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

30 CORSO PRATICO COL COMPUTER

421826

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

F4 F5 F6 F7 F8 BATTERY LOW

diretta 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
Eidos (Bruno Motta, Carmine Stragapede), UGO SALVI, GIANCARLO MAURI, 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

IL LINGUAGGIO COBOL (II)

Ancor oggi è uno tra i più usati linguaggi nonostante le sue strutture di controllo non siano completamente congruenti con i principi della programmazione strutturata.

Astrazione nelle strutture di controllo

Il linguaggio COBOL si presenta estremamente ricco di strutture di controllo, anche se queste non sono completamente congruenti con i principi della programmazione strutturata.
Innanzitutto, è disponibile l'istruzione di salto GO TO, che permette di trasferire il controllo all'etichetta di un paragrafo, secondo la forma:

GO TO nome-paragrafo.

È presente una struttura IF che ha la forma:

IF condizione istruzione-1 ELSE istruzione-2

ove la clausola ELSE è opzionale.

Interessante su questa struttura è l'uso della locuzione NEXT SENTENCE, che indica di proseguire con la parte di programma che direttamente segue la struttura; così:

```
IF A>B NEXT SENTENCE ELSE GO TO PAR
```

corrisponde a una struttura IF... THEN... ELSE in cui, invece di essere vuota la seconda alternativa, è vuota la prima.
Le strutture iterative sono invece legate esclusivamente all'istruzione PERFORM. Evidentemente il programmatore COBOL sarà in grado di costruire qualunque struttura iterativa con l'uso di istruzioni IF e GO TO, ma le uniche strutture messe effettivamente a disposizione dal linguaggio hanno la forma:

```
PERFORM nome-paragrafo condizioni di iterazione.
```

L'istruzione PERFORM non è necessariamente legata a un'iterazione; essa può essere considerata un normale richiamo di un sottoprogramma, e l'assenza di indicazioni sull'iterazione implica che l'esecuzione del paragrafo avvenga una volta sola.
Inoltre, come abbiamo visto nell'esempio, è possibile adottare l'espressione:

```
PERFORM nome-par-1 THRU nome-par-2
```

che esegue le istruzioni comprese tra le due etichette; ciò è particolarmente utile quando le istruzioni da eseguire contengono al loro interno ulteriori etichette che spezzano il paragrafo in più paragrafi da un punto di vista della sintassi, pur corrispondendo a una trasformazione unitaria.
Le condizioni di iterazione esprimibili sono molteplici; ne forniamo qui alcuni esempi:

```
PERFORM P 10 TIMES
```

effettua l'esecuzione del paragrafo P 10 volte di seguito;

```
PERFORM P UNTIL X=Y
```

effettua l'iterazione del paragrafo fino al momento in cui la condizione X=Y risulti vera;

```
PERFORM P VARYING I FROM 1 BY 2 UNTIL I>100
```

effettua l'iterazione enumerativa di P facendo variare I da 1 a 101, incrementando ogni volta I di 2.
Si osservi che l'istruzione è più generale che non una semplice iterazione enumerativa, in quanto la condizione espressa non è necessariamente legata al valore del contatore; per esempio, la forma:

```
PERFORM P VARYING I FROM 1 BY 1
UNTIL TROVATO = "VERO".
```

indica la presenza di un indice, ma non corrisponde a un'istruzione enumerativa, in quanto non è noto a priori il numero di iterazioni da effettuare: in generale le condizioni possono essere molteplici, legate da AND e OR.

```
PERFORM P VARYING I FROM 1 BY UNTIL I>10
AFTER J FROM 1 BY 1 UNTIL J>10
```

effettua l'iterazione enumerativa su I, e all'interno di questa effettua l'iterazione su J, tipicamente per la scansione di una tabella 10 per 10.

SECOR ALT # — S O U R C E L I S T I N G — COBOL REF # — SECOB & COBOL COMMENTS — SCOB REF#

```
00304        $PROC STAMPA.                                                                  REF BY  #00288
00305       *- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
00306       * STAMPA
00307       *          CERCA DESCRIZIONE DEL PRODOTTO IN TABELLA; CALCOLA IL
00308       *          TOTALE DELL'IMPORTO DI TUTTI I MOVIMENTI; STAMPA L'INTE-
00309       *          STAZIONE DEL MODULO E STAMPA I TOTALI
00310       *- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
00311       * COND. IN: ARTICOLO DA STAMPARE; PRIMO RECORD MOVIMENTO LETTO
00312       *
00313       * COND.OUT:ARTICOLO TRATTATO;STAMPE EFFETTUATE;NUOVO RECORD LETTO
00314       *- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
00315       *
00316        $DD
00317       *
00318        01  RIGA-TOT REDEFINES/RIGA.                                                    CONTAINS     38 CHARACTERS     6 WORDS
                                                                                           REF BY   00288  00289
00319          02 TOT-ART-ST              PIC B(2)9(11).                                     STARTS IN CHARACTER POSITION      1
                                                                                           REF BY   00286
00320          02 IVA-ST                  PIC B(4)9(11).                                     STARTS IN CHARACTER POSITION     14
                                                                                           REF BY   00287
00321          02 FILLER                  PIC X(6).                                          STARTS IN CHARACTER POSITION     29
00322        01  IND-EL-TAB               PIC 99.                                          CONTAINS      2 CHARACTERS     1 WORDS
                                                                                           REF BY   00251  00261  00263  00274  00287
00323       *
00324        01  TOTALE                   PIC 9(11).                                       CONTAINS     11 CHARACTERS     2 WORDS
                                                                                           REF BY   00252  00275  00286  00287
00325       *
00326        01  TROVATO-IN-TAB           PIC X.                                           CONTAINS      1 CHARACTERS     1 WORDS
                                                                                           REF BY   00253  00258  00263
00327          88 PRESENTE-IN-TAB VALUE IS "Y".
00328        01  RTAB REDEFINES/TABELLA-ARTICOLI.                                          CONTAINS    850 CHARACTERS   142 WORDS
00329          03  EL-TAB-ART  OCCURS 50.
00330              04  TIPO-REC-TAB        PIC X.                                         STARTS IN CHARACTER POSITION      1
00331              04  COD-ART-TAB         PIC X(8).                                      STARTS IN CHARACTER POSITION      2
                                                                                           REF BY   00263
00332              04  PRU-TAB             PIC 9(6).                                      STARTS IN CHARACTER POSITION     10
                                                                                           REF BY   00274
00333              04 PERC-IVA-TAB         PIC 99.                                        STARTS IN CHARACTER POSITION     16
                                                                                           REF BY   00287
00334       *
00335        01  IMPONIBILE               PIC 9(10).                                       CONTAINS     10 CHARACTERS     2 WORDS
                                                                                           REF BY   00274  00275
00336       *
00337          02 DESCRIZ REDEFINES/REC-MOV     DESCR.
00338              03 COD-MOV              PIC X(8).                                      STARTS IN CHARACTER POSITION     26
00339              03 QTA                  PIC 9(4).                                      STARTS IN CHARACTER POSITION     34
                                                                                           REF BY   00274
00340              03 FILLER               PIC X(18).                                     STARTS IN CHARACTER POSITION     38
00341        $PD
00342          $DO INIZIAL-TOT                                                  P00250  REF TO  #00352
00343          $ITERUNTIL (PRESENTE-IN-TAB)                                      00257
                                                                                         REF TO   00086  00266
00344            $DO TR-ELEM-TAB                                                P00260  REF TO  #00356
00345          $REPEAT
00346          $ITERUNTIL OK-ARTICOLO NOT EQUAL DEP-K-ART OR EOF-MOV)            00270  REF TO   00035  00079
                                                                                         REF TO   00058  00279
00347            $DO TR-MOV                                                     P00273  REF TO  #00361
00348          $REPEAT
00349          $PERFORM TR-INTESTAZ                                                    REF TO  #00373
00350          $DO STAMPA-TOT                                                  P00265  REF TO  #00366
00351        $COR
00352        INIZIAL-TOT.                                                      REF BY  #00342
00353            MOVE ZERO TO IND-EL-TAB                                       REF TO   00084                                  00251
00354            MOVE ZERO TO TOTALE                                           REF TO   00085                                  00252
00355            MOVE "N" TO TROVATO-IN-TAB.                                   REF TO   00086                                  00253
00356        TR-ELEM-TAB.                                                      REF BY  #00344
00357            ADD 1 TO IND-EL-TAB.                                          REF TO   00084                                  00261
00358            IF DEP-K-ART = COD-ART-TAB (IND-EL-TAB)                                                                       00262
00359               MOVE "Y" TO TROVATO-IN-TAB.                                REF TO   00068  00079  00084  00086            00263
                                                                                *    EF FORMAT OR RADIX CONVERSION REQUIRED
                                                                               ON SUBSCRIPT
00360       *
00361        TR-MOV.                                                           REF BY  #00347
00362            MULTIPLY PRU-TAB (IND-EL-TAB) BY QTA GIVING IMPONIBILE        REF TO   00039  00069  00084  00086            00274
00363            ADD IMPONIBILE TO TOTALE                                      REF TO   00085  00088                          00275
00364            READ RMOV AT END MOVE "Y" TO IND-EOF-MOV.                                                                    P00276
                                                                               REF TO   00058
00365       *
00366        STAMPA-TOT.                                                       REF BY  #00350
00367            MOVE TOTALE TO TOT-ART-ST                                     REF TO   00045  00085                          00286
00368            COMPUTE IVA-ST = (TOTALE * PERC-IVA-TAB (IND-EL-TAB)) / 100   REF TO   00046  00070  00084  00085            00287
00369            WRITE RIGA-TOT                                                REF TO   00041  00044                          00288
00370            MOVE SPACES TO RIGA-TOT                                       REF TO   00044                                 00289
00371            ADD 1 TO CONTA-RIGHE.                                         REF TO   00057                                 00290
00372        $ENDPROC
```

COBOL e programmazione strutturata

Con il diffondersi della programmazione strutturata numerose critiche sono state fatte ai linguaggi di programmazione che non erano orientati a tale disciplina.
Ciò ha comportato una certa diffusione di precompilatori, cioè di programmi che accettavano come ingresso un programma contenente, oltre le frasi del linguaggio usato, anche altre frasi non esistenti nel linguaggio e tipiche della programmazione strutturata, che fornivano come uscita un programma equivalente al precedente, in cui tutte le istruzioni "estese" erano state tradotte nel linguaggio di partenza. È stato così possibile adottare principi strutturati anche in COBOL, proprio mediante tali precompilatori.
Un esempio di precompilatore per la programmazione strutturata top down è costituito dal programma SECOB, messo a punto dall'Istituto di Cibernetica dell'Università di Milano, dal Credito Italiano e dalla Banca Provinciale Lombarda.

L'esempio che riportiamo ne mostra le caratteristiche fondamentali, che si possono riassumere in:

- organizzazione di un programma in "moduli" (quello riportato è uno di tali moduli), richiamabili con PERFORM
- ogni modulo ha una propria DATA DIVISION (individuata dalla parola chiave $DD (riga 316), una propria PROCEDURE DIVISION (individuata da $PD alla riga 341) e una parte di istruzioni COBOL (individuata da $COB alla riga 351)
- la parte $PD contiene solo frasi estese che definiscono la struttura del programma
- la frase $DO richiama una porzione COBOL presente nella parte $COB
- la frase SPERFORM richiama un altro modulo
- l'iterazione ha la forma $ITERUNTIL...$REPEAT
- altre strutture di controllo sono disponibili
- la parte $COB contiene paragrafi COBOL direttamente richiamabili con il verbo $DO.

Si noti la doppia numerazione che evidenzia la passata di precompilazione: per esempio, la riga 346 del testo corrisponde effettivamente alla 346-esima linea inserita, ma nella precompilazione risulterà la 270-esima linea del testo del programma COBOL fornito al compilatore COBOL (come è evidenziato dal numero che segue l'istruzione).
Si noti anche la possibilità di raffinamenti successivi dei dati: la linea 318 ridefinisce con maggior dettaglio la struttura del campo RIGA, definito precedentemente in moduli di livello più alto.

Modularità e sottoprogrammi

Abbiamo visto come un programma COBOL abbia una PROCEDURE DIVISION che contiene la parte di istruzioni. Questa può essere suddivisa in ulteriori SECTION che presentano le seguenti caratteristiche:

- ognuna di esse è dichiarata da una frase nome-paragrafo SECTION;
- i nomi di paragrafo all'interno di una SECTION sono locali a quella SECTION; così è possibile avere nomi doppi, purché appartengano a sezioni diverse, senza ambiguità;
- ogni SECTION risulta essere un'unità di caricamento (ovvero, come si dice in gergo, un OVERLAY): all'inizio dell'esecuzione del programma la SECTION di partenza è l'unica a essere presente nella memoria del calcolatore, e in genere sarà residente in memoria per tutta l'esecuzione; le altre SECTION vengono invece caricate dinamicamente all'atto della loro invocazione (che può essere fatta con PERFORM o con GO TO) e cancellate al momento del loro abbandono; il programmatore può, mediante opportune frasi COBOL, descrivere esattamente il comportamento desiderato su tali overlay, in modo da ottimizzare l'uso della memoria della macchina, facendo eseguire programmi la cui

Il formalismo usato dal COBOL per descriverne la sintassi si discosta da quello normalmente usato per altri linguaggi di programmazione.
Per esempio, la sintassi della semplice istruzione DISPLAY, per evidenziare un messaggio, ha la forma:

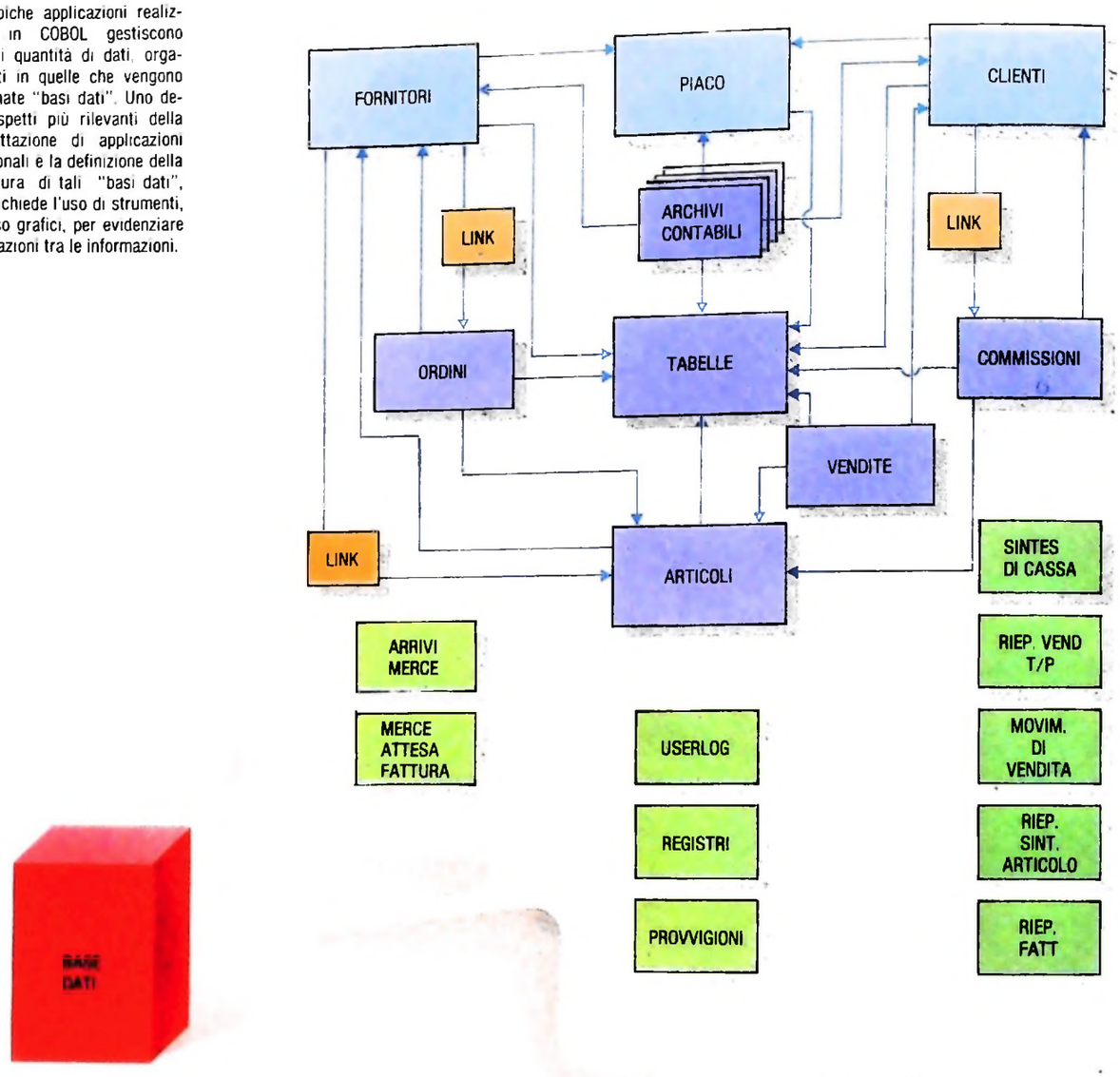
ove:
- le parole scritte in maiuscolo sono parole COBOL riservate
- quelle sottolineate sono obbligatorie
- le parole scritte in minuscolo individuano parole che saranno definite dal programmatore (si tratta, in sostanza, di "categorie sintattiche" che qualificano il tipo di elemento inseribile)
- gli elementi contenuti tra parentesi graffe sono obbligatori e in alternativa
- gli elementi racchiusi tra parentesi quadre sono opzionali e, se più di uno, sono anche in alternativa
- i puntini "..." indicano che l'ultimo elemento può essere ripetuto più volte.

Così risultano istruzioni legali:

```
DISPLAY A
DISPLAY 100
DISPLAY A B
DISPLAY A 100 B C D
```

e così via.

Le tipiche applicazioni realizzate in COBOL gestiscono grandi quantità di dati, organizzati in quelle che vengono chiamate "basi dati". Uno degli aspetti più rilevanti della progettazione di applicazioni gestionali è la definizione della struttura di tali "basi dati", che richiede l'uso di strumenti, spesso grafici, per evidenziare le relazioni tra le informazioni.



occupazione globale è molto maggiore della memoria disponibile.

I meccanismi di modularizzazione più vicini allo sviluppo top down sono invece costituiti dalle istruzioni:

- PERFORM
- CALL

Abbiamo già visto abbondanti esempi della prima; sappiamo come essa permetta il richiamo di paragrafi che condividono completamente la memoria tra di loro: tutte le variabili sono accessibili da tutti i paragrafi.

L'istruzione CALL invece richiama un programma COBOL compilato separatamente; la sua forma tipica è:

CALL PROG USING A,B,C ove

A, B e C sono nomi di variabili che fungono da parametri.

Osservazioni conclusive

Il COBOL si presenta come un linguaggio estremamente potente e ricco, con una sintassi molto ampia; la sua età è rivelata dai numerosi aspetti di visibilità della struttura fisica della macchina e dalla ridotta capacità di astrazione sui tipi di dati. Il linguaggio è ancora oggi uno dei più diffusi e usati.

IL COMPUTER NEGLI UFFICI

L'introduzione negli uffici di elaboratori flessibili e potenti trasforma non solo i compiti amministrativi, ma l'intera attività di controllo di un'organizzazione.

Le attività connesse alla gestione di informazioni hanno acquisito un'importanza sempre maggiore nell'economia e nella vita sociale dei paesi industrializzati, avendo beneficiato in modo naturale dello sviluppo di tecnologie elettroniche dedicate all'elaborazione dell'informazione. Da un lato computer e altri dispositivi digitali vengono introdotti in quantità e potenze crescenti, dall'altro si va ampliando lo spettro delle applicazioni realizzate per la risoluzione di problemi gestionali e di comunicazione. Il tipo di attività che ne risulta massimamente influenzato e assistito è quello che riguarda la "lavorazione" di beni di natura non materiale come informazioni e conoscenza. Ne consegue che l'ambito principale in cui le soluzioni tecnologiche più avanzate vengono sperimentate e proposte è l'ufficio, che negli ultimi cento anni ha visto avvicendarsi tutti i mutamenti tecnologici che hanno influito sull'utilizzo dell'informazione, e che di riflesso hanno contribuito a modificare l'organizzazione stessa del lavoro.

Un ufficio è un luogo dedicato a numerose attività: leggere, scrivere, pensare e comunicare ne sono le principali. Vi si amministra un'organizzazione in ogni suo livello di dettaglio, preparando piani e valutando proposte e rapporti, avviando pratiche per un percorso di successive trasformazioni fino all'archiviazione, secondo procedure spesso standardizzate. Per ciascuna di tali attività esistono ora strumenti software e hardware in grado di ripetere il successo che ebbero prodotti di tecnologie ormai datate, originariamente adottati nell'ambiente degli uffici e divenuti gradualmente di impiego quotidiano anche in settori non strettamente legati a tali attività. Non si tratta infatti di effettuare una conversione, peraltro non semplice, dal supporto cartaceo a quello elettronico, mantenendo inalterata la struttura delle procedure che occorre seguire, bensì d'incrementare la produttività degli uffi-

HEWLETT-PACKARD

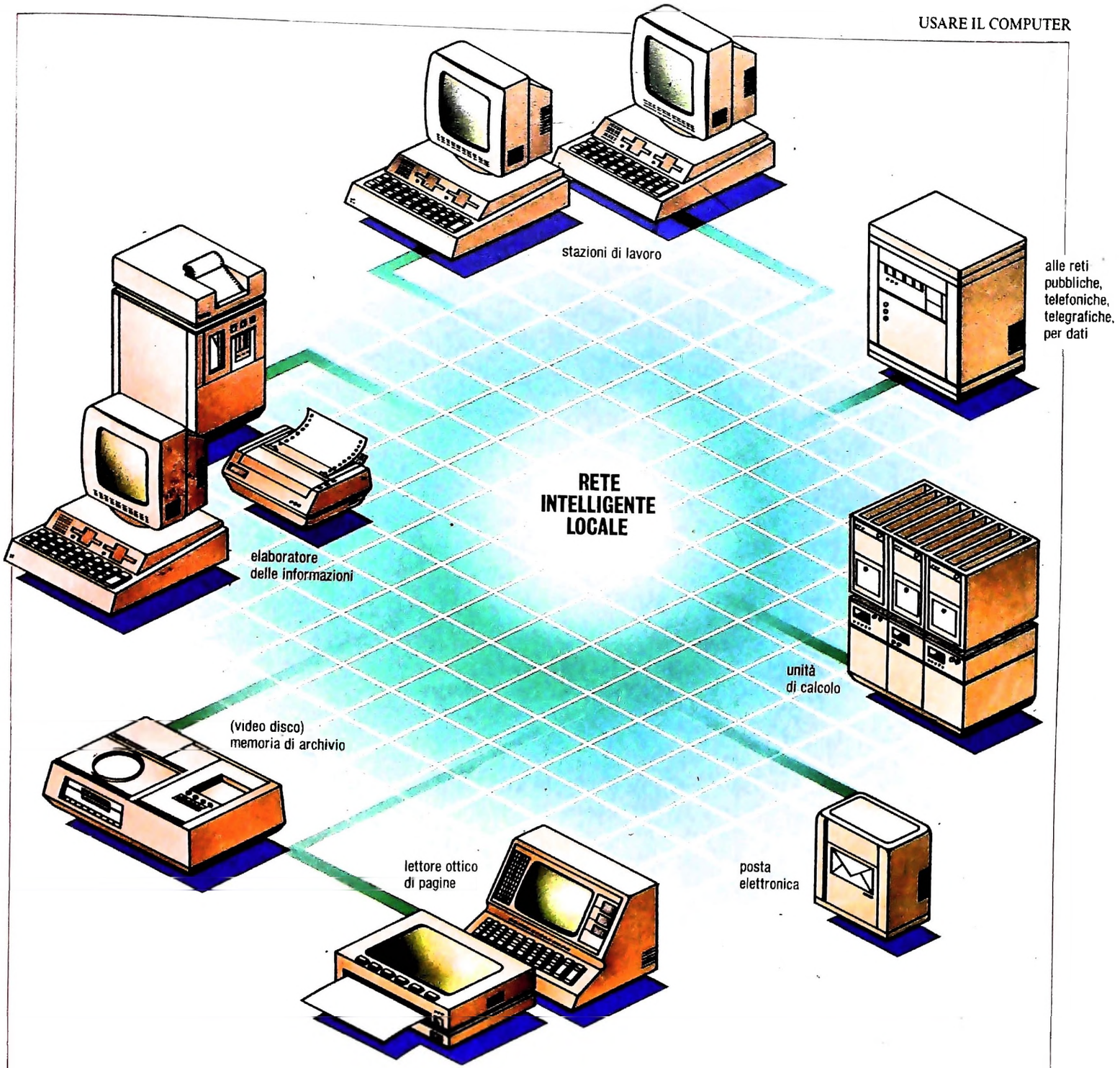
ci e migliorare la qualità del servizio da svolgere. Un impiego più efficace di mezzi di comunicazione e l'introduzione negli uffici di elaboratori flessibili e potenti trasforma non solo i compiti amministrativi, ma l'intera attività di controllo di un'organizzazione.

La meccanizzazione del lavoro d'ufficio ha dunque una lunga storia. Nella seconda metà del secolo scorso la tecnologia più diffusa per la compilazione di qualsiasi documento era la scrittura manuale con pennino d'acciaio. Un enorme salto di qualità venne compiuto con l'introduzione negli uffici, a breve distanza l'uno dall'altro, di dispositivi di grande impatto e valore tecnologico quali telegrafo, telefono e macchina per scrivere. Quest'ultima ebbe un ingresso nel mondo del lavoro parecchio simile a quello degli odierni elaboratori. Disponibile come prototipo artigianale fin dal 1850, raggiunse livelli di produzione industriali e bassi costi solo alla fine del secolo, attraversando un notevole perfezionamento tecnico che la rese ampiamente utilizzabile in casa e in ufficio.

Successivi avanzamenti tecnologici portarono all'introduzione di telescriventi, macchine per scrivere elettriche, centralini interni, copiatrici automatiche e macchine per elaborazione dati a schede perforate; ciò rese sempre più l'ufficio un crocevia di fitte comunicazioni e di attività dalle caratteristiche diversificate e complesse. Il tipo di organizzazione del lavoro negli uffici si è parallelamente trasformato, confermando che, nel lavoro dell'uomo, ogni cambiamento tecnologico è in grado di produrre un impatto significativo sul modello organizzativo che viene usato per la gestione complessiva del lavoro. L'ufficio preindustriale, tipica forma organizzativa delle attuali piccole aziende e studi professionali, lascia grande spazio all'iniziativa del singolo impiegato o dirigente, e sottolinea l'importanza di validi rapporti interpersonali. Risulta tuttavia inefficiente nel trattare vaste quantità di transazioni e dati. L'ufficio industriale, per ovviare a tali inconvenienti, è strutturato secondo il modello della catena di montaggio, con una larga base impiegatizia che svolge una mole notevole di lavoro ripetitivo su pratiche standard, connessa a piccole sezioni di specialisti per il trattamento delle eccezioni alle procedure. È l'ambiente noioso e senza stimoli dei grandi uffici, dove il flusso delle informazioni è lento e il servizio è svolto in modo insoddisfacente.

La consapevolezza di tali limitazioni, unita alla disponibilità di elaboratori mano a mano più potenti per la gestione delle informazioni, ha portato alcuni enti, come banche e aziende

Lo sviluppo delle tecnologie elettroniche sta rivoluzionando i metodi e l'ambiente stesso di lavoro: nella pagina precedente, una visione d'insieme di un centro operativo. In questa pagina, il computer che funge da "intelligenza organizzativa" raccogliendo e distribuendo i diversi tipi di informazione (schema della pagina accanto). L'applicazione del computer consente di far scorrere una quantità notevole di informazioni al secondo e può offrire configurazioni a utente singolo e multiplo.



APPLE COMPUTER

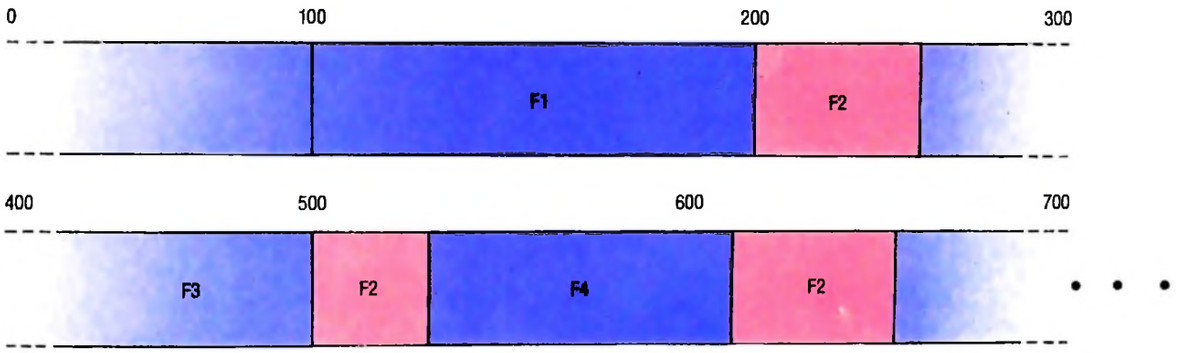
del terziario più avanzato, a individuare, negli anni Settanta, nuovi modelli organizzativi più consoni alle tendenze di sviluppo informatiche. Dapprima è stato affrontato il problema di automatizzare attività di stretta routine (controlli su conti e assegni, procedure contabili, registrazione di dati) utilizzando i grossi calcolatori dei centri di elaborazione dati (D.P.); in seguito, con l'introduzione di macchine per elaborare testi (word processors) e di duttili personal computer, è stata rinnovata l'immagine dell'ufficio industriale pieno di traffico, riportando il lavoro d'ufficio a valide caratteristiche preindustriali, ma con una produttività nettamente accresciuta e possibilità di soluzioni nuove. Grazie ad esempio alla possibilità di effettuare collegamenti remoti all'elaboratore della propria società, attraverso reti di comunicazione di estensione adeguata, un impiegato potrebbe svolgere il proprio lavoro sull'elaboratore di casa, in visita a un cliente o comunque non nell'ufficio. Si crea così il concetto di ufficio virtuale, non più un luogo dove recarsi per un numero fissato di ore, bensì un flessibile ambiente di lavoro ricreabile dove sia possibile trasportare un terminale e allacciarlo a un'opportuna linea di comunicazione, ad esempio telefonica.
Esiste, quindi, la concreta possibilità di automatizzare le attività che si realizzano in un ufficio. L'"office automation" vuole indicare l'obiettivo di un ufficio informatico dove ogni attività, sia essa standardizzata o meno, è completamente integrata con le risorse di elaborazione e i sistemi di comunicazione presenti, in modo da creare sistemi per la gestione dell'informazione in grado di assistere l'impiegato nelle diverse fasi del suo lavoro.
L'impiegato è incentivato alla produzione grazie a una minore ripetitività; si pone quindi l'uguaglianza "ufficio informatico = meno routine, meno errori, più velocità e qualità, meno costi". Si tratta infatti di evitare la specializzazione che caratterizza l'ufficio industriale (la cui formulazione è dovuta a Frederick W. Taylor, che si occupò agli inizi del secolo dell'ottimizzazione dei tempi di lavoro) e la conseguente mancanza di percezione complessiva del lavoro svolto, mantenendone tuttavia i pregi in termini di capacità di carico di lavoro.

L'ufficio informatico

L'ufficio informatico rappresenta un sistema strutturato di procedure in primo luogo per trattare testi e accedere a basi di dati personalizzate e in secondo luogo per gestire comunicazioni. Ciò è realizzabile attraverso una rete integrata che supporti: funzioni di word processing per generare corrispondenza interna e a carattere ufficiale, un sistema elettronico di scambio di messaggi per la comunicazione personale e inoltre un efficiente ambiente di archiviazione di dossier e dati, alcuni servizi personali (come agende e supporti per la programmazione degli impegni) e, infine, servizi di tipo collettivo che si basino sul sistema di comunicazione dell'azienda e permettano la trasmissione di facsimili, teleconferenze, collegamenti con archivi aziendali remoti e banche dati esterne. Con buona approssimazione, l'hardware relativo sarebbe costituito da un gruppo di stazioni di lavoro, comprendenti un terminale di buone capacità grafiche, una stampante e un dispositivo di memorizzazione locale, da una banca dati costantemente aggiornata, da collegamenti con un sistema di comunicazione interno e verso l'esterno, e da notevole quantità di strumenti software a disposizione, massimamente integrati tra loro. Le stazioni dovrebbero possedere un'intelligenza locale (tipico l'uso di elaboratori personali connessi in rete o elaboratori di recente concezione basati su potenti microprocessori che permettono una ridotta multiutenza) per supportare, a discrezione dell'impiegato, le più svariate applicazioni personali, e notevoli qualità di adattabilità ed ergonomia per agevolare il lavoro.
Tra le caratteristiche sopra elencate, che verranno riprese nel dettaglio in successivi articoli, il punto critico, allo stato attuale della tecnologia, è rappresentato dalle funzioni di comunicazione. Alcune rilevazioni mostrano che il tempo impiegato da manager in comunicazioni verbali e scritte raggiunge percentuali vicine al 90%, che verrebbero influenzate positivamente dall'introduzione di un sistema elettronico di gestione di comunicazioni e testi. Il punto nodale è la necessità di aumentare la produttività degli uffici per affrontare i crescenti costi di gestione di organizzazioni in cui aumentano la complessità dei processi decisionali e i bisogni informativi. L'"office automation" dovrebbe rappresentare un'ottima via da percorrere per superare una tale situazione. La tecnologia relativa alle comunicazioni rappresenta il fattore innovativo più rilevante nella progettazione di sistemi per l'O.A., in quanto molti parametri di valutazione del lavoro svolto negli uffici dipendono dalla velocità con cui viaggiano le informazioni e dal carico ottimale che i canali di comunicazione sono in grado di reggere; occorre conoscere quanti utenti il sistema di comunicazione è in grado di servire, in riferimento al tipo di lavoro che viene richiesto. Ad esempio, è vitale per un'azienda che la corrispondenza venga trattata efficientemente, in modo che sia consegnata con prontezza all'impiegato o dirigente interessato. Un servizio di posta elettronica a livello geografico (differente dal sistema di scambio di messaggi locale) ben integrato con il sistema informativo risolverebbe ogni difficoltà, poiché la missiva, giunta in azienda sotto forma digitale, sarebbe indirizzata automaticamente alla stazione di lavoro del destinatario, la quale diventa anche una casella postale elettronica. Ciò esige un'eccellente abilità del sistema di far scorrere quantità notevoli di informazioni al secondo, specie se è richiesta la trasmissione di lunghi documenti (rapporti, contratti, listini). La difficoltà di impiego di questi nuovi mezzi è legata alla interfaccia poco colloquiale del sistema con l'utente, che di solito è impressionato dalla differenza che riscontra rispetto alle abituali procedure d'ufficio, e alla complessità intrinseca dei sistemi fortemente distribuiti. Questa infatti si oppone ad un frequente mutamento delle caratteristiche dei sistemi stessi, contrariamente alla normale tendenza di ristrutturazione organizzativa che è tipica degli uffici.
Si pensi tuttavia che siamo ancora in una fase iniziale dell'effettiva esplosione dell'O.A. e che i benefici più consistenti saranno apprezzabili solo a partire dai prossimi anni.

Ancora sui modi per operare su file

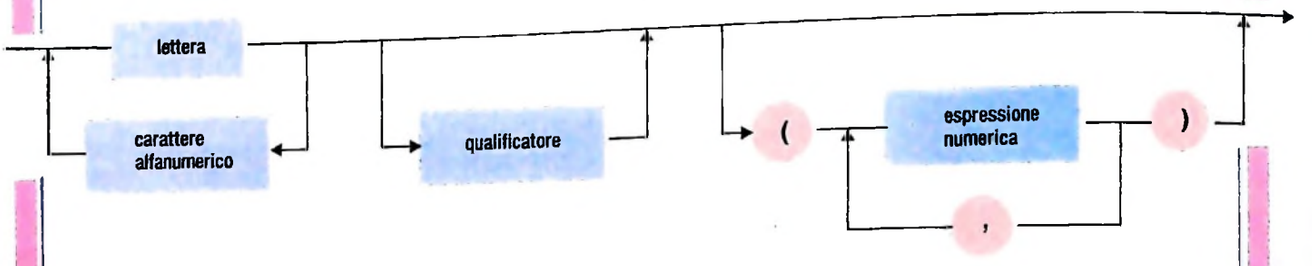
Abbiamo già accennato alla possibilità di "leggere" e di "registrare" dati, da e su file, usando primitive di linguaggio che in Pascal sono rispettivamente GET e PUT. In realtà non possiamo eseguire una GET o una PUT senza prima esserci posti nella condizione di operare sul file prescelto. Cerchiamo di capire l'esigenza di questa predisposizione: abbiamo visto che un file è una struttura dati nata dall'esigenza di "ospitare" grandi quantità di informazioni e che proprio per questo motivo tradizionalmente risiede su una memoria ausiliaria. Quando allora vogliamo operare su un file abbiamo esigenza prima di tutto di sapere in quale "posizione" del supporto di memoria usato è stato memorizzato. Inoltre, sempre a causa della dimensione di queste strutture di dati, accade talvolta che un file sia in realtà spezzato in più parti sul supporto di memoria. Può accadere infatti che tale supporto contenga già altri file e che gli spazi disponibili tra un file e l'altro non siano sufficienti a ospitare il nostro. Se però la somma degli spazi liberi mette a disposizione nel suo insieme la memoria necessaria, allora spesso il file viene distribuito su più parti. Se per esempio rappresentiamo la memoria come una sequenza di byte a ciascuno dei quali è associato un "indirizzo", cioè un numero che, esattamente come un numero civico, consente di riconoscerlo dagli altri e indichiamo come F1, F2, F3, F4 i file in essa contenuti possiamo avere una situazione come la seguente:



che corrisponde al fatto che i file F1, F3 e F4 erano preesistenti a F2; così quando quest'ultimo è stato allocato, cioè quando ne è stata richiesta la creazione, non era disponibile uno spazio unico, ma un insieme di spazi più piccoli che sommati raggiungevano la dimensione richiesta.
Questo non è che un esempio delle informazioni necessarie per poter operare su un file. Tali informazioni sono contenute in un "descrittore" che specifica, tra l'altro, in quali locazioni di memoria il file si trova e qual è la dimensione di ciascuna. Quando infatti effettuiamo un'operazione di scrittura o di lettura in realtà comandiamo un organo che va a posizionarsi su un indirizzo preciso di memoria. Nei linguaggi di alto livello non vediamo né l'operazione di individuazione di tale indirizzo né il comando al suddetto organo, perché come abbiamo già visto, queste e altre operazioni sono in realtà effettuate da istruzioni di basso livello, che vengono "riassunte" dalle macroistruzioni del nostro linguaggio. È necessario però che tali istruzioni di basso livello dispongano delle informazioni suddette. Pertanto è necessario,

Gli identificatori delle variabili

SINTASSI



SEMANTICA

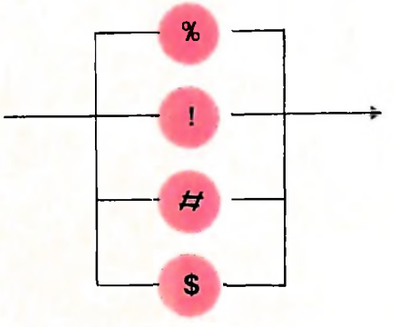
Una variabile è identificata da una successione di caratteri alfanumerici (cioè o lettere o numeri) di cui il primo deve essere necessariamente una lettera.

Di tale sequenza solo i primi due caratteri sono significativi, mentre i restanti sono da considerare di puro ausilio mnemonico. Così:

```
10 LET RA=100          10 LET RAGGIO=100
```

sono istruzioni che si equivalgono. Si noti che sono vietati nomi di variabili che corrispondono a parole chiave del linguaggio, come IF, THEN, OR ecc.

Una variabile può essere qualificata rispetto al tipo da un carattere terminante il nome, scelto tra i seguenti:



con il seguente significato:

% la variabile è intera e può assumere i valori compresi tra -32768 e 32767;

! la variabile è reale in precisione semplice e può assumere valori compresi tra -10^62, mentre i valori più piccoli rappresentabili ammettono ordini di grandezza di 10^-64; la precisione semplice mette a disposizione 7 cifre decimali significative, di cui 6 sono visualizzate dalla stampa e la settima è usata per arrotondare la sesta;

la variabile è reale in doppia precisione, ovvero ha gli stessi limiti di una variabile in precisione semplice, ma mette a disposizione 16 cifre significative (di cui 15 possono essere visualizzate e la sedicesima è usata per arrotondare la quindicesima);

$ la variabile è una stringa alfanumerica e può contenere da 0 a 255 caratteri. La sequenza di 0 caratteri, indicata con "", è detta stringa "nulla" o "vuota".

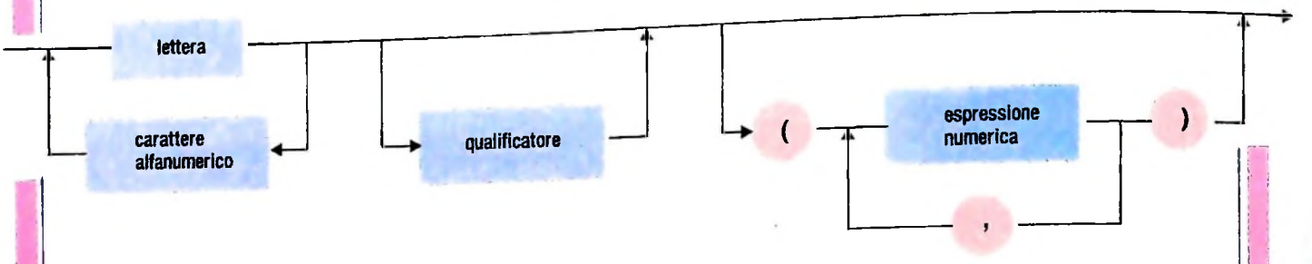
In assenza di specifiche indicazioni, le variabili prive di qualificatore sono tutte considerate in doppia precisione.

Una variabile che sia elemento di un array deve essere seguita dalla specificazione dei valori assunti dagli indici, che devono essere in numero uguale a quello indicato nell'istruzione DIM e ne devono rispettare i limiti. Un'espressione può essere usata come indice: in tal caso il risultato viene troncato. Così:

A (2.999) equivale a A (2)

Gli identificatori delle variabili

SINTASSI

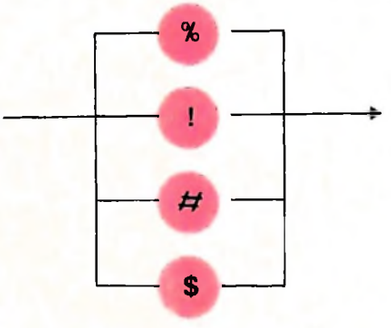


SEMANTICA

Una variabile è identificata da una successione di caratteri alfanumerici (cioè o lettere o numeri) di cui il primo deve essere necessariamente una lettera.
Di tale sequenza solo i primi due caratteri sono significativi, mentre i restanti sono da considerare di puro ausilio mnemonico. Così:

```
10 LET RA=100          10 LET RAGGIO=100
```

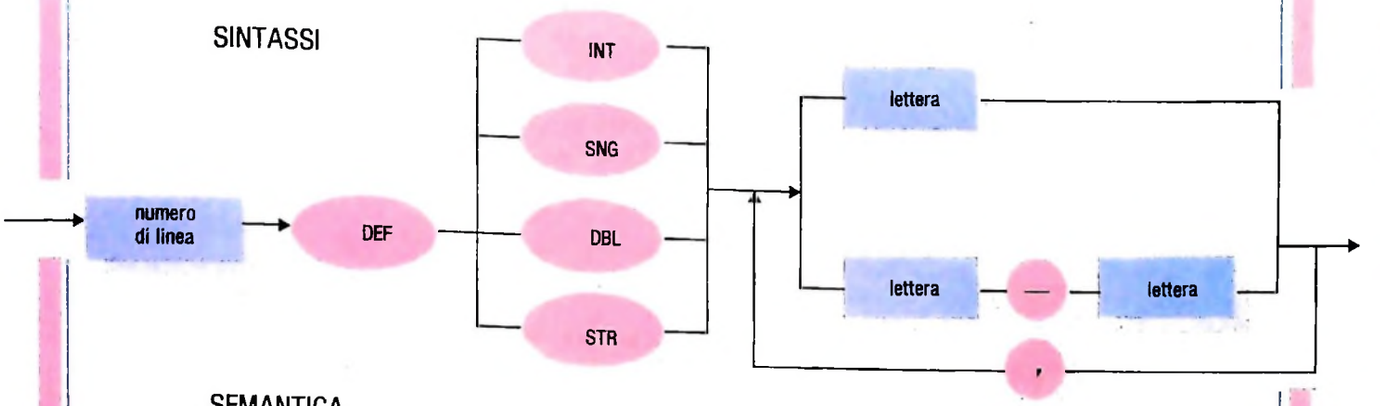
sono istruzioni che si equivalgono. Si noti che sono vietati nomi di variabili che corrispondono a parole chiave del linguaggio, come IF, THEN, OR ecc.
Una variabile può essere qualificata rispetto al tipo da un carattere terminante il nome, scelto tra i seguenti:



con il seguente significato:
% la variabile è intera e può assumere i valori compresi tra -32768 e 32767;
! la variabile è reale in precisione semplice e può assumere valori compresi tra -10^62, mentre i valori più piccoli rappresentabili ammettono ordini di grandezza di 10^-64; la precisione semplice mette a disposizione 7 cifre decimali significative, di cui 6 sono visualizzate dalla stampa e la settima è usata per arrotondare la sesta;
# la variabile è reale in doppia precisione, ovvero ha gli stessi limiti di una variabile in precisione semplice, ma mette a disposizione 16 cifre significative (di cui 15 possono essere visualizzate e la sedicesima è usata per arrotondare la quindicesima);
$ la variabile è una stringa alfanumerica e può contenere da 0 a 255 caratteri. La sequenza di 0 caratteri, indicata con "", è detta stringa "nulla" o "vuota".
In assenza di specifiche indicazioni, le variabili prive di qualificatore sono tutte considerate in doppia precisione.
Una variabile che sia elemento di un array deve essere seguita dalla specificazione dei valori assunti dagli indici, che devono essere in numero uguale a quello indicato nell'istruzione DIM e ne devono rispettare i limiti. Un'espressione può essere usata come indice: in tal caso il risultato viene troncato. Così:
A (2.999) equivale a A (2)

Definizioni di tipo

SINTASSI



SEMANTICA

L'istruzione stabilisce il tipo delle variabili che iniziano con una certa lettera, indipendentemente dalla presenza di un qualificatore del nome. Così:

```
10 DEFINT A-K,S
```

indica che tutte le variabili il cui identificatore inizia con una lettera compresa nell'intervallo A-K, estremi inclusi, o con la lettera S, sono da considerare intere, indipendentemente dalla presenza o meno di un qualificatore.

DEFINT specifica variabili INTERE
DEFSNC specifica variabili in SINGOLA PRECISIONE
DEFDBL specifica variabili in DOPPIA PRECISIONE
DEFSTR specifica variabili stringa.

prima di operare su un file, mettere a disposizione tutte le informazioni contenute nel descrittore del file. Questa operazione di trasferimento di informazioni è effettuata in quasi tutti i linguaggi di alto livello da una istruzione chiamata OPEN.
Il descrittore di cui abbiamo parlato deve evidentemente essere sempre aggiornato, deve cioè contenere una descrizione corretta del file. Per questo motivo ogni volta che viene completata una elaborazione del file, è necessario eseguire una istruzione detta CLOSE che provvede a tali aggiornamenti.

Come individuare la fine di un file: la funzione EOF

Supponiamo di avere registrato una serie di informazioni su un file per esempio di numeri. Vogliamo adesso verificare le informazioni che abbiamo registrato chiedendone una lista. Per fare ciò costruiremo un programma che si riposiziona all'inizio del file e legge e visualizza elemento per elemento fino all'esaurimento del file stesso. L'algoritmo sarebbe allora il seguente:

```
GET elemento
WHILE elemento disponibile DO
    BEGIN
        visualizza elemento
        GET elemento
END;
```

Ma come facciamo in realtà a sapere quando gli elementi sono esauriti? Il problema non è indifferente poiché se non abbiamo tale informazione rischiamo di continuare a leggere informazioni che non appartengono in realtà al nostro file. Occorre dunque poter individuare il "confine" tra il nostro file e il resto della memoria. Questo problema può essere risolto in modi diversi, che però si possono tutti ricondurre a due elementi essenziali:

- la fine del file è indicata da un opportuno segnalatore, che è registrato in coda al file e che è noto e riconoscibile dalle microistruzioni che effettuano le operazioni di lettura;
- il descrittore del file sa qual è l'occupazione di memoria del file e consente quindi di individuare l'ultimo elemento.

In realtà tutte queste sono informazioni che l'utente che dispone di un linguaggio di alto livello non vede mai. Esiste infatti in tutti i linguaggi di alto livello la possibilità di conoscere in ogni momento se si è o meno alla fine del file. Questa informazione è spesso contenuta in una variabile booleana di nome EOF che vale quindi VERO quando ci si trova alla fine del file e FALSO negli altri casi.
In Pascal esiste una funzione predefinita che si chiama anch'essa EOF e che applicata a una variabile di tipo file ritorna al valore VERO quando il file è esaurito e FALSO negli altri casi.
Con l'uso di tale funzione l'algoritmo sopra descritto risulta allora così:

```
OPEN F
GET F^
WHILE not EOF (F) DO
    BEGIN
        visualizza F^
        GET F^
    END;
```

Si noti che come già altre volte abbiamo usato il Pascal solo come riferimento e che pertanto il precedente non è un programma accettabile da un compilatore Pascal.

Cosa abbiamo imparato

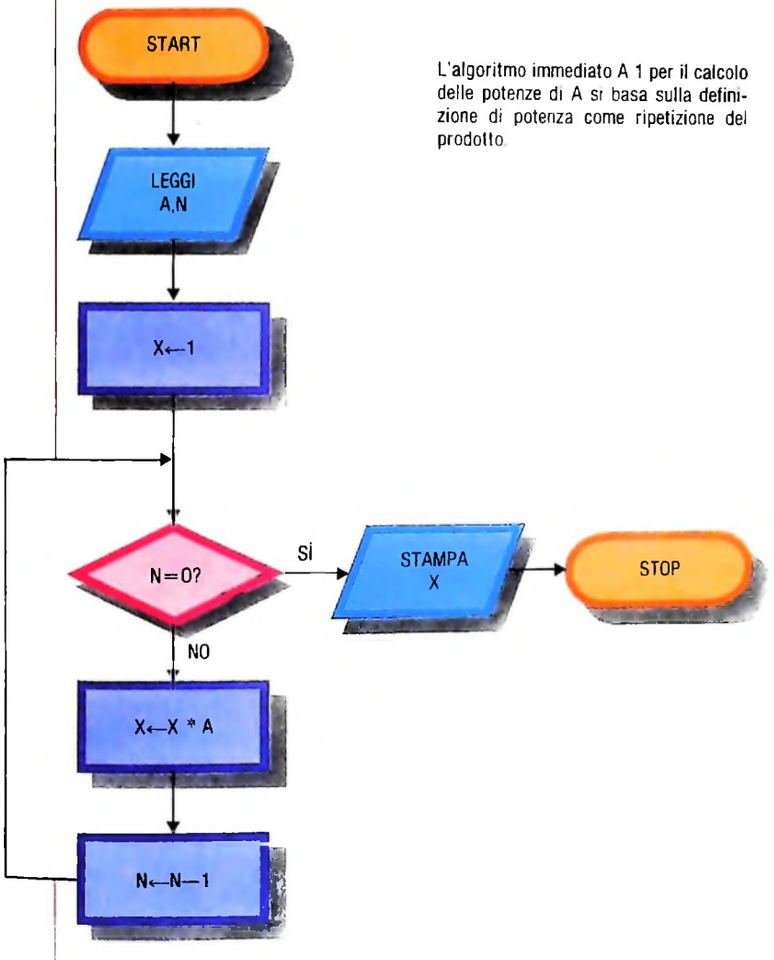
In questa lezione abbiamo visto:

- Come operare su file con le istruzioni OPEN e CLOSE;
- Come individuare la fine di un file con la funzione EOF;
- La sintassi degli identificatori di variabili in BASIC;
- La definizione di tipi in BASIC;
- Come si dichiara il tipo di una variabile BASIC con l'uso di un qualificatore che ne termina il nome;
- Come si stabilisce il tipo di una variabile a partire dalla prima lettera del suo identificatore.

TEORIA DELLA COMPLESSITÀ COMPUTAZIONALE

Uno strumento per distinguere problemi "facili" e problemi "difficili".

Supponiamo di dover scrivere un algoritmo che, ricevendo in ingresso due numeri interi A ed N, calcoli la potenza A^N. Chiunque non abbia dimenticato la definizione informale di A^N come "A moltiplicato per se stesso N volte", col caso particolare $A^0 = 1$, non avrà nessuna difficoltà a scrivere un algoritmo analogo a quello presentato qui sotto. L'algoritmo A1 si basa su un ciclo controllato dalla variabile N: a ogni esecuzione del ciclo si moltiplica per A la variabile X, il cui valore iniziale è 1, e si decrementa di 1 la variabile N, finché quest'ultima non assume il valore 0. È facile verificare che l'algoritmo è corretto, cioè che per ogni valore di N, compreso lo 0, calcola effettivamente la potenza ennesima di A.



L'algoritmo immediato A 1 per il calcolo delle potenze di A si basa sulla definizione di potenza come ripetizione del prodotto.

Consideriamo ora l'algoritmo A2 della pagina seguente. Quante persone sarebbero in grado, senza suggerimenti, di capire cosa fa questo algoritmo? Sicuramente ben poche; eppure, questo algoritmo calcola esattamente la stessa funzione del precedente, la potenza. Ci si può allora chiedere quale mente contorta abbia ideato questo algoritmo incomprensibile e perché lo abbia fatto. Per rispondere, basta osservare la tabella di pag. 471, che riporta il numero di operazioni aritmetiche occorrenti per calcolare A^N con i due algoritmi, in corrispondenza di diversi valori dell'esponente.

Mentre per A1 il numero di operazioni è sempre il doppio dell'esponente (infatti si eseguono un prodotto ed una sottrazione per ogni ciclo, ed i cicli sono N), con A2 il numero di operazioni da eseguire cresce molto più lentamente, seguendo una curva di tipo logaritmico anziché lineare. Per N = 100, a esempio, A2 arriva al risultato con un numero di operazioni (e quindi un tempo) che è circa un ottavo rispetto a quello di A1: un bel vantaggio, che diventa sempre più rilevante al crescere di N.

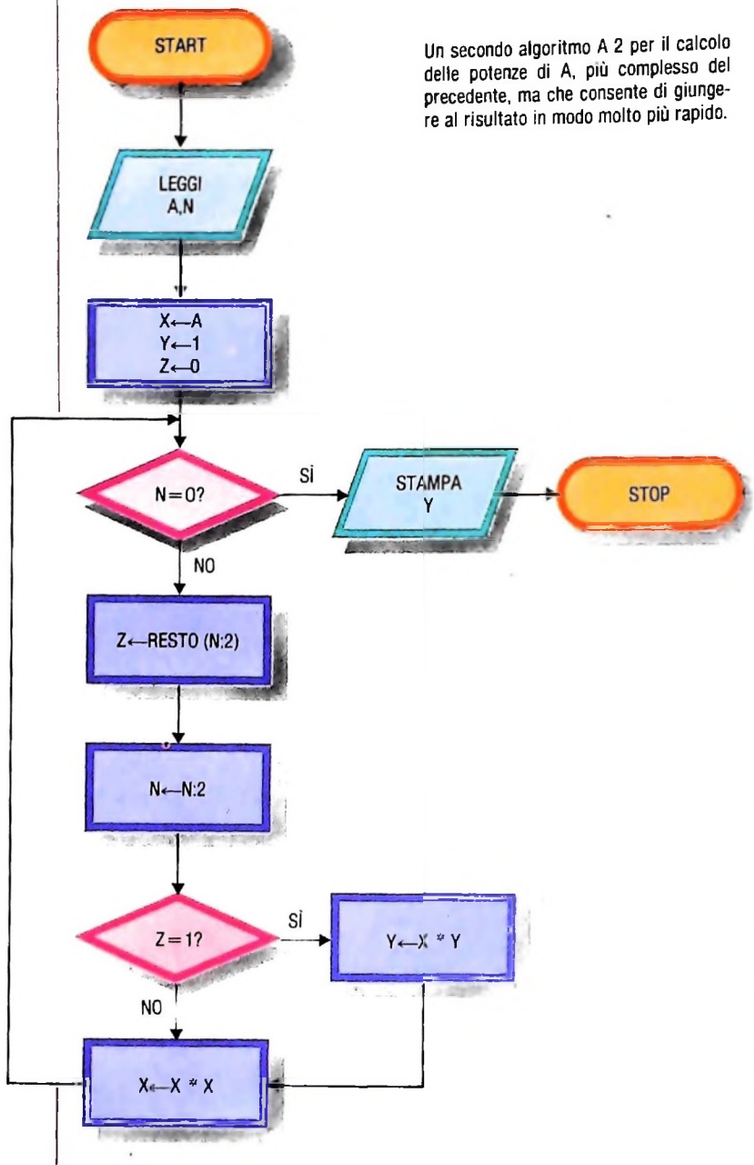
L'idea su cui si basa l'algoritmo A2 è semplice e possiamo presentarla con un esempio. Se vogliamo calcolare A^{12}, non è necessario che calcoliamo tutte le potenze intermedie; basta infatti calcolare A^6 e moltiplicarlo per se stesso, risparmiando così 5 moltiplicazioni. Ma A^6 può essere calcolato moltiplicando per se stesso A^3, e così via.

L'algoritmo A2 non è altro che l'applicazione sistematica di un'idea analoga. Esso parte dalla osservazione che ogni numero N può essere visto come somma di potenze di 2, attraverso la sua rappresentazione binaria:

$N = \sum_{i=1}^{n} b_i \, 2^{i-1}$ ove b_i è la i-esima cifra da destra nella rappresentazione binaria di N.

Per esempio, $N = 13 = 1.2^3 + 1.2^2 + 0.2^1 + 1.2^0$.

Se N è esponente di una potenza, applicando le prop[illegible] delle potenze, otteniamo un prodotto di potenze i c[illegible] nenti sono potenze di 2. In particolare, il fattore



Un secondo algoritmo A 2 per il calcolo delle potenze di A, più complesso del precedente, ma che consente di giungere al risultato in modo molto più rapido.

presente solo se la i-esima cifra binaria (da destra) di N è 1. Per esempio, $A^{13} = A^{2^3+2^2+2^0} = A^8 \cdot A^4 \cdot A$

A questo punto, possiamo calcolare A^N calcolando, attraverso la variabile X che ad ogni passo viene moltiplicata per se stessa, solo le potenze della forma A^2, A^4, A^8, ... (ed è qui che avviene il risparmio) e moltiplicando la variabile di uscita Y per $A^{2^{i-1}}$ solo se b_i ha valore 1. La determinazione delle cifre binarie di n avviene attraverso le istruzioni Z ← resto (N:2), che determina le cifre binarie successive, a partire da quella più a destra, di N e N ← N:2, che dà la parte intera del quoziente tra N e 2.

Algoritmi e risorse

L'esempio visto mette in evidenza un aspetto della soluzione algoritmica dei problemi di cui non abbiamo fin qui tenuto conto. In precedenza, abbiamo visto come distinguere i problemi insolubili dai problemi solubili, attraverso la formalizzazione della nozione di algoritmo in termini di macchina di Turing; ma la macchina di Turing dispone di un nastro illimitato e può continuare a lavorare senza vincoli temporali. È insomma un modello ideale, che non tiene conto dei vincoli fisici legati alla disponibilità di risorse. In realtà, risolvere problemi attraverso un algoritmo ha un costo in termini di risorse utilizzate; a esempio il tempo di calcolo o la quantità di memoria: è necessario essere in grado di valutare questo costo e, se possibile, di ridurlo al minimo, utilizzando, come abbiamo fatto sopra, algoritmi più raffinati (ma attenzione! anche il tempo occorrente per trovare algoritmi ultraraffinati è una risorsa).

Intorno a questa problematica si è sviluppata una teoria che, oltre a fornire indicazioni di estrema utilità per la realizzazione pratica di algoritmi efficienti, rappresenta anche uno dei più originali contributi che l'informatica abbia dato alla comprensione di fenomeni complessi e alla cultura scientifica in generale: la teoria della complessità computazionale.

La complessità degli algoritmi

Il primo problema che la teoria della complessità si è posta è quello di valutare in termini quantitativi le prestazioni, e quindi l'efficienza, di un algoritmo, fornendo da un lato definizioni significative delle nozioni di "risorsa", "costo", "efficienza", e dall'altro tecniche di analisi sufficientemente potenti e generali.

Cosa si debba intendere per "risorsa" è stato stabilito in modo estremamente rigoroso da M. Blum, con un insieme di assiomi che caratterizzano in modo preciso le proprietà che deve avere, in astratto, qualunque tipo di risorsa. In questa sede non è possibile presentare e discutere il significato di questi assiomi, per cui dovremo limitarci a prendere in considerazione i due tipi di risorsa che più interessano in concreto:

— il tempo impiegato per eseguire l'algoritmo

— la quantità di memoria occorrente per portare a termine l'esecuzione.

Parlare di tempo impiegato per eseguire un algoritmo è evidentemente molto impreciso; infatti, si intuisce abbastanza facilmente che il tempo dipende anche dal dato iniziale fornito all'algoritmo. Inoltre, se per tempo intendiamo il tempo fisico, misurato con un orologio, che intercorre a esempio tra il momento della lettura dei dati e quello della stampa dei risultati, avremo valori anche molto diversi a seconda della potenza dell'elaboratore utilizzato per eseguire l'algoritmo.

Per eliminare la dipendenza dalla macchina, possiamo far riferimento non al tempo di calcolo effettivo, ma a un tempo "ideale", dipendente esclusivamente dalla struttura dell'algoritmo: una misura di questo tipo è data dal numero totale di operazioni eseguite, eventualmente suddivise per tipo (somme, prodotti, confronti, ...). In questo modo, è facile risalire al tempo effettivo quando si conosca il tempo impiegato da un particolare elaboratore per eseguire un'operazione.

Per quanto riguarda il modo in cui il tempo di calcolo può dipendere dai dati di ingresso, dal nostro esempio iniziale

possiamo trarre alcune considerazioni generali estremamente importanti.
Consideriamo ancora l'algoritmo della pagina a lato. Il numero di operazioni da eseguire dipende dal numero di ripetizioni del ciclo; ma il ciclo viene ripetuto tante volte quante sono le cifre della rappresentazione binaria dell'esponente N. Per esempio, per N=30 il ciclo viene ripetuto 5 volte, per N =40 6 volte, per N=1000 10 volte.
L'elemento più importante per determinare il numero complessivo di operazioni è quindi la "dimensione" dell'esponente (cioè il numero di cifre binarie necessarie per rappresentarlo). Possiamo allora evidenziare la dipendenza del tempo dai dati di ingresso assegnando a ogni dato una dimensione, in genere un numero intero positivo proporzionale alla quantità di memoria necessaria per contenere i dati, e determinando quindi il tempo di computazione come funzione della dimensione dei dati. Tuttavia, a parità di dimensione, abbiamo non lo stesso numero di operazioni, ma "circa" lo stesso numero. Nel nostro esempio, ogni ciclo consta di tre o quattro operazioni, a seconda che la cifra binaria calcolata nel corso di quel ciclo sia 0 o 1. Così, per N=40 bastano 20 operazioni (poiché la rappresentazione di 40 in base 2 è 101000), mentre per N=63, rappresentato da 111111 e quindi di ugual dimensione, ne occorrono 24.
Per rendere univoca la funzione di complessità, si possono fare due scelte, considerando, per ogni dimensione n:
— il dato "peggiore", cioè quello che dà luogo alla computazione più lunga; oppure:
— la complessità media tra tutti i dati di ugual dimensione.
Poiché la complessità nel caso peggiore è più importante in molte situazioni reali, ed è in genere più semplice da determinare, ci riferiremo nel seguito a essa. Considerazioni analoghe si possono fare per la complessità in spazio; si può quindi dare la seguente definizione.

La *complessità in tempo* (nel caso peggiore) di un algoritmo A è la funzione:
T(n) = numero massimo di operazioni elementari richieste da una computazione su un dato di dimensione n.
La *complessità in spazio* (nel caso peggiore) di un algoritmo A è la funzione:
S(n) = quantità massima di memoria occupata nel corso di una computazione su un dato di dimensione n.

Nel nostro esempio, è facile verificare che:
— la complessità in tempo nel caso peggiore è T(n)=4n. Infatti se tutte le cifre binarie dell'esponente valgono 1, ad ogni ciclo si eseguono 4 operazioni;
— la complessità in media, in questo caso facile da calcolare, è T(n)=3.5n. Infatti, in media, avremo un ugual numero di cifre binarie uguali a 1 o a 0 e quindi un ugual numero di cicli con 3 o 4 operazioni.
Un'ultima precisazione è necessaria per quanto riguarda le nozioni di operazione elementare e di quantità di memoria. Per definire in modo preciso queste nozioni, occorre far riferimento a un particolare modello di computazione, con un insieme di operazioni primitive la cui esecuzione richiede, per definizione, tempo unitario. È essenziale che il modello di computazione sia standardizzato, in modo da poter valutare in modo omogeneo la complessità degli algoritmi. A esempio, le valutazioni che abbiamo fatto sui due algoritmi per il calcolo della potenza non sono omogenee, poiché nel primo le operazioni sono essenzialmente prodotti, mentre nel secondo entrano massicciamente le divisioni. Inoltre, le operazioni primitive devono essere effettivamente elementari, altrimenti sarebbe facile calcolare la potenza, ma lo stesso varrebbe per ogni altra funzione computabile, con una sola operazione, assumendo un modello che abbia la potenza come operazione primitiva.
Ancora una volta, la macchina di Turing si presenta come il modello di riferimento ideale: potremo allora definire il tempo di computazione di un algoritmo come il numero di mosse eseguite dalla macchina di Turing che lo formalizza, in funzione della dimensione dei dati di ingresso, e lo spazio co-

n	5	10	30	40	50	100	1000
$f_1(n)$	10	20	60	80	100	200	2000
$f_2(n)$	11	14	19	20	21	24	36

Al crescere del valore dell'esponente n, il numero di operazioni occorrenti per calcolare a^n con l'algoritmo A 1 cresce molto più rapidamente di quello occorrente con A 2.

FUNZIONE	TIPO	n=1	n=5	n=10	n=20	n=50	n=100
$4n$	lineare	$4 \cdot 10^{-9}$ secondi	$2 \cdot 10^{-8}$ secondi	$4 \cdot 10^{-8}$ secondi	$8 \cdot 10^{-8}$ secondi	$2 \cdot 10^{-7}$ secondi	$4 \cdot 10^{-7}$ secondi
$2n^2$	quadratica	$2 \cdot 10^{-9}$ secondi	$5 \cdot 10^{-8}$ secondi	$2 \cdot 10^{-7}$ secondi	$8 \cdot 10^{-7}$ secondi	$5 \cdot 10^{-6}$ secondi	$2 \cdot 10^{-5}$ secondi
n^3	cubica	10^{-9} secondi	$1.25 \cdot 10^{-7}$ secondi	10^{-6} secondi	$8 \cdot 10^{-6}$ secondi	$1.25 \cdot 10^{-4}$ secondi	10^{-3} secondi
2^n	esponenziale	$2 \cdot 10^{-9}$ secondi	$3.2 \cdot 10^{-8}$ secondi	10^{-6} secondi	10^{-3} secondi	312 ore = 13 giorni	$4 \cdot 10^{11}$ secoli
3^n	esponenziale	$3 \cdot 10^{-9}$ secondi	$2.43 \cdot 10^{-7}$ secondi	$5.9 \cdot 10^{-5}$ secondi	3,5 secondi	$2 \cdot 10^{5}$ secoli	$1.6 \cdot 10^{29}$ secoli

Nella tabella è riportato il tempo di computazione effettivo di un calcolatore in grado di eseguire un miliardo di operazioni al secondo per l'esecuzione di algoritmi di diversa complessità. È evidente la vera e propria esplosione del tempo di calcolo per funzioni di complessità di tipo esponenziale.

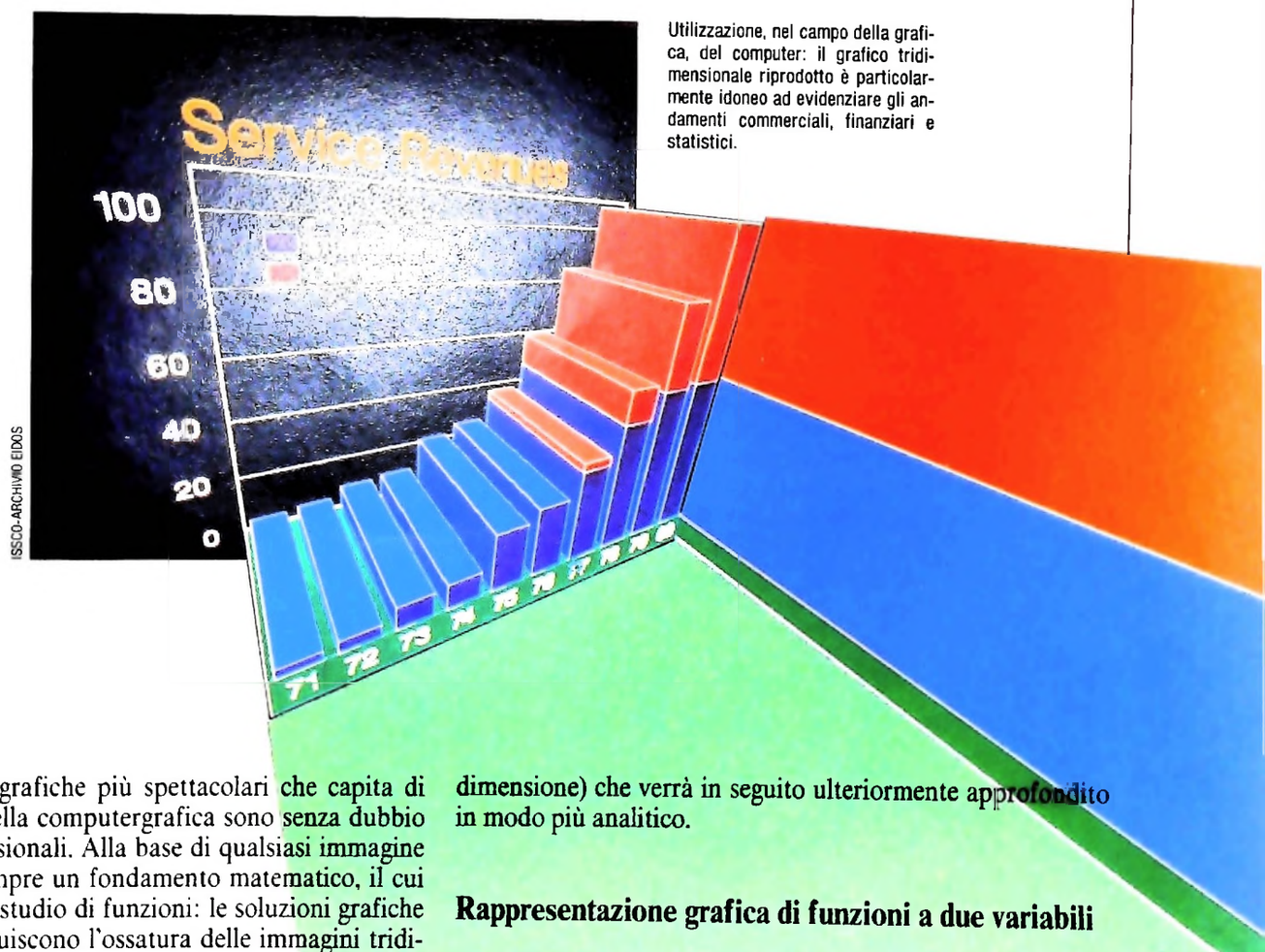
me il numero di caselle del nastro utilizzate nel corso della computazione.
La tabella di pag. 471 ci dà l'occasione per fare un'altra osservazione.
Finché N non supera 5, l'algoritmo A1 è preferibile ad A2, mentre A2 diventa vantaggioso da 6 in poi. I pochi casi in cui A1 è preferibile sono però del tutto trascurabili rispetto all'infinità di casi in cui è più efficiente A2. In genere quindi tra due funzioni di complessità $f(n)$ e $g(n)$ viene considerata preferibile quella che ha il valore più basso a partire da un certo n in poi, cioè quella che, con termine matematico, è "asintoticamente" minore.
Ancora, poiché le possibili funzioni di complessità sono infinite, è opportuno, almeno per un primo livello di analisi, raccoglierle in un numero ridotto di classi, evidenziando l'andamento generale della funzione (cioè la sua rapidità di crescita) e trascurando dettagli inessenziali. In genere, data una funzione di complessità della forma $f(n) = K_1 f_1(n) + \ldots + K_j f_j(n)$, si prende in considerazione solo il termine che cresce più rapidamente; inoltre, si considerano equivalenti funzioni che differiscono solo per il coefficiente K. Si parlerà così di funzioni:
— logaritmiche: $f(n) = K.\log n$
— lineari: $f(n) = K.\ n$
— quadratiche: $f(n) = K.\ n^2$
— polinomiali: $f(n) = K.\ n^h$
— esponenziali: $f(n) = K^n$

Algoritmi "buoni" e algoritmi "cattivi"

Ora che sappiamo che gli algoritmi hanno un costo e che siamo in grado di darne una valutazione quantitativa, possiamo decidere se il costo è accettabile rispetto al servizio offerto o meno, possiamo cioè distinguere gli algoritmi inefficienti dagli algoritmi efficienti. Naturalmente, il criterio di discriminazione è arbitrario, ma per una serie di considerazioni di carattere sia teorico che pratico oggi c'è un sostanziale accordo su un criterio proposto da Edmonds nel 1965.
Tralasciando per il momento le considerazioni teoriche, possiamo concentrarci sulla tabella di questa pagina che sintetizza le motivazioni pratiche uniformandole alla scelta di Edmonds. La tabella prende in considerazione cinque distinte funzioni di complessità, e per ognuna di queste fornisce il tempo effettivo di calcolo, ipotizzando di avere un elaboratore in grado di eseguire un miliardo di operazioni al secondo, per diverse dimensioni dei dati di ingresso.
Dalla tabella risulta evidente che finché la funzione di complessità è di tipo polinomiale (come $4n$, $2n^2$, n^3) si ottiene il risultato finale in tempi ragionevoli anche per dati elevati. Se la funzione di complessità è invece di tipo esponenziale, si ha una vera e propria esplosione del tempo di calcolo, che diventa rapidamente dell'ordine degli anni o dei secoli anche per dati di dimensioni non eccessive.
Questa situazione non verrebbe modificata nemmeno da un sostanziale incremento della velocità degli elaboratori. Se poniamo il vincolo di un'ora come tempo di computazione accettabile, possiamo dire che un aumento di 1000 volte della velocità di calcolo farebbe crescere solo di dieci unità la dimensione dei problemi risolubili nel tempo dato da algoritmi di complessità 2^n, mentre la decuplicherebbe se la funzione di complessità fosse n.
Possiamo allora concludere enunciando la seguente convenzione.
Consideriamo "buoni" o efficienti tutti gli algoritmi che hanno complessità al più polinomiale.
Consideriamo "cattivi" o inefficienti tutti gli algoritmi che hanno complessità esponenziale o superesponenziale.

UN'IMMAGINE IN TRE DIMENSIONI

Un primo approccio alla visualizzazione di immagini tridimensionali, o meglio che simulano la terza dimensione, generando la percezione visiva della profondità.



ISSCO-ARCHIVIO EIDOS

Utilizzazione, nel campo della grafica, del computer: il grafico tridimensionale riprodotto è particolarmente idoneo ad evidenziare gli andamenti commerciali, finanziari e statistici.

Le rappresentazioni grafiche più spettacolari che capita di vedere nell'ambito della computergrafica sono senza dubbio le immagini tridimensionali. Alla base di qualsiasi immagine di questo tipo c'è sempre un fondamento matematico, il cui strumento tipico è lo studio di funzioni: le soluzioni grafiche di tali funzioni costituiscono l'ossatura delle immagini tridimensionali.
In questa lezione viene fornito un semplice esempio di risoluzione grafica di una funzione a due variabili compatibile con lo schermo dell'M10 e che costituisce una simulazione tridimensionale. Vuole essere questo un primo rapido approccio ad un tipo di problematica (simulazione di una terza dimensione) che verrà in seguito ulteriormente approfondito in modo più analitico.

Rappresentazione grafica di funzioni a due variabili

In una precedente lezione è stato introdotto il concetto di funzione; più precisamente abbiamo definito il legame che intercorre tra una variabile x, detta Variabile Indipendente, e una variabile y detta Variabile Dipendente.
Abbiamo anche verificato che in questo caso sono sufficienti due dimensioni (l'asse delle x e l'asse delle y) per effettuare

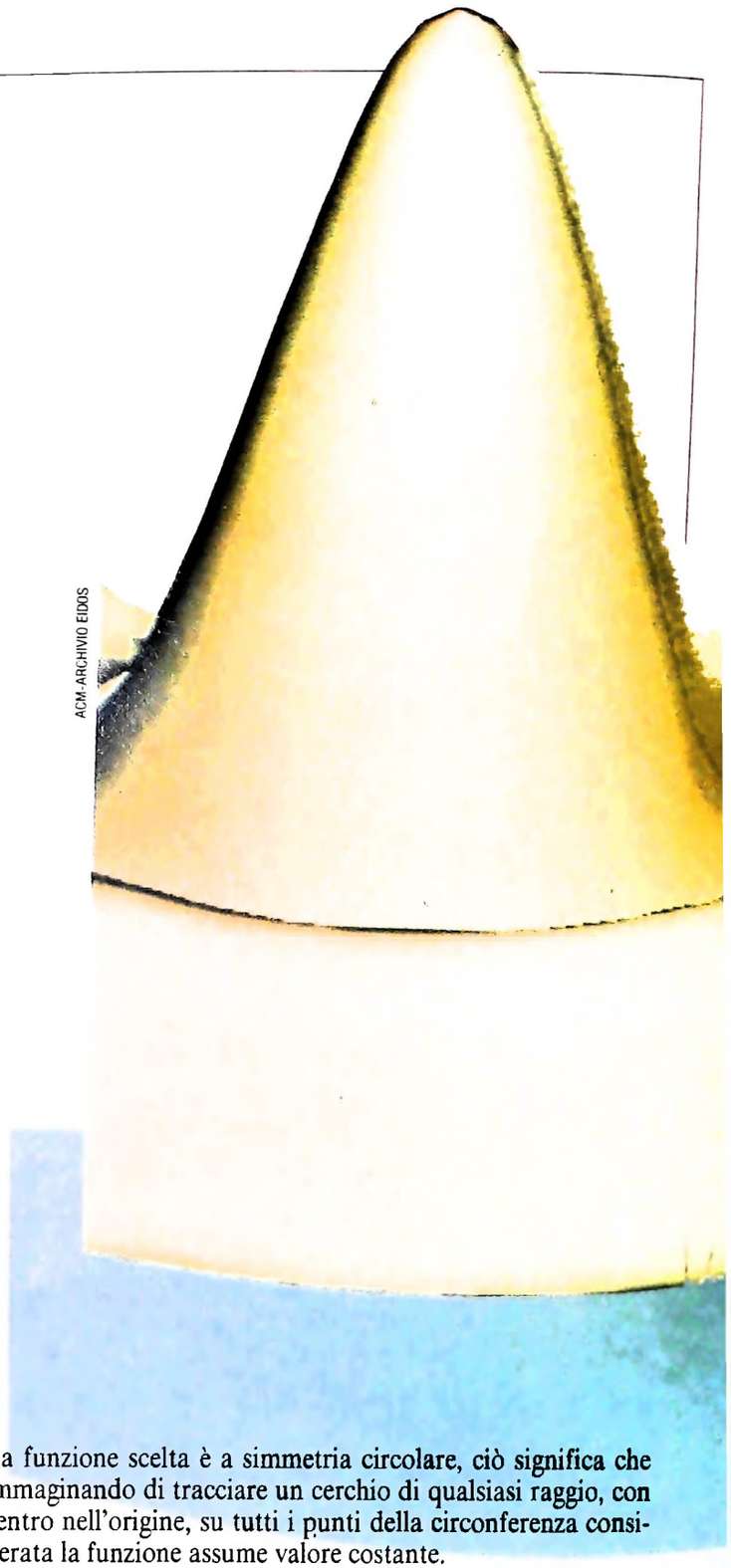
la rappresentazione grafica. Ma cosa accade se le variabili indipendenti aumentano? La risposta è immediata: bisogna aumentare le dimensioni dello spazio. Ad esempio, per una funzione a due variabili indipendenti, del tipo z=f(x,y), serviranno per la sua rappresentazione grafica 3 assi ortogonali: i due precedenti, più quello per le z. Ossia si passa da due a tre dimensioni. Analoga estensione vale per funzioni a più di due variabili.
In questa lezione vogliamo quindi introdurre un ampliamento dei concetti precedentemente trattati, affrontando il caso in cui la variabile dipendente y non sia solo funzione di x, ma anche di altre variabili.
Ci limiteremo però ad analizzare funzioni a due variabili indipendenti, dato che oltre questo limite non è semplice andare nella rappresentazione sullo schermo di un computer che è bidimensionale. Infatti mentre per le funzioni a due sole variabili la rappresentazione grafica è facilmente realizzabile nel piano e quindi altrettanto facilmente sullo schermo di un calcolatore, nel caso di funzioni a due variabili indipendenti si deve affrontare il problema di raffigurare lo spazio a tre dimensioni su un piano bidimensionale.
Quest'ostacolo, apparentemente molto complesso, viene risolto utilizzando un importante concetto della geometria euclidea: "la proiezione ortogonale"; ossia dato un generico punto nello spazio, questo può essere riportato o meglio, proiettato su un piano, semplicemente pensando di congiungere il piano con il punto dato tramite una linea perpendicolare al piano stesso; il punto d'intersezione tra il piano e la linea rappresenta la proiezione cercata. Utilizzando questo concetto è in generale possibile, data una qualunque forma tridimensionale (ad esempio il grafico di una funzione in due variabili indipendenti), ottenerne la corrispondente rappresentazione in due dimensioni.
Grande è l'uso che viene fatto delle funzioni a due variabili e svariati sono i campi di applicazione di questo strumento matematico.
Un semplice esempio è dato dal seguente problema: "si rilevi il peso di un campione finito di persone in funzione della loro altezza e età". Questo esempio, di genere statistico, può essere rappresentato nella forma:

Peso = f(altezza, età)

Solitamente è più opportuno effettuare uno studio di tali funzioni dal punto di vista grafico dato che non è molto agevole affidarsi solamente a un'analisi di tipo numerico.

Il programma

La rappresentazione di opportune funzioni matematiche a due variabili indipendenti costituisce uno dei più classici esempi di visualizzazione di forme tridimensionali, in quanto producono un effetto ottico particolarmente gradevole.
Si è quindi pensato di realizzare un programma che fornisce la rappresentazione di una funzione matematica la cui definizione formale è visibile alle linee 1000 e 1010 del listato.


ACM-ARCHIVIO EIDOS

La funzione scelta è a simmetria circolare, ciò significa che immaginando di tracciare un cerchio di qualsiasi raggio, con centro nell'origine, su tutti i punti della circonferenza considerata la funzione assume valore costante.
È inoltre possibile adottare un particolare metodo di visualizzazione che permette di ottimizzarne i tempi: se si valuta la funzione supponendo che essa abbia origine al centro dello schermo, è possibile sfruttare la simmetria sopra citata calcolandone il valore in un semi-schermo e ribaltando poi specularmente i valori così trovati.
Il passo di campionamento delle funzioni è direttamente dipendente dai valori costanti fissati nel programma e può quindi essere supposto variabile.
Sarebbe anche possibile sfruttare maggiormente la simmetria circolare, ma per semplicità ci limiteremo a questo utilizzo.

Rappresentazione tridimensionale di una funzione matematica a più variabili indipendenti. Immagini di questo tipo sono molto utilizzate per lo sviluppo di quel ramo matematico che va sotto il nome di topologia, ossia lo studio delle proprietà dello spazio.

```
20 CLS
25 '***INIZIALIZZAZIONE VARIABILI******
30 DIM C(32):A1=4:A2=16:A=70
40 A=A*3.14159/180 '*CONV.IN RADIANTI**
50 CS=COS(A):SN=SIN(A)
55 '***CALCOLO DELLA FUNZIONE**********
60 FOR I=0 TO 63 STEP .5
65 '***FOR I=-63 TO 0 STEP .5
70 ST=-100
80 X=I/A2
90 XA=INT(X*30+.5)
100 NR=8
110 FOR J=-NR TO 0
120 Y=J/A1
130 GOSUB 1000
140 C(-J)=Z
150 GOSUB 1500
160 NEXT J
170 GOSUB 2000
180 NEXT I
185 '***FINE PROGRAMMA*****************
190 BEEP:BEEP
200 PRINT @0,"F"
210 GOTO 210
215 '**********************************
218 '**CALCOLO COORDINATA Z*************
1000 S=.7*SQR(X^2+Y^2)
1010 Z=COS(S)+COS(2*S)/2+COS(6*S)/4
1020 Z=Z*1.05
1030 RETURN
1040 '**********************************
1490 '***PLOTTAGGIO FUNZIONE************
1495 '***CON TRASFORMAZIONE IN 3D******
1500 Y=INT(17*(CS*Y+SN*Z)+.5)
1510 IF Y>ST THEN ST=Y ELSE RETURN
1520 PSET(119-XA,31-Y)
1530 PSET(119+XA,31-Y)
1540 RETURN
1550 '**********************************
1900 '***PLOTTAGGIO ARRAY COORD. Z*****
2000 FOR J=0 TO NR
2010 Z=C(J)
2020 Y=J/A1
2030 GOSUB 1500
2040 NEXT J
2050 RETURN
2100 '**********************************
```

Il listato

Nella stesura del programma si sono utilizzate delle Keyword (letteralmente: Parole Chiave; è questa la normale definizione che comprende tutte le "parole riservate" di un certo linguaggio) il cui significato generale dovrebbe essere ormai noto. È comunque da notare l'uso particolare che in alcuni casi si è reso necessario.

1) Nella stesura dei commenti è stato utilizzato il carattere "'" al posto della parola REM con evidente risparmio in termini di spazio.

2) Si è fatto un uso particolare dei cicli di FOR... NEXT utilizzando STEP non interi e incrementi da valori di partenza negativi. Sono a questo scopo da notare le linee 60 e 65. Nella linea 60 si utilizza un normale ciclo di FOR... NEXT a incremento con partenza da zero; sostituendo alla linea 60 la linea 65 (ciò è possibile inserendo all'inizio della 60 un carattere "'" e togliendo i caratteri "***" dalla linea 65) si ottiene la visualizzazione con origine alle estremità laterali dello schermo, al posto di avere la partenza del disegno a centro-schermo.

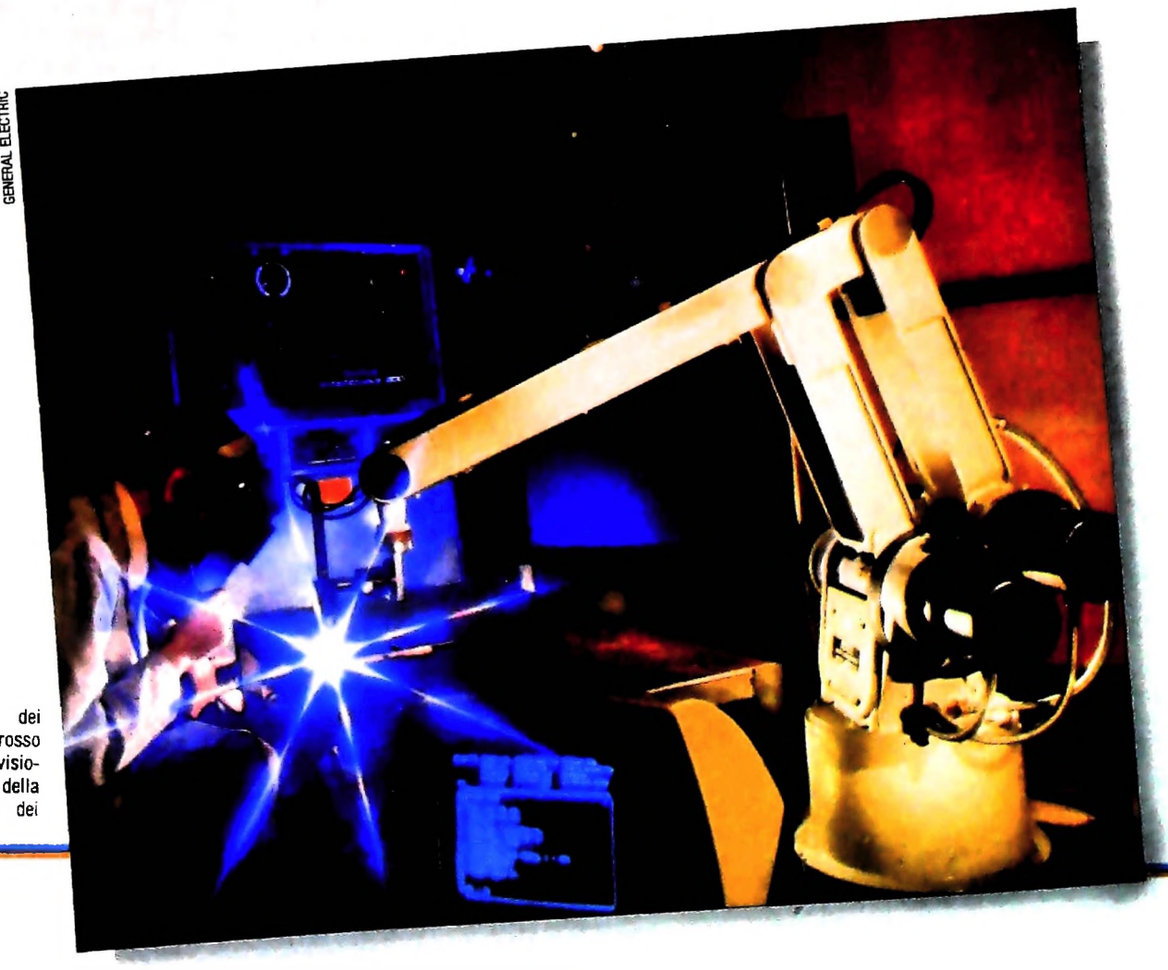
3) Alla linea 1000 è possibile intervenire sul coefficiente numerico (il cui valore è stato posto uguale a 0.7) in modo tale da variare l'ampiezza delle ondulazioni della funzione.
4) I punti della funzione vengono stampati a gruppi; ciò è ottenuto caricando i valori della funzione in un vettore denominato con la lettera C (dimensionamento della linea 30).
5) L'operazione di proiezione dei punti sullo schermo viene effettuata nella subroutine con partenza alla linea 1500.
6) L'immagine sullo schermo è mantenuta dalla linea 210 che, ciclando su se stessa, evita l'immissione di comandi inopportuni.

L'esecuzione del programma richiede, a causa dell'elevato numero dei calcoli, un tempo di esecuzione di circa 20 minuti al termine dei quali viene emesso un segnale acustico e compare una F in alto a sinistra ad indicare la fine del programma, dal quale si esce digitando i tasti Shift+Break.

Robotica

Addestrare un robot al compito per cui verrà utilizzato è un'operazione che richiede svariate tecniche, tra cui, la più diffusa a tutt'oggi, è quella di toglierlo dalla linea di produzione e, attraverso una manipolazione diretta, fargli apprendere i movimenti che dovrà poi compiere nella fase operativa.
Questo metodo, che certamente nella maggior parte dei casi è assai semplice, presenta numerosi svantaggi, tra i quali il primo è il costo. Togliere dalla linea di produzione un robot significa spesso interrompere l'intera produzione, e per evitar ciò si può soltanto ricorrere a un secondo modello, utilizzato esclusivamente per l'addestramento.
Il secondo grande svantaggio di un approccio diretto all'addestramento del robot deriva dal fatto che non si può prevedere e verificare in alcun modo se le operazioni che dovrà svolgere potranno effettivamente essere compiute o se sarà necessaria qualche modifica, sia nell'organizzazione del ciclo produttivo, sia nel posizionamento del robot, sia infine nella scelta del tipo stesso di robot.

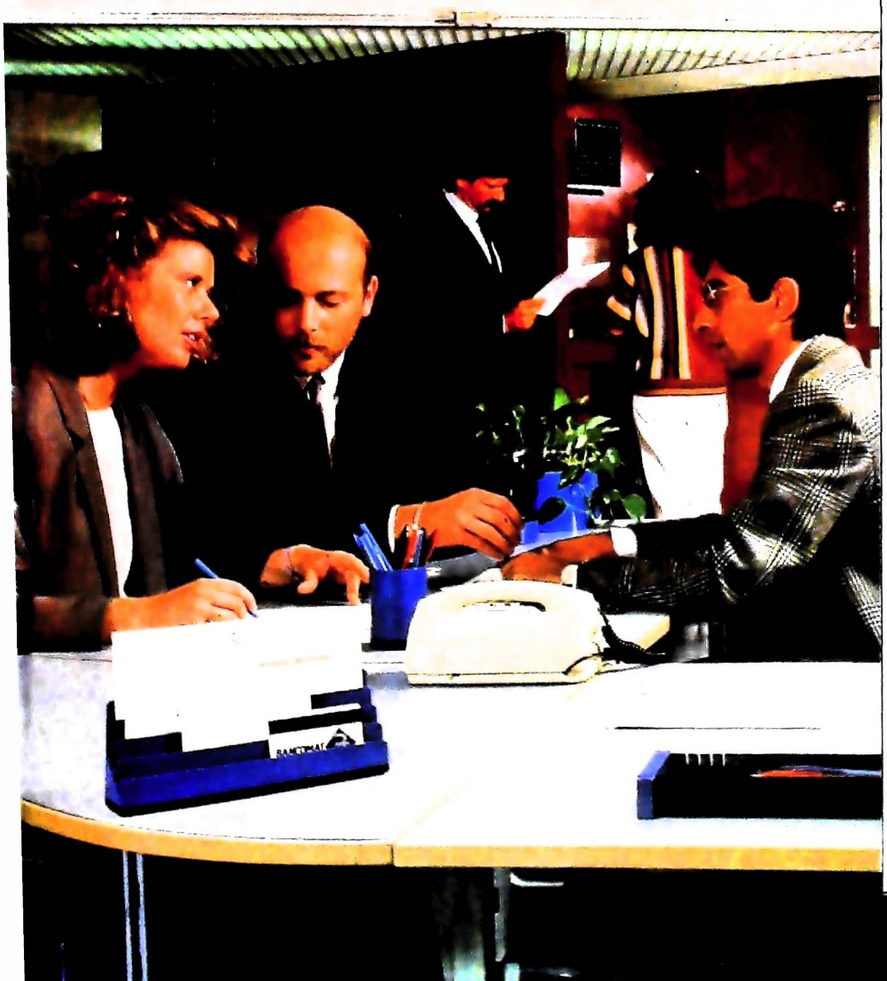
Per ovviare a queste limitazioni sta diventando sempre più diffusa una tecnica di verifica delle sequenze operative di un robot attraverso tecniche di simulazione grafica.
L'approccio consiste nel creare un modello grafico completo, anche se sintetico, della scena operativa e del robot. Successivamente viene analizzata la sequenza operativa e verificata la capacità del robot di raggiungere i punti desiderati secondo un percorso ottimale.
In questo modo si ha una visione diretta sul display del simulatore grafico dell'attività del robot ed è possibile intervenire cambiando sia la posizione degli oggetti della scena (ad esempio per evitare che il robot urti ostacoli durante i suoi movimenti) sia la posizione stessa del robot sulla scena.
A conclusione di queste attività di verifica e di ottimizzazione della sequenza operativa, il robot può venire istruito direttamente facendo generare al sistema di simulazione il programma che i processori di controllo del robot saranno poi in grado di interpretare ed eseguire.


GENERAL ELECTRIC

Sperimentazione dei sensori all'infrarosso in un sistema di visione e controllo della scena operativa dei robot.

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.