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

38 CORSO PRATICO COL COMPUTER

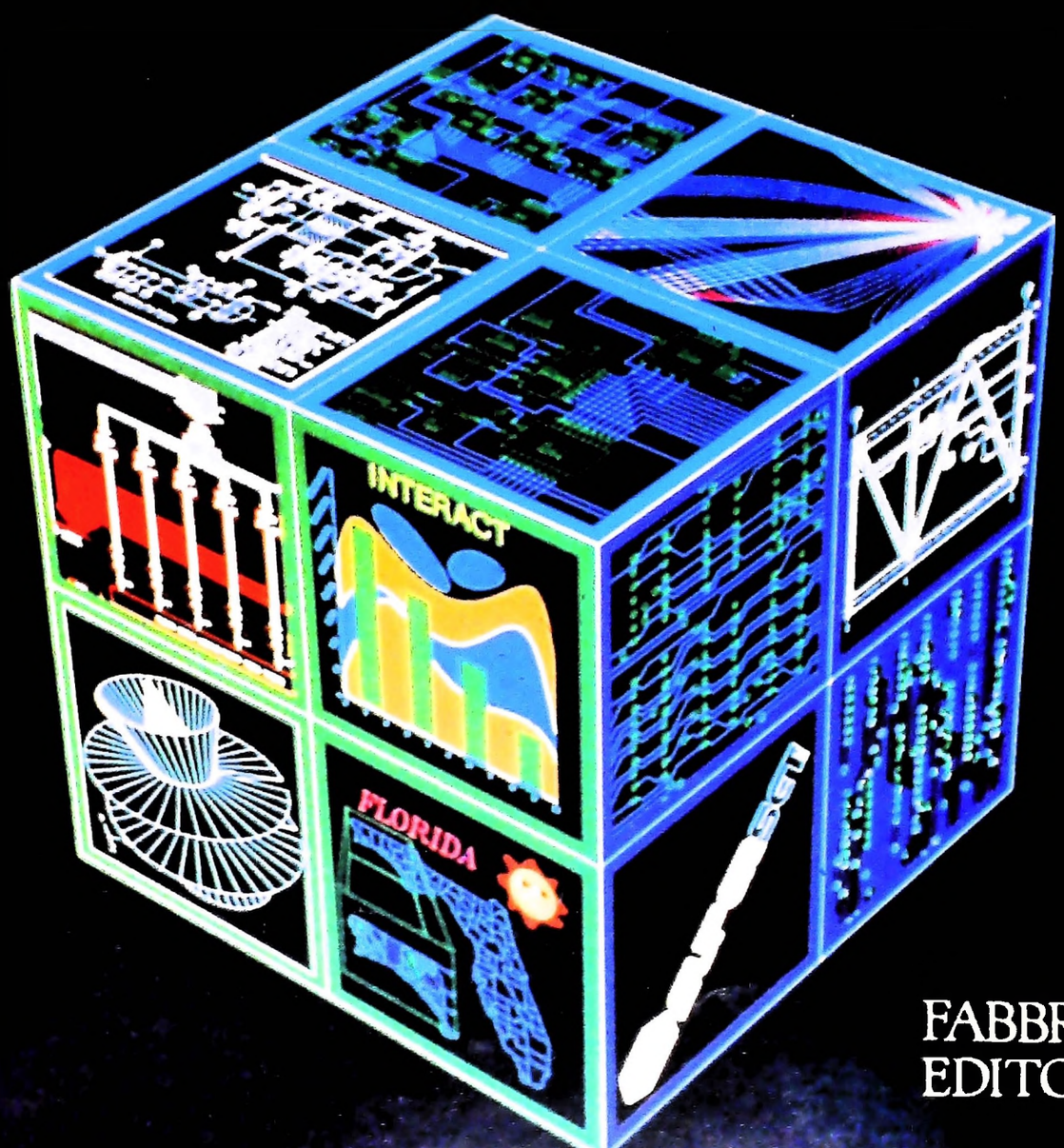
421909

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
GIANCARLO MAURI, CLAUDIO PARMELLI
ETTORE DECIO, LUCA SPAMPINATO
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

LA VERIFICA DEI PROGRAMMI

Come accertarsi che un programma funzioni esattamente come era stato previsto.

Il problema della verifica

Le metodologie di programmazione, pur riducendo le possibilità di errori logici, non ne garantiscono l'assenza e non aiutano comunque a evitare altri tipi di errore. Per questo, è necessario affrontare il problema della correttezza anche dal secondo punto di vista presentato nella lezione precedente, che può essere ricondotto alle seguenti questioni:
— dato un problema P e un algoritmo (programma) A che dovrebbe risolverlo, come possiamo stabilire se A è corretto, cioè se risolve effettivamente P?
— nel caso in cui A risulti scorretto, come possiamo individuare gli errori e correggerli?
Il primo punto è quello di maggior rilevanza teorica. Così come è stato formulato, è evidentemente un tipico problema di computabilità: si tratta infatti di stabilire l'esistenza di un algoritmo generale che, ricevendo in ingresso un problema e un algoritmo codificati in forma opportuna, scriva "sì" se l'algoritmo è corretto relativamente al problema e "no" in caso contrario.
Diciamo subito che l'esistenza di un tale algoritmo è da escludere, cioè che il problema della correttezza è indecidibile nei termini visti. Ciò non significa che bisogna rinunciare ad affrontarlo. Infatti l'indecidibilità riguarda un aspetto facilmente isolabile del problema della correttezza, la terminazione, e inoltre è possibile sviluppare metodi parziali.

Il testing dei programmi

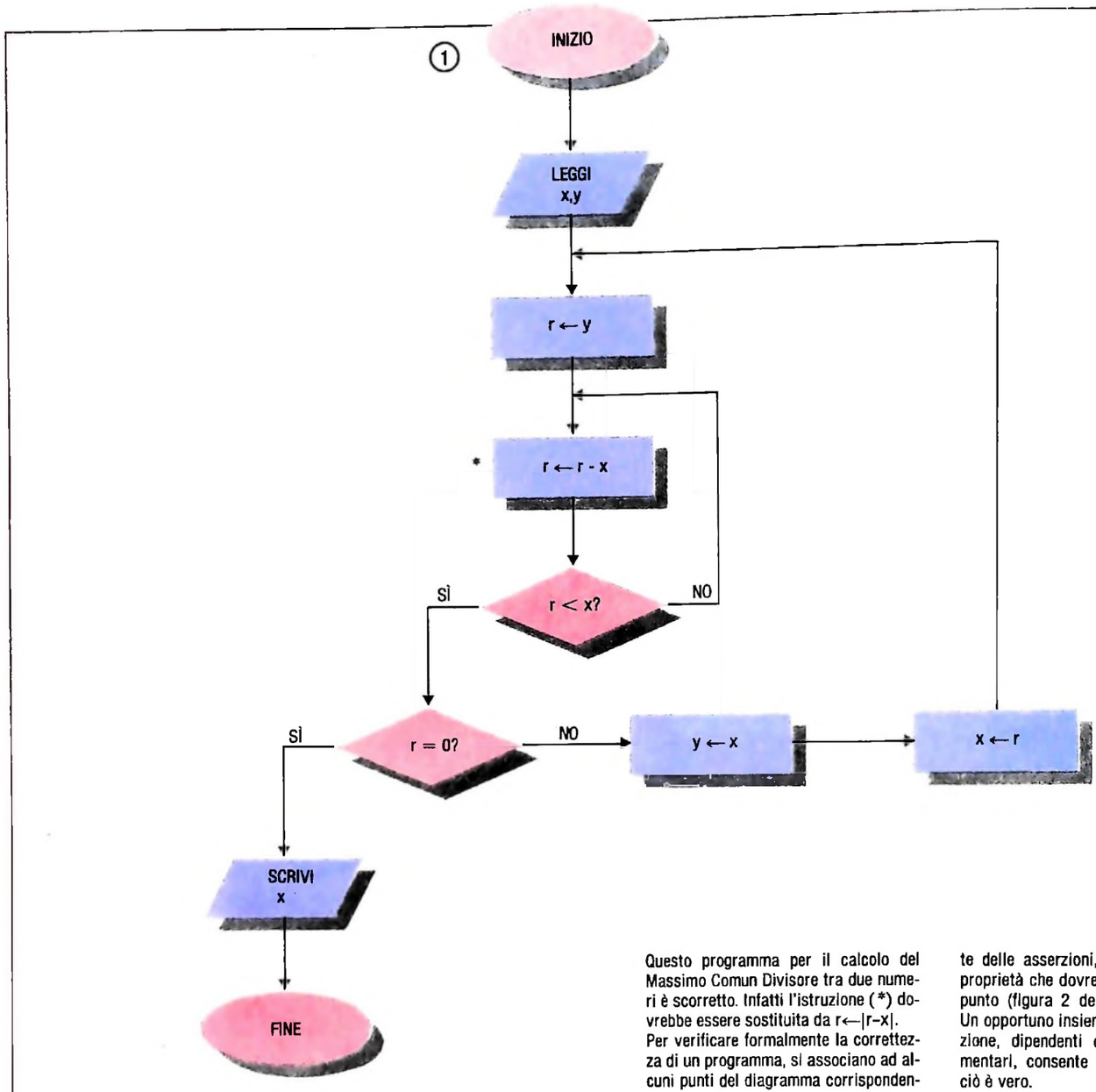
Un primo e più diffuso modo per affrontare il problema è di tipo empirico, e va sotto il nome di "testing" dei programmi. Il testing consiste essenzialmente nell'esecuzione di un programma in un ambiente controllato, in cui sia possibile verificare la corrispondenza tra risultati effettivi e risultati attesi. Poiché non ha senso, né è in genere possibile, provare il programma su tutti i dati, il punto cruciale riguarda la scelta di dati di test significativi, che cioè rappresentino ampie classi di dati.
Un primo criterio di scelta può basarsi sulla funzionalità del programma: si tratta di individuare classi di dati che dovrebbero dare origine a risultati "analoghi", e scegliere quindi un dato di prova per ogni classe.
Si supponga per esempio di voler sottoporre a test un programma per il calcolo del Massimo Comun Divisore tra due numeri x e y. Si potrebbe scegliere un dato di test per ognuno dei casi qui riportati:

$x>0$ e $y>0$;
$x>0$ e $y<0$ o viceversa;
$x=0$ e $y\neq 0$ o viceversa;
x primo e y primo;
x multiplo di y o viceversa ecc.

Con questo criterio si ignora completamente la struttura interna del programma. Un approccio diverso è basato sulla scelta dei dati di test guidata dalla struttura di controllo del programma, in modo tale che:
— ogni istruzione sia eseguita almeno una volta;
— ogni possibile cammino di controllo nel programma sia seguito almeno una volta.
Ognuno dei due criteri visti lascia però aperti ampi margini di errore (si veda per esempio la figura 1 della pagina seguente); e anche la loro combinazione non è sufficiente per dare la garanzia assoluta di correttezza.

La verifica formale dei programmi

L'ideale, come già si è detto per la sintesi dei programmi, è disporre di metodi formali in grado di dedurre, con un procedimento logico rigoroso a partire dalle informazioni iniziali disponibili, che l'algoritmo è corretto. Il più noto di questi metodi è il metodo delle asserzioni sviluppato da Floyd e Hoare.
L'idea su cui si basa il metodo è semplice: poiché ogni programma è ottenuto componendo secondo regole e strutture di controllo precise un insieme di "comandi" elementari, se si assegna un significato computazionale a ogni comando elementare (precisando per esempio come il comando trasforma un certo insieme di proprietà dei dati) e si forniscono regole adeguate per comporre questi significati computazionali, dovrebbe essere possibile associare a ogni programma, come suo significato complessivo, la funzione che esso computa; dal confronto di questa funzione con quella desiderata si verifica immediatamente se il programma è corretto.
Per realizzare praticamente questa idea, si procede associando ad alcuni punti "critici" del programma (rappresentato per esempio da un diagramma a blocchi) delle "asserzioni" che descrivono alcune proprietà e relazioni tra i valori delle variabili che dovrebbero essere vere in quel punto, e verificando quindi se effettivamente i comandi elementari garanti-



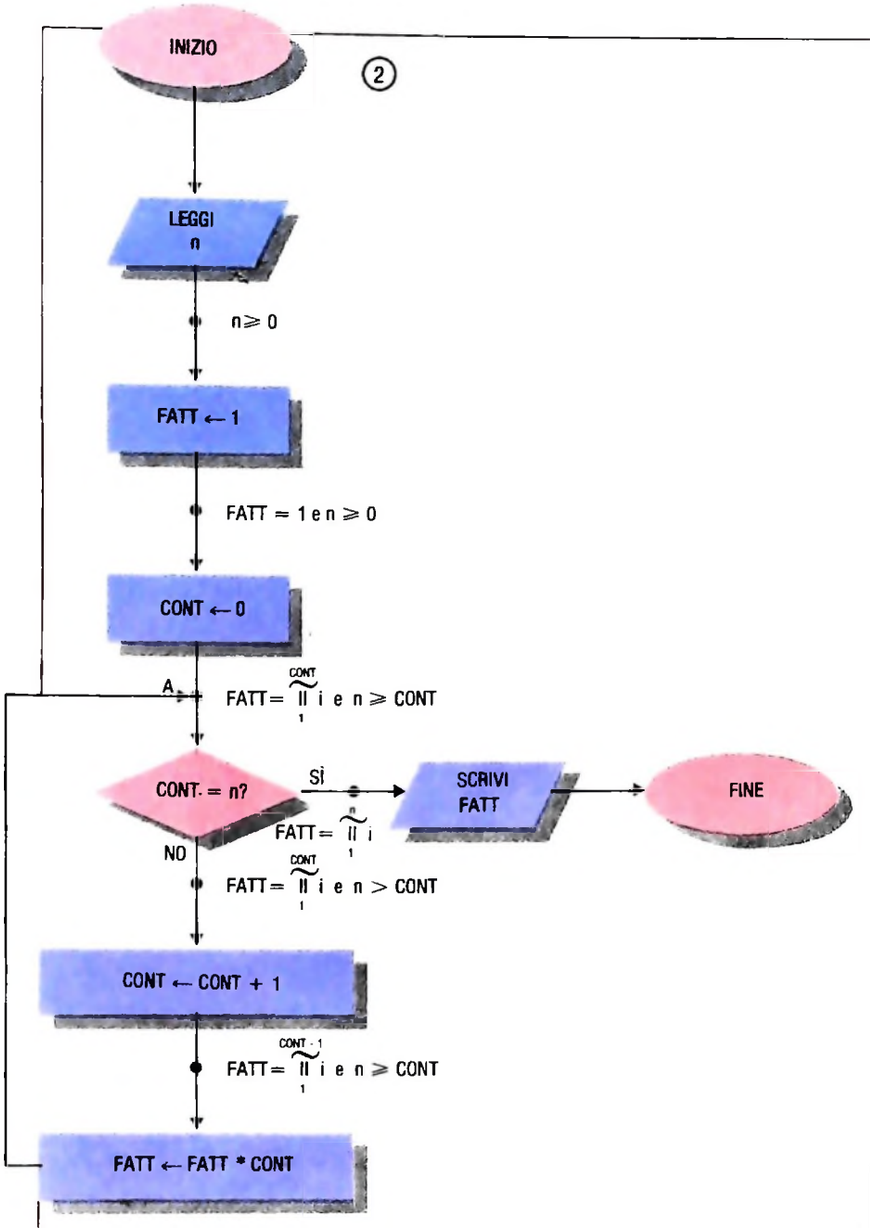
Questo programma per il calcolo del Massimo Comun Divisore tra due numeri è scorretto. Infatti l'istruzione (*) dovrebbe essere sostituita da r←|r-x|. Per verificare formalmente la correttezza di un programma, si associano ad alcuni punti del diagramma corrispondente delle asserzioni, le quali enunciano proprietà che dovrebbero valere in quel punto (figura 2 della pagina accanto). Un opportuno insieme di regole di deduzione, dipendenti dalle istruzioni elementari, consente poi di verificare se ciò è vero.

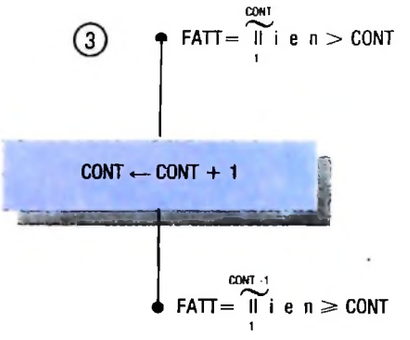
scono la validità di tutte queste proprietà. Ovviamente, avremo due asserzioni fondamentali: la prima, $p_0(x)$, posta all'inizio del programma, descrive i dati di ingresso "leciti"; la seconda, $p_F(x,y)$, al termine del programma, descrive la relazione voluta tra dati e risultati, cioè la funzione che il programma dovrebbe computare.
Per esempio, dato il programma per il calcolo del fattoriale già visto nella lezione precedente (figura 2 nella pagina accanto), l'asserzione di ingresso preciserà che sono ammessi solo numeri non negativi ($n>0$), mentre l'asserzione d'uscita sarà $FATT = n! = \overset{CONT}{\underset{1}{\|}} i$ (il simbolo $\overset{CONT}{\underset{1}{\|}} i$, detto di produttoria, indica il prodotto di tutti i numeri interi compresi tra 1 e n).
Possiamo a questo punto definire in modo formale la correttezza e la terminazione di un programma P rispetto alle asserzioni p_0 e p_F:

— P termina se per ogni dato di ingresso x che verifica il predicato $p_0(x)$ la computazione raggiunge l'istruzione di fine;
— P è parzialmente corretto se per ogni dato x che verifica $p_0(x)$. Se P termina con l'uscita y allora vale $p_F(x,y)$;
— P è totalmente corretto se per ogni dato x per cui $p_0(x)$ è vero termina con un'uscita y tale che $p_F(x,y)$ è vero.
Per dimostrare che un programma è totalmente corretto (che è ciò che ci interessa) conviene in genere dimostrare prima la correttezza parziale e poi la terminazione di P.

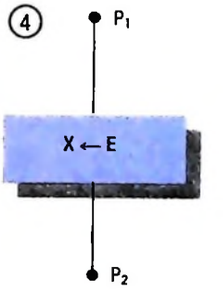
Le prove di correttezza parziale

Per dimostrare che, per tutte le esecuzioni di P che derivano, dalla validità dell'asserzione iniziale si deduce quella dell'asserzione finale, si spezza la dimostrazione in una serie di pas-

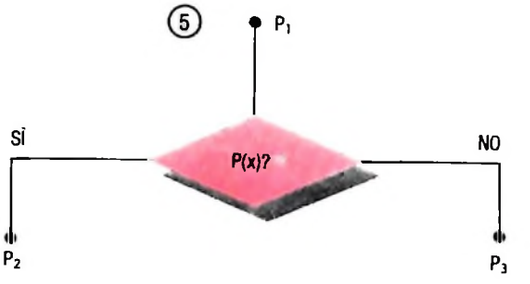




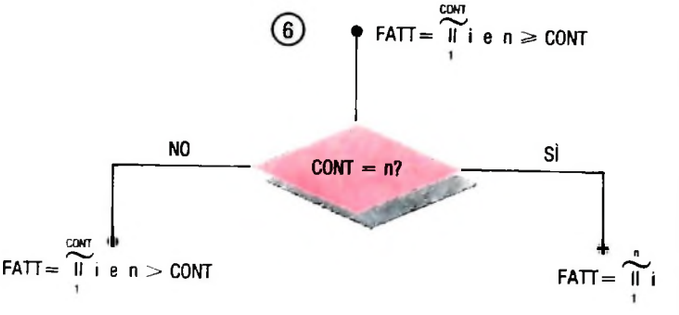
Sostituendo CONT con CONT + 1 nella post-asserzione, si ottiene esattamente la pre-asserzione.



Sostituendo a tutte le occorrenze di X in p_2 la formula E si ottiene un predicato che è implicato da p_1.



Perché il test sia corretto, p_1 e P devono implicare p_2, mentre p_1 e $\neg$ P devono implicare p_3.

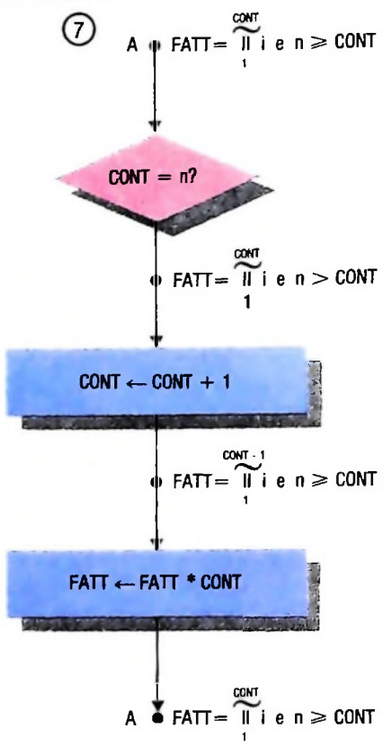


Un esempio di come un test modifica le asserzioni.

si successivi, legati ai singoli comandi elementari. Ognuno di questi passi consiste nel dimostrare che, dopo l'esecuzione di un comando C, come conseguenza della validità di tutte le asserzioni precedenti e dell'esecuzione di C, vale una particolare asserzione A(x). In questo modo, si può vedere come l'esecuzione dei singoli comandi modifica man mano l'asserzione iniziale trasformandola in quella finale.

Il caso più semplice è quello in cui il programma sia costituito da istruzioni di assegnamento in successione lineare. In questo caso, basta semplicemente controllare, risalendo indietro lungo il programma, che l'asserzione che segue ogni istruzione sia conseguenza di quella che la precede, tenendo conto dei nuovi valori assegnati alle variabili.

Consideriamo per esempio l'assegnamento di figura 3; che proprietà deve valere prima dell'esecuzione di questa istruzione per assicurare che $FATT = \prod_{1}^{CONT-1} i$ sia vero dopo? La ri-



Il ciclo della figura 2 della pagina precedente è stato tagliato nel punto A. L'asserzione associata al punto A deve valere sia all'inizio che al termine di ogni esecuzione del ciclo. Per questo si dice "invariante".

sposta è che deve essere vero il predicato ottenuto sostituendo CONT con CONT+1, cioè FATT = $\prod_{1}^{CONT+1-1} i = \prod_{1}^{CONT} i$.

In generale, dato un assegnamento $X \leftarrow E$ con pre-asserzione p_1 e post-asserzione p_2 come in figura 4, ove E è una qualsiasi espressione, l'assegnamento è corretto se, sostituendo a tutte le occorrenze di X nella post-asserzione l'espressione E, si ottiene un predicato la cui validità è conseguenza della validità della pre-asserzione. Il secondo caso da affrontare riguarda le istruzioni di selezione. Benché un'istruzione di selezione del tipo mostrato nella figura 5 non cambi i valori delle variabili, essa ci consente di acquisire informazioni più precise sulle loro proprietà. Infatti, mentre prima di eseguire l'operazione non sappiamo se la proprietà P(x) vale o non vale, all'uscita dal blocco di selezione questa informazione è nota e può essere utilizzata. Perché l'istruzione di selezione sia corretta dovremo quindi poter dedurre la post-asserzione p_2 associata all'uscita positiva del blocco della pre-asserzione p_1 e dalla proprietà P(x) e la post-asserzione p_3 associata all'uscita negativa dalla pre-asserzione e dalla negata della proprietà P(x). In formule, dovrà valere:

$p_1(x) \wedge P(x) \supset p_2(x)$

$p_1(x) \wedge \neg P(x) \supset p_3(x)$

Nell'esempio della figura 2, la pre-asserzione CONT≤n diventa, dopo l'esecuzione del test CONT=n?, rispettivamente CONT=n e CONT<n (figura 6).

Un'ultima osservazione riguarda i cicli; per trattarli, è necessario spezzarli fissando un "punto di taglio" e applicando le regole viste sopra al pezzo di programma che, partendo dal punto di taglio, riporta al punto di taglio stesso. Questo è evidentemente molto semplice se il diagramma che rappresenta il programma è strutturato. La figura 7 mostra come può essere tagliato il ciclo del diagramma di figura 2.

Possiamo riassumere quanto detto enunciando il

METODO DELLE ASSERZIONI INDUTTIVE (FLOYD)

Per provare la correttezza di un programma P rispetto alla asserzione iniziale $p_o(x)$ ed all'asserzione finale p_F occorre:

— trovare un opportuno insieme di asserzioni induttive da associare ai vari punti del programma;

— tagliare i cicli;

— verificare che per ogni blocco di assegnamento o di selezione le asserzioni di uscita siano conseguenza delle asserzioni di ingresso e dell'istruzione eseguita.

La scelta delle asserzioni "giuste" richiede una notevole esperienza e una chiara comprensione del programma, e può avvenire solo su base euristica, non automatizzabile. In genere, partendo da un'idea anche approssimativa delle relazioni tra le variabili che sono importanti a un certo punto del programma, si scelgono delle asserzioni provvisorie. Si verifica quindi se ogni asserzione può essere dedotta dalle precedenti e, in caso contrario, si cerca di rendere queste più forti in modo da consentire la deduzione e così via. La parte di verifica è invece facilmente automatizzabile.

La terminazione

Per completare la prova di correttezza di un programma, dopo averne dimostrata la correttezza parziale, occorre dimostrarne la terminazione; si è già detto che il problema della terminazione è in generale indecidibile. Tuttavia è possibile in parecchi casi arrivare a una dimostrazione in modo abbastanza semplice, col seguente procedimento:

— si individuano tutti i cicli del programma;

— per ogni ciclo si individua una funzione f(x) di una qualche variabile di programma x con le seguenti proprietà:

— f(x) è un numero intero;

— inizialmente f(x)≥0;

— a ogni esecuzione del ciclo, f(x) decresce di almeno una unità;

— si esce dal ciclo se e solo se f(x) raggiunge il valore 0.

Evidentemente, poiché i numeri interi sono ordinati, una funzione siffatta raggiungerà sicuramente il valore 0, garantendo quindi la terminazione.

Nell'esempio riportato a pagina precedente, la funzione f(CONT))=n-CONT ha le proprietà volute, e quindi il programma termina.

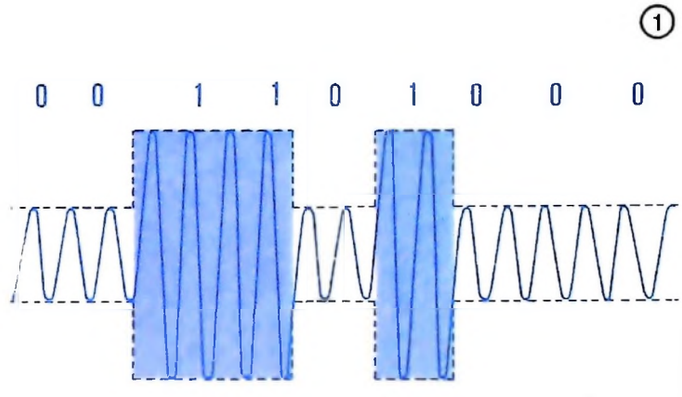
Non sempre si può individuare una funzione di questo tipo per ogni ciclo: in questo caso, non siamo in grado di dire né se il programma termina né se non termina.

MODEM E INTERFACCE

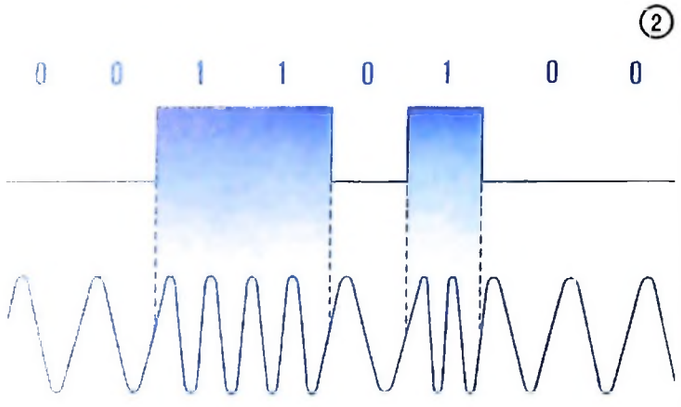
Come convertire i segnali digitali in un'onda in grado di percorrere le linee telefoniche.

Le tecniche di modulazione

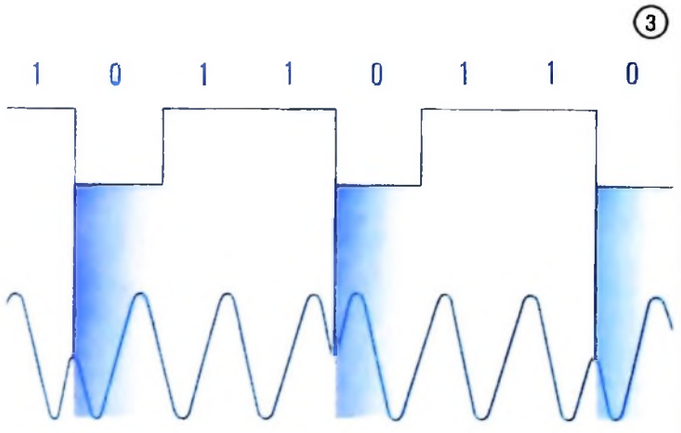
Per usare i circuiti telefonici per la trasmissione dati, in generale, dobbiamo convertire i segnali digitali in un'onda con forma tale da consentirle di percorrere senza eccessive deformazioni la linea telefonica: questo stesso segnale sarà poi convertito all'altro estremo della linea di nuovo in forma digitale per essere immesso nell'elaboratore oppure utilizzato dal terminale.
Per la trasmissione lungo le linee telefoniche conviene impiegare segnali con forma d'onda sinusoidale: un segnale di questo tipo può contenere l'informazione digitale ed essere trasferito tramite una portante *(carrier wave)* lungo la linea di trasmissione. Quindi, all'estremo di partenza, con un procedimento chiamato modulazione, si trasforma l'informazione digitale avviandola sulla portante, mentre all'altro estremo della linea si provvederà, con un procedimento chiamato demodulazione, a rilevare i dati trasferiti dalla portante.
Il procedimento di modulazione viene realizzato alterando, opportunamente, le caratteristiche principali del segnale; analizziamo quindi le più comuni tecniche di modulazione.
Modulazione di ampiezza (AM). Permette la trasmissione di informazioni binarie associando ai valori 1 e 0 due ampiezze diverse della frequenza portante trasmessa in linea. La modulazione di ampiezza offre la possibilità di avere più di due livelli, permettendo quindi la trasmissione di più bit nello stesso intervallo temporale (figura 1).
La tecnica *qam* (Quadrature Amplitude Modulation) è proprio basata su un sistema di codifica che permette di ottenere elevati *throughput* (9600 bit/s e oltre) anche con la limitata larghezza di banda delle linee foniche (circa 3 KHz). (Per *throughput* si intende, generalmente, la quantità di informazioni elaborate o trasmesse.)
Modulazione di frequenza (FM). In questo tipo di modulazione il segnale portante è modulato in frequenze diverse. Per esempio la portante può essere modulata tra 1200 Hz e 2200 Hz (senza alterarne l'ampiezza) secondo il segnale digitale binario. Le specifiche frequenze utilizzate dipendono dai dispositivi trasmittenti e riceventi utilizzati; per alcuni tipi di dispositivi il segnale logico 0 è rappresentato con la frequenza di 1200 Hz mentre con la frequenza di 2200 Hz è rappresentato il segnale 1 (figura 2).
Quando la FM è utilizzata per inviare informazioni binarie in forma bipolare, viene denominata Frequency Shift Keyins (FKM); in tale sistema la portante (normalmente di 1700



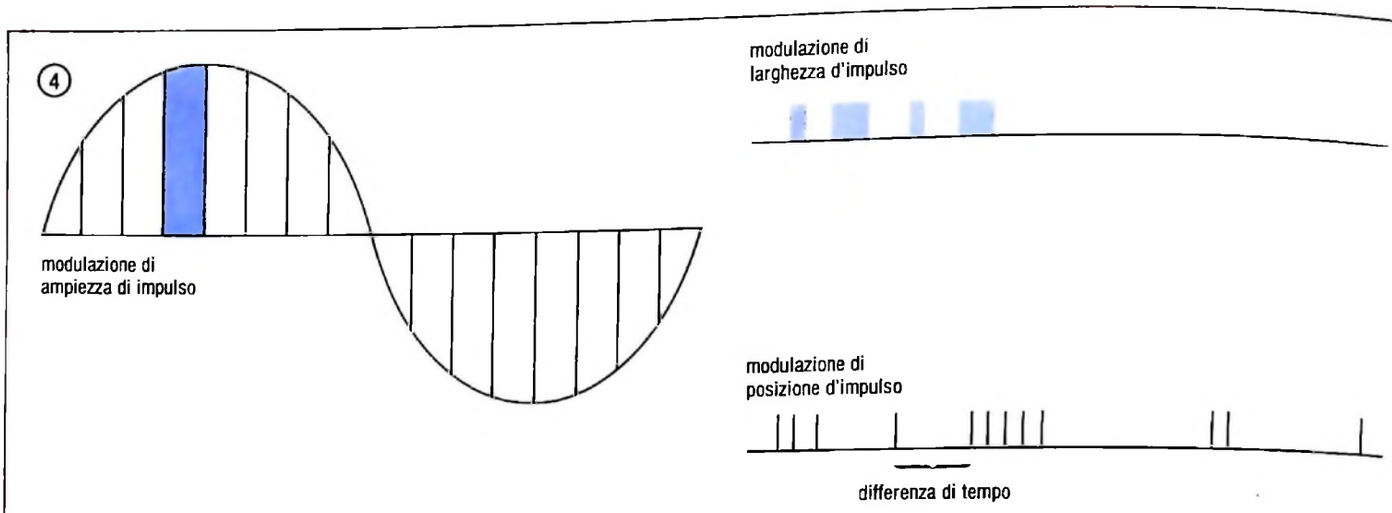
Modulazione d'ampiezza.



Modulazione di frequenza.



Modulazione di fase.



Tre tecniche di modulazione a impulsi codificati (PCM). Un impulso codificato rappresenta valori discreti di porzioni istantanee del segnale.

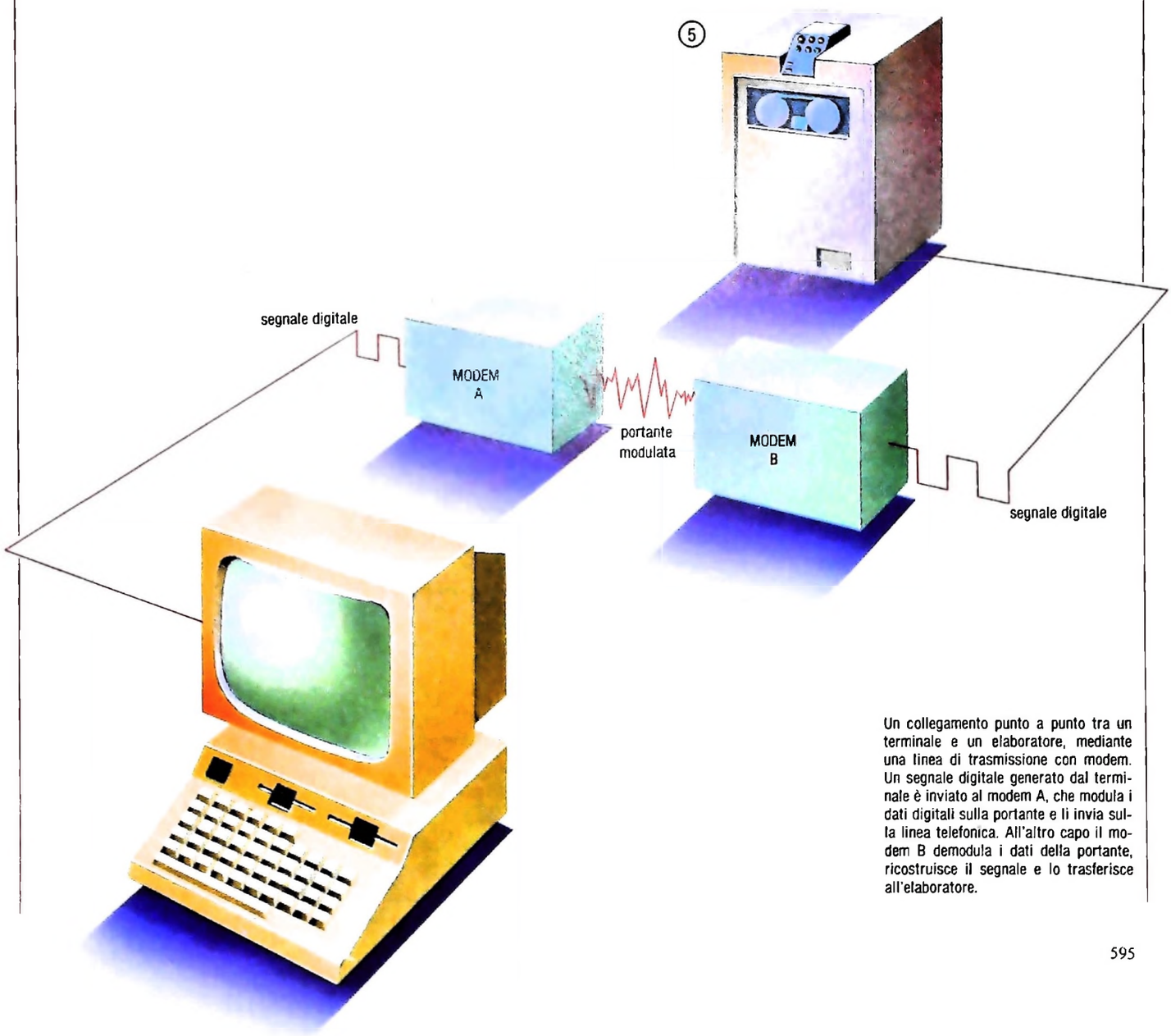
Hz) viene generalmente modulata con più o meno 500 Hz in modo da poter rappresentare i due valori binari 0 e 1 (come visto prima).
La tecnica FKM è generalmente utilizzata per dispositivi a bassa velocità come le telescriventi che operano a velocità inferiori a 1800 bps.
Modulazione di fase (PM). Negli ultimi tempi sta cominciando a soppiantare le precedenti tecniche per quanto riguarda le trasmissioni ad alta velocità (oltre i 2000 bps), in quanto è soggetta a minori disturbi.
Nella modulazione di fase, la fase del segnale trasmesso viene ruotata di un certo numero di gradi in relazione ai bit che debbono essere trasmessi; per esempio in un modem a due fasi il segnale è ruotato di 180 gradi (360 : 2) a seconda che si voglia indicare il segnale binario 0 oppure 1. Se non avvengono rotazioni, il segnale viene interpretato come una serie di 0 o di 1.
La figura 3 di pagina precedente, rappresenta la Differential Phase Shift Keying (DPSK), in questa tecnica ogni volta che il dispositivo ricevente incontra una rotazione di fase di 180 gradi, assegna il valore binario 0; mentre in tutti gli altri istanti associa il valore 1.
In genere i dispositivi a modulazione di fase, operano in quattro o otto fasi, permettendo un flusso di dati sulla linea doppio o triplo.
Modulazione di impulsi codificati (PCM). Un impulso codificato è utilizzato per rappresentare valori quantizzati di porzioni istantanee del segnale; vi sono quindi diverse tecniche per modificare le caratteristiche degli impulsi e la figura 4 mostra tre diverse modulazioni di impulsi utilizzate per le trasmissioni digitali.
Nella modulazione di impulso, l'informazione è convertita in un flusso di bit notevolmente simili ai dati di un computer. Per esempio, nella modulazione di ampiezza di impulso (figura 4), il segnale è prima convertito in una serie di impulsi le cui ampiezze corrispondono alle ampiezze istantanee del segnale; tali impulsi vengono quindi utilizzati per modulare in ampiezza il segnale portante con il risultato che la portante modulata consiste in una serie di impulsi aventi ampiezza corrispondente all'informazione da trasmettere.
Delle quattro tecniche di modulazione illustrate, questa è la meno affetta da disturbi e quindi viene utilizzata per diminuire la probabilità di errore durante la trasmissione.

Modem

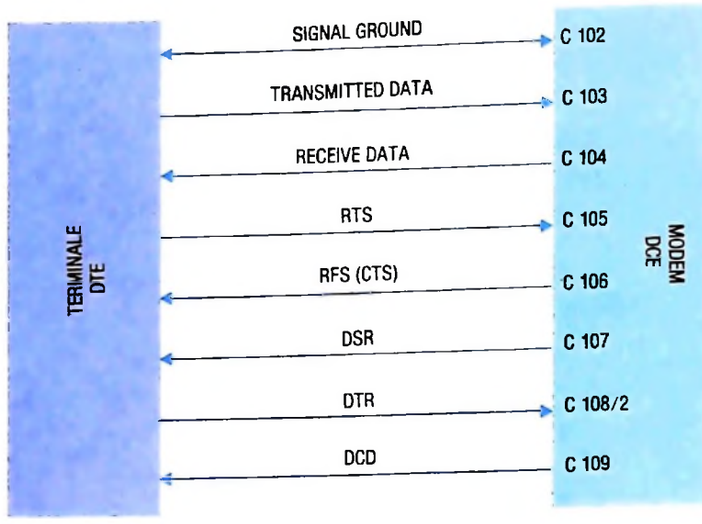
Modulazione e demodulazione dei dati digitali sono eseguite da un dispositivo chiamato *modem* (dalla contrazione delle parole (Modulator-Demodulator); talvolta lo stesso dispositivo prende il nome di *data-set*.
Esistono molti tipi di modem: half-duplex, full-duplex, sincroni e asincroni; in origine i modem erano oggetto separato della rete, mentre oggi molti terminali hanno un modem integrato (che però, in Italia, non è ammesso).
La figura 5 alla pagina seguente schematizza un collegamento punto a punto tra un terminale e un elaboratore con una linea di comunicazione con modem: il terminale genera un segnale digitale che è inviato al modem A, il quale modula i dati digitali sulla portante e li invia lungo la linea telefonica. Il modem B demodula i dati della portante e ricostruisce il segnale con forma quadra, e questo viene immesso nell'elaboratore.
I modem possono rallentare in modo significativo la sequenza di trasmissione dei dati.
La linea di trasmissione che collega i due modem può essere a due fili o a quattro fili. In precedenza si era detto che per stabilire un canale di comunicazione sono necessari quattro fili; in alcuni casi tuttavia (a velocità relativamente basse cioè 600-1200 bps) i modem possono manipolare elettronicamente le caratteristiche elettriche della trasmissione a due fili e derivare canali a due vie da un sistema a due fili.

Interfaccia modem/terminale V24/RS-232

Con il termine di interfaccia viene normalmente definita la demarcazione tra due parti tra loro collegate di un unico sistema. Per far sí che le due parti lavorino in perfetta sintonia è necessario che l'una segua un insieme di regole, o "specifiche", complementare all'insieme con il quale lavora l'altra parte. In genere sono definiti diversi livelli di specifiche di interfaccia, ma affinché le due parti possano essere collegate è indispensabile che si verifichi la compatibilità a ogni livello. Il livello più basso è quello meccanico: occorre che le prese dei vari componenti impiegati per collegare le due parti del sistema combacino perfettamente. Il livello immediatamente superiore è quello elettronico: occorre che ciascuna parte abbia il numero di fili dovuto e che ogni filo porti il segnale corretto nel momento voluto.
Analizzando lo schema riportato nella figura 6 della pagina seguente, si possono individuare le interfacce tra il modem A e il terminale e tra il modem B e l'elaboratore. L'interfaccia tra il terminale e il modem è rappresentata da un certo numero di fili che portano i segnali dati e i segnali di controllo.
Sul mercato possiamo trovare centinaia di terminali diversi e una vasta gamma di modem. Per evitare che si verifichino problemi di incompatibilità sono state definite delle specifiche di alcune interfacce standard: tra queste la più nota è la Ccitt V24 che è definita per velocità fino a 20000 bps, sufficiente per la maggior parte delle nostre esigenze.
Le specifiche V24/RS-232 definiscono il numero di fili impiegati per collegare il modem al terminale, i segnali elettrici inviati lungo quei fili e i livelli di segnale.
La figura in alto alla pagina seguente riporta gli elementi più importanti dell'interfaccia V24/RS-232: a sinistra del diagramma troviamo il Data Terminal Equipment (DTE) che sta a indicare un generico terminale o l'elaboratore collegato con il modem (DCE) riportato sulla destra. Le linee, che nello schema costituiscono l'interfaccia, portano i seguenti segnali (i numeri sono riferiti ai circuiti di scambio all'interno del modem):



Un collegamento punto a punto tra un terminale e un elaboratore, mediante una linea di trasmissione con modem. Un segnale digitale generato dal terminale è inviato al modem A, che modula i dati digitali sulla portante e li invia sulla linea telefonica. All'altro capo il modem B demodula i dati della portante, ricostruisce il segnale e lo trasferisce all'elaboratore.



⑥ Gli elementi principali di un'interfaccia V 24/RS-232 fra un terminale e un modem. Questo tipo d'interfaccia standard è definito per velocità fino a 20 000 bit per secondo. Sulle linee che rappresentano l'interfaccia sono indicati i segnali portati.

— SIGNAL GROUND -C102- (ritorno comune). Costituisce il circuito di ritorno comune di massa.
— REQUEST-TO-SEND - RTS -C105- (richiesta di invio). Questo segnale è inviato dal DTE al modem e viene interpretato come richiesta del DTE per iniziare a trasmettere dati.
— READY-FOR-SENDING - RFS (CLEAR-TO-SEND - CTS) -C106- (pronto a inviare). Questo segnale proviene dal modem e informa il DTE che esso può iniziare a trasmettere i dati.
— TRANSMITTED-DATA -C103- (dati trasmessi). Questo filo che va dal DTE al modem, porta i segnali digitali che devono essere modulati sulla portante.
— RECEIVE-DATA -C104- (ricevi i dati). Questo filo porta i dati digitali dal modem al terminale; si tratta dei dati che il modem ha demodulato dalla portante.
— DATA-CARRIER-DETECT - DCD -C109- (individuazione della portante dati). Questo segnale va dal modem al terminale e lo avvisa che il modem riceve correttamente la portante e che è pronto a demodulare i dati.
— DATA-SET-READY - DSR -C107- (DCE pronto). Segnala al DTE che il DCE è stato attivato ed è pronto per lo scambio di eventuali segnali di controllo.
— DATA-TERMINAL-READY - DTR -C108/2- (DTE pronto). Segnala che il DTE è attivato e prepara il DCE a effettuare operazioni di connessione. La condizione di off (0 logico), terminata la trasmissione in corso, segnala al DCE la disattivazione del DTE.
Alcuni dei circuiti di scambio hanno corrispondenza, nella parte frontale del modem, con dei led rossi (siglati nello stesso modo dei circuiti), la cui condizione di acceso segnala il corretto funzionamento della connessione; tali circuiti sono: C106,C107,C109.

Accoppiatori acustici

Un altro dispositivo per interfacciare i terminali con le linee di comunicazione è l'accoppiatore acustico (*acoustic coupler*). Tale dispositivo permette di utilizzare un normale telefono come interfaccia tra il terminale e la linea, convertendo i segnali digitali, provenienti dal computer, in segnali analogici, in questo caso in suoni udibili che possono essere trasmessi lungo una linea telefonica per mezzo della "cornetta" del telefono.
Inizialmente per poter stabilire la comunicazione, bisogna formare il numero di telefono al quale è collegato il centro di elaborazione: nel momento in cui l'elaboratore risponde, si inserisce la cornetta del telefono nell'accoppiatore acustico. In molti terminali gli accoppiatori acustici sono integrati e quindi basta inserire la cornetta del telefono negli speciali alloggiamenti del terminale per trasferire i suoni udibili dalla cornetta al terminale.
Altri accoppiatori acustici (come quello utilizzato per M10) hanno un'interfaccia V24/RS-232 e possono quindi da una parte "interfacciare" direttamente il terminale, mentre l'altra uscita avrà un'interfaccia acustica che convertirà i segnali digitali in suoni udibili che possono essere captati dalla cornetta del telefono.
I dispositivi ad accoppiamento acustico non sono progettati in realtà in modo da poter funzionare per lungo tempo in un unico posto, a causa del modo con cui sono costruite le cornette telefoniche.
Infatti nel microtelefono sono contenuti dei granuli di carbone che vibrando quando sono sollecitati dalle onde sonore della nostra voce, danno origine al segnale che viene inviato sulla linea telefonica.
Nel caso di una conversazione di lunga durata succede che i granuli di carbone vengono compressi e il segnale che essi creano subisce un disturbo.
Difficilmente percepiamo durante la nostre conversazioni questi disturbi, perché, sia pur impercettibilmente, continuiamo a muovere la cornetta del telefono con la mano e il volto. In questo modo vengono mossi i granuli di carbone del microtelefono con il risultato che difficilmente la qualità del segnale degrada.
Ma nel nostro caso, la cornetta del telefono è fissata sull'accoppiatore acustico, e i granuli non possono essere mossi; di conseguenza il segnale subisce una progressiva distorsione.

Lezione 37

Una versione semplificata di algoritmo di MERGE

Nelle lezioni precedenti abbiamo esaminato un problema di MERGE di due archivi, che abbiamo risolto con un algoritmo molto vicino a quello che avremmo impiegato nella pratica e quindi, da questo punto di vista, molto intuitivo, ma forse un po' complesso da un punto di vista realizzativo.
In questa lezione vedremo una variante dell'algoritmo già visto, che, grazie ad alcuni "accorgimenti", permette di semplificare notevolmente il programma. Ciò che complica l'algoritmo precedentemente considerato è il trattamento di fine file: quando infatti uno dei due archivi viene esaurito, dobbiamo eseguire una parte di programma che provvede a riportare sul file di output le informazioni rimanenti dell'altro archivio. Ciò richiede la costruzione di parti di programma aggiuntive a quella che effettua il MERGE, che viene eseguita fin tanto che sono disponibili dati su entrambi i file.
Tutto si semplifica però se riusciamo ad assimilare il caso di fine file al caso di informazione disponibile.
A questo scopo ricorriamo a un semplice "trucco": quando incontriamo la condizione di fine file e non disponiamo quindi più di un nominativo da confrontare, assegniamo alla variabile corrispondente un valore "molto alto", che cioè in tutti i confronti con i nominativi letti dall'altro file risulti sempre maggiore. A questo scopo dovremo scegliere una stringa che supponiamo non compaia mai come nominativo, per esempio una sequenza opportunamente lunga di "z".
In tal modo, anche quando abbiamo raggiunto la fine del file, continuiamo in realtà a eseguire la stessa porzione di codice, fino a quando anche l'altro file è esaurito. Interrompiamo allora l'iterazione di confronti e terminiamo l'esecuzione.

La struttura dell'algoritmo

Vediamo dunque la struttura del nuovo algoritmo, in linguaggio ispirato al Pascal:

```
open file A;
open file B;
open file C
leggea;
leggeb
WHILE NOT (A^=B^ = massimo valore) DO
        CASE
                A^=B^   : BEGIN
                            scrive B^ su C;
                            leggea;
                            leggeb
                            END
                A^<B^   : BEGIN
                            scrive A^ su C;
                            leggea
                          END
```

```
                    A^>B^  : BEGIN
                                scrive B^ su C;
                                leggeb
                             END
            ENDCASE
close file A;
close file B;
close file C;
```

dove "leggea" è così realizzata

```
legge A;
IF EOF(A) THEN
        A^: = massimo valore;
```

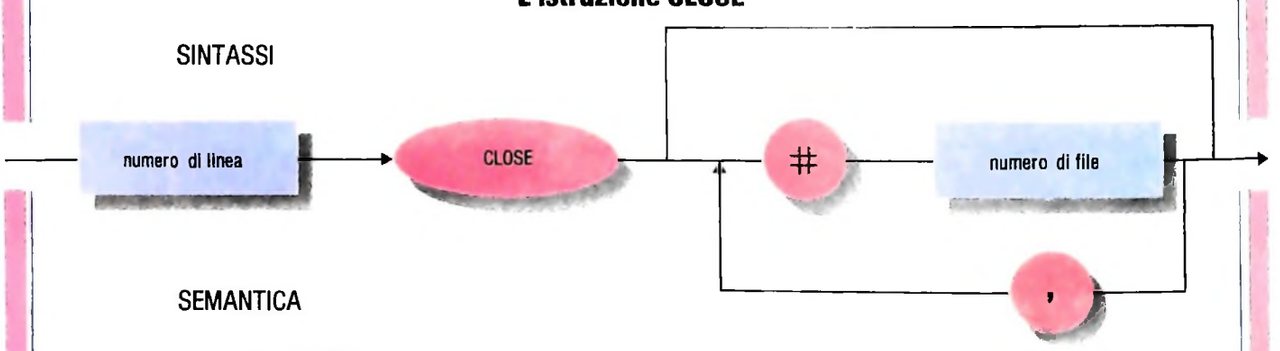
e "leggeb" è così realizzata:

```
legge B;
IF EOF(B) THEN
                B^: = massimo valore;
```

Come si vede in questo modo l'algoritmo è notevolmente semplificato.
Si noti anche che viene controllata la condizione di fine file sulla prima lettura, che ci consente quindi di eseguire il programma anche se i due archivi di partenza sono privi di dati.
Si noti che l'iterazione sul MERGE termina quando entrambi i nominativi contengono il valore che individua la fine del file.

L'istruzione CLOSE

SINTASSI



SEMANTICA

L'istruzione CLOSE permette di concludere le operazioni di trattamento di un file qualsiasi sia la periferica su cui esso risiede.
In particolare è possibile con una sola istruzione CLOSE "chiudere" più di un file (si ricordi che se il numero di files usati è superiore a 1 deve essere specificato nell'istruzione MAXFILES).
È inoltre possibile "chiudere" tutti i file specificando la sola istruzione CLOSE senza operandi.
Si noti comunque che la "chiusura" di tutti i file avviene automaticamente nei seguenti casi:

- quando viene eseguita una istruzione END;
- quando viene eseguito uno dei seguenti comandi:
  - NEW: per generare un nuovo programma;
  - RUN: per eseguire un programma;
  - LOAD: per caricare un programma da file in ambiente BASIC.

Il modulo BASIC

Riportiamo ora la traduzione BASIC del programma sopra mostrato. Si noti che è stato costruito assegnando i numeri di linea a partire da 1000 in modo da poter essere sostituito al modulo corrispondente nel programma realizzato nelle lezioni precedenti:

```
1000 ' Merge
1010 ' Apertura files
1020 OPEN "RAM:A" FOR INPUT AS #1
1030 OPEN "RAM:B" FOR INPUT AS #2
1040 OPEN "RAM:C" FOR OUTPUT AS #3
1050 ' Files disponibili
1060 ' Lettura record di A e di B
1070 GOSUB 1400 ' Legge A
1080 GOSUB 1500 ' Legge B
1090 ' While not (N$=N1$=massimo valore) do
1100 IF N$=N1$ AND N$="ZZZZZZZZZZZZZZZZZZZZ"
THEN 1300
1110 ' Case
1120 ' N$=N1$
1130 IF N$<>N1$ THEN 1170
1140 PRINT #3,N1$;",";T1
1150 GOSUB 1400 ' Legge A
1160 GOSUB 1500 ' Legge B
1165 GOTO 1262
1170 ' N$<N1$
1180 IF N$>N1$ THEN 1240
1190 PRINT #3,N$;",";T
1200 GOSUB 1400 ' Legge A
1210 GOTO 1262
1240 ' N$>N1$
1250 PRINT #3,N1$;",";T1
1260 GOSUB 1500 ' Legge B
1262 ' Endcase
1265 GOTO 1090
1300 ' Endwhile
1310 CLOSE #1
1320 CLOSE #2
1330 CLOSE #3
1340 END
1400 ' Legge A
1410 ' If eof(A) then N$=massimo valore
1420 IF NOT EOF(1) THEN 1450
1430 LET N$="ZZZZZZZZZZZZZZZZZZZZ"
1440 GOTO 1490
1450 ' Else
1460 INPUT #1,N$,T
1490 RETURN
1500 ' Legge B
1510 ' If eof(B) then N1$=massimo valore
1520 IF NOT EOF(2) THEN 1550
1530 LET N1$="ZZZZZZZZZZZZZZZZZZZZ"
1540 GOTO 1590
```

```
1550 ' Else
1560 INPUT #2,N1$,T1
1590 RETURN
```

Nel modulo sopra riportato, la lettura dei due file è realizzata con due moduli di livello più basso, che provvedono anche a verificare la condizione di fine file e, in tal caso, ad assegnare il "massimo valore" alle due variabili che contengono il nominativo.
Si noti, come abbiamo già mostrato in altri casi, come nella "traduzione" dallo schema di programma al BASIC, intervengano alcuni cambiamenti legati alla struttura del linguaggio.

L'istruzione Maxfiles

Per eseguire correttamente il programma di MERGE che abbiamo costruito è necessario dichiarare il numero di file che il programma stesso usa. In mancanza di tale dichiarativa, l'apertura di più file provoca un errore di tipo BN, corrispondente al fatto che è stato attribuito a un file un numero non ammesso.
L'indicazione del numero di file che si intende usare viene fornita tramite l'istruzione MAXFILES.
Tale istruzione può essere fornita all'interno del programma o "direttamente", come "comando" al di fuori di un programma e senza specificare il numero di linea.
Poiché nel nostro caso il numero di file è tre, potremo quindi avere:

```
1 MAXFILES=3
```

o potremo fornire il comando "direttamente":

```
Ok
MAXFILES=3
Ok
```

Cosa abbiamo imparato

In questa lezione abbiamo visto:

- una variante dell'algoritmo di MERGE tra due file ordinati;
- l'uso di MAXFILES sia come istruzione inserita nel programma o come "comando" fornito "direttamente", per specificare il numero di file su cui si intende operare;
- la sintassi completa dell'istruzione CLOSE.

PROGETTO D'ARCHITETTURA NEL SOFTWARE

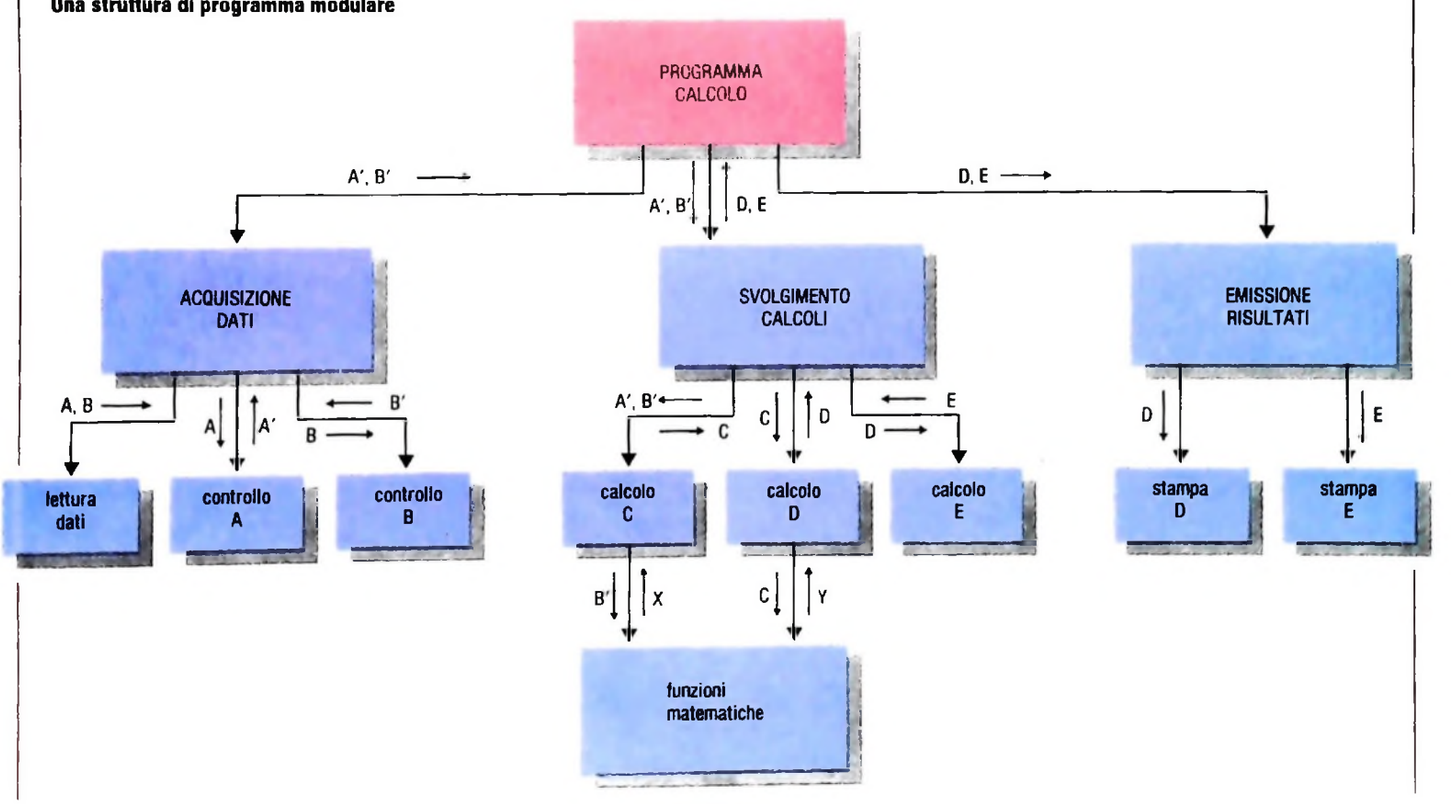
Dopo aver individuato le specifiche funzionali, occorre definire la struttura architetturale del prodotto da sviluppare.

Due sono i livelli a cui quest'attività, che spesso viene chiamata "disegno", si colloca: un primo livello è relativo alla organizzazione del prodotto in sottosistemi, o comunque in macro-blocchi (si parla di "programming in the large", ovvero *programmazione in grande*); un secondo livello è invece relativo alla struttura di dettaglio del singolo modulo e programma (si parla di "programming in the small", ovvero *programmazione in piccolo*).
In ambedue le attività si hanno a disposizione metodi e strumenti che guidano i progettisti all'ottenimento di prodotti di qualità.

Programmazione in grande

La programmazione di alto livello è un momento estremamente rilevante: è qui che nascono caratteristiche come la *manutenibilità* e la *modificabilità* dei programmi, garantendo la possibilità di fare evolvere il programma nel tempo, adattandolo a ogni nuova esigenza.
In quest'ambito ha sempre avuto particolare rilevanza la "programmazione modulare", cioè l'organizzazione di un grande programma in diversi moduli comunicanti tra di loro, scritti separatamente e successivamente integrati. Il pro-

Una struttura di programma modulare

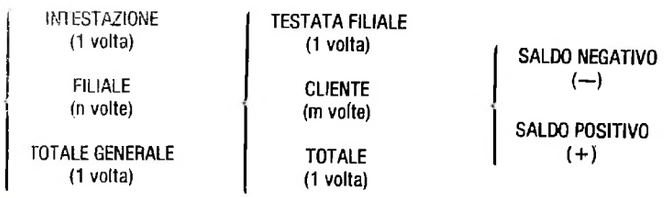


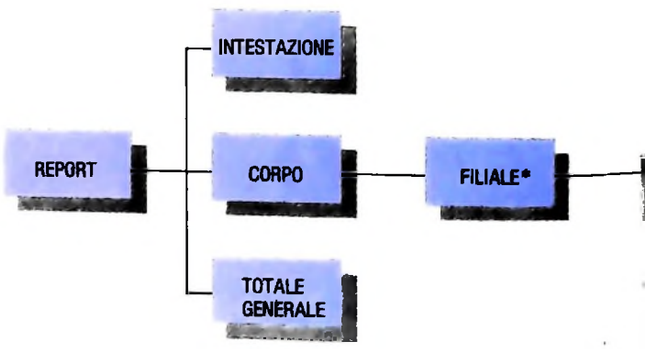
gramma si può strutturare anche gerarchicamente: la figura a pagina precedente ne illustra un esempio.
Tra le tecniche più significative di programmazione modulare, il cosiddetto metodo di *structured design*, (progettazione strutturata) dovuto a Yourdon, Myers e Constantine, è particolarmente interessante.
Il metodo fornisce un insieme di indicazioni di passi attraverso cui procedere, raggiungendo una struttura modulare in cui siano evidenziate tutte le relazioni di controllo e di dati tra i vari moduli. Nell'esempio riportato, un semplice PROGRAMMA CALCOLO richiama tre moduli di ACQUISIZIONE DATI, di SVOLGIMENTO CALCOLI e di EMISSIONE RISULTATI; esso riceve dal primo due dati A e B, che passa al secondo da cui riceve D ed E, che passa al terzo. Il modulo ACQUISIZIONE DATI, a sua volta, invoca altri moduli e così via. Il metodo della structured design pone l'attenzione su come i vari moduli sono stati individuati e progettati, e li esamina dal punto di vista di due attributi: la *coesione* e l'*accoppiamento*.
Per *coesione* s'intende il grado di legame che c'è tra le varie parti interne a un modulo: si individuano 7 livelli di coesione, di cui il più basso (coesione incidentale) evidenzia che non sono presenti validi motivi per avere messo insieme in un modulo un certo insieme di operazioni, mentre il più alto (coesione funzionale) evidenzia che in un modulo c'è tutto e solo ciò che è legato allo svolgimento di un'unica funzione (per esempio, la lettura dei dati da elaborare).
Un elevato grado di coesione di un modulo ne garantisce, in genere, la sua correttezza (meno confusione = meno errori), la sua manutenibilità (a fronte di modifiche al sistema, un solo modulo è coinvolto per una specifica funzione), la riusabilità in altri programmi.
Per *accoppiamento* s'intende invece il grado di legame che esiste tra due moduli diversi: poiché il nostro obiettivo è di costruire un prodotto facile da modificare e da manutenere, è opportuno che un modulo non sappia nulla su come sono fatti i moduli che richiama e che non sappia nulla su come sono fatti quelli da cui è richiamato; cioè è opportuno che l'accoppiamento tra i moduli sia minimo.
Anche in questo caso vengono individuati 7 livelli di accoppiamento, dal peggiore, in cui un modulo richiama un altro modulo sulla base della conoscenza fisica di come esso è fatto (in un linguaggio Assembler, il richiamo di un modulo saltandone la prima istruzione, che sappiamo effettuare operazioni che non ci interessano), al migliore, in cui un modulo comunica con un altro fornendogli un messaggio che corrisponde alla richiesta di effettuare un'operazione, senza che i due condividano neppure i dati da elaborare.

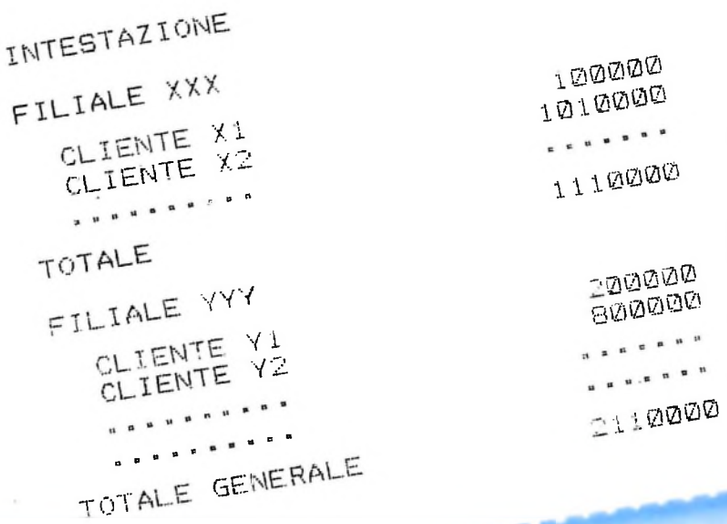
Programmazione in piccolo

Anche nell'ambito del singolo programma è importante la struttura architetturale di dettaglio; una buona strutturazione rende i programmi leggibili dai programmatori, facilita il reperimento di eventuali errori, permette di far sì che un programma possa essere preso in carico anche da un programmatore diverso da chi l'ha originariamente scritto.
La "programmazione strutturata", cioè la disciplina che impone lo sviluppo dei programmi top down con l'uso di strutture di controllo standard e di strutture di dati, è stata una prima risposta a queste esigenze; successivamente è stato messo a punto sulla base di tali principi un certo insieme di metodologie. Voler classificare tali metodologie è complesso; tuttavia si può fornire un'indicazione di massima degli obiettivi cui si rivolgono.
Possiamo grossolanamente classificare i problemi da risolvere come *facili* o *difficili*, intendendo con i primi quelli per i quali la definizione del problema contiene implicitamente la tecnica di risoluzione, e con i secondi quelli che richiedono, per la soluzione, un'attività creativa d'invenzione. Possiamo suddividere i problemi anche in *semplici* e *complessi*, intendendo con i primi quelli che hanno una ridotta quantità di informazioni da elaborare e comunque informazioni con correlazioni semplici, con i secondi quelli con molti dati, eventualmente correlati in modo molto complesso.
Un esempio di problema semplice e facile è la media di 100 valori: la definizione del problema contiene in sé, per una persona anche di ridotta scolarità, il modo con cui si deve raggiungere la soluzione.

La descrizione secondo Warnier





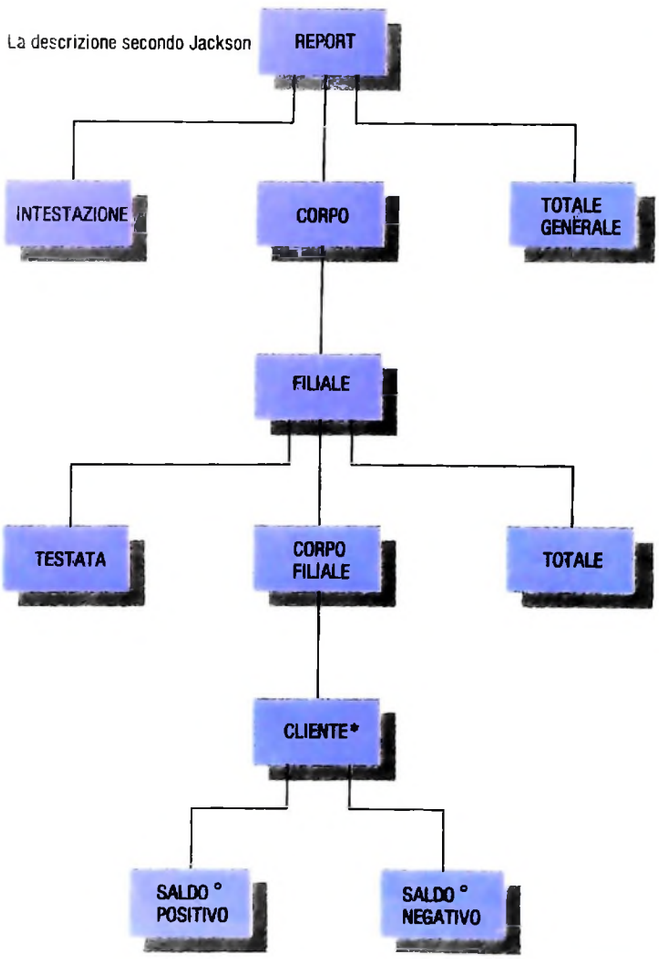


La rappresentazione dei dati in WARNIER, JACKSON e PHOS

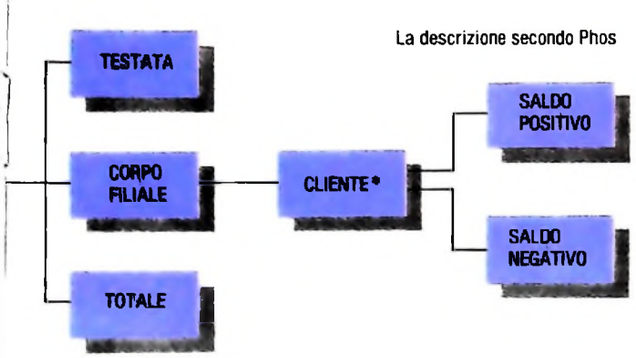
Un ipotetico resoconto di una società, che elenca il saldo contabile di tutti i clienti per tutte le filiali, con evidenziati i totali per ogni filiale oltre quello generale, secondo il metodo Warnier, Jackson e Phos. I saldi negativi sono evidenziati con la stampa del segno "—" che segue il valore.

Un problema semplice ma difficile può essere invece il calcolo del Massimo Comun Divisore tra due interi: la definizione di MCD, infatti, come insieme di tutti i fattori comuni di due numeri, presupporrebbe la scomposizione dei due numeri in fattori primi e una successiva ricerca, mentre esistono algoritmi estremamente efficaci ma poco intuitivi, dovuti, secondo la tradizione storica, alla creatività di Euclide (come quello illustrato nel riquadro della pagina seguente).
Un problema facile ma complesso può essere quello di una contabilità aziendale: infatti si tratta semplicemente di attribuire un certo insieme di valori a determinati insiemi, e poi effettuarne somme, fornendo informazioni che non presentano alcuna necessità d'invenzione; tuttavia la grande quantità di informazioni e la correlazione tra le stesse (un acquisto è registrato con riferimento a un fornitore, la cui descrizione potrà essere reperita in un complesso archivio; analogamente per una vendita; spesso i riferimenti sono effettuati mediante codici ecc.) rendono la soluzione di difficile dominio.
Infine, un esempio di problema difficile e complesso può essere quello della costruzione di un compilatore: si tratta qui di gestire una grande quantità di dati in generale estremamente correlati (il programma come insieme complesso di istruzioni con riferimenti da istruzione a istruzione, variabili,

La descrizione secondo Jackson



La descrizione secondo Phos



tipi ecc., e anche con riferimenti ad altri sottoprogrammi o ad altre entità esterne), con un compito tutt'altro che intuitivo: l'analisi sintattica e la traduzione in un linguaggio di più basso livello.
Si noti che spesso i problemi difficili sono stati molto studiati, così d'avere a disposizione modelli di soluzione che fanno rientrare i problemi in classi di algoritmi noti, e che quindi rischiano di diventare "facili": si pensi per esempio agli algoritmi di ordinamento.
Per affrontare i problemi facili e semplici non è necessaria alcuna specifica tecnica; per affrontare quelli difficili sono necessarie cultura e inventiva: tecniche come la programmazione strutturata facilitano procedimenti mentali ordinati e risultano quindi un valido supporto al reperimento della soluzione; per affrontare quelli facili ma complessi è invece possibile usare opportune metodologie, che guidino con precisione il programmatore alla costruzione del programma.
Tra le varie metodologie disponibili con tali caratteristiche alcune si pongono l'obiettivo di dedurre la struttura di dettaglio del programma a partire dalle caratteristiche dei dati che devono essere elaborati. In sostanza si tratta di descrivere i dati in ingresso e in uscita al programma in termini di:

- successione di elementi di tipo diverso
- alternativa tra elementi di tipo diverso
- ripetizione di elementi dello stesso tipo

e di associare al trattamento delle varie combinazioni di dati le adeguate strutture di elaborazione:

- la sequenza per i dati in successione
- la selezione per i dati in alternativa
- l'iterazione per i dati ripetuti.

Le tre metodologie più diffuse con tale approccio sono:

- *metodo Warnier*, che prende il nome del suo ideatore, il francese J.D. Warnier. Il metodo usa alcuni elementi della teoria degli insiemi per esaminare le strutture dei dati in ingresso e in uscita; quindi costruisce una struttura logica onnicomprensiva di tutti i dati in ingresso (indipendentemente dal fatto che si trovino su file diversi) e analogamente si comporta per i risultati in uscita. Da queste strutture deriva il flow chart del programma, che poi riempie con le specifiche istruzioni di lettura, scrittura, gestione dei file, delle stampe, dei contatori ecc., lasciando al programmatore l'inserimento delle sole istruzioni relative al calcolo che deve essere effettuato dal programma (cioè quelle istruzioni che dipendono dallo specifico problema e su cui il metodo non può evidentemente pronunciarsi, come, per esempio, l'aliquota IVA da applicare). Il metodo è molto preciso e garantisce ottimi risultati, anche se spesso risulta leggermente laborioso e impositivo.
- *metodo Jackson*: dovuto all'inglese M. A. Jackson. Si differenzia dal precedente per puntare a una maggiore intuitività, lasciando una certa autonomia al programmatore. Il metodo parte dalla descrizione grafica mediante alberature di tutti i file in ingresso e in uscita presi uno per uno, e chiede al programmatore di trovare una struttura di programma che si sovrapponga, almeno parzialmente, a ciascuna delle singole alberature; quindi richiede una codifica del programma in un opportuno pseudolinguaggio e fornisce indicazioni per l'inserimento delle operazioni base, esattamente come nel metodo Warnier, ma senza una confrontabile decisione. Sono a disposizione anche "precompilatori" dello pseudolinguaggio, che trasformano il programma così scritto in linguaggio COBOL. È senza dubbio il metodo più adottato.
- *metodo PHOS*: sviluppato nell'ambito di una collaborazione tra l'Università di Milano e la Honeywell Information System Italia, deriva il suo nome dal fatto che ottiene la struttura del programma a partire dalla struttura dei risultati (Program Hierarchy from Output Structure); in seguito confronta la struttura così ottenuta con quella dei dati in ingresso, classificando un certo insieme di situazioni e fornendo ricette di intervento per adattare la struttura precedente ai dati da elaborare. Non fa uso di pseudolinguaggi e fornisce, come i due precedenti, indicazioni sulle istruzioni da inserire nel programma per gestire file, contatori, stampe ecc.

Metodi come quelli accennati guidano fortemente il programmatore, cosicché programmi scritti da programmatori diversi si assomigliano molto: ciò aumenta la possibilità di "trasportare" un programma da un programmatore a un altro, e finisce per costituire un vero e proprio standard culturale e di linguaggio all'interno dell'azienda.

Un esempio di problema semplice ma difficile

Algoritmo di Euclide per il calcolo del Massimo Comun Divisore tra due numeri.
L'algoritmo, scritto in uno pseudolinguaggio di programmazione, è il seguente:

```
leggi A e B
WHILE A<>B DO
    IF A>B THEN A=A-B
           ELSE B=B-A
    ENDIF
REPEAT
stampa A
```

Infatti, dati per esempio i due valori 70 e 21, si svolgono i seguenti passaggi:

```
A=70 B=21 diversi
A>B e quindi A=49
A=49 B=21 diversi
A>B e quindi A=28
A=28 B=21 diversi
A>B e quindi A=7
A=7  B=21 diversi
A<B e quindi B=14
A=7  B=14 diversi
A<B e quindi B=7
A=7  B=7  uguali
stampa il risultato: 7
```

Il metodo, estremamente semplice da realizzare e veloce da eseguire, è basato sulla seguente proprietà del Massimo Comun Divisore:

MCD (A,B)=MCD (A,abs (A–B))

che non è assolutamente intuitiva.

IL LINGUAGGIO LISP (III)

Concludiamo la panoramica sul linguaggio LISP perfezionando l'esempio del dizionario.

Riprendiamo l'esempio del dizionario apportandovi miglioramenti ed estensioni. Anche se l'esempio in sé non ha una rilevanza assoluta (esistono metodi ben più efficienti e completi per gestire un dizionario automatico che debba essere usato nella pratica quotidiana) ci è utile per illustrare i metodi e gli approcci che possono essere usati in ambiente LISP nello sviluppo di sistemi ben più complessi e sofisticati.
Una prima estensione che può essere apportata al dizionario permette una più facile interazione con lo stesso. Si tratta di integrare le funzioni che abbiamo scritto realizzando la parte del sistema che interagisce con l'utente, domandandogli che operazione intende richiedere e su quale vocabolo.
Vediamo prima quali funzioni mette a disposizione il LISP per comunicare con l'ambiente esterno. Una di queste è già stata usata: si tratta della PRINT. Questo comando permette di scrivere sul terminale il valore di un'espressione LISP. Per esempio: (PRINT 'print) emetterà sul terminale il simbolo print. L'intera espressione (PRINT 'print) ritornerà come valore lo stesso simbolo print. Similmente (PRINT (CAR '(Diego Armando Maradona))) causerà la stampa del simbolo Diego, che è il valore dell'espressione argomento della PRINT, e ritornerà come valore ancora Diego. Nei sistemi LISP in uso corrente, esistono diverse altre funzioni che permettono un accurato controllo delle stampe, ma nel nostro caso la PRINT è sufficiente. Per quanto riguarda l'input la funzione di base è la READ. La sua invocazione ritorna come valore la prima espressione LISP che viene immessa al terminale. L'esecuzione del programma che contiene una chiamata alla READ viene sospesa fino al momento in cui l'utente termina di digitare l'espressione attesa.
Per esempio la valutazione di (CDR (READ)) farà sì che il sistema attenda l'immissione di una nuova espressione. Se l'utente scrive l'espressione (Paulo Roberto Falcao) il valore dell'intera espressione è (Roberto Falcao).
Veniamo ora all'estensione del dizionario. Per questo definiamo una funzione top-level che richiederà all'utente nell'ordine il vocabolo e l'operazione richiesta. La funzione si riferisce a un dizionario globale denominato DIZIONARIO.

```
(SETQ top-level
   (LAMBDA ()
      (PRINT"Da quale lingua?")
      (SETQ lingua (READ))
      (PRINT"Quale vocabolo?")
(SETQ vocabolo (READ))
```

```
(COND ((EQ lingua 'inglese)
       (PRINT
         (traduzione-italiana vocabolo DIZIONARIO)))
      ((EQ lingua 'italiano)
       (PRINT
         (traduzione-inglese vocabolo DIZIONARIO)))
      (True"(PRINT"Lingua sconosciuta")))
(PRINT"Un'altra traduzione?")
(COND ((EQ (READ) 'si)
       (top-level))
       (True"(PRINT"Arrivederci caro")))))
```

Qualche parola di spiegazione. All'inizio vengono richieste le informazioni necessarie [linee 3, 4, 5, 6] che vengono conservate associandole a un nome. Quindi viene presa una decisione sulla funzione di traduzione da invocare in base a quanto è stato specificato dall'utente [linee 7-10]. Dopo aver invocato la funzione opportuna riferendosi al vocabolario globale, l'intero procedimento viene ripetuto, sempre che ciò sia desiderato dall'utente [linee 13-15]. È interessante notare come la ripetizione della sequenza di operazioni sia ottenuta tramite una ricorsione, laddove con un linguaggio tradizionale sarebbe stato necessario usare o una coppia label-goto, oppure un apposito costrutto di iterazione, come per esempio un while o un for in Pascal.
Questa estensione al programma non presenta di per sé caratteristiche particolarmente originali; infatti il modo di procedere nelle sequenze domanda-risposta è molto simile a quello che si sarebbe adottato con un linguaggio tradizionale. Tuttavia è possibile modificare ulteriormente l'esempio in modo da sfruttare le particolari caratteristiche del LISP. L'idea è quella di utilizzare la simmetria riconoscibile nella struttura che abbiamo scelto per rappresentare il dizionario. Vogliamo unificare le due funzioni di traduzione traduzione-inglese e traduzione-italiana in un'unica funzione generale di ricerca. Un metodo informale per generalizzare definizioni di funzioni, in modo che il comportamento dipenda più sensibilmente dal modo in cui vengono chiamate, è quello di mettere tra gli argomenti della definizione alcune delle funzioni che vengono invocate internamente. La nuova funzione traduzione riportata sotto, per esempio, è indipendente dalla lingua di partenza. Questa è determinata dall'elemento delle coppie (vocabolo-italiano vocabolo-inglese) che viene considerato rispettivamente durante la ricerca e l'ac-

cesso. Tali operazioni vengono realizzate per mezzo di oculate combinazioni di invocazioni delle funzioni **inglese** e **italiano**. Per rendere la nuova funzione di traduzione indipendente dalla lingua sarà quindi necessario aggiungere agli argomenti della funzione le procedure di accesso, cosicché è compito di chi invoca la procedura specificare di volta in volta le opportune funzioni di accesso e quindi il comportamento desiderato.

```
(SETQ traduzione
      (LAMBDA (da a vocabolo diz)
        (COND ((NULL diz) (PRINT vocabolo)
                          (PRINT"?"))

              ((EQ (da (CAR diz)) vocabolo)
                   (a (CAR diz)))
              (True (traduzione da a vocabolo (CDR diz))))))
```

La struttura della funzione **traduzione** è sostanzialmente identica a quella di **traduzione-italiana** e **traduzione-inglese**, con la differenza che dove in esse venivano invocate in modo prefissato le funzioni **italiano** e **inglese**, ora compaiono i nomi di funzioni che saranno passati come argomento. Così per ottenere un comportamento equivalente a quello di **traduzione-inglese** sul vocabolo **due** è sufficiente invocare **traduzione** in questo modo:

(traduzione italiano inglese 'due DIZIONARIO)

dove **italiano** e **inglese** sono i nomi cui sono associate le definizioni delle funzioni descritte nel precedente articolo. Per simulare il comportamento di **traduzione-italiana** potremo invece scrivere:

(traduzione inglese italiano 'two DIZIONARIO)

L'esempio mostra come in LISP sia possibile trattare in modo omogeneo dati e programmi: in questo caso le due funzioni di accesso sono trattate come normali espressioni, alla stessa stregua di 'due, all'invocazione della funzione di traduzione, e come vere funzioni nel corso della valutazione. Questa è una delle principali caratteristiche del LISP e viene ampiamente sfruttata nella costruzione di sistemi anche complessi. Risulta particolarmente utile, per esempio, poter associare definizioni di funzioni non soltanto a nomi ma anche a proprietà da essi possedute. Un tipico caso è quello di **top-level**, che deve scegliere in che modo invocare la funzione **traduzione** al fine di ottenere la traduzione nella corretta direzione. La decisione viene presa in base al simbolo che l'utente specifica come lingua dalla quale tradurre.
Vediamo come.

```
(SETQ top-level
   (LAMBDA ()

     (PRINT"Da quale lingua?")
     (SETQ lingua (READ))

     (PRINT"Quale vocabolo?")
     (SETQ voc (READ))

     (COND ((EQ lingua 'inglese)
            (PRINT

              (traduzione inglese italiano voc DIZIONARIO)))
           ((EQ lingua 'italiano)
            (PRINT
              (traduzione italiano inglese voc DIZIONARIO)))
           (True"(PRINT"Lingua sconosciuta")))
     (PRINT"Un'altra traduzione?")

     (COND ((EQ (READ) 'si)
            (top-level))

           (True"(PRINT"Arrivederci caro")))))
```

La sostanziale differenza rispetto alla versione precedente è proprio nelle righe 9 e 12. Al posto di **traduzione-inglese** e **traduzione-italiana** abbiamo una invocazione della funzione **traduzione** con gli argomenti **inglese** e **italiano** e **italiano** e **inglese** rispettivamente.
Il simbolo letto determina quali funzioni passare come argomento alla **traduzione**. Sarebbe quindi naturale associare al simbolo stesso, che denota la lingua d'origine, direttamente le definizioni delle funzioni che devono essere comunicate alla **traduzione**. Un potente mezzo per rappresentare associazioni di questo tipo è costituito dalle **property list**. Nel nostro caso possiamo comunicare al sistema queste informazioni con le seguenti espressioni, dove la funzione **put** è stata esaminata negli articoli precedenti.

```
(PUT 'inglese 'DA '(LAMBDA (voce) (secondo voce)))
(PUT  'inglese 'A '(LAMBDA (voce) (CAR voce)))
(PUT  'italiano 'DA '(LAMBDA (voce) (CAR voce)))
(PUT  'italiano 'A  '(LAMBDA (voce) (secondo voce)))
```

A questo punto per rintracciare le corrette primitive di accesso, è sufficiente riferirsi alla lingua d'origine e al tipo di accesso (**DA** o **A**).
Di conseguenza la funzione **top-level** si semplifica concettualmente e si generalizza con la nuova definizione.

```
(SETQ top-level
   (LAMBDA ()

     (PRINT"Da quale lingua?")
     (SETQ lingua (READ))

     (PRINT"Quale vocabolo?")
     (SETQ voc (READ))

     (PRINT (traduzione
                  (GET lingua 'DA)
                  (GET lingua 'A)
                  voc

                    DIZIONARIO))
     (PRINT"Un'altra traduzione?")

     (COND ((EQ (READ) 'si)
            (top-level))

           (True"(PRINT"Arrivederci caro")))))
```

Per comprendere il funzionamento di questa funzione vediamo come procede la traduzione di **three**. Il vocabolo viene indicato dall'utente quando richiesto e associato al nome **voc**. La stessa cosa avviene per **inglese** associato a **lingua**. A questo punto viene direttamente invocata **traduzione**. I parametri opportuni vengono individuati valutando le invo-

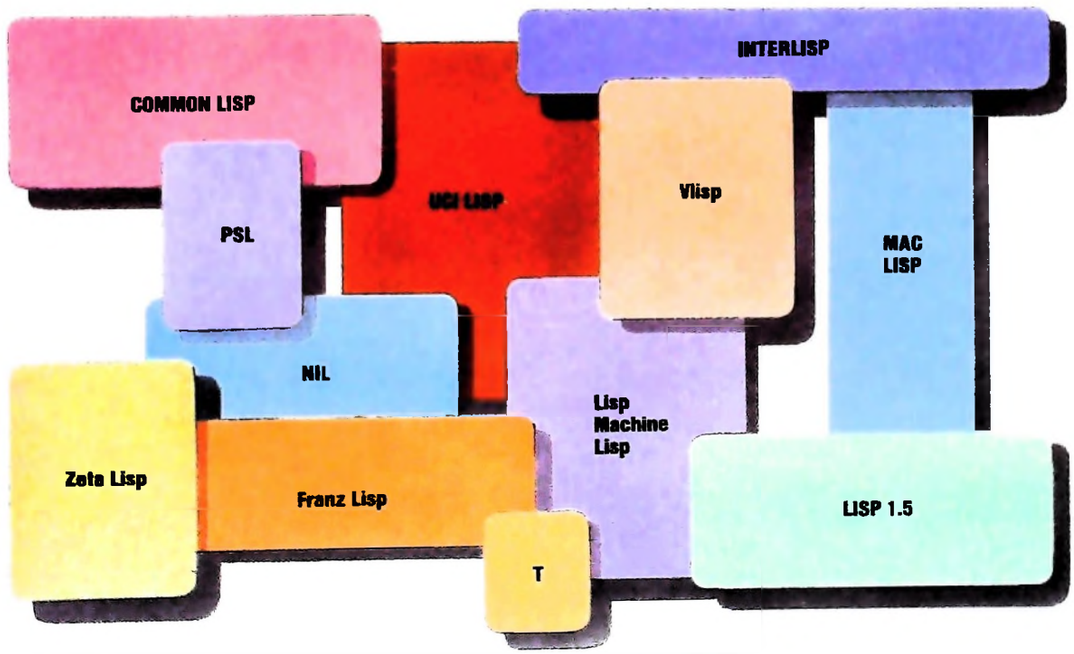
cazioni di GET, cioè accedendo alle proprietà del simbolo che identifica la lingua d'origine. La prima espressione ritornerà la funzione che accede al secondo elemento di una voce di dizionario (la parte inglese) (LAMBDA (voce) (secondo voce)); la seconda quella che accede alla parte italiana della voce (LAMBDA (voce) (CAR voce)). Traduzione viene così invocata correttamente senza che top-level debba incaricarsi di esaminare tutti i possibili casi di richiesta.
Un'ulteriore estensione di questo semplice esempio può essere utile per mostrare le caratteristiche di flessibilità del LISP e per prendere contatto con una delle tecniche di programmazione più diffuse tra gli utilizzatori di questo linguaggio. Si tratta di fare in modo che sia possibile aumentare dinamicamente il numero di informazioni contenute nel vocabolario mentre lo si usa. La funzione traduzione si incaricherà di verificare se il vocabolo di cui è richiesta la traduzione è contenuto nel vocabolario corrente ed eventualmente di richiedere all'utente l'informazione mancante.

```
(SETQ traduzione
      (LAMBDA (da a vocabolo diz)
        (COND ((NULL diz) (aggiungi vocabolo)
              ((EQ (da (CAR diz)) vocabolo)
                   (a (CAR diz)))
              (True (traduzione da a vocabolo (CDR diz))))))
(SETQ aggiungi
      (LAMBDA (vocabolo)
```

```
(PRINT"Non e' nota la traduzione di")
(PRINT vocabolo)
(PRINT"Il vocabolo in italiano?")
(SETQ ITALIANO (READ))
(PRINT"e in inglese?")
(SETQ INGLESE (READ))
(SETQ DIZIONARIO
      (nuova-voce
            (CONS ITALIANO (CONS INGLESE ()))
            DIZIONARIO))))
```

L'aspetto significativo di questa nuova versione dell'esercizio è ancora una volta la semplicità con la quale possono venire espresse le manipolazioni simboliche. Infatti, mentre utilizzando linguaggi più tradizionali è necessario prevedere in modo dettagliato le risorse di memoria che saranno necessarie al buon funzionamento del programma in ogni possibile caso, con il LISP il problema della gestione di tali risorse è automaticamente risolto dal sistema in modo dinamico.
Se il programma di traduzione nella sua ultima versione fosse stato scritto in un linguaggio tradizionale, quale per esempio il Pascal, avremmo dovuto dichiarare all'inizio del programma una struttura dati (per esempio un vettore costituito da record adatti a rappresentare ciascuno una coppia vocabolo-traduzione) che costituisse l'equivalente del DIZIONARIO che abbiamo sinora usato. Naturalmente non ci sono

Alcuni sistemi LISP esistenti



particolari difficoltà in questo modo di procedere, salvo che il vettore Pascal deve essere dimensionato a priori, ed è quindi necessario prevedere con buona precisione la dimensione massima raggiungibile dal dizionario; inoltre questo vettore viene comunque allocato e occupa spazio di memoria, anche se il dizionario contiene effettivamente soltanto tre o quattro vocaboli.
Una diversa possibilità è quella di gestire in modo dinamico l'allocazione e deallocazione di memoria, in modo da mantenere in ogni istante soltanto le strutture che rappresentano realmente una voce di dizionario: la difficoltà è in questo caso la gestione esplicita delle operazioni di richiesta e rilascio di memoria, che è completamente a carico del programmatore (questi avrà a disposizione in Pascal funzioni quali new e dispose).
Tutto ciò non deve invece preoccupare il programmatore LISP. È infatti il sistema stesso che provvede alla gestione automatica della memoria; questa viene allocata opportunamente a ogni applicazione della funzione CONS ai suoi argomenti, in modo da creare la nuova lista richiesta. Chi scrive programmi non deve occuparsi di questi dettagli.
È utile infine ricordare che non si può parlare di un "linguaggio LISP" standard nello stesso modo in cui si parla di "Pascal standard". È più giusto parlare di "sistemi LISP", dove ogni sistema può avere caratteristiche anche sostanzialmente diverse da quelle di altri sistemi. In questa serie di articoli abbiamo delineato le caratteristiche comuni alla maggior parte dei LISP esistenti: un sistema reale fornirà, oltre a tali primitive di base (i cui nomi peraltro possono essere diversi da quelli usati qui), un ricco ambiente di programmazione con caratteristiche autonome che devono essere di volta in volta esplorate.

Glossario

Bit di parità. *Un bit in più che viene aggiunto a un byte, al fine di poter controllare se i contenuti del byte sono stati trasmessi o immagazzinati correttamente. Ove la parità sia di tipo dispari, il bit di parità viene fissato a 1 o a 0, secondo i casi, in modo che la somma dei bit che costituiscono il byte risulti un numero dispari; in caso la parità scelta sia invece di tipo pari, si fissa il bit di parità a 1 o a 0 in modo che la somma dei bit che costituiscono il byte sia un numero pari. Se, in un controllo successivo, i bit di un byte, sommati al relativo bit di parità, non danno somma pari (o dispari) secondo quanto determinato dal tipo di parità scelto, si può sapere che è sopravvenuto un errore.*

Bit di stop. *Nella trasmissione di dati, bit usato per la sincronizzazione, che segue ogni byte trasmesso in modo asincrono attraverso una porta seriale. Alcuni sistemi richiedono anche più di un bit di stop, ma ambedue gli estremi di ciascun collegamento di comunicazione debbono utilizzare lo stesso numero di bit di stop, che vengono automaticamente aggiunti alla sorgente ed eliminati alla destinazione da appositi circuiti hardware.*

Delimitatore. *Un carattere usato per indicare l'inizio o la fine di una stringa di caratteri. In un programma di word processing, per esempio, una coppia di delimitatori potrebbe servire per identificare l'inizio e la fine di un blocco di testo che deve essere cancellato, trasferito o copiato. I delimitatori stessi non fanno parte della stringa.*

EAROM - *Sigla di Electrically Alterable Read-Only Memory, memoria a sola lettura elettricamente modificabile. Un tipo di memoria a sola lettura (ROM) che può essere cancellata e riprogrammata più volte. È simile alla EPROM, ma viene cancellata mediante corrente elettrica, anziché mediante luce ultravioletta. È una memoria costosa, usata in applicazioni particolari, come il controllo di macchine utensili, dove è periodicamente necessario modificare completamente i programmi di base.*

joystick. *Un dispositivo di input che ha la forma di una piccola leva o di una barra, montata su un'apposita base. La leva può essere spostata in tutte le direzioni e il suo movimento viene trasformato in impulsi elettrici, interpretati dall'unità di elaborazione come comandi di movimento del cursore o di qualunque altro simbolo predefinito sullo schermo. Il joystick è ampiamente usato come dispositivo di controllo nei giochi (può essere munito anche di un pulsante per le operazioni di "fuoco") per i piccoli calcolatori, ma è usato anche, in campo professionale, come dispositivo di input, al posto della tastiera, per esempio, per programmi di grafica e di progettazione assistita dal calcolatore.*

Linker. *Un programma che prende file di programma in linguaggio macchina prodotti da compilatori o da assemblatori e li collega (*link *in inglese significa 'collegare') in un unico file di programma eseguibile.*

Multielaborazione - *Situazione in cui un sistema di calcolo comprende più unità di elaborazione che lavorano in parallelo sotto la supervisione di un unico sistema operativo, condividendo unità di memoria e dispositivi di ingresso e uscita, e con organi di comunicazione che consentono il passaggio di dati da ciascuna unità di elaborazione alle altre. I vantaggi della multielaborazione sono la maggiore capacità complessiva del sistema (che pertanto può eseguire compiti più complessi) e la sua maggiore "disponibilità": un guasto a una delle unità di elaborazione infatti non blocca automaticamente tutto il sistema.*

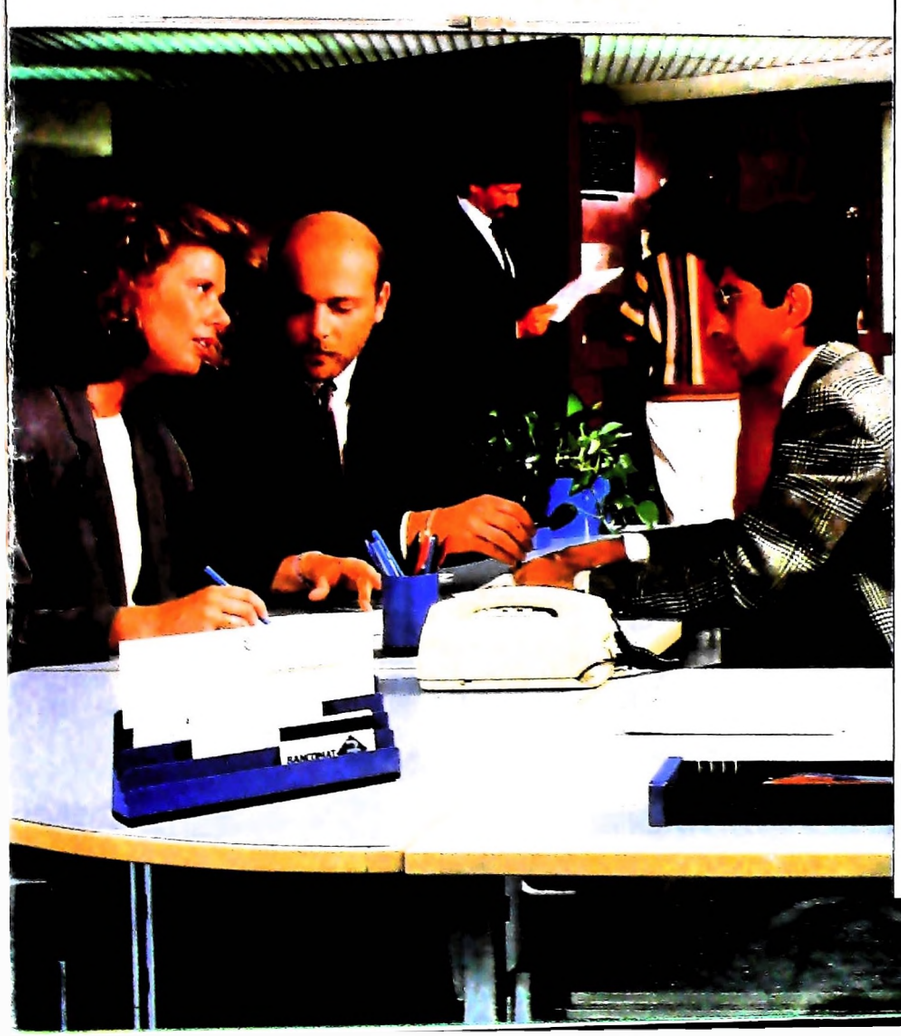
Pascal - *Linguaggio di programmazione di alto livello sviluppato nel 1969 da Niklaus Wirth, pensato originariamente come linguaggio didattico per l'insegnamento della programmazione come attività sistematica, che ha poi trovato larga applicazione al di fuori dell'insegnamento, anche sui piccoli calcolatori personali. È un linguaggio ricco ed elegante, orientato specificamente alla programmazione strutturata, che permette anche la realizzazione di programmi ricorsivi.*

PL/I - *Sigla di Programming Language/One. Linguaggio di programmazione di alto livello, sviluppato negli anni 1963-64 da un comitato fondato dalla IBM. Si proponeva come linguaggio universale, adatto sia per la programmazione scientifica, sia per quella commerciale, sia ancora per la programmazione di sistemi. Ha preso caratteristiche dai tre maggiori linguaggi precedenti, FORTRAN, ALGOL e COBOL, ma ha avuto un successo limitato e soprattutto grazie al coinvolgimento diretto della IBM nella sua definizione.*

PEEK. *Un'istruzione del BASIC e di alcuni altri linguaggi di programmazione, che consente all'utente di conoscere i contenuti di una specifica locazione di memoria della macchina.*

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

PERSONAL COMPUTER OLIVETTI M10

L'UFFICIO DA VIAGGIO

Anche in leasing con Olivetti Leasing.

olivetti

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

NOME/COGNOME

VIA/N.

CAP/CITTÀ

TELEFONO