	0	ASL B
1	0	0	0	1	ASR A
1	0	0	1	1	ASR B
1	0	1	0	0	LSR A
1	0	1	0	1	LSR B
1	0	1	1	0	DEC A
1	0	1	1	1	DEC B
1	1	0	0	0	INC A
1	1	0	0	1	INC B
1	0	0	1	0	LIBERE
1	0	0	1	1	
		•			
		•			
		•			
1	1	1	1	1	

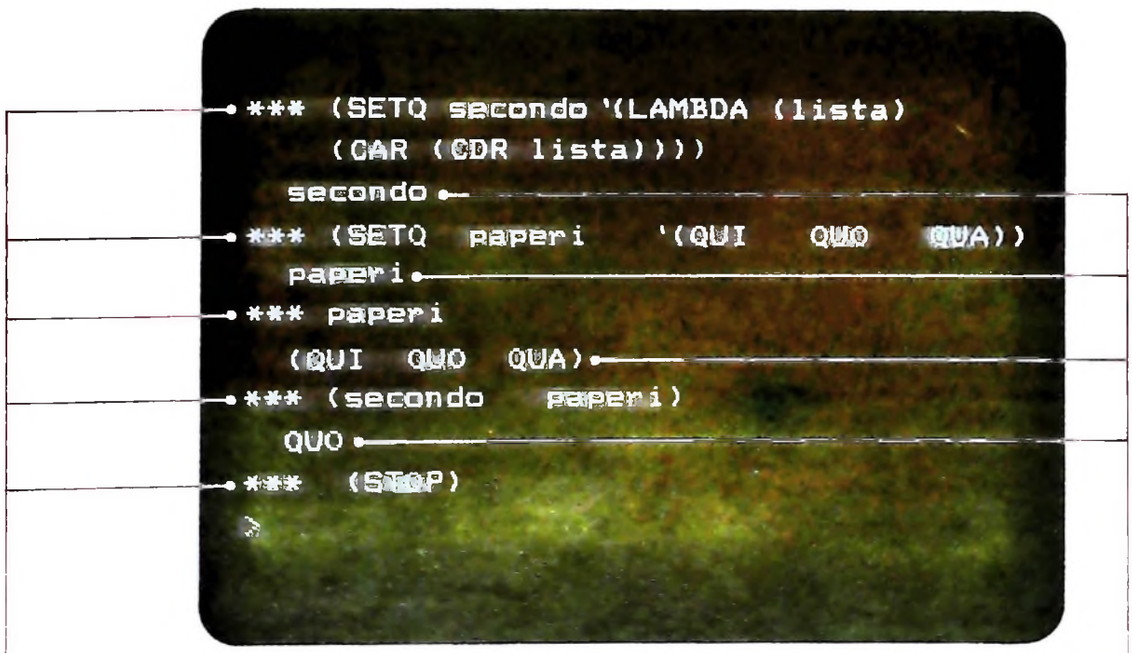
IL LINGUAGGIO LISP (I)

Un linguaggio di programmazione che ricopre ancora un ruolo primario nell'evoluzione dell'Intelligenza Artificiale.

Sicuramente il Lisp non è l'unico linguaggio di programmazione utilizzato in A.I., anzi si è accennato in un precedente articolo come praticamente ogni gruppo di ricerca abbia sviluppato il proprio particolare ambiente di programmazione. Ma l'unico che sia riuscito a resistere alla rapida obsolescenza che di solito aggredisce questi prodotti, mantenendo inalterate le proprie caratteristiche fondamentali, è proprio il Lisp. Anzi, gli ultimi anni hanno visto un rinnovato interesse per questo linguaggio, che oggi ricopre saldamente la prima posizione tra gli strumenti di sviluppo di "applicazioni intelligenti".

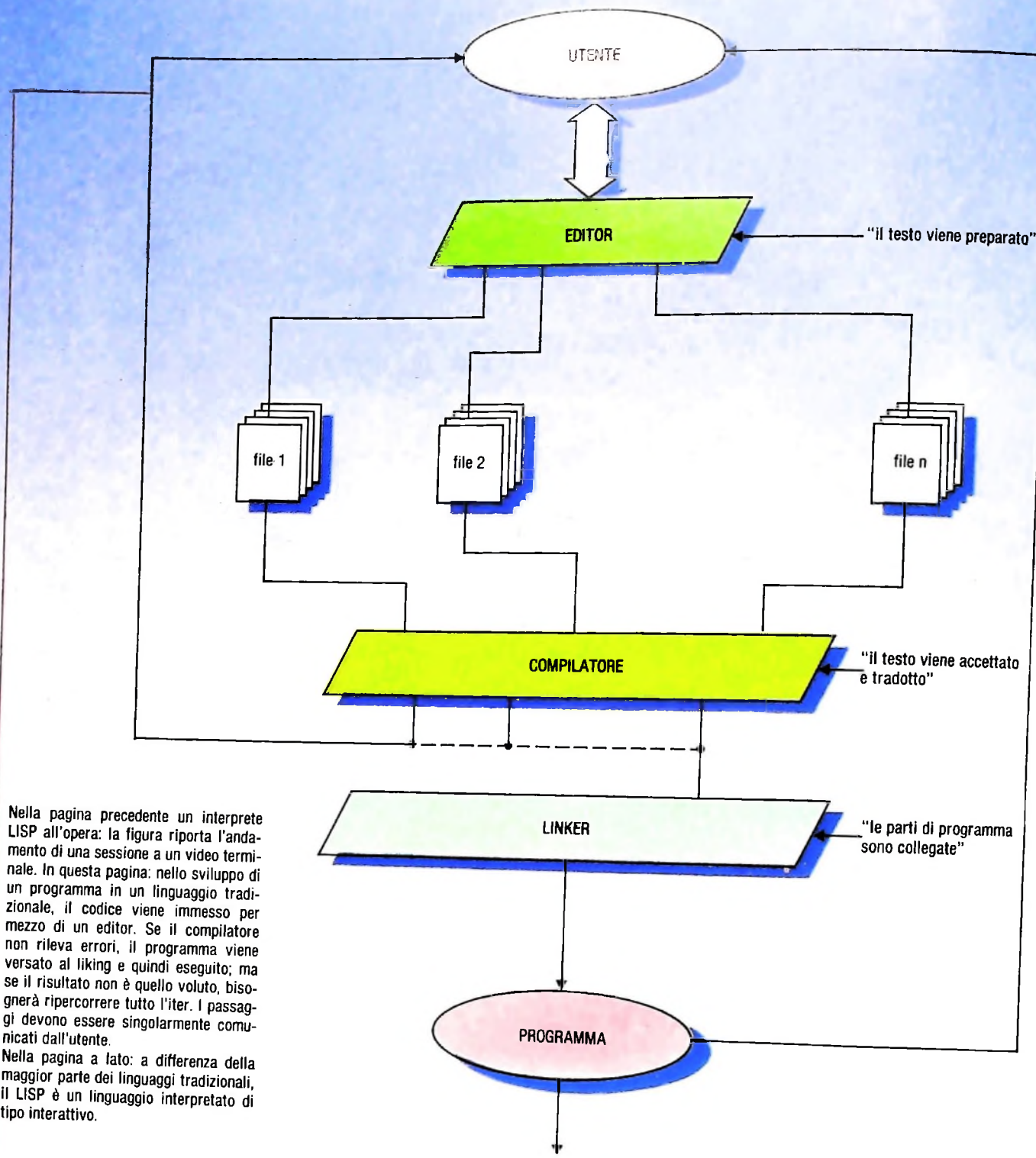
I sistemi LISP

Parlare del LISP come di un linguaggio di programmazione è in un certo senso restrittivo.
Per comprendere la ragione di ciò è necessario esaminare brevemente come procede lo sviluppo di un programma in un linguaggio tradizionale. Il codice viene immesso, suddiviso in uno o più file, mediante l'uso di un editor. I file sono gestiti dal sistema operativo della macchina ospite e vengono passati come input a un compilatore con un comando esplicito. In assenza di errori segnalati dal compilatore (errori sintattici), è possibile procedere alla fase di linking. Finalmente il programma di partenza è pronto per essere eseguito, sempre con un comando esplicito. Se non si ottiene il risultato desiderato, è necessario ripercorrere tutti i passi del ciclo (figura a pagina 562). Si può notare che lo sviluppo del programma viene gestito dalla cooperazione tra sistema operativo, utente e compilatore dello specifico linguaggio. In un ambiente LISP al contrario, l'utente, nel corso dello sviluppo dei suoi programmi, deve interagire soltanto con il sistema LISP stesso. Infatti egli si trova di fronte principalmente a un interprete, che rende più vicine le fasi di produzione ed esecuzione del codice, affiancato da un sistema interattivo



il sistema mette *** quando attende input da parte dell'utente

risposte del sistema



Nella pagina precedente un interprete LISP all'opera: la figura riporta l'andamento di una sessione a un video terminale. In questa pagina: nello sviluppo di un programma in un linguaggio tradizionale, il codice viene immesso per mezzo di un editor. Se il compilatore non rileva errori, il programma viene versato al liking e quindi eseguito; ma se il risultato non è quello voluto, bisognerà ripercorrere tutto l'iter. I passaggi devono essere singolarmente comunicati dall'utente.
Nella pagina a lato: a differenza della maggior parte dei linguaggi tradizionali, il LISP è un linguaggio interpretato di tipo interattivo.

che provvede a gestire sia i rapporti con il sistema operativo, pressoché invisibile all'utente, sia le risorse fisiche della macchina (figura a pagina 563). Parleremo quindi più correttamente di sistemi LISP.

Struttura di un sistema LISP

Nell'analisi dei tipici sistemi LISP è possibile individuare tre parti costituenti distinte.

Struttura associativa. Questa parte del sistema svolge una duplice funzione. Da un lato permette all'utente di memorizzare e ritrovare informazioni utili per l'attività di programmazione, dall'altro provvede alla gestione delle uniche strutture dati del linguaggio, *le liste* e *gli atomi*, facendosi anche carico delle operazioni di gestione delle risorse di memoria (il nome LISP deriva dalle due parole "List Processing", ovvero "manipolazione di liste").
Nel LISP un atomo è una qualunque stringa di caratteri, tipo *Atlantide, Vidanken, gu*, cioè un nome che svolge funzione di identificatore. Una lista, invece, può essere:
a) la lista vuota (), indicata anche dal nome NIL;
b) una parentesi aperta (, seguita da un numero arbitrario di atomi o liste, seguito da una parentesi chiusa).
Sono liste correttamente formate le seguenti: (Usare il computer), (Quo vadis?), (Il mio nome è Bond (James Bond)), (Paperone (Paperino (Qui Quo Qua)) (Topolino (Tip Tap))).
In realtà le liste non sono soltanto la struttura di rappresen-

tazione dei dati, ma anche quella dei programmi stessi, nonché il mezzo attraverso cui avviene la comunicazione tra utente e sistema.
Questa si realizza per mezzo di liste il cui primo elemento è il nome di un comando, seguito dai suoi argomenti. Quando l'utente fornisce al sistema una lista, questo esegue le operazioni specificate, restituisce un risultato e richiede la prossima lista. Esaminiamo brevemente i comandi disponibili per richiedere manipolazioni su liste: questi sono CAR, CDR e CONS.
Il comando CAR permette di ottenere il primo elemento della lista che costituisce il suo argomento. Per esempio, sottoponendo al sistema la lista (CAR '(cucurucucu paloma)) si ottiene come risposta cucurucucu. Il carattere "'" specifica al sistema che quanto segue deve essere inteso testualmente così come appare. Infatti, trattandosi di una lista avrebbe ottenuto altrimenti il trattamento riservato a comandi con argomenti. Per esempio:
(CAR (CAR '((papé satan) (papé satan) aleppe)))
produce l'atomo papé; invece
(CAR '(CAR '(papé satan))) produce l'atomo CAR.
Complementare al CAR è il comando CDR che, data una lista, la restituisce privata del suo primo elemento. Quindi da (CDR ' (uno nessuno centomila)) si ottiene (nessuno centomila) e da (CAR (CDR '(suonate (le vostre trombe) che noi suoneremo (le nostre campane))))
si ha (le vostre trombe). È necessario però anche costruire le liste: esiste a questo scopo la funzione CONS che appunto genera una lista a partire dai suoi argomenti ciascuno dei quali può essere una lista o un atomo. Questi diverranno rispettivamente il primo elemento e il resto della nuova lista.
(CONS 'veni '(vidi vici)) produce (veni vidi vici) e
(CONS '(qualche volta) ' ((qualche volta) è meglio di (sempre))) produce ((qualche volta) (qualche volta) è meglio di (sempre)).
È notevole il risultato che
(CONS (CAR '(James Bond)) (CDR '(James Bond)))
risulta in (James Bond) e questo vale per ogni lista.
Per maneggiare in modo significativo le liste è utile disporre di alcuni predicati fondamentali. NULL riconosce la lista vuota. (NULL ( )) produce l'atomo True che rappresenta il valore di verità VERO. Invece (NULL '(Memento Audere Semper)) ritorna l'atomo NIL che, particolarità del LISP, oltre a indicare la lista vuota, rappresenta anche il valore di verità FALSO. È possibile poi distinguere gli atomi dalle liste tramite il predicato ATOM. Avremo quindi
(ATOM 'Napoleone) che dà True e
(ATOM '(Napoleone Bonaparte)) che dà NIL.
Infine è indispensabile il predicato EQ che confronta tra loro due atomi. (EQ 'Fabbri' Fabbri) produce True mentre
(EQ (Gruppo Editoriale Fabbri)' (Gruppo Editoriale Fabbri)) produce NIL, perché l'operatore EQ può confrontare soltanto atomi. Un altro fondamentale compito della struttura associativa è quello di gestire in modo abbastanza rudimentale, ma sufficientemente efficace, una base di dati con tutte le informazioni necessarie alle attività di programmazione. Il modo più semplice e generale per conservare e accedere a informazioni è sicuramente quello di attribuire loro un nome, che verrà usato per indicare le informazioni stesse. Nel contesto del LISP i simboli saranno atomi e gli oggetti che rappresentano le informazioni liste o atomi. Vediamo ora le primitive con cui il Lisp crea tali legami.
Per stabilire un legame tra un simbolo e un oggetto è disponibile la funzione SETQ. Per esempio
(SETQ Fiori '(ortensia gladiolo mimosa)).
Da questo momento in poi quando il sistema LISP si troverà di fronte l'atomo Fiori estrarrà automaticamente la lista associata (ortensia gladiolo mimosa). Quindi sottoponendo all'interprete il solo simbolo Fiori otterremo come risposta la lista associata. Inoltre se proponiamo (CAR Fiori) avremo ortensia, cioè il primo elemento della lista associata a Fiori. Questo permette di capire l'importanza del carattere "'" premesso a un oggetto LISP: soltanto quando un oggetto è preceduto da "'" viene considerato in quanto tale, altrimenti il sistema applica un procedimento che nel caso di un atomo consiste nel rintracciarne l'oggetto associato e nel caso di una lista di considerarla come un comando seguito da argomenti. Tutto ciò può essere ulteriormente chiarito considerando le due seguenti espressioni:
1) (ATOM 'Fiori) 2) (ATOM Fiori).
Nel caso 1) il sistema risponderà True (Fiori e un atomo) mentre nel caso 2) risponderà NIL (il valore associato a Fiori è una lista).
Spesso gli oggetti per i quali esistono informazioni da memorizzare possiedono particolari caratteristiche attraverso le quali è possibile raggruppare le informazioni. Per esempio il piano completo dell'opera 'Corso pratico col computer', messo sotto forma di lista, potrebbe essere associato all'ato-

CORSO PRATICO COL COMPUTER
Insieme delle proprietà

PROPRIETÀ	*VALORI DELLE PROPRIETÀ*
EDITORE	GRUPPO EDITORIALE FABBRI
FORMA EDITORIALE	CORSO A FASCICOLI
FREQUENZA	SETTIMANALE
COLLABORAZIONI	Olivetti, Banco di Roma
PREZZO A FASCICOLO	L. 2000

mo, Corso-pratico-col-computer che ne rappresenta il nome. Altre caratteristiche del corso sono state raggruppate nella tabella qui sopra. Tale tabella si può conservare, modificare e consultare, in modo estremamente naturale, utilizzando gli appositi comandi. La funzione PUT permette di associare un valore ad un atomo relativamente a una particolare caratteristica. Così per memorizzare la tabella sarà sufficiente sottoporre al sistema i seguenti comandi:
(PUT 'Corso-pratico-col-computer 'editore '(G. E. Fabbri)),
(PUT 'Corso-pratico-col-computer 'forma-editoriale '(fascicoli)),
(PUT 'Corso-pratico-col-computer 'frequenza '(settimanale)) e così via per le altre colonne. Potremo poi utilizzare la funzione GET per accedere a queste informazioni. Per esempio per ritrovare la frequenza di pubblicazione del corso è sufficiente sottoporre al sistema l'espressione:
(GET 'Corso-pratico-col-computer 'frequenza).
Data l'inflazione galoppante è possibile che dopo alcuni mesi il prezzo di copertina del corso subisca una variazione: per tenere conto della nuova situazione è sufficiente aggiornare con una nuova PUT la base di dati. Non ci si troverebbe impreparati neppure nell'eventualità che la pubblicazione diventasse gratuita. Per rimuovere l'informazione relativa al prezzo, si userà il comando:
(REM 'Corso-pratico-col-computer 'prezzo-a-fascicolo)
dove REM (REMOVE) rimuove la proprietà indicata.

Interprete funzionale. Il LISP è però soprattutto un linguaggio di programmazione. Caratteristica di ogni linguaggio di programmazione è la possibilità di definire strutture dati e programmi che le manipolano. I programmi sono usualmente, secondo una pratica comune, suddivisi ordinatamente in procedure. In LISP anche queste sono rappresentate con liste. È necessario quindi un modo per identificare una procedura. Questo avviene attraverso il simbolo LAMBDA. Vediamo, per esempio, come è definita la procedura che accede al secondo elemento di una lista: (LAMBDA (args) (CAR (CDR args))). Il secondo elemento di questa lista, (args), è la lista dei parametri della procedura, cioè dei nomi simbolici con i quali sono indicati nel seguito della definizione gli argomenti con i quali sarà utilizzata. Il resto della lista è detto corpo della definizione. In questo caso l'argomento dovrà essere soltanto uno. Per dare un nome alla nostra definizione possiamo ricorrere al comando SETQ. Sottoponendo all'interprete l'espressione:
(SETQ secondo '(LAMBDA (lista) (CAR (CDR lista)))),
avremo quindi associato all'atomo secondo la procedura descritta proprio come avremmo fatto con un qualunque altro valore.
Questa è una delle caratteristiche peculiari del LISP: la rappresentazione di definizioni di procedura tramite liste permette quella rappresentazione interna strutturale dei programmi di cui si parlava in un articolo precedente.

Procedura definitiva: vediamo ora come utilizzare la procedura definitiva. Ci si comporta come per un comando primitivo del tipo CAR, ATOM, EQ ecc. Quindi:
(secondo '(Qui Quo Qua)) ci restituirà Quo.
Aldilà delle considerazioni intuitive vediamo come avviene la produzione di questo risultato. Il sistema Lisp ricevendo una lista con un atomo come primo elemento, ne estrae il valore associato e, se si tratta della definizione di una funzione, applica tale definizione agli altri elementi della lista considerati come argomenti. Si può pensare che l'applicazione avvenga in questo modo: il sistema sostituisce nel corpo della definizione a tutte le occorrenze di lista, l'argomento. Nel nostro caso quindi si avrà: (secondo (Qui Quo Qua)) che si espande, attraverso la precedente definizione di secondo in (CAR (CDR (Qui Quo Qua)). Il sistema procederà poi considerando la lista così ottenuta come se fosse stata proposta dall'utente.

Lezione 35

MERGE tra due file ordinati: il modulo di MERGE

Riprendiamo la costruzione del programma di MERGE, che avevamo iniziato nella lezione precedente.
Completato il modulo che effettua il caricamento dei dati negli archivi di partenza, passiamo all'implementazione del modulo di MERGE.
Avevamo anche già tracciato lo schema del programma nel consueto linguaggio, che si ispira al Pascal.
Nella costruzione del programma seguiamo il metodo TOP DOWN: in questa filosofia stiamo effettuando i seguenti passi:
- siamo partiti dall'individuazione dei moduli che costituiscono il programma complessivo (caricamento degli archivi, MERGE, visualizzazione del file di output);
- abbiamo quindi costruito il programma BASIC che richiama tramite tre istruzioni GOSUB in cascata i relativi moduli;
- abbiamo tracciato lo schema del modulo di caricamento;
- sulla base di tale schema abbiamo costruito il relativo modulo BASIC.

Procediamo ora alla costruzione del modulo di MERGE, seguendo anche in questa fase un atteggiamento TOP DOWN.
Cominciamo dunque a riprodurre in BASIC la struttura dell'algoritmo, ricorrendo anche all'uso di commenti.

```
1000 ' Merge dei due files
1010 ' Apertura files
1030 OPEN "RAM:A" FOR INPUT AS #1
1040 OPEN "RAM:B" FOR INPUT AS #2
1050 OPEN "RAM:C" FOR OUTPUT AS #3
1055 ' Files disponibili
1057 ' Lettura primo record di A e di B
1060 INPUT #1,N$,T
1070 INPUT #2,N1$,T1
1075 ' Disponibili record di A e di B
1080 ' Execute until finea,fineb
1090 ' If N$=N1$ then
1170 ' Else
1180 ' If N$<N1$ then
1240 ' Else
1380 GOTO 1080
1390 ' Thencase
1400 ' When finea
1500 ' When fineb
1600 ' Endexecute
1800 CLOSE #1
1810 CLOSE #2
1820 CLOSE #3
```

Per la lettura dei dati dai due file useremo due coppie di variabili, che contengono rispettivamente il nominativo e il numero telefonico e che chiamiamo N$ e T per il file A e N1$ e T1 per il file B.

In questo modo abbiamo costruito la struttura portante dell'algoritmo, usando direttamente le istruzioni BASIC nei punti più semplici e ricorrendo invece a commenti nei punti più complessi: questi commenti fungono ora da promemoria delle

parti di programma ancora da sviluppare. Dobbiamo ora dare un contenuto BASIC a questi promemoria.
Cominciamo dalla parte che effettua i confronti tra i nominativi.

Si noti, come già visto in altri casi, che nella traduzione in BASIC, le condizioni che compaiono nelle istruzioni IF sono invertite.

```
1000 ' Merge dei due files
1010 ' Apertura files
1030 OPEN "RAM:A" FOR INPUT AS #1
1040 OPEN "RAM:B" FOR INPUT AS #2
1050 OPEN "RAM:C" FOR OUTPUT AS #3
1055 ' Files disponibili
1057 ' Lettura primo record di A e di B
1060 INPUT #1,N$,T
1070 INPUT #2,N1$,T1
1075 ' Disponibili record di A e di B
1080 ' Execute until finea,fineb
1090 ' If N$=N1$ then
1100 IF N$<>N1$ THEN 1170
1105 ' Event finea if eof(A)
1110 IF EOF(1) THEN 1400
1115 INPUT #1,N$,T
1120 PRINT #3,N1$;",";T1
1130 ' Event fineb if eof(B)
1140 IF EOF(2) THEN 1500
1150 INPUT #2,N1$,T1
1160 GOTO 1380
1170 ' Else
1180 ' If N$<N1$ then
1190 IF N$>N1$ THEN 1240
1200 PRINT #3,N$;",";T
1210 ' Event finea if eof(A)
1220 IF EOF(1)THEN 1400
1230 INPUT #1,N$,T
1235 GOTO 1380
1240 'Else
1250 PRINT #3,N1$;",";T1
1260 ' Event fineb if eof(B)
1270 IF EOF(2) THEN 1500
1280 INPUT #2,N1$,T1
1380 GOTO 1080
1390 ' Thencase
1400 ' When finea
1500 ' When fineb
1600 ' Endexecute
1800 CLOSE #1
1810 CLOSE #2
1820 CLOSE #3
1900 RETURN
```

Nella traduzione dal modello dell'algoritmo al programma BASIC dobbiamo soprattutto fare attenzione a rispettare le caratteristiche specifiche del linguaggio di

programmazione, che a volte impone qualche piccolo cambiamento rispetto allo schema di partenza: così per esempio nel caso della funzione EOF il BASIC ci impone una verifica dello stato del file prima di ogni lettura.
Osserviamo l'uso dell'istruzione INPUT, che avevamo già visto precedentemente per la lettura di dati da file piuttosto che da tastiera: in questo caso le informazioni da leggere sono due, l'una alfanumerica corrispondente al nominativo e l'altra numerica corrispondente al numero telefonico: abbiamo quindi usato due variabili, una di tipo stringa e l'altra numerica.
Non è ora difficile completare il modulo con le istruzioni che realizzano il trattamento degli eventi di esaurimento dei due file:

```
1400 ' When finea
1410 ' Record di B da scrivere
1420 PRINT #3,N1$;",";T1
1430 ' While not eof(B) do
1440 IF EOF(2) THEN 1470
1445 INPUT #2,N1$,T1
1450 PRINT #3,N1$;",";T1
1460 GOTO 1430
1470 ' Endwhile
1480 GOTO 1600
1500 ' When fineb
1510 ' Record di A da scrivere
1520 PRINT #3,N$;",";T
1530 ' While not eof(A) do
1540 IF EOF(1) THEN 1590
1550 INPUT #1,N$,T
1555 PRINT #3,N$;",";T
1560 GOTO 1530
1590 ' Endwhile
1600 ' Endexecute
```

Una prima esecuzione

Giunti a questo punto dello sviluppo del programma siamo già in grado di effettuarne un'esecuzione parziale, verificando il corretto funzionamento dei due moduli già costruiti. Per evitare comunque che il richiamo del modulo non ancora disponibile porti a un'indicazione d'errore, inseriamo anche la prima e l'ultima istruzione del modulo di visualizzazione:

```
2000 ' Visualizzazione archivio di output
2500 RETURN
```

A questo punto il programma può essere eseguito. Se alla richiesta di caricamento dei due archivi forniamo le seguenti informazioni nel primo caso:

```
          GENERAZIONE PRIMO ARCHIVIO
Nominativo? BELLI
Numero telefonico? 8409807
```

```
Nominativo? BIANCHI
Numero telefonico? 215432
Nominativo? CITTI
Numero telefonico? 453400
Nominativo? STOP
```

e le seguenti per il secondo:

```
          GENERAZIONE SECONDO ARCHIVIO
Nominativo? ASTUTI
Numero telefonico? 122343
Nominativo? BIANCHI
Numero telefonico? 908765
Nominativo? STOP
```

al termine dell'esecuzione potremo avere prova della correttezza dei due moduli verificando il contenuto rispettivamente dei due file di partenza e di quello usato per il MERGE. Torniamo al menù e troveremo infatti tre nuovi file, di nome rispettivamente A.DO, B.DO e C.DO. Se spostiamo il cursore sul nome del file da verificare e premiamo il tasto ENTER, ci sarà visualizzato il contenuto del file relativo. Proviamo a verificarlo nel caso del file A e leggeremo le seguenti informazioni:

```
BELLI, 8409807
BIANCHI, 215432
CITTI, 453400
```

Notiamo in particolare la presenza della virgola tra il nominativo e il numero telefonico, che avevamo inserito per consentire all'istruzione INPUT di distinguere le due informazioni. Visualizziamo ora il contenuto del file C:

```
ASTUTI, 122343
BELLI, 8409807
BIANCHI, 908765
CITTI, 453400
```

Come avevamo deciso, nel caso del nominativo Bianchi, che era presente in entrambi i file è stata riportata l'informazione presente nell'archivio B come mostra il numero telefonico associato. La verifica del programma sarà ora facilitata dalla realizzazione del modulo di stampa, che visualizza il contenuto del file prodotto dal MERGE.

Cosa abbiamo imparato

In questa lezione abbiamo visto:

- come si costruisce un programma di MERGE tra due file un procedimento TOP DOWN.

LA CORRETTEZZA DEI PROGRAMMI

Garantire che un programma funzioni in modo corretto è un problema di non facile soluzione.

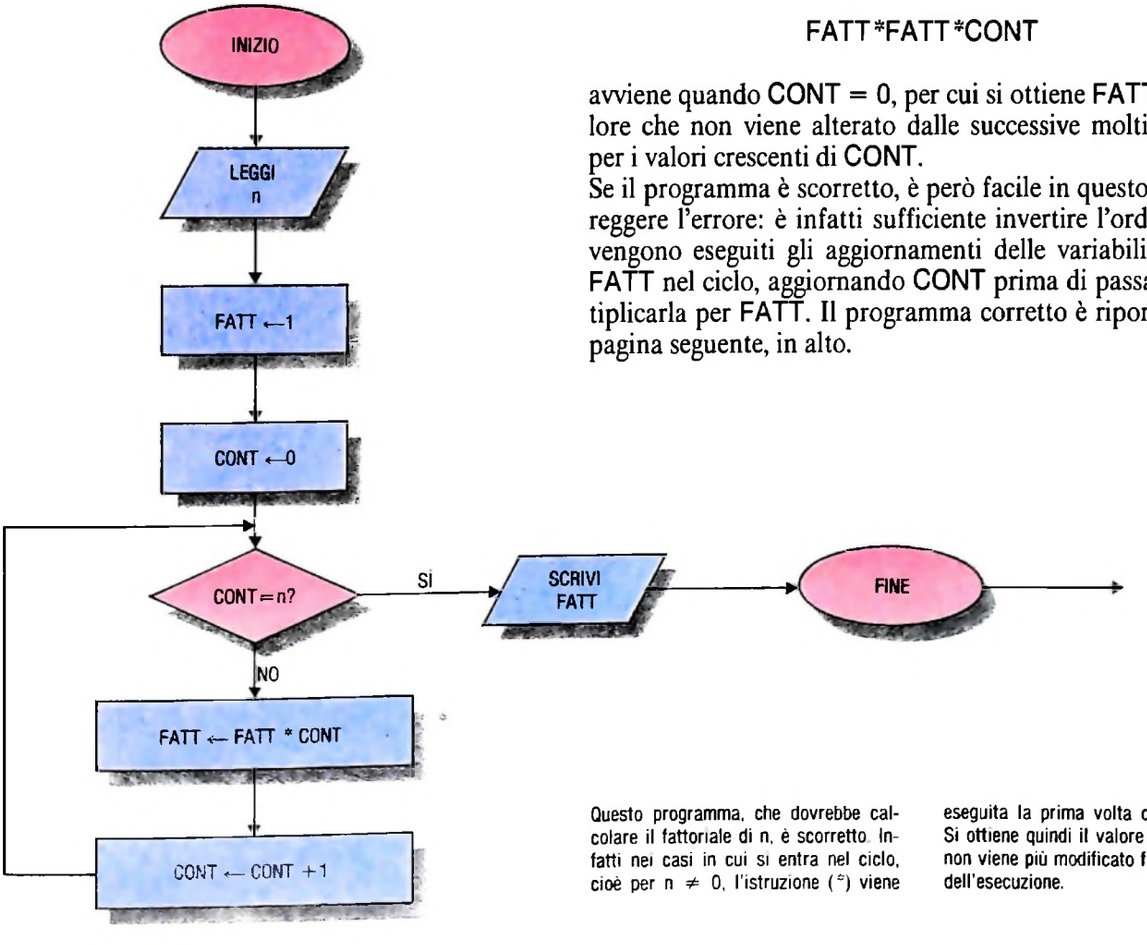
Un programma scorretto

Il programma rappresentato qui sotto dovrebbe calcolare il fattoriale del numero n, cioè il prodotto dei primi n numeri interi positivi, n! = 1*2*3*...* (n-1)*n (n! si legge "n fattoriale"; per convenzione, si pone 0! = 1).

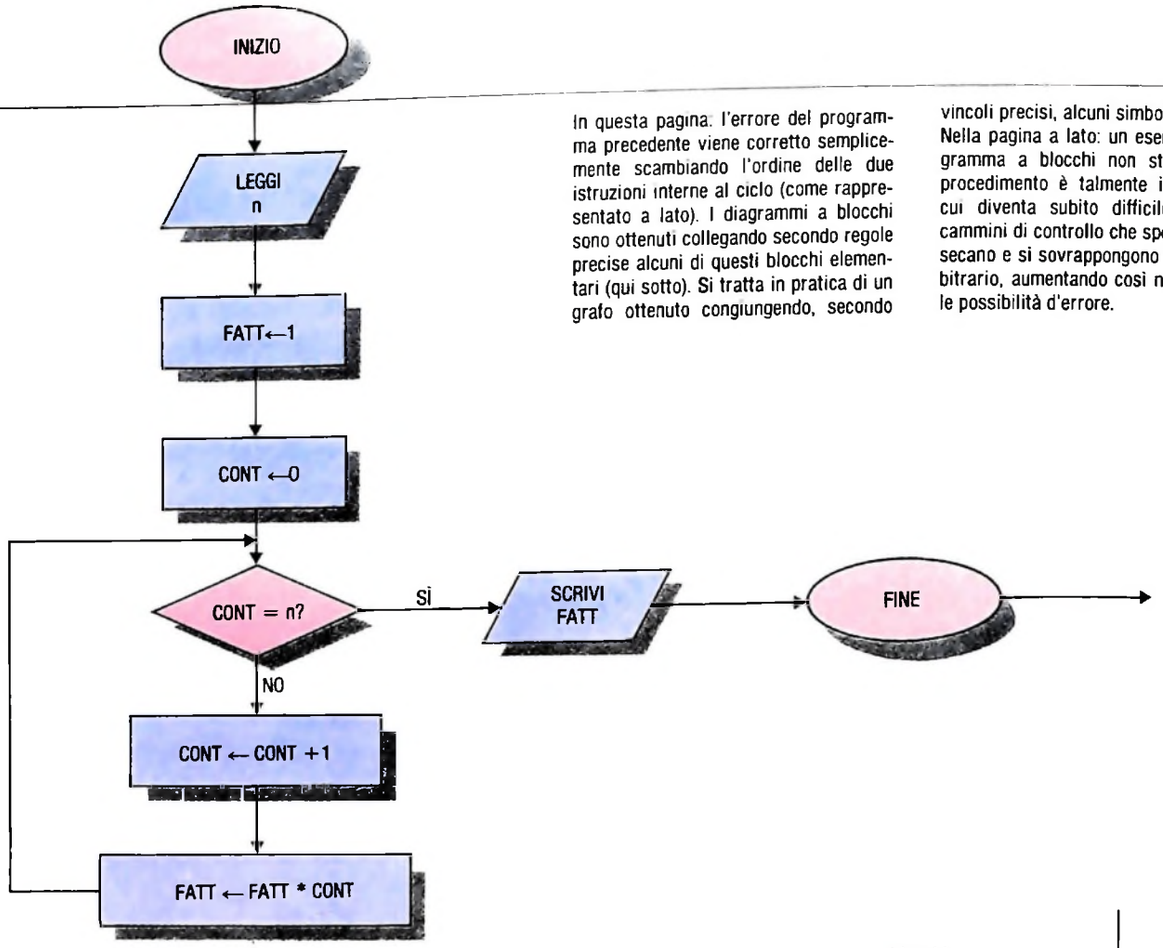
Il programma consta essenzialmente di un ciclo, controllato dal contatore CONT, da cui si esce dopo n iterazioni, cioè quando CONT, che parte da 0 e viene incrementato di 1 ad ogni iterazione, raggiunge il valore n. All'interno del ciclo, oltre all'incremento della variabile di controllo CONT, abbiamo l'aggiornamento della variabile FATT, che inizialmente ha il valore 1 e viene poi moltiplicata per i valori successivamente assunti da CONT.
In realtà, il programma funziona correttamente solo per n = 0, per cui dà il valore 0! = 1. Per tutti gli altri valori, invece, si ottiene in uscita il valore 0. Infatti, la prima esecuzione dell'istruzione

FATT*FATT*CONT

avviene quando CONT = 0, per cui si ottiene FATT = 0, valore che non viene alterato dalle successive moltiplicazioni per i valori crescenti di CONT.
Se il programma è scorretto, è però facile in questo caso correggere l'errore: è infatti sufficiente invertire l'ordine in cui vengono eseguiti gli aggiornamenti delle variabili CONT e FATT nel ciclo, aggiornando CONT prima di passare a moltiplicarla per FATT. Il programma corretto è riportato nella pagina seguente, in alto.



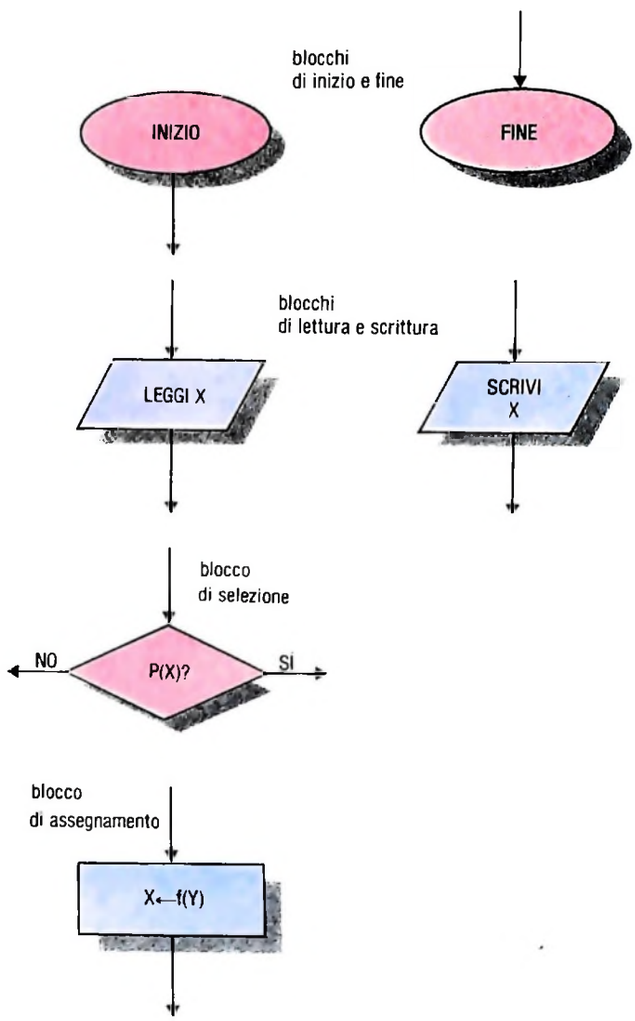
Questo programma, che dovrebbe calcolare il fattoriale di n, è scorretto. Infatti nei casi in cui si entra nel ciclo, cioè per n ≠ 0, l'istruzione (*) viene eseguita la prima volta con CONT = 0. Si ottiene quindi il valore FATT = 0 che non viene più modificato fino al termine dell'esecuzione.

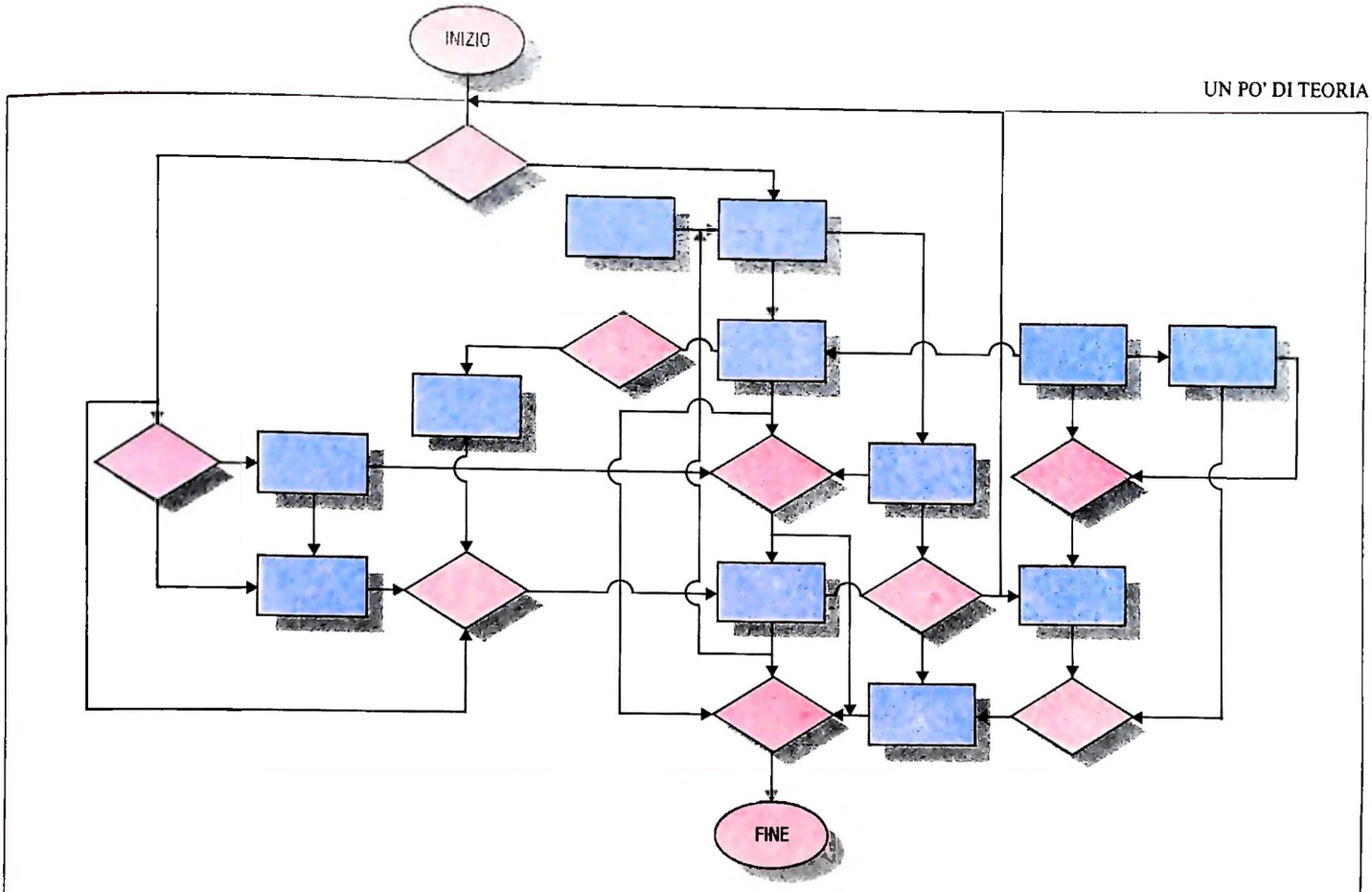


In questa pagina: l'errore del programma precedente viene corretto semplicemente scambiando l'ordine delle due istruzioni interne al ciclo (come rappresentato a lato). I diagrammi a blocchi sono ottenuti collegando secondo regole precise alcuni di questi blocchi elementari (qui sotto). Si tratta in pratica di un grafo ottenuto congiungendo, secondo vincoli precisi, alcuni simboli grafici. Nella pagina a lato: un esempio di diagramma a blocchi non strutturato. Il procedimento è talmente intricato per cui diventa subito difficile seguire i cammini di controllo che spesso s'intersecano e si sovrappongono in modo arbitrario, aumentando così notevolmente le possibilità d'errore.

Il problema della correttezza

Errori anche banali come quello visto sono la norma piuttosto che l'eccezione nell' attività di programmazione, nel senso che la prima versione di un programma anche non eccessivamente complesso contiene quasi sempre diversi errori di vario genere. I più frequenti sono gli errori logici nella struttura di controllo del programma, per esempio nella corretta sequenza delle istruzioni (come quello presente nel nostro esempio), nelle condizioni di uscita dai cicli, oppure la mancata copertura di tutte le possibilità in una o più istruzioni di selezione nidificate e così via. Altri errori frequenti sono gli errori di computazione (singole istruzioni sbagliate, errori di segno, ...) e gli errori di ingresso/uscita e di gestione dei dati, come nei casi in cui un dato o un risultato parziale viene letto o scritto in memoria ad un indirizzo sbagliato.
Di qualunque tipo siano gli errori, il risultato finale è che il programma non fa esattamente ciò per cui è stato scritto, cioè, almeno per alcuni dati di ingresso, dà risultati diversi da quelli desiderati. Si osservi che questo non significa che il programma non funzioni: non è raro il caso in cui in programmi complessi come i sistemi operativi si scoprano errori dopo un lungo periodo di funzionamento corretto, semplicemente perché i dati per cui il programma è scorretto sono così particolari che si presentano molto raramente. Questo rende più subdolo il pericolo che si cela dietro la scorrettezza dei programmi: si pensi a cosa succederebbe se un programma di controllo dei sistemi di sicurezza di una centrale nucleare, perfettamente funzionante in situazioni normali, rea-





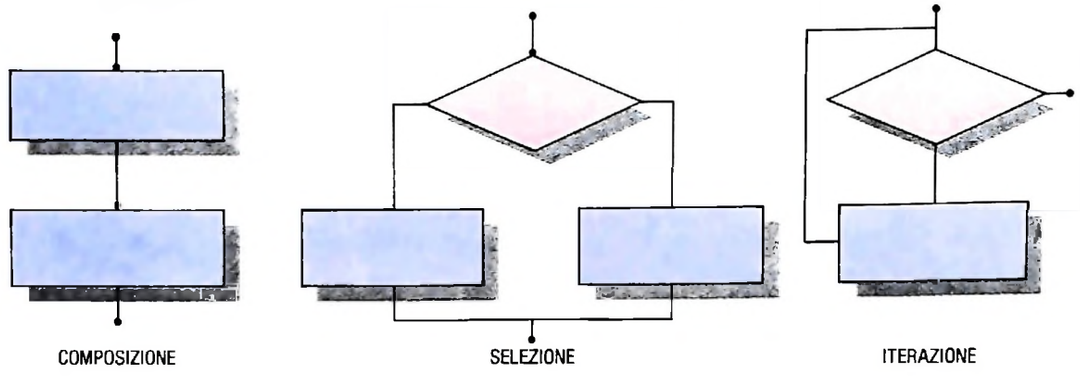
gisse in modo scorretto proprio nell'eventualità, estremamente improbabile, di un'emergenza effettiva!
Quello che ci interessa è quindi garantire la correttezza dei programmi. In modo più preciso, diremo che:
un programma P è corretto se per ogni dato d'ingresso x fornisce in uscita il valore desiderato f (x).
A prima vista, questo potrebbe sembrare un compito abbastanza facile, rispetto per esempio alla stesura dell'algoritmo di soluzione e alla sua codifica in un linguaggio di programmazione. Che le cose non siano così semplici è facilmente intuibile se si pensa che l'ultima missione dello Space Shuttle è stata annullata per la scoperta di un errore che si nascondeva tra i circa quaranta milioni di istruzioni dei programmi di controllo del volo. Non sembrerà allora strano che circa la metà del costo complessivo di produzione di un programma sia attribuibile alle attività di controllo della correttezza (verifica) contro solo il 15% per il progetto dell'algoritmo e il 10% per la codifica.
Le dimensioni anche economiche del problema hanno quindi stimolato la ricerca di tecniche e strumenti per garantire la correttezza dei programmi.

La sintesi dei programmi

Il problema della correttezza dei programmi può essere affrontato a due diversi livelli:
— individuare tecniche che consentano di scrivere programmi sicuramente corretti; oppure:
— individuare tecniche per scoprire e correggere gli errori in programmi già scritti.
Ovviamente, l'ideale sarebbe avere la possibilità di sviluppare in modo automatico, a partire da una formulazione rigorosa del problema da risolvere, per esempio attraverso una formula logica che esprime in modo preciso la relazione esistente tra dati e risultati, un programma che risolva correttamente il problema formulato. Si parla in questo caso di sintesi dei programmi.
Benché da tempo siano in corso ricerche sulla sintesi automatica dei programmi, i risultati sono finora scarsi, soprattutto per quanto riguarda programmi di grosse dimensioni, e restano ancora aperti problemi non indifferenti.

Le metodologie di programmazione

In assenza di una teoria della sintesi dei programmi che dia risultati apprezzabili in pratica, è possibile affrontare il problema in termini più pragmatici. Sono così nate le metodologie di programmazione, cui possiamo accennare brevemente, anche se non rientrano a rigore nell'informatica teorica.
Una metodologia di programmazione è un insieme di criteri e strumenti che aiutano a costruire un programma procedendo in modo sistematico, così da ridurre al minimo la possibilità di commettere errori logici.
La più semplice e più nota metodologia di programmazione è la programmazione strutturata, che vincola all'uso "disciplinato" delle strutture di controllo.



Le tre strutture fondamentali della programmazione strutturata. Ogni struttura ha un solo punto d'ingresso e un solo punto d'uscita; inoltre contiene blocchi che a loro volta possono essere diagrammi strutturati.

Abbiamo fin qui utilizzato, per rappresentare i programmi, i diagrammi a blocchi, strumento grafico di facile comprensione di cui diamo ora una definizione precisa.
Un diagramma a blocchi è un grafo ottenuto collegando tra loro un certo numero di simboli grafici scelti tra quelli rappresentati nella figura a pagina 566, in basso, con i seguenti vincoli:
— c'è un solo simbolo di inizio;
— ogni freccia che esce da un simbolo è collegata a una e una sola freccia che entra in un altro simbolo;
— ogni simbolo può essere raggiunto dal simbolo di inizio seguendo le frecce;
— da ogni simbolo si può raggiungere un simbolo di fine;
— è opportuno che vi sia un solo simbolo di fine e che i simboli di lettura dei dati di ingresso e di scrittura dei risultati vengano collocati rispettivamente subito dopo l'inizio e subito prima della fine.
Ogni diagramma a blocchi D rappresenta un programma che computa una funzione f_D. Partendo dal simbolo di inizio, si seguono le frecce eseguendo le istruzioni man mano incontrate. In particolare, quando s'incontra un simbolo di selezione si esce seguendo la freccia "sì" se il predicato relativo è vero con i valori correnti delle variabili, la freccia "no" altrimenti. L'esecuzione termina se si raggiunge il simbolo di fine. In tale caso, il valore della funzione $f_D(x)$ è il valore y stampato.
Questa definizione consente di costruire diagrammi a blocchi con connessioni estremamente libere e poco ordinate, che diventano difficili da seguire, poiché s'intrecciano in modo arbitrario, come nella figura a pagina precedente. Questo corrisponde, nei linguaggi di programmazione, all'uso libero delle istruzioni di salto incondizionato "goto".
Per rendere comprensibili i programmi rappresentati in questa forma, e quindi facilitare la correzione e ridurre le probabilità di errore, è necessario imporre vincoli più forti sulla struttura dei diagrammi a blocchi, prendendo in considerazione solo diagrammi ottenuti da un insieme ridotto di "strutture di controllo" fondamentali.
Nella struttura dei programmi si possono identificare tre di queste "strutture di controllo". La prima è l'esecuzione di istruzioni in sequenza. La seconda è la selezione dell'istruzione da eseguire sulla base di un test sui valori correnti delle variabili. La terza è infine la ripetizione o iterazione di un insieme di istruzioni per un certo numero di volte.
Graficamente, si hanno gli schemi rappresentati nella figura in alto, ove ogni rettangolo rappresenta un blocco che potrebbe a sua volta essere un diagramma complesso. Si osservi che ogni schema ha un solo punto di ingresso e un solo punto di uscita: questo impedisce rinvii incontrollati da un punto del programma a un altro, rendendo il programma stesso più leggibile.
Si pone a questo punto un problema teorico non indifferente, che è quello della potenza computazionale di queste strutture di controllo: il linguaggio dei diagrammi a blocchi con l'uso esclusivo delle tre regole di composizione tra blocchi della figura in alto consente di esprimere tutte le funzioni computabili, come il linguaggio dei diagrammi a blocchi con la possibilità di rinvii non controllati, oppure no? Una risposta affermativa a questa domanda è stata data da C. Bohm e G. Jacopini nel 1966, con la dimostrazione del seguente:

TEOREMA

Dato un qualsiasi diagramma a blocchi D che computa la funzione f_D esiste un diagramma a blocchi costruito utilizzando esclusivamente le strutture della figura in alto che computa la stessa funzione.

Su questo risultato teorico, trasferito nei linguaggi di programmazione più recenti (Pascal, ADA, ...), che dispongono di strutture di controllo che impongono una precisa disciplina di programmazione, poggia la possibilità di scrivere programmi attraverso raffinamenti successivi. In sostanza, si tratta di decomporre il problema iniziale in sottoproblemi più semplici, e di decomporre questi a loro volta, e così via, fino a ottenere sottoproblemi di dimensione tale da essere facilmente comprensibili e "maneggiabili". Questo ha ovvi riflessi sulla correttezza, poiché a ogni livello si tratta di garantire che sono stati risolti correttamente i sottoproblemi e che inoltre è corretto il modo in cui vengono messi insieme per risolvere il problema di livello superiore.

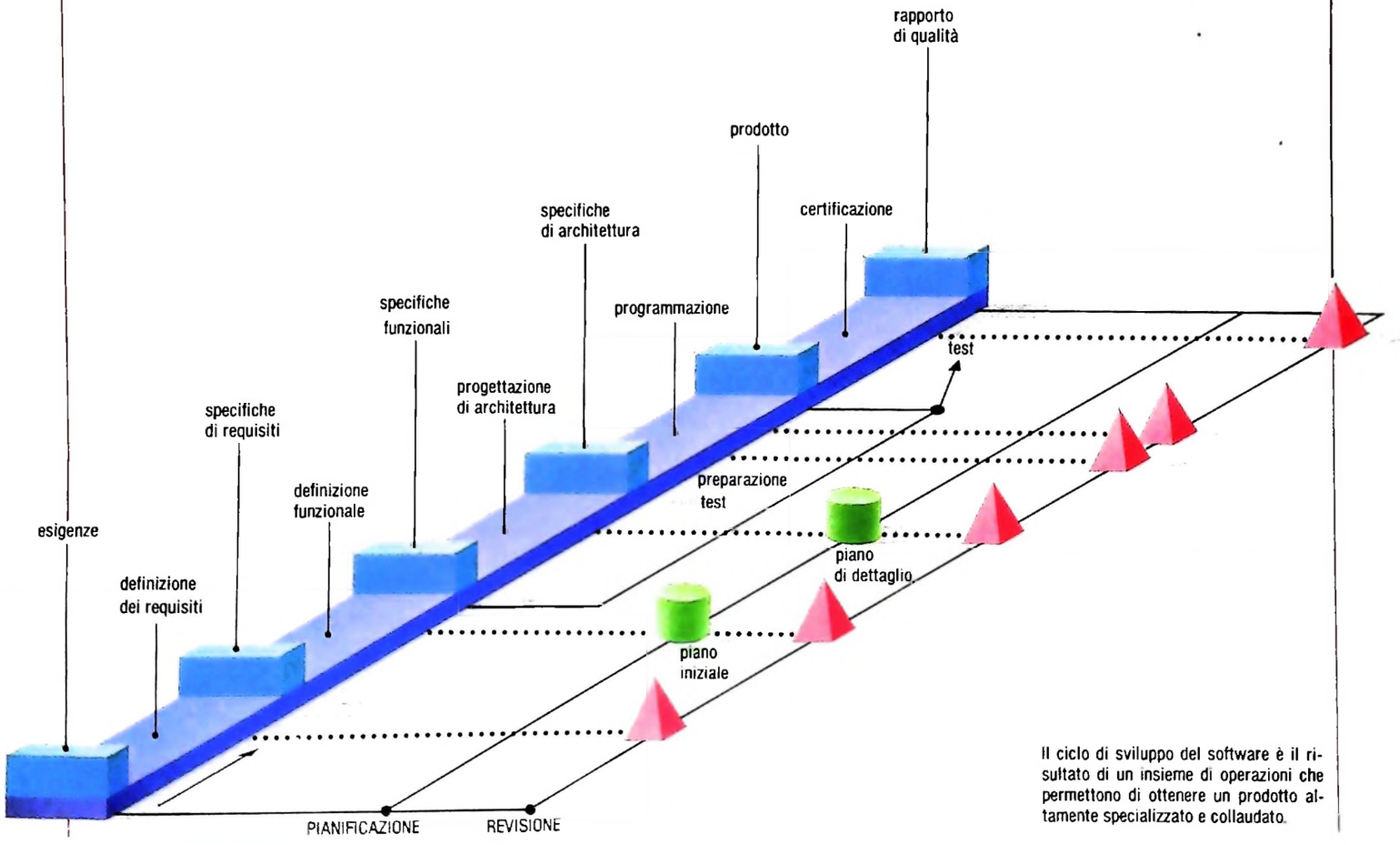
L'INGEGNERIA DEL SOFTWARE

Come si produce software di qualità, a costi contenuti e in tempi controllati.

Intorno alla fine degli anni Sessanta i crescenti costi dello sviluppo di programmi, la loro sempre maggiore complessità, l'esperienza di colossali fallimenti nella realizzazione di sistemi hanno provocato uno specifico interesse per le problematiche relative alla produzione di software di qualità, a costi contenuti, realizzato in tempi controllati. Da tale interesse è nato intorno al 1968 il termine di "ingegneria del software", che si è via via riempito di contenuti, fino allo stato attuale, in cui, pur lontani da un completo consolidamento della disciplina, sono ormai stati raggiunti un certo numero di "punti fermi".
Una delle prime considerazioni emerse dalla nuova sensibilità ottenuta sulla produzione di software è stata la rilevanza dell'aspetto organizzativo nella produzione: sviluppare grandi programmi vuole dire sviluppare complessi prodotti in cui sono coinvolte contemporaneamente molte persone; quindi è fondamentale avere una precisa organizzazione del lavoro, in cui siano presenti fasi operative, ruoli, prodotti intermedi.

Ciclo di sviluppo

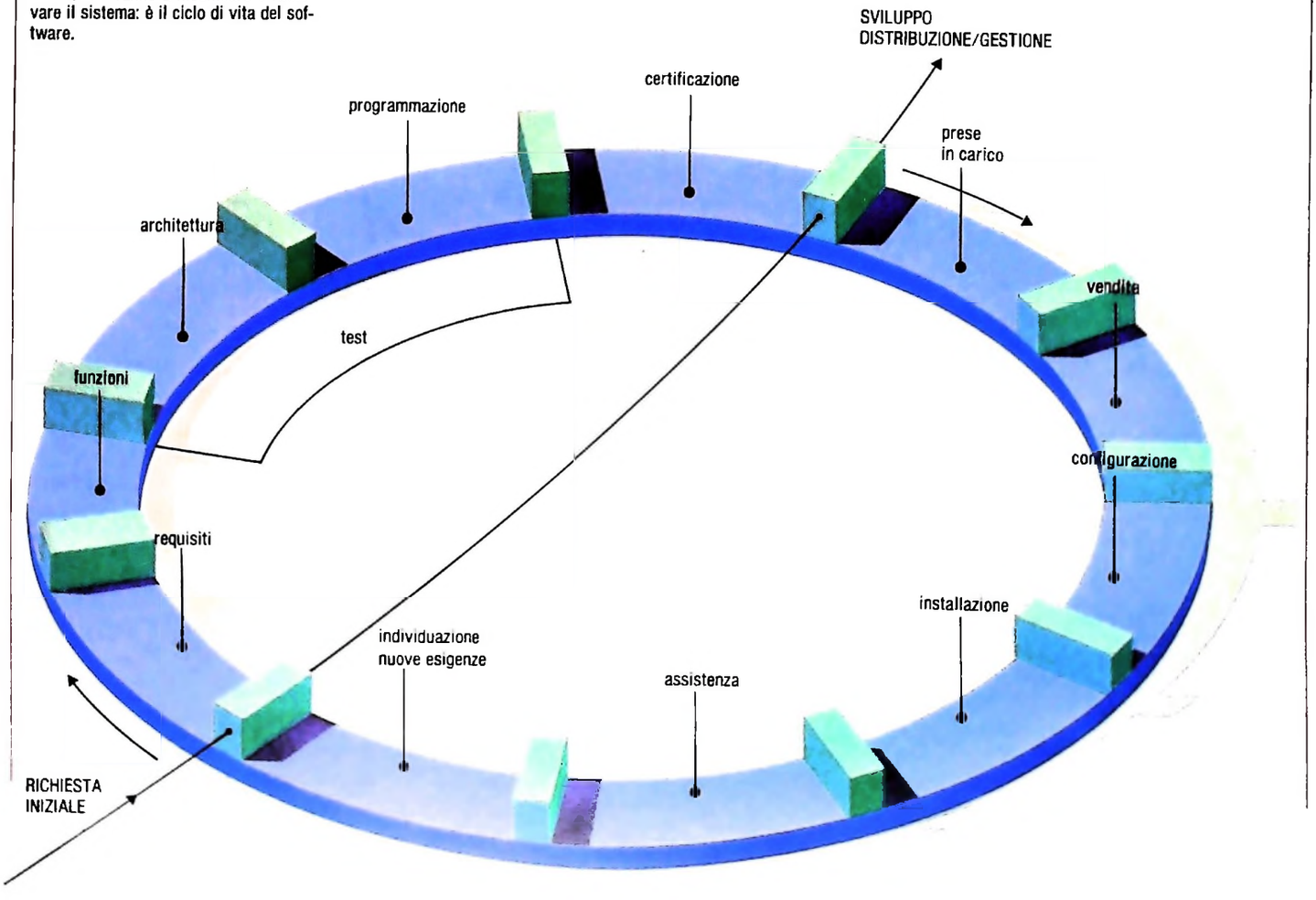
Si è così cominciato a parlare di "ciclo di sviluppo del software", legando questo termine a uno schema di lavoro come quello rappresentato nella figura in basso.
Si possono individuare nello schema sequenze di fasi di lavo-



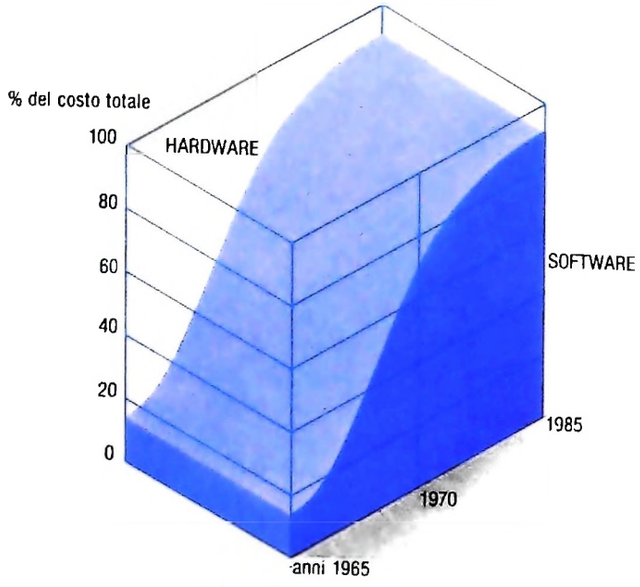
Il ciclo di sviluppo del software è il risultato di un insieme di operazioni che permettono di ottenere un prodotto altamente specializzato e collaudato.

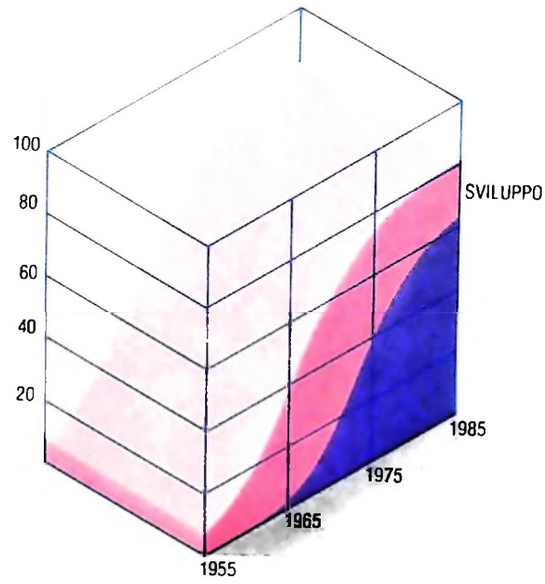
ro che permettono di ottenere il prodotto finale, e un certo insieme di attività di gestione, che ne garantiscano il corretto svolgimento.
A partire da un certo insieme di indicazioni che il mercato evidenzia, un'attività di "definizione dei requisiti" permette di ottenere un documento di specifiche dei requisiti. Quest'attività ha il compito di mettere bene a fuoco il problema che deve essere risolto e che era stato solamente percepito, in modo da definire con precisione lo scopo che dovrà avere un certo prodotto software. Se per esempio c'è la percezione di opportunità di sviluppare un programma di ausilio al lavoro del medico, sarà compito di questa fase la definizione precisa del tipo di utente, se ci si dovrà occupare di cartelle cliniche o di analisi automatica di elettrocardiogrammi, se il prodotto dovrà occuparsi o no di effettuare calcoli statistici, o se si dovrà fornire qualche tipo di contabilità, e così via.
Questa fase si occupa quindi del *perché* noi vogliamo sviluppare un prodotto.
Una volta ottenute le specifiche di requisiti, ha inizio una successiva fase di lavoro, per la "definizione funzionale" del prodotto, che porterà alla costruzione di un documento di specifiche funzionali. Si tratta di un documento che descrive con dettaglio tutte le operazioni che l'utente finale del prodotto potrà effettuare, indicando con molta precisione anche tutte le modalità di interazione, i comandi, la forma in cui dovranno essere fornite le informazioni, il formato con cui verranno ottenuti i risultati, gli eventuali messaggi d'errore a fronte di qualche operazione scorretta da parte dell'utente, e così via. Questa fase ha quindi il compito di specificare *cosa* vogliamo che venga costruito.
Dalle specifiche funzionali si parte per definire l'architettura del prodotto, con quell'attività che chiamiamo "progettazione di architettura". Si tratta cioè di definire la struttura del programma da sviluppare, in termini di moduli, sottoprogrammi, struttura delle aree dati e dei file, modalità di comunicazione tra i vari moduli, e così via. In questa fase definiamo quindi *come* fare il prodotto.
Le specifiche di architettura sono ora il punto di partenza per la costruzione dei vari programmi e sottoprogrammi nel linguaggio di programmazione; il risultato visibile dell'attività di programmazione è costituito quindi dalle varie liste di compilazione dei moduli scritti.
A questo punto è necessario procedere alla fase di "certificazione" del prodotto, cioè a un'attività che verifichi che il programma così come è risultato dall'attività di sviluppo è corretto rispetto a quanto indicato nelle specifiche funzionali. L'unico modo per effettuare tale verifica è quello di eseguire il prodotto con un certo insieme di dati-campione per i quali sono precedentemente noti i risultati, si controlla così che effettivamente tali risultati corrispondano a quelli forniti dall'esecuzione. Il risultato di questa attività sarà un rapporto di qualità che indica quanti controlli sono stati fatti, quali i re-

Dalla richiesta iniziale alla gestione del software, fase in cui nascono esigenze nuove, che vanno a modificare o rinnovare il sistema: è il ciclo di vita del software.



Investimenti nell' hardware e nel software, e ripartizioni dei costi



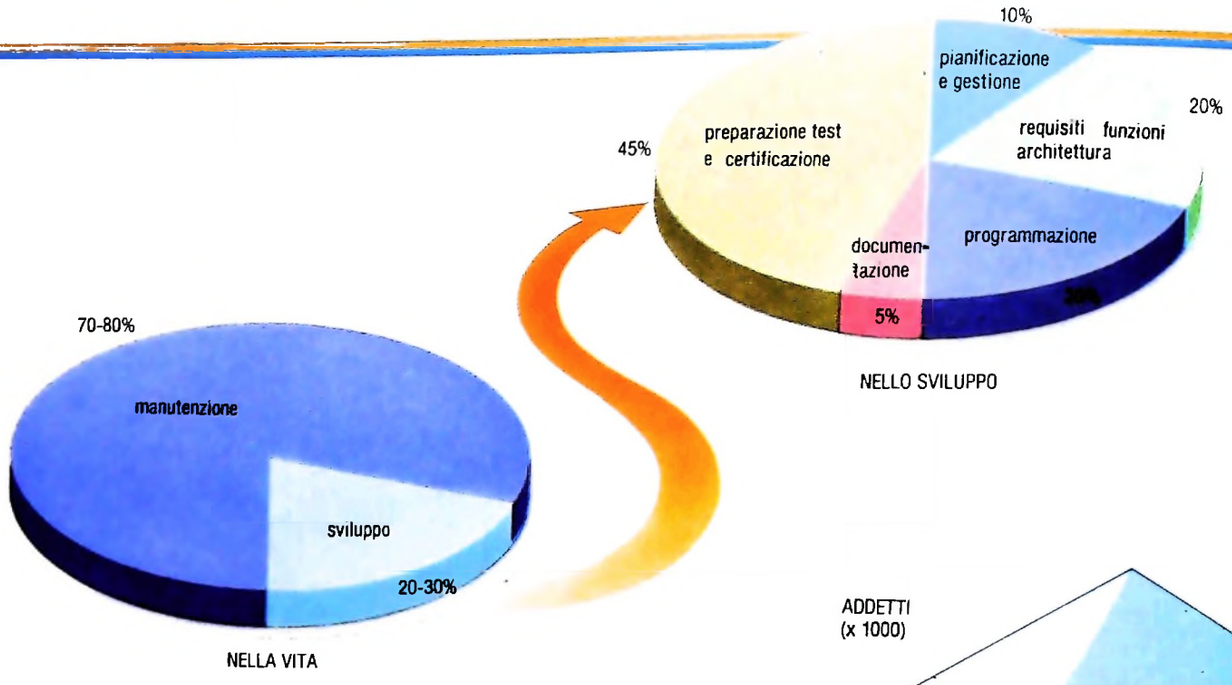


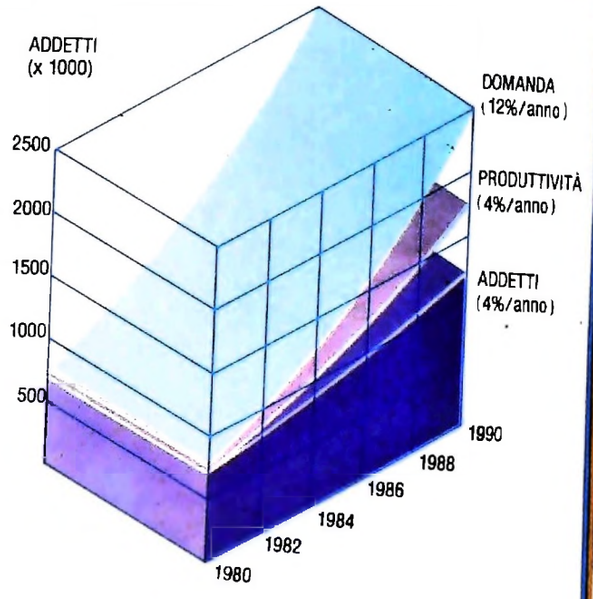
Nel corso degli ultimi vent'anni la diminuzione dei costi delle apparecchiature hardware e la crescente complessità dei sistemi software hanno determinato un'inversione nella linea di tendenza degli investimenti relativi: mentre a metà degli anni Sessanta gli investimenti in software erano meno del 20% del totale, attualmente questi coprono circa l'80% degli investimenti nell'elaborazione dell'informazione (figura sopra).
Di più, osservando l'andamento degli investimenti per il solo software, osserviamo come, mentre a metà degli anni Sessanta la maggior parte degli sforzi era legata a nuovi prodotti, oggi il costo della manutenzione arriva a coprire dal 60 all'80% degli investimenti globali nel software.
Una ripartizione indicativa dei costi nella produzione di software è riportata dagli schemi della pagina seguente, in alto.
Da questa si vede come la fase di preparazione delle prove e di certificazione porti via quasi la metà dei costi di sviluppo, mentre la codifica dei programmi, che potrebbe sembrare una parte rilevante, ha in realtà costi relativamente contenuti.

lativi esiti, quali errori sono stati rilevati, e così via. Tuttavia la certificazione del prodotto non contempla la messa a punto di tutti i dati di prova, cioè di quelli che vengono chiamati i "test" del prodotto, che devono essere già disponibili prima che essa abbia inizio; come si vede dallo schema del ciclo di sviluppo, un'attività di preparazione dei test è sviluppata parallelamente alle fasi di realizzazione del prodotto.
La fase di "preparazione dei test" ha il compito di definire tutte le prove che devono essere effettuate e di costruire i test relativi; poiché l'obiettivo è quello di verificare che il prodotto si comporti esattamente come era richiesto dalle specifiche funzionali, quest'attività ha inizio proprio nel momento in cui tale documento è disponibile. Il fatto di svolgere questa fase parallelamente alle altre permette di anticipare la certificazione, che può iniziare quando prodotto e test sono disponibili.
Accanto alle attività viste, che sono legate allo sviluppo vero e proprio del prodotto, ne esistono altre con un contenuto più legato ad aspetti organizzativi e gestionali, che sono quelle di "pianificazione" e di "revisione". La prima si occupa di definire tempi e risorse necessarie per sviluppare il prodotto, indicando a grandi linee prima (piano iniziale) e con dettaglio poi (piano di dettaglio) le date in cui si prevede il completamento delle singole attività. La seconda verifica invece periodicamente che i vari prodotti intermedi (documenti, liste ecc) dell'attività siano adeguati nella qualità e nei contenuti, e rispettino le scadenze previste.

Il ciclo di vita del software

In realtà lo sviluppo del software è solo una delle attività industriali che sono visibili in un'azienda del settore; infatti dopo che il prodotto è stato completato e dichiarato corretto





I due grafici a torta in alto presentano la ripartizione dei costi nella produzione di software. Il diagramma evidenzia invece il rapporto fra domanda di software e produttività; in base ai dati dei primi anni Ottanta è prevedibile un ulteriore richiesta che, pur incrementando la produttività, sarà difficile evadere.

Proprio a causa dei crescenti investimenti nel software si è sviluppata la disciplina dell'"ingegneria del software", tendente a migliorare il rapporto costo/qualità di tali prodotti; inoltre un aumento della produttività pro capite nello sviluppo di software è d'importanza capitale per la società: studi condotti dal Ministero della Difesa degli Stati Uniti mostrano come la domanda da parte del mercato di sviluppo di software cresce ogni anno del 12% (figura a lato), a fronte di una crescita del 4% solamente degli addetti a tale tipo di produzione; anche con una crescita di produttività del 4% all'anno (prevista a fronte di investimenti che verranno fatti negli Stati Uniti per importi di diversi miliardi di dollari) non si riuscirà a soddisfare la domanda.

e distribuibile, inizia una vera e propria attività di gestione del prodotto, di commercializzazione, di personalizzazione, che sfocia spesso in una individuazione di nuove esigenze, che portano a un'estensione del prodotto iniziale. Questa attività di estensione e di modifica (perché no, anche di correzione di eventuali errori emersi durante l'uso) è indicata spesso con il termine di "manutenzione", anche se si tratta di un'attività che presenta le stesse caratteristiche di un vero e proprio piano di sviluppo: infatti un programma non è un'apparecchiatura che si può usurare o guastare (come un apparato meccanico per cui il termine "manutenzione" ha una precisa connotazione), ma può solamente evidenziare malfunzionamenti non individuati precedentemente, o richiedere nuove funzioni. Ne risulta un "ciclo di vita" del prodotto che ha le caratteristiche di quello illustrato a pagina 570, che mostra un andamento ciclico in cui un evento iniziale esterno causa lo sviluppo iniziale di un prodotto; questo rilasciato dallo sviluppo, passa nella fase di "distribuzione/gestione", che prevede:

- la presa in carico del prodotto rilasciato da parte di un adeguato ufficio;
- la vendita del prodotto;
- la configurazione del prodotto venduto (in genere un programma non è realizzato come insieme rigido di funzioni, ma può essere venduto con adeguate personalizzazioni; per esempio con alcune funzioni incluse e altre eliminate perché non interessano il cliente, o con aree di memoria più o meno estese a seconda del volume di lavoro richiesto dal cliente, e così via; in questo senso parliamo di "configurazione");
- installazione del prodotto presso il cliente;
- assistenza all'uso;
- individuazione di nuove richieste.

In seguito esamineremo con dettaglio le caratteristiche delle singole fasi di sviluppo.

UN NUOVO MODO DI USARE LA BANCA.

Conto corrente più

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

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

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

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

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

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

Veniteci a trovare, ci conosceremo meglio.

BANCO DI ROMA
CONOSCIAMOCI MEGLIO.

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

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

PERSONAL COMPUTER OLIVETTI M10

L'UFFICIO DA VIAGGIO

Anche in leasing con Olivetti Leasing.

Personal Computer, Via Meravigli 12.

NOME COGNOME

CAP CITTÀ

TELEFONO

olivetti