CADEL

37 CORSO PRATICO COL COMPUTER

421891

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

F4 F5 F6 F7 F8

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 del Calcolatori all'Università Autonoma Metropolitana di Città del Messico), GOFFREDO HAUS, MARCO MAIOCCHI, DANIELE MARINI, GIANCARLO MAURI, CLAUDIO PARMELLI, ENNIO PROVERA

Testi
ADRIANO DE LUCA, DANIELE MARINI, ETTORE DECIO, LUCA SPAMPINATO
Etnoteam (ADRIANA BICEGO)

Tavole
Logicai Studio Communication
Il Corso di Programmazione e BASIC è stato realizzato da Etnoteam S.p.A., Milano
Computergrafica è stato realizzato da Eidos, S.c.r.l., Milano
Usare il Computer è stato realizzato in collaborazione con PARSEC S.N.C. - Milano

Direttore Editoriale
ORSOLA FENGHI

Redazione
CARLA VERGANI
LOGICAL STUDIO COMMUNICATION

Art Director
CESARE BARONI

Impaginazione
BRUNO DE CHECCHI
PAOLA ROZZA

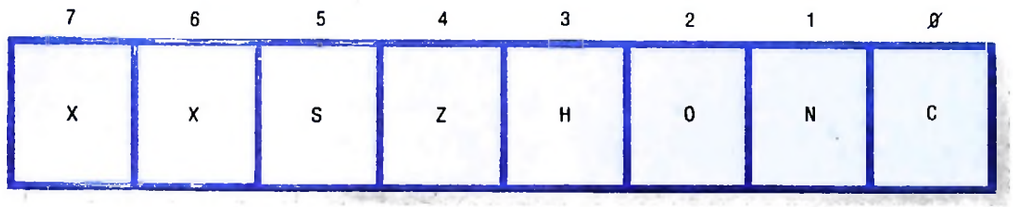
Programmazione Editoriale
ROSANNA ZERBARINI
GIOVANNA BREGGÈ

Segretarie di Redazione
RENATA FRIGOLI
LUCIA MONTANARI

L'USO DELLO STACK (II)

Un'ulteriore utilizzazione dello STACK, il registro di stato che, con i suoi otto bit, fornisce informazioni sullo stato dei microprocessori.

Status Flag (registro di stato)

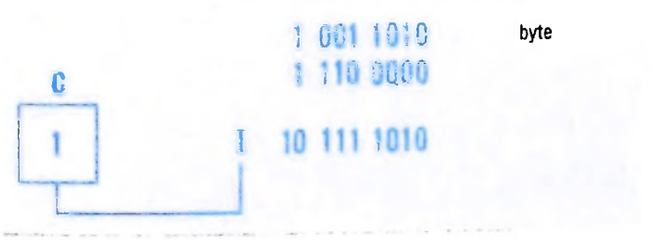


Questo registro (figura sopra) fornisce informazioni sullo stato del microprocessore dopo ogni esecuzione di istruzione. È composto da 8 bit due dei quali non utilizzati, che hanno i seguenti significati:

C = Carry (Riporto)
N = Somma/Sottrazione
O = Overflow (Eccesso)
Z = Zero
S = Segno

"C" Carry

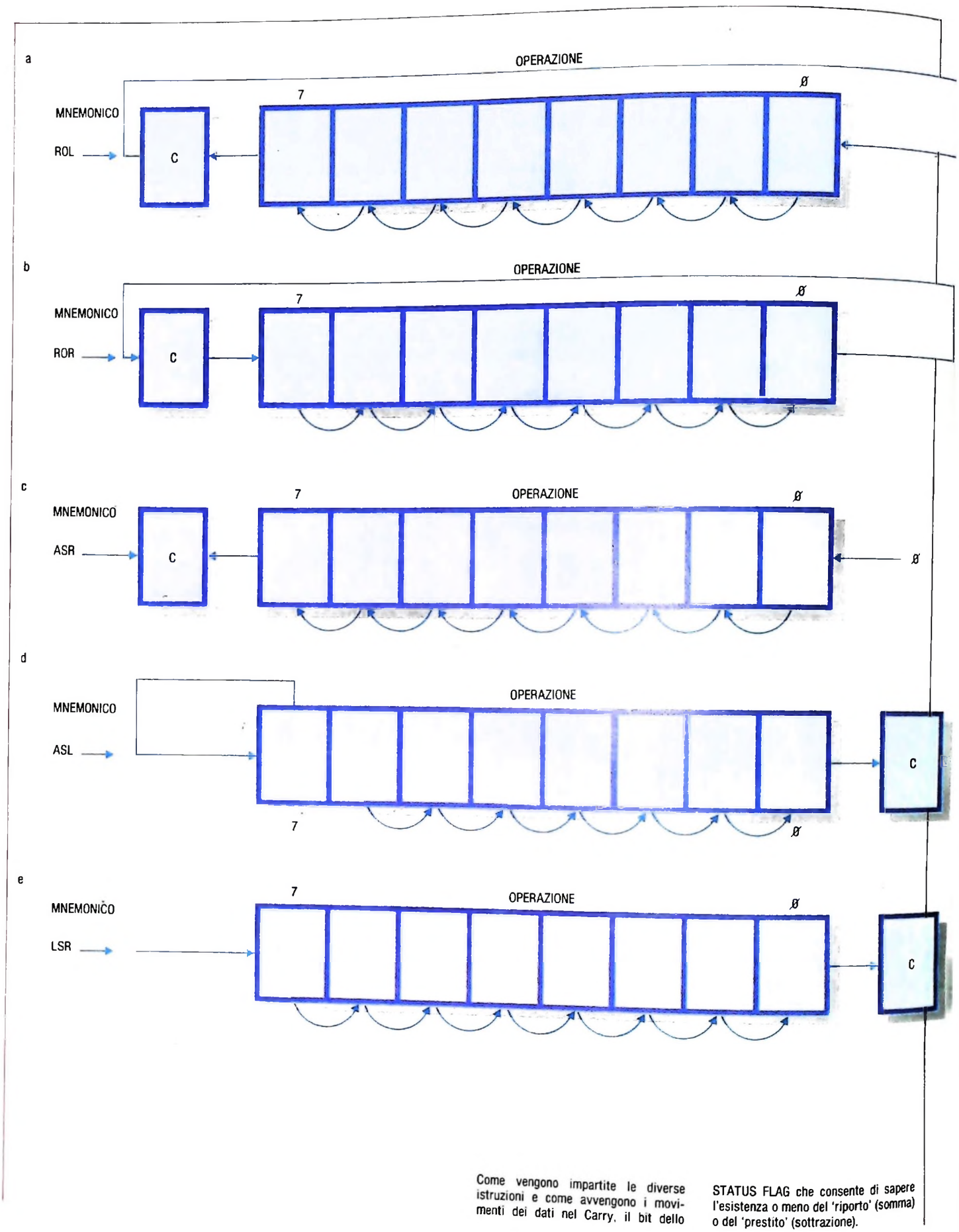
Il bit del Carry è posto a 1 o a 0, a seconda dell'operazione eseguita. Per le istruzioni di somma, per esempio, la generazione del Carry = 1 è dovuta al riporto, se esiste, fra la somma dei due numeri, come nell'esempio che segue:



Lo stesso procedimento avviene quando si deve eseguire una sottrazione: in questo caso vuol dire che nell'eseguire l'operazione siamo stati costretti ad "andare a prendere in prestito una unità".
La possibilità di sapere attraverso il "C" dello STATUS FLAG l'esistenza o no del "riporto" o "prestito" permette di estendere facilmente la grandezza dei numeri da uno a due byte, per esempio.
Quando nelle operazioni suddette non c'è riporto o prestito, nel Carry viene depositato uno zero.
Quando ci troviamo in presenza di istruzioni di rotazione come nella figura a della pagina seguente l'istruzione ROL vuol dire: ruotare il contenuto del registro (A o B) a sinistra attraverso il Carry. In questo caso il contenuto della casella "C" nello STATUS FLAG è uguale al bit che gli viene trasmesso dal registro.
Con l'istruzione ROR (figura b) succede la stessa cosa.
Le altre istruzioni che modificano ancora il "C" sono le istruzioni di SHIFT (scorrimento) come l'istruzione ASL che vuol dire: "spostamento a sinistra" (figura d), mentre ASR (figura c) vuol dire: "spostamento a destra".
L'ultima istruzione presa in considerazione dalla nostra Uamicro III è la LSR (figura e) che vuol dire "spostamento a destra", però di altra forma.

"N" Somma/Sottrazione

Questo bit viene messo a zero quando si sta eseguendo una operazione di somma, a 1 quando si esegue una sottrazione.

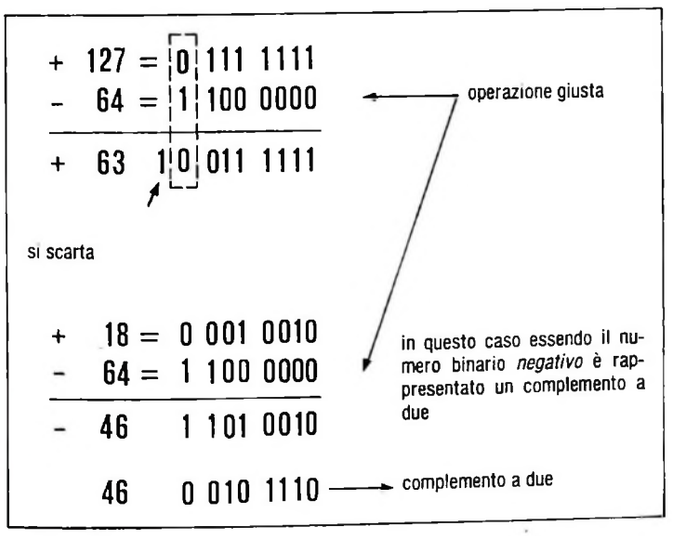


Come vengono impartite le diverse istruzioni e come avvengono i movimenti dei dati nel Carry, il bit dello STATUS FLAG che consente di sapere l'esistenza o meno del 'riporto' (somma) o del 'prestito' (sottrazione).

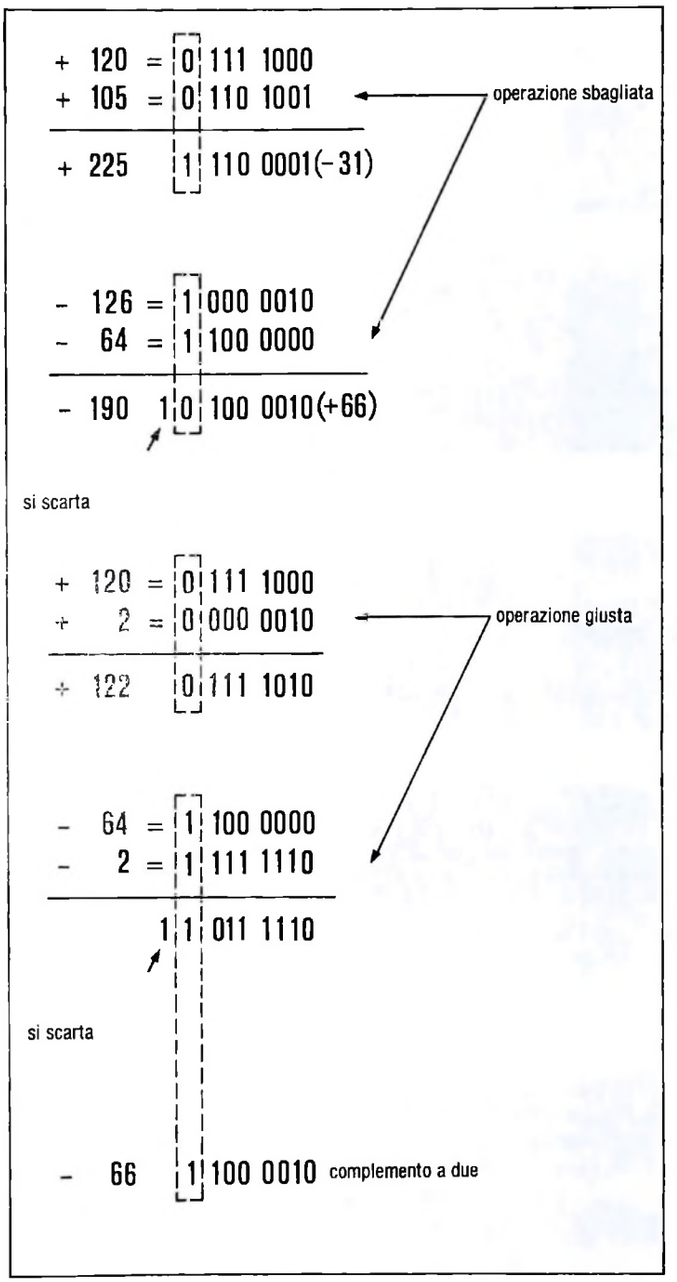
"O" Overflow (traboccamento)

Nelle operazioni aritmetiche questo bit indica un "traboccamento", cioè che il risultato dell'operazione è più grande del massimo numero possibile (+ 127) o è minore del numero minimo (-127). Questa condizione di "eccesso" è determinata dall'esame dei segni degli operandi, vediamo come. Prima di tutto è utile ricordare che gli ALU sono capaci solo di sommare e che la sottrazione, come sappiamo, si effettua sommando al primo numero il complemento a due del secondo. Questo determina, come prima cosa, che i numeri rappresentati in un byte vanno al massimo da + 0 a + 127 per i numeri positivi e da - 0 a - 127 per i numeri negativi, ricordando, ancora una volta, che un numero si considera positivo quando ha il bit più a sinistra uguale a zero, mentre si considera negativo quando è uguale a uno, e che questo bit non entra nella determinazione dell'equivalente decimale se non come segno. Nella figura di pag. 576, a destra, possiamo quindi vedere la suddivisione dei numeri positivi e negativi. Va notato, comunque, che il valore assoluto dei numeri binari negativi si ricava facendo il complemento a due degli stessi. Per esempio, facendo il complemento a due del binario

Dopo queste premesse ritorniamo al nostro problema di *overflow*.
Quando gli operandi di una somma sono di segno differente, il che vuol dire, chiaramente, che siamo in presenza di una sottrazione, allora è impossibile che si generi un *overflow* come possiamo vedere dagli esempi che seguono:

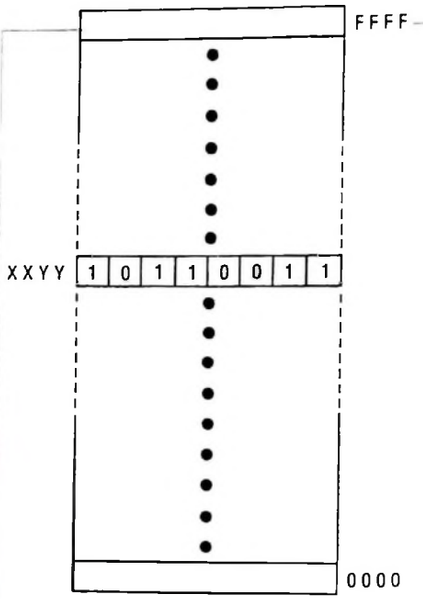


Quando gli operandi di una somma sono dello stesso segno, entrambi positivi o negativi, allora si genera un *overflow*, quindi un errore, quando il segno del risultato è differente da quello degli operandi.
Tutto va bene quando il segno del risultato è uguale a quello degli operandi.
Osserviamo infatti gli esempi che seguono:



"Z" Zero

Questo bit è messo a 0 o a 1 come conseguenza del risultato di alcune operazioni. Con le operazioni aritmetiche e logiche (figure b, c, d di pagina 574) a 8 bit il bit Z viene messo a



Sopra, una parola in memoria con indirizzo XXYY e l'uso del bit Z dello STATUS FLAG per testare i valori di ogni bit. A lato, rappresentazione dei numeri positivi e negativi (complemento a due).

SEGNO	NUMERO BINARIO	NUMERO DECIMALE
0	000 0000	+0
0	000 0001	+1
:	:	:
:	:	:
:	:	:
:	:	:
0	111 1110	+126
0	111 1111	+127
1	000 0000	-0
1	000 0001	-127
1	000 0010	-126
:	:	:
:	:	:
:	:	:
:	:	:
1	111 1110	-2
1	111 1111	-1

complemento a due

uno quando come risultato delle operazioni suddette nel registro sono presenti soltanto zeri. Se invece non avviene questo viene depositato uno zero.
Nelle istruzioni che testano i singoli bit nelle parole in memoria o nei registri il valore della casella Z sarà uguale all'inverso del valore testato.
Per esempio (figura in alto a sinistra) se vogliamo sapere il valore del bit di posizione 0 allora l'istruzione che testa il bit, che nel nostro caso è uguale a 1, mette nella casella Z dello STATUS FLAG il valore 0.

Al contrario sarebbe successo se per esempio avesse testato il bit di posizione 2 0 3.

"S" Segno

Questo bit immagazzina lo stato del bit più significativo dell'accumulatore (MSB). Come sappiamo un numero positivo è identificato con uno 0, mentre un numero negativo con un 1 (figura in alto a destra).

Glossario

ASCII - *Sigla di American Standard Code for Information Interchange (codice americano standard per lo scambio di informazioni). Il più diffuso fra i codici internazionali che rappresentano numeri, lettere e simboli con valori binari univoci, comprensibili ed elaborabili dal computer. È anche il principale fra i codici di trasmissione dati fra sistemi di elaborazione.*

Compatibilità - *Si dice che due computer sono compatibili se i programmi scritti per l'uno possono essere eseguiti anche sull'altro. La compatibilità totale fra due sistemi è rara: più spesso si limita ad alcuni aspetti, e non è detto che sia biunivoca, che cioè anche i programmi scritti sul secondo possano essere eseguiti sul primo.*

Transistor - *Un componente elettronico che può svolgere funzioni di commutatore o di amplificatori. Il transistor, presente nei piccoli apparecchi radioriceventi e negli amplificatori, è il componente fondamentale dei circuiti integrati, sui quali si trova in forma miniaturizzata. Su un chip di silicio di pochi millimetri di lato possono trovare posto anche migliaia di transistor.*

Volatile - *Si dice di una memoria i cui contenuti si perdono quando viene tolta l'alimentazione. La memoria RAM di un computer è in genere volatile; è non volatile, però, la RAM dei calcolatori portatili (come l'M10 Olivetti), che conserva dati e programmi immessi dall'utente anche a calcolatore spento.*

Winchester - *Una tecnologia di costruzione di dischi rigidi per la realizzazione di sistemi di memoria di massa per calcolatori, sviluppata negli anni Settanta nei laboratori della IBM. Un disco Winchester è un disco magnetico rigido in un contenitore sigillato; le operazioni di lettura e scrittura sono effettuate da una testina che non viene mai in contatto con il disco stesso, con possibilità di danneggiamento nulle. Il contenitore sigillato impedisce la penetrazione di agenti contaminanti dall'esterno. Grazie a queste caratteristiche, un disco Winchester può immagazzinare grandi quantità di informazioni (da 5 a 20 milioni di Kbyte e anche più).*

LA DEFINIZIONE DEI REQUISITI E DELLE FUNZIONI

È il momento in cui si determinano gli obiettivi e le funzioni necessarie per risolvere il problema individuato.

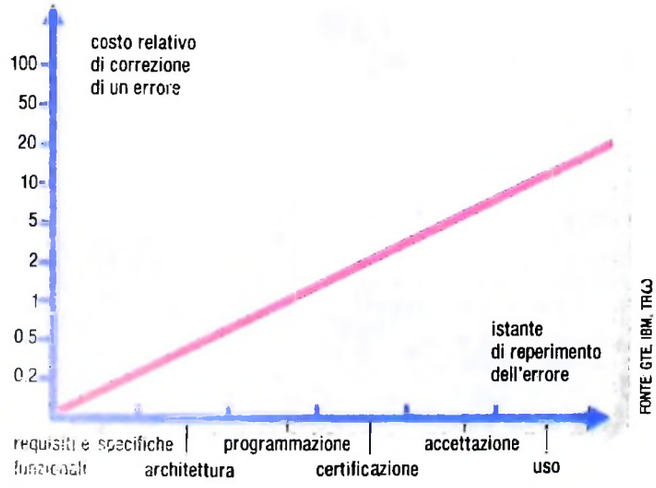
Esaminiamo con più dettaglio le prime fasi dell'attività di sviluppo del software: si tratta della definizione dei *requisiti* (cioè l'individuazione precisa e la delimitazione del problema da risolvere, il "perché" del prodotto da sviluppare) e della definizione delle *specifiche funzionali* (cioè la descrizione completa di come il prodotto dovrà essere usato, il *cosa* deve essere fatto).

Le prime fasi nello sviluppo di software sono quelle che ancora oggi sono meno comprese, meno stabili, con un minore numero di strumenti a disposizione e più delicate. Commettere un errore in queste fasi è estremamente pericoloso, in quanto si viene a determinare lo sviluppo di un prodotto che non risponde agli obiettivi o che addirittura non fornisce le funzioni necessarie per risolvere il problema individuato. Questa caratteristica è chiaramente documentata dallo schema della figura a lato, che illustra l'andamento del costo di correzione di un errore trovato in un prodotto software, in dipendenza del momento in cui esso viene reperito: il costo di correzione di un errore trovato durante l'uso del prodotto può essere decine di volte superiore a quello per errori scoperti durante la programmazione, e centinaia di volte superiore a quelli scoperti durante le prime fasi di definizione dei requisiti e delle specifiche funzionali.

Ciò può essere spiegato col fatto che errori commessi durante la programmazione richiedono cambiamenti ai programmi, ma lasciano inalterati i risultati delle attività precedenti (come le specifiche funzionali), mentre errori commessi nelle prime fasi non trovano nelle successive strumenti in grado di evidenziarli, se non durante l'uso da parte di un utente che verifica l'inadeguatezza del prodotto, e comportano rifacimenti di specifiche, e conseguentemente interventi architetturali e di programmazione, con naturalmente una moltiplicazione dei costi.

Per tali motivi si sono portati avanti molti studi che forniscono ai progettisti di software metodi e strumenti atti ad aiutarli proprio in queste fasi iniziali.

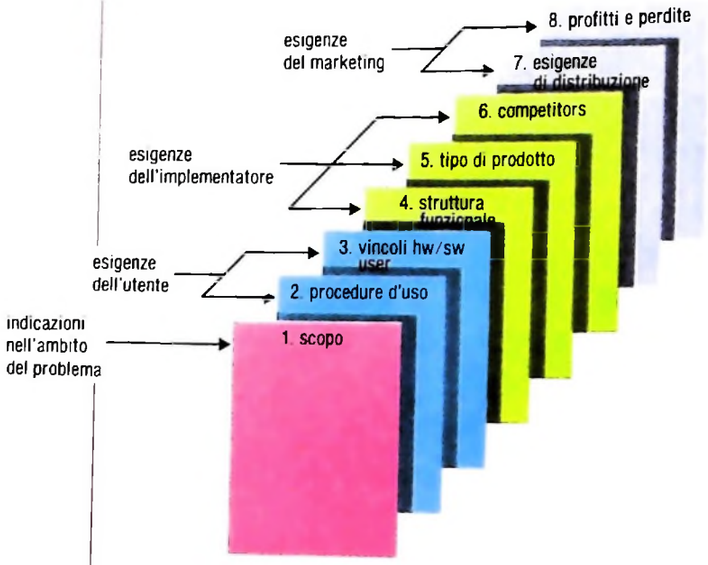
I vari metodi messi a disposizione per queste fasi si trovano a cavallo tra l'analisi di aspetti di organizzazione aziendale e l'attività di definizione di prodotti software: infatti la definizione dei requisiti di un prodotto e la descrizione delle sue modalità di funzionamento possono essere o meno adeguate in funzione di una specifica realtà aziendale e la valutazione di adeguatezza può essere fatta solo calando in una descrizione di tale realtà la descrizione del modo con cui i nuovi strumenti permettono di effettuare le operazioni di tutti i giorni.



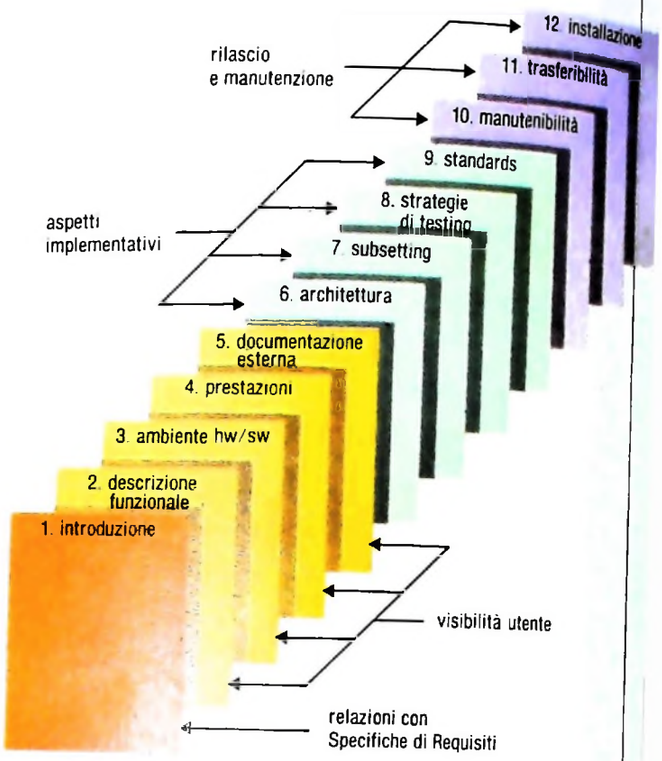
La definizione dei requisiti e delle funzioni. Il grafico individua i costi di correzione di un errore in rapporto alla fase in cui esso viene individuato.

La linea di tendenza attuale si spinge verso la definizione di *linguaggi di specifica*, che, così come i linguaggi di programmazione permettono di descrivere in modo non ambiguo algoritmi, dovrebbero descrivere formalmente senza ambiguità le caratteristiche funzionali di un sistema, con possibilità di validazione, di verifiche di consistenza, e così via. Sforzi in tali direzioni sono tuttavia da considerare ancora come attività di ricerca, mentre le attuali metodologie a disposizione nell'ambito della normale produzione tendono a fornire strumenti di guida all'esame dei problemi, in modo da ridurre il rischio di commettere errori per mancato approfondimento di analisi.

Un esempio di standard di Specifiche di Requisiti (SR)



Un esempio di standard di Specifiche Funzionali (SF)



Sopra, un esempio di standard di Specifiche di Requisiti (SR); è molto importante individuare in partenza la reale situazione dell'ambiente in cui si vuole intervenire (lo scopo). A lato, uno standard di Specifiche Funzionali (SF), cioè la descrizione completa di come il prodotto, individuato precedentemente dalle specifiche di requisiti, dovrà essere usato.

Gli standard di documentazione

In un tale ambito possono giocare un ruolo molto rilevante strumenti molto semplici, come gli "standard" di documentazione. Organizzando l'ambiente di sviluppo di software secondo un ciclo come quello precedentemente descritto, è previsto che al termine delle due fasi iniziali di lavoro siano disponibili due documenti di specifiche di requisiti e di specifiche funzionali; questi due documenti devono sottostare ad alcune regole, che costituiscono lo standard specifico dell'ambiente:

- ciascun documento deve essere organizzato in capitoli e paragrafi, i cui titoli sono fissati dallo standard;
- nessun capitolo o paragrafo può essere omesso, anche se il suo contenuto non è applicabile al tema descritto: in tal caso si dichiarerà esplicitamente in tale paragrafo o capitolo che esso è stato lasciato intenzionalmente vuoto, perché non applicabile o da definire successivamente.

In tal modo:

- chi scrive il documento ha una guida relativa ai temi da trattare, senza correre il rischio di non approfondire alcuni aspetti per pura dimenticanza;
- chi verifica lo stato del documento è in grado facilmente di evidenziarne la completezza o le carenze.

Nelle figure in alto sono illustrati due tipici esempi di indici di specifiche standard.

Le metodologie

È difficile descrivere in poco tempo le caratteristiche delle metodologie disponibili per le prime fasi di sviluppo; peraltro sarebbe poco significativo farne un lungo elenco senza alcuna spiegazione.

Ci limitiamo qui a indicare che esse sono normalmente costituite da una precisa procedura operativa che, attraverso vari passi, permette di ottenere il risultato richiesto.

Molto spesso sono adottati specifici strumenti grafici di descrizione e viene richiesta la stesura di precisi documenti. Talvolta diverse metodologie tendono a spaziare su fasi diverse: alcune si occupano solo del problema dei requisiti, altre sconfinano anche verso la definizione funzionale, altre metodologie ancora si occupano solo di quest'ultima fase fornendo però indicazioni sull'architettura del prodotto, e così via.

Ancora, molto spesso i metodi proposti sono legati alle caratteristiche del problema da risolvere: nello sviluppo di prodotti software per applicazioni amministrative e gestionali c'è una grossa enfasi su aspetti di organizzazione del lavoro e su strutture di dati mentre nello sviluppo di sistemi per controllo di processi industriali (come per esempio il controllo di una centrale elettrica o di una raffineria) l'attenzione è spostata sulla descrizione dinamica dei processi da controllare, e così via.

Le reti di Petri

Tra gli strumenti che hanno incontrato successo nella definizione di requisiti e di specifiche funzionali, e che sono poi stati inseriti in opportune metodologie ci sono le "reti di Petri". Si tratta di strumenti grafici che possono, a volontà, essere affiancati da una descrizione altamente formalizzata in termini matematici, che permettono di descrivere staticamente e dinamicamente il modo di evolvere di un sistema (per esempio, del sistema "azienda-prodotto software da sviluppare per la sua meccanizzazione"). Introdotte dal professore universitario tedesco Petri, esse sono "grafi" in cui sono presenti due tipi di "nodi" collegati da archi orientati: i *posti* e le *transizioni* (figura in basso a sinistra).

Interpretando le transizioni come attività e i posti come risorse da consumare (se sono in entrata a una transizione) o da produrre (se sono in uscita) ovvero come condizioni che devono essere verificate, è possibile descrivere sistemi anche molto complessi con estrema chiarezza. Osserviamo, con un esempio derivato da una semplificazione di un caso reale, la capacità delle reti di Petri di modellare sistemi in cui attività umane ed automatiche si integrino (figura in basso a destra).

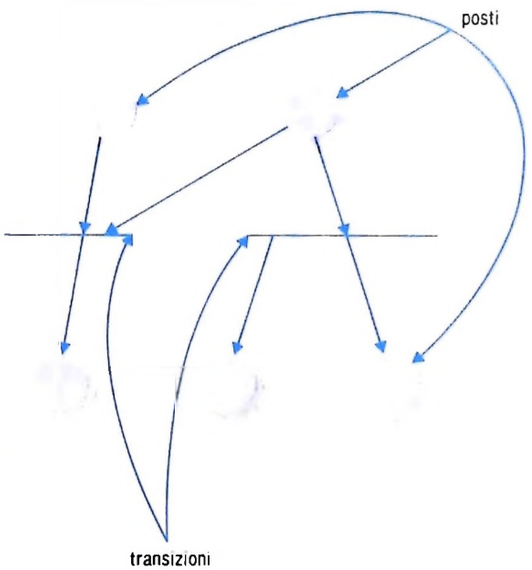
L'esempio mostra le operazioni di "acquisizione merci" da parte di un magazzino nell'ambito di un'azienda.

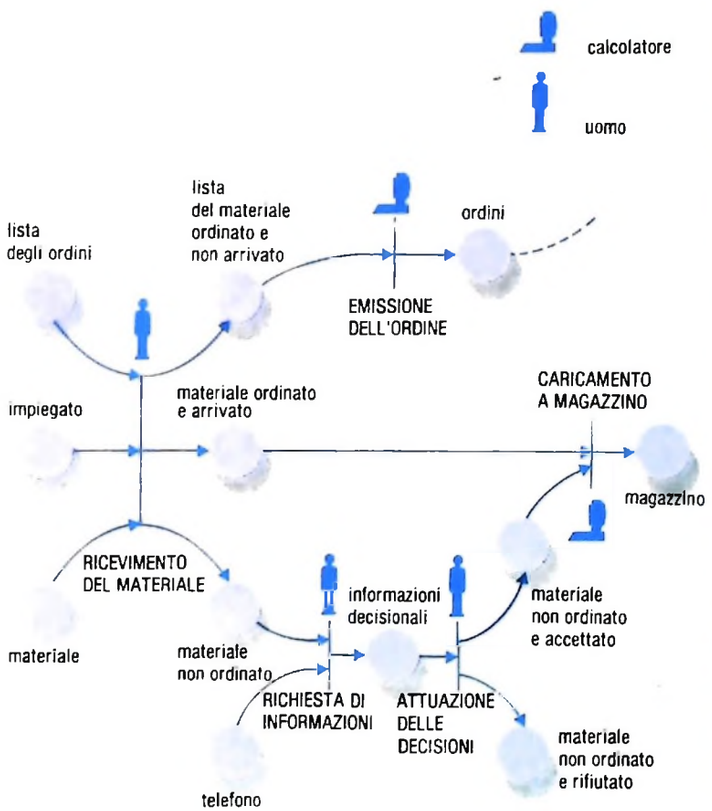
L'attività di "ricevimento del materiale" prevede le risorse di: un impiegato, una lista di ordini effettuati e il materiale che arriva; tale attività porterà come risultato ad una lista di merce ordinata e non pervenuta, alla merce ordinata e pervenuta, e a merce che, per errore o per altri motivi, è giunta senza essere stata ordinata.

La prima attività, manuale, alimenta quindi in parallelo altre tre attività; la prima di queste, l'"emissione dell'ordine", è automatica, ovvero effettuata mediante elaboratore, e parte direttamente dalla lista del materiale ordinato e non pervenuto: gli ordini risultanti verranno poi "consumati" da un'altra attività aziendale qui non evidenziata; la seconda attività, il "caricamento a magazzino", non può aver luogo in quanto non tutte le sue precondizioni sono soddisfatte; la terza, la "richiesta informazioni", ha invece luogo e consiste in una telefonata alla persona competente per definire il comportamento sulla merce non ordinata.

Le informazioni derivanti da tale telefonata permettono di suddividere il materiale in merce da rifiutare e merce da caricare in ogni caso a magazzino, permettendo lo svolgimento dell'attività precedentemente in attesa.

Si osservi che in un tale schema vengono messi in evidenza non solo aspetti di automazione e manualità, ma anche aspetti di competenza, problemi logistici (il telefono non può essere in un capannone a 300 metri di distanza !), relazioni fra le varie funzioni aziendali, gli addetti necessari ecc.

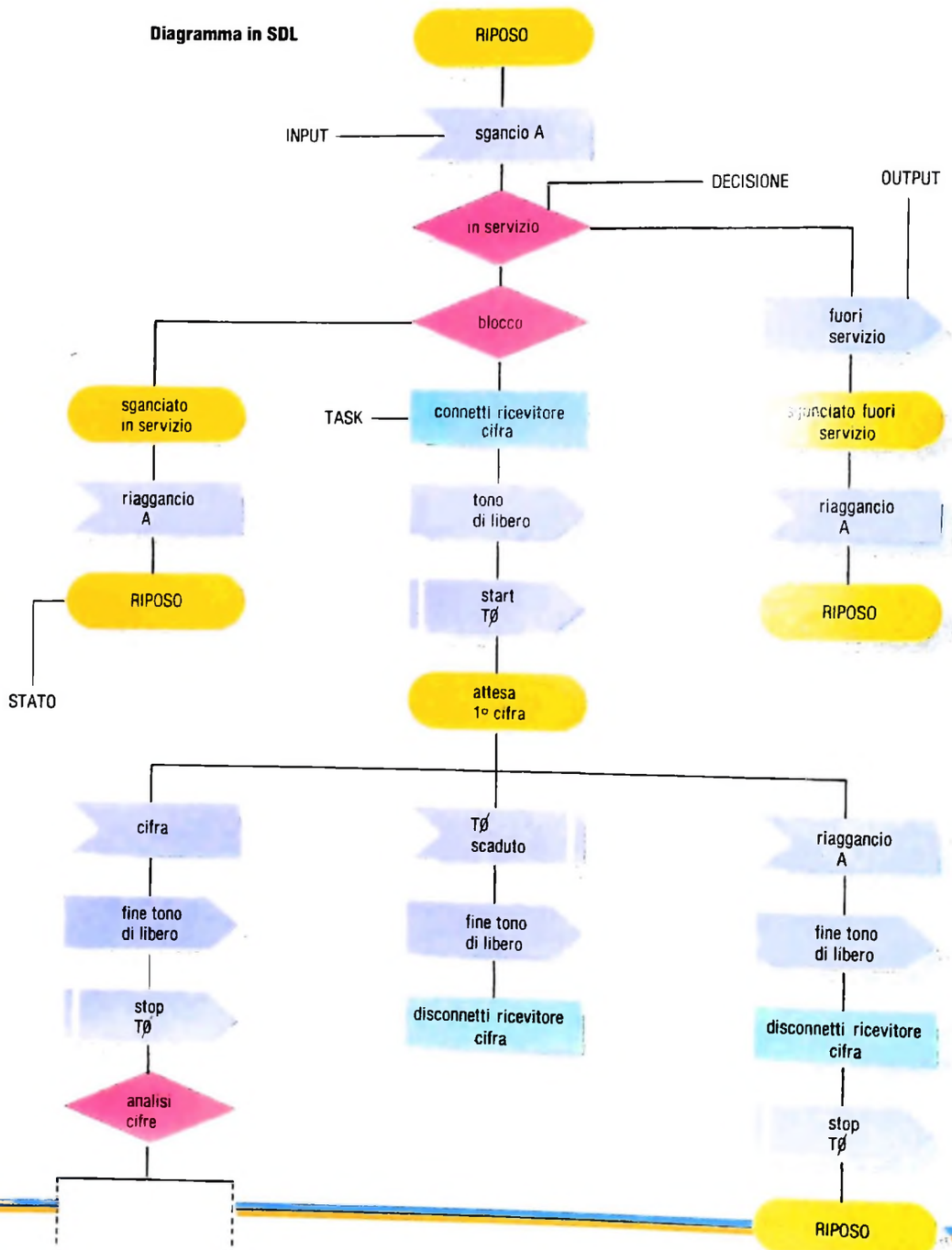




SDL: Specification and description language

Un altro esempio interessante di strumento per la descrizione delle specifiche funzionali è l'SDL, messo a punto dal CCITT (International Telegraph and Telephone Consultative Committee), che è un organo consultivo internazionale che si occupa del problema dello sviluppo di software per le centrali telefoniche, per le reti ecc.
Esso si basa su un certo insieme di concetti base rappresentati graficamente, quali ad esempio il concetto di STATO (condizione in cui si trova un processo in attesa di un evento esterno), di INPUT (ovvero un segnale un arrivo riconoscibile da un processo), di OUTPUT (generazione di un segnale verso qualche processo), di DECISIONE (verifica di una condizione per scegliere come proseguire l'evoluzione del processo), di TASK (azione svolta all'interno del processo) e così via.
Il diagramma riportato illustra con SDL il comportamento di un processo telefonico che deve reagire a un utente che vuole effettuare una telefonata.
Il processo è in riposo; quando arriva il segnale di sgancio, che indica che l'utente ha sollevato il ricevitore, verifica la possibilità di fornire il servizio correttamente e (ramo centrale) si predispone alla ricezione delle cifre del numero telefonico, inviando subito dopo il segnale di libero; quindi viene fatto partire un "cronometro" che misura il tempo che intercorre prima dell'arrivo della prima cifra; se la cifra arriva, si prosegue con il primo ramo; se scade il tempo previsto prima che la cifra arrivi, si prosegue sconnettendo il ricevitore, e così via.

Diagramma in SDL



Lezione 36

Il programma di MERGE

Abbiamo visto nelle due lezioni precedenti la costruzione di un programma di MERGE, sviluppato secondo il metodo TOP DOWN. Riportiamo ora il modulo finale, che realizza la visualizzazione del file prodotto dal MERGE. Poiché abbiamo già esaminato problemi di scansione sequenziale di un file, riportiamo direttamente il modulo BASIC:

```
2000 ' Visualizzazione arch. output
2010 CLS
2020 PRINT "  VISUALIZZAZIONE ARCHIVIO FINALE"
2030 OPEN "RAM:C" FOR INPUT AS #1
2040 ' While not eof(C) do
2050 IF EOF(1) THEN 2100
2060 INPUT #1,N2$,T2
2070 PRINT N2$,T2
2080 GOTO 2040
2100 ' Endwhile
2110 CLOSE #1
2500 RETURN
```

Questo modulo ci sarà molto utile per verificare la correttezza del programma senza ricorrere alla visualizzazione dei file dal menù principale.

Come verificare la correttezza del programma

Il programma che abbiamo costruito deve prevedere un insieme di casi particolari per realizzare il MERGE tra i due archivi. Solo se tutti questi casi vengono provati possiamo dire che il programma è corretto. A questo scopo dobbiamo individuare un insieme di prove che verifichino tutti i casi previsti.
Per prima cosa però incominciamo a costruire un elenco dei casi da provare:

1. il file A fornisce un nominativo che è minore di quello fornito dal file B

2. il file A fornisce un nominativo che è maggiore di quello fornito dal file B

3. il file A contiene un nominativo presente anche su B

4. il file A è più "corto" del file B

5. il file A è più "lungo" del file B

6. i due file contengono lo stesso numero
   di nominativi.

Quella che abbiamo descritto è una "check list" delle funzioni del programma. Come nel caso del controllo per esempio di un aereo prima di un volo, è necessario disporre di un elenco delle verifiche da effettuare per garantire che il prodotto, nel caso il velivolo, funzionerà correttamente.
A questo punto bisogna indicare, a fianco di ogni funzione, le operazioni che ci consentiranno di provarla. Nel nostro caso si tratterà di individuare un insieme opportuno di dati per il primo e il secondo archivio.
Ora, se facciamo un po' di attenzione, possiamo generare insiemi di nominativi per i due archivi che permettano di verificare più funzioni in una volta. Per esempio la seguente sequenza di dati:

file A	file B
Bianchi	Bianchi
Rossi	

verifica sia il caso di nominativi uguali sui due file sia il caso di file A più lungo del file B.
Cominciamo allora coll'individuare funzioni che possono essere verificate con uno stesso insieme di dati. Ciascuno di questi insiemi costituisce una prova o test, a cui pertanto assegneremo un nome.
Assegnamo quindi un test a ogni funzionalità, tenendo già conto del fatto che potremo pensare a test che verifichino più funzioni contemporaneamente.
Possiamo allora riscrivere la check list completa delle prove:

FUNZIONALITÀ	TEST
1. Il file A fornisce un nominativo che è minore di quello fornito dal file B	Test1
2. il file A fornisce un nominativo che è maggiore di quello fornito dal file B	Test2
3. il file A contiene un nominativo presente anche su B	Test3
4. il file A è più "corto" del file B	Test2
5. il file A è più "lungo" del file B	Test1
6. il file A contiene lo stesso numero di nominativi di B	Test4

Definiamo ora i dati relativi a ciascun test in modo che rispettino i requisiti di verifica delle funzioni attribuite a ciascuno. Usiamo per brevità singole lettere per i no-

La funzione EOF

Abbiamo già visto, in una serie di esempi, l'uso della funzione EOF, per verificare la fine dei dati in un file in fase di lettura.

La funzione EOF, applicata dopo una OPEN o una INPUT, fornisce il valore 0 se il file contiene informazioni e può quindi essere letto e il valore -1 in caso contrario.
Così dato il seguente file:

```
CONTIENE UNA SOLA STRINGA
```

se eseguiamo il seguente programma:

```
10 OPEN "RAM:TRYEOF" FOR INPUT AS #1
20 ' While not eof(TRYEOF) do
30 IF EOF(1) THEN 100
40 PRINT "valore di EOF: ";EOF(1)
50 INPUT #1,A$
60 PRINT "stringa letta: ";A$
70 GOTO 20
100 ' Endwhile
110 PRINT "valore di EOF: ";EOF(1)
120 CLOSE #1
130 END
```

otterremo il seguente risultato:

```
Ok
RUN
valore di EOF:  0
stringa letta: CONTIENE UNA SOLA STRINGA

valore di EOF: -1
Ok
```

Si noti la differenza tra l'esaurimento delle informazioni contenute in un file, che si verifica o se questo è vuoto o quando tutti i dati sono ormai stati letti, e l'esaurimento dello spazio di memoria disponibile quando si è in fase di registrazione. Questo secondo caso produce un errore di overflow.
Così se eseguiamo il seguente programma (e in particolare abbiamo già informazioni registrate nel nostro M10):

```
10 OPEN "RAM:FILL" FOR OUTPUT AS #1
20 FOR I=1 TO 3200
30 PRINT #1,"1234567890"
40 NEXT I
50 CLOSE #1
60 END
```

otterremo il seguente risultato:

```
Ok
run
?OM Error in 30
Ok
```

minativi e numeri progressivi a partire da 1 per i numeri telefonici. In questo modo verifichiamo anche che nel caso di nominativo doppio non solo questo venga prelevato una sola volta, ma sia scelto quello proveniente dal file B.
Riportiamo di seguito i dati di prova di ciascun test e i risultati attesi in corrispondenza:

TEST1

FILE A	FILE B
A 1	B 4
C 2	D 5
Z 3	

A	1
B	4
C	2
D	5
Z	3

TEST2

FILE A	FILE B
B 1	A 3
D 2	C 4
	E 5
	F 6

A	3
B	1
C	4
D	2
E	5
F	6

TEST3

FILE A	FILE B
A 1	B 4
C 2	D 5
F 3	E 6
F 7	

A	1
B	4
C	2
D	5
E	6
F	3

TEST4

FILE A	FILE B
B 1	A 3
C 2	D 4

A	3
B	1
C	2
D	4

Cosa abbiamo imparato

In questa lezione abbiamo visto:

- come si verifica la correttezza di un programma, attraverso la messa a punto di un insieme di prove;
- la differenza tra la fine dei dati in un file in fase di lettura e la fine dello spazio di memoria disponibile in fase di registrazione.

minativi e numeri progressivi a partire da 1 per i numeri telefonici. In questo modo verifichiamo anche che nel caso di nominativo doppio non solo questo venga prelevato una sola volta, ma sia scelto quello proveniente dal file B.
Riportiamo di seguito i dati di prova di ciascun test e i risultati attesi in corrispondenza:

TEST1

FILE A	FILE B
A 1	B 4
C 2	D 5
Z 3	

A	1
B	4
C	2
D	5
Z	3

TEST2

FILE A	FILE B
B 1	A 3
D 2	C 4
	E 5
	F 6

A	3
B	1
C	4
D	2
E	5
F	6

TEST3

FILE A	FILE B
A 1	B 4
C 2	D 5
F 3	E 6
F 7	

A	1
B	4
C	2
D	5
E	6
F	3

TEST4

FILE A	FILE B
B 1	A 3
C 2	D 4

A	3
B	1
C	2
D	4

Cosa abbiamo imparato

In questa lezione abbiamo visto:

- come si verifica la correttezza di un programma, attraverso la messa a punto di un insieme di prove;
- la differenza tra la fine dei dati in un file in fase di lettura e la fine dello spazio di memoria disponibile in fase di registrazione.

IL LINGUAGGIO LISP (II)

Ulteriori considerazioni e alcuni esempi per meglio evidenziare le caratteristiche peculiari di questo linguaggio.

Abbiamo visto precedentemente l'esempio dell'applicazione di una semplice funzione LISP ai propri argomenti. In queste pagine esamineremo alcune definizioni leggermente più complesse prima di dedicarci a un esempio che, pur non essendo particolarmente sofisticato, mette bene in evidenza alcune delle caratteristiche peculiari del LISP.
Per definire funzioni che eseguono operazioni diverse al variare della natura dei propri argomenti è necessario, come in tutti i linguaggi, disporre di un meccanismo di selezione (per esempio in Pascal si usa il costrutto if... then... else).
In LISP esiste un costrutto analogo ma più generale, COND. La selezione si ottiene facendo seguire al simbolo COND, nella definizione di una funzione, una successione di coppie, ciascuna delle quali rappresenta una delle alternative. Il primo elemento della coppia è il predicato che, se risulta vero, causa l'esecuzione del secondo elemento della coppia. Per esempio, si può migliorare la definizione della funzione che accede al secondo elemento di una lista, esaminata nel precedente articolo, aggiungendo una verifica che evita l'applicazione dei selettori CAR e CDR alla lista vuota. La nuova definizione è perciò

```
(SETQ  secondo  '(LAMBDA  (lista)
                    (COND  ((NULL   lista)  errore))
                           (True   (CAR (CDR lista))))
```

Quando secondo viene applicata, verranno considerate le alternative in ordine. Il predicato NULL è applicato all'argomento, e se risulta vero, cioè la lista è vuota, viene segnalato un errore. Altrimenti viene esaminata la seconda alternativa e quindi valutato il predicato True. Per definizione questo è sempre vero e quindi se la lista non è vuota si accederà senz'altro al suo secondo elemento con la valutazione di (CAR (CDR lista)).
Abbiamo visto come la definizione di una funzione non sia che una lista come tutte le altre: non esiste infatti alcuna differenza, se vengono considerate come strutture dati, tra la lista (come quando fuori piove) e la lista (LAMBDA (x) (ATOM (CAR x))). Dare un nome a una funzione equivale quindi a dare un nome a una struttura dati qualunque: questo si ottiene come abbiamo visto con il comando SETQ. Dal momento dell'esecuzione di questo comando in poi per utilizzare la funzione è sufficiente usarne il nome. Questo vale anche se si vuole utilizzare la funzione all'interno della definizione di un'altra funzione. Per esempio si può definire la funzione che restituisce il terzo elemento di una lista, terzo, utilizzando la funzione secondo. Si avrà così

```
(SETQ terzo '(LAMBDA (x)  (secondo (CDR x)))).
(terzo '(Piccolo   mondo  antico))
```

ora, restituirà antico.
In questo modo si possono definire le funzioni di più alto livello utilizzando nomi di funzioni di livello più basso che non sono ancora state definite. A questo modo estremamente libero di trattare le funzioni si contrappone quello, molto più rigido, dei linguaggi tradizionali: in Pascal, per esempio, prima di utilizzare una qualsiasi funzione bisogna dichiararne il nome e il tipo.
È quindi possibile, all'interno della lista che descrive una funzione LISP, inserire il nome stesso della funzione che si sta definendo.
Le funzioni che sono definite utilizzando il proprio nome vengono usualmente dette *ricorsive*.
Per quanto il concetto così espresso possa sembrare confuso o di scarsa utilità vedremo nei prossimi esempi come in verità la ricorsione corrisponda a un modo semplice e naturale di pensare alla risoluzione di certi problemi.
Come esempio consideriamo il compito di misurare la lunghezza di una lista, intesa come numero degli elementi costituenti. Vediamo prima, a livello discorsivo, l'algoritmo più immediato. Il caso più semplice è che la lista sia vuota.
Quindi:

1) se la lista è vuota, lunghezza = 0.

In tutti gli altri casi si può affermare che la lunghezza di una lista è superiore di uno alla lunghezza della lista privata del suo primo elemento. Quindi:

2) lista' = lista senza il primo elemento
lunghezza di lista = 1 + lunghezza di lista'

Tutto ciò trova una naturale espressione in LISP. Infatti, usando il costrutto COND e la ricorsione, come suggerito dalla precedente descrizione informale, si ottiene la seguente, compatta, definizione.

```
(SETQ lunghezza (LAMBDA (lista)
                    (COND ((NULL lista)  0)
                          (True (INC  (lunghezza (CDR lista)))))))
```

Fino ad ora, non abbiamo parlato di dati numerici poiché il campo di applicazione significativo per il LISP non è il calcolo numerico. In realtà tutti i sistemi LISP comprendono un insieme di primitive, a volte anche sofisticate, per la manipolazione di numeri. Per i nostri scopi, è sufficiente disporre dei numeri interi. INC è appunto la funzione di sistema che incrementa di uno il proprio argomento. Come si può notare, il nome della funzione che si sta definendo viene usato come qualunque altro nel corpo della stessa. Per provare la nostra funzione sottoponiamo all'interprete le espressioni:

```
(lunghezza ())
```

ottenendo 0 e

```
(lunghezza  '(questi sono quattro elementi)
```

che darà come risultato il numero 4.
Alla funzione lunghezza viene passata, come argomento nel secondo caso, la lista iniziale che viene a mano a mano, ad ogni successiva applicazione, privata di un elemento. Questo avviene perché la chiamata è combinata con l'applicazione del selettore CDR, nell'ultima linea della definizione. La prima delle due alternative serve proprio, oltre che per calcolare il corretto valore per le liste vuote, ad arrestare questo processo di semplificazione dell'argomento quando non ci sono più elementi nella lista.
Veniamo ora ad un esempio un po' più complicato e interessante. Vogliamo costruirci un piccolo dizionario italiano-inglese in cui collocare, via via, le parole di cui apprendiamo il significato. La struttura più semplice per un tale scopo è una lista di voci. Ogni voce sarà formata da una parola italiana e dalla sua traduzione inglese. Una voce sarà quindi anch'essa una lista. Un elementare dizionario iniziale, legato al nome DIZIONARIO sarà per esempio:

```
(SETQ DIZIONARIO '((uno  one)  (due  two)  (tre  three) ...))
```

Definiamo la funzione che ritrova la traduzione di una parola italiana data in un dizionario.

```
(SETQ traduzione-inglese
        (LAMBDA (voc diz)
          (COND ((NULL diz)  (PRINT "vocabolo ignoto"))
                ((EQ (italiano (CAR diz))  voc)
                          (inglese (CAR diz)))
                (True (traduzione-inglese voc (CDR diz))))))
```

Esaminiamo ora questa definizione. La funzione accetta come argomenti un vocabolo e un dizionario. Se il dizionario non contiene voci, il vocabolo non ha traduzione nota.
Altrimenti se il vocabolo è la parola italiana della prima voce non ci resta che ritornare come valore la controparte inglese della voce. Se questo non è il caso, si procede con la ricerca applicando il medesimo procedimento al dizionario mutilato della sua prima voce. La prima alternativa serve così anche per segnalare che tutte le voci sono state esaminate senza che la parola sia stata trovata.
Ci servono ora le funzioni per accedere alla parola italiana e alla sua traduzione all'interno di una voce.

```
(SETQ italiano (LAMBDA (voce) (CAR voce)))
(SETQ inglese (LAMBDA (voce) (secondo voce)))
```

Per chiarire il funzionamento di queste procedure, esaminiamo per esempio il comportamento del sistema durante una traduzione.

```
(traduzione-inglese 'due DIZIONARIO)
```

La funzione di traduzione, dopo aver verificato che il dizionario non è vuoto [terza riga della definizione], passa a considerare la prima voce: (uno one), cioè il CAR della lista DIZIONARIO.
Da questa viene estratto il vocabolo italiano mediante una applicazione del selettore italiano, che a sua volta si serve di CAR [quarta riga]. EQ ritorna in questo caso NUL e quindi viene considerata la prossima alternativa che prevede l'applicazione della stessa funzione traduzione-inglese alle rimanenti voci del dizionario, in questo caso ((due two) (tre three) ...) [quinta riga]. Quindi si riprendono a considerare le alternative dall'inizio. Questa volta la seconda alternativa ha successo. EQ verifica infatti l'uguaglianza del vocabolo due e della parte italiana della voce (due two) che si trova ora in prima posizione.
Si ricorre quindi al selettore inglese per ottenere la traduzione da ritornare come valore all'utente.

È interessante notare come il nostro dizionario sia perfettamente simmetrico. È semplice quindi scrivere la funzione che utilizza lo stesso dizionario e lo stesso schema della funzione traduzione-inglese per tradurre dall'inglese all'italiano. Avremo:

```
(SETQ traduzione-italiana
      (LAMBDA (voc diz)
          (COND ((NULL diz)  (PRINT "unknown word"))
                ((EQ (inglese (CAR diz))  voc)
                    (italiano (CAR diz)))
                (True (traduzione-italiana voc (CDR diz))))))
```

Il funzionamento di questa procedura nonché la sua struttura sono molto simili a quelli di traduzione-inglese. In seguito vedremo come questa affinità darà luogo a una serie di interessanti considerazioni.
Come possiamo vedere le parole in un dizionario "elettronico" non devono essere ricercate manualmente, ma soprattutto esiste la possibilità di costruire programmi che dinamicamente agiscono su di esso realizzando utili trasformazioni e arricchimenti.
La prima esigenza dinamica è sicuramente quella di integrare il dizionario con nuove voci.

```
(SETQ  nuova-voce
      (LAMBDA (voce diz)
          (CONS  voce  diz)))
```

La funzione nuova-voce è molto semplice; si limita a restituire un nuovo dizionario, uguale a quello passato come argomento, a meno del fatto che la nuova voce compare nella prima posizione.
Per aggiungere effettivamente una voce a DIZIONARIO, la si dovrà chiamare come segue:

```
(SETQ  DIZIONARIO  (nuova-voce  '(dieci  ten)  DIZIONARIO))
```

dopo di che il DIZIONARIO avrà il seguente aspetto:

((dieci ten) (uno one) ...).

Può capitare che una voce non sia più di uso corrente; sarà quindi necessario rimuoverla. Per fare questo definiamo una procedura che accetta come argomenti un dizionario e un vocabolo (indifferentemente inglese o italiano) e ritorna un nuovo dizionario dal quale è stata rimossa la prima voce che contiene il vocabolo.

```
(SETQ rimozione
      (LAMBDA (voc  diz)
          (COND ( (NULL diz)  diz)
                ( (EQ voc (italiano  (CAR diz))
                      (CDR  diz) )
                ( (EQ voc (inglese (CAR diz))
                      (CDR  diz) )
                (True (CONS (CAR diz)
                            (rimozione  voc  (CDR diz))) )))
```

Facciamo ora un breve commento.
Se il vocabolo è la parte italiana della prima voce del dizionario [seconda riga], viene ritornato il resto dello stesso.
La medesima operazione avviene nel caso si tratti della parte inglese [quarta riga].
Altrimenti la voce correntemente in prima posizione deve comparire nel dizionario risultante, quindi viene aggiunta al dizionario ottenuto tentando di eseguire la rimozione sulle altre voci [settima riga].
In pratica il dizionario viene così "smontato" fino alla voce da rimuovere; se la voce non è presente, viene ricostruito dall'inizio.
Può accadere che una parola italiana abbia più traduzioni inglesi nello stesso dizionario, o viceversa.
Estendiamo perciò la funzione di rimozione in modo che agisca su tutte le voci in cui compare la parola indicata. È sufficiente per questo far proseguire l'operazione di "smontaggio" estendendola su tutto il dizionario, senza fermarsi dopo la prima occorrenza.

```
(SETQ rimozione
      (LAMBDA (voc  diz)
          (COND ( (NULL diz)  diz)
                ( (EQ voc (italiano (CAR diz))
                      (rimozione (CDR  diz) ))
                ( (EQ voc (inglese (CAR diz))
                      (rimozione (CDR  diz) ))
                  (True (CONS (CAR diz)
                              (rimozione voc (CDR diz))) )))
```

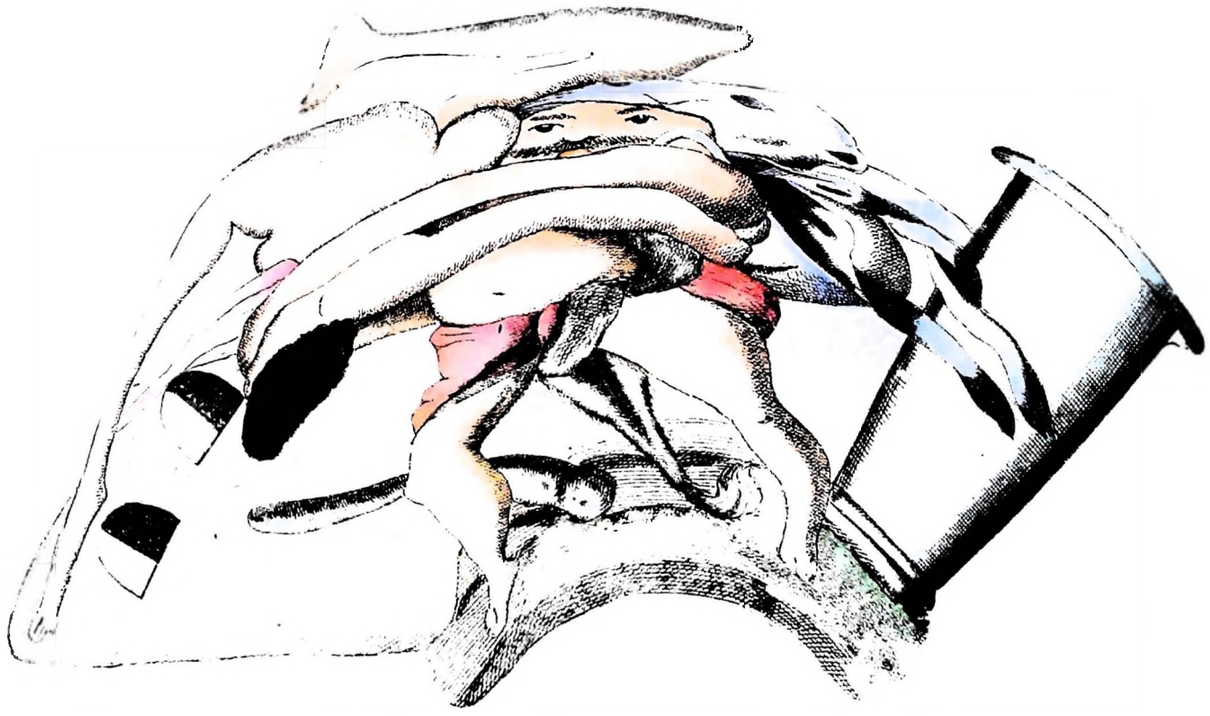
Anche qui, per ottenere una effettiva modifica del dizionario, si eseguirà ad esempio la chiamata:

```
(SETQ DIZIONARIO (rimozione 'due DIZIONARIO) )
```

Nel prossimo articolo concluderemo la panoramica sul LISP perfezionando l'esempio del dizionario con l'introduzione di alcune estensioni al programma che ci permetteranno di illustrare i metodi e gli approcci che possono essere usati in ambiente LISP per sviluppare un sistema complesso.

ANCORA SULLE ANAMORFOSI

Le trasformazioni anamorfiche possono essere utilizzate anche per scopi scientifici e non solo per il loro aspetto fantastico.



In precedenza abbiamo presentato il tema delle anamorfosi, sottolineando l'aspetto fantastico dei risultati ottenibili, ma le trasformazioni anamorfiche possono essere utilizzate per scopi scientifici più seri. In effetti la fisica matematica si avvale di trasformazioni di tipo anamorfico per aiutare lo studioso ad ambientare un determinato problema fisico in un contesto in cui risulta trattabile.

Immaginate per esempio di dover studiare il moto di una particella elementare all'interno di un campo magnetico molto complicato, generato da diverse sorgenti. In generale un problema del genere è difficilissimo, se non impossibile, da risolvere. Vengono in questo caso in aiuto le trasformazioni cosiddette "conformi", che altro non sono che una generalizzazione delle trasformazioni anamorfiche, le quali permettono di trasformare il campo così complicato, in un campo più semplice, nel quale le equazioni del moto si possono risolvere con tecniche note.

Fermandoci a queste brevi osservazioni, torniamo al nostro problema grafico e vediamo come si possono realizzare le anamorfosi con un semplice calcolatore. Gli esempi che mostreremo sono elementari e consistono semplicemente nell'illustrare come viene trasformata una griglia ortogonale da una anamorfosi conica o cilindrica (figura a pag. 587).

Supponiamo di disegnare una griglia ortogonale che si estende dalle coordinate 90,24 alle coordinate 150,40, con un passo, secondo l'asse X, di 3 unità e secondo l'asse Y di 8 unità. Abbiamo scelto questi valori perché si possono visualizzare facilmente sullo schermo dell' M10.

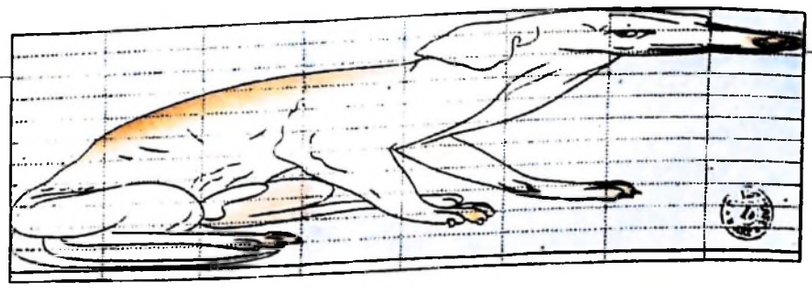
L'anamorfosi cilindrica consiste nel trasformare tutti i vertici della nostra griglia come se stirassimo circolarmente un fo-

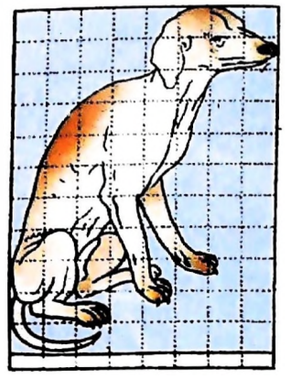
Ancora alcune trasformazioni anamorfiche (XVIII secolo) che danno immagini fantastiche: nella pagina accanto, anamorfosi cilindrica di due lottatori e, qui sotto, un cavaliere, anch'esso "visto" mediante un'anamorfosi cilindrica. (Le due incisioni, probabilmente tedesche, sono conservate al Philadelphia Museum, USA).

glio di gomma su cui essa è disegnata. Dal punto di vista della geometria analitica, è dimostrabile che un punto della griglia, che ha inizialmente coordinate X0,Y0, dopo la trasformazione, sarà ancora un vertice della griglia; le sue coordinate saranno esprimibili in funzione del nuovo sistema di riferimento, composto dai cerchi concentrici (corrispondenti alle rette parallele all'asse X) e dalle rette uscenti dall'origine e ortogonali a tali cerchi (corrispondenti alle vecchie coordinate X, ovvero alle rette parallele all'asse Y).
Esprimiamo tutto ciò con l'aiuto delle coordinate polari, che chiamiamo RO e THETA. Nella figura trasformata RO coincide con il valore che aveva Y nella figura originaria, mentre il valore di THETA coincide con quello che aveva X a meno di un intero angolo giro.
Ora per trovare le coordinate cartesiane della figura trasformata, basta ricordare la relazione:

```
X = RO * COS(THETA)
Y = RO * SIN(THETA)
```

Nel programma si eseguono proprio queste trasformazioni, dopo aver però centrato la griglia ortogonale nell'origine del sistema di riferimento. In realtà per completare il discorso occorre tenere presente la periodicità delle coordinate angolari: ogni volta che compiamo un intero giro di 360 gradi non è più possibile distinguere un punto con la stessa coordinata RO. Occorre perciò limitare l'intervallo di variazione al massimo in un angolo giro.
Nel programma abbiamo limitato la variazione degli angoli in un intervallo di 180 gradi, che abbiamo espresso con il va-





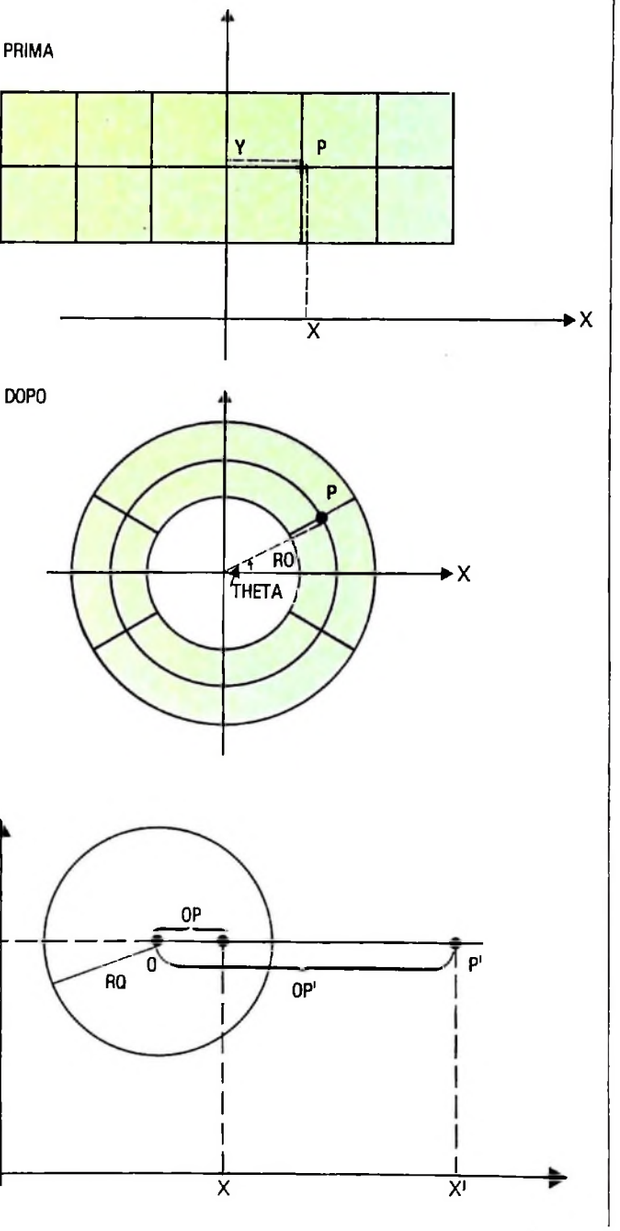
Le trasformazioni anamorfiche di questa pagina rappresentano i primi tentativi di 'deformazione' delle immagini; il cane viene allungato secondo precise proporzioni geometriche, come pure la 'deformazione' verticale del ritratto di Carlo I (Londra, Collezione Anthony d'Ottay).

lore 3.1415 approssimante pigreca. Nell'istruzione 212 si effettua questa limitazione, imponendo che l'intervallo da 90 a 150 unità (pari a 60 unità) sia proporzionale all'angolo piatto. Successivamente si trasla la griglia in modo che il centro coincida con lo zero delle X (istruzione 221). Dopo la trasformazione bisogna ricordarsi di rimettere le cose a posto, poiché il nostro schermo è limitato a valori compresi tra 0 e 239 e tra 0 e 63. Per evitare che i valori di Y escano da questo intervallo, si è moltiplicata la Y per un fattore di scala (S1), mentre si riporta l'origine a posto traslando la X di +120.
Provate il programma premendo il tasto C dopo la comparsa del menù e la griglia verrà ridisegnata secondo lo schema riportato in figura. Ripetendo la trasformazione più volte, si otterranno figure sempre più deformate.
Diverso è il caso dell'anamorfosi conica, chiamata anche inversione circolare.
Infatti ripetendo due volte la trasformazione, la griglia viene ricomposta, a meno di errori dovuti all'approssimazione dell'elaboratore o all'eventuale assenza di punti singolari (in particolare lo zero). L'inversione circolare consiste in una trasformazione che impone una specifica relazione tra le distanze dall'origine di due punti, uno interno e l'altro esterno, ad un cerchio di raggio R ma allineati con l'origine. La relazione è la seguente:

$$R^2 = OP * OP'$$

Dove OP è il segmento che congiunge il punto originario con il centro del cerchio, e OP' quello che congiunge il punto trasformato. In pratica, come già detto, l'inversione consiste in un ideale rovesciamento del semipiano interno al cerchio e di quello esterno, in modo da scambiare i ruoli rispettivi.
A tale scopo il calcolo da effettuare consiste nell'esprimere le coordinate del punto trasformato in funzione di quelle del punto originario, rispettando la formula sopra scritta. In

```
1 CLS
2 DIM X(15),Y(15),U(15),V(15): A$=""
3 S1=.65 'FATTORE DI SCALA PER CILINDRICA
4 S2=1 'FATTORE DI SCALA PER INVERSIONE
10 GOSUB 1000 'INIZIALIZZA DATI
20 GOSUB 1200 'DISEGNA FIGURA
30 PRINT "ANAMORFOSI: C>ILINDRICA,I>NVERSIONE      CONICA,F>INE"
35 A$=INKEY$
40 IF A$="" THEN 35 ELSE IF A$="F" THEN 999 ELSE IF A$="C" THEN 200 ELSE IF
A$="I" THEN 500 ELSE GOTO 30
200 'CILINDRICA
212 ROT = 3.1415/60
220 FOR I=1 TO 15
221 X(I)=X(I)-270
222 X=X(I)*ROT: Y=Y(I)
230 U(I)=Y*COS(X)
240 V(I)=Y*SIN(X)
250 X(I)=U(I)+120
260 Y(I)=(V(I)+54)*S1
270 NEXT I
290 GOSUB 1200
300 GOTO 30
500 'INVERSIONE CONICA
510 RQ=15^2 'RAGGIO DEL CERCHIO DI INVERSIONE
520 FOR I= 1 TO 15
525 X(I)=X(I)-120:Y(I)=Y(I)-32
530 VET= (X(I)^2+Y(I)^2)^.5
532 IF VET=0 THEN VET=.0001
540 RO=RQ/VET
550 U(I)=RO*X(I)/VET
560 V(I)=RO*Y(I)/VET
570 X(I)=U(I)*S2+120
580 Y(I)=V(I)*S2+32
590 NEXT I
600 GOSUB 1200
610 GOTO 30
999 END
1000 'INIZIALIZZA DATI
1010 C=90
1020 FOR I=1 TO 5
1030 X(I)=C+15*(I-1)
1040 X(I+5)=X(I)
1050 X(I+10)=X(I)
1060 NEXT I
1070 FOR I=0 TO 2
1080 C=24 + 8*I
1090 FOR J=1 TO 5
1100 Y(5*I+J)=C
1110 NEXT J
1120 NEXT I
1130 RETURN
1200 'DISEGNA FIGURA
1201 CLS
1210 FOR I=0TO 2
1220 FOR J=1 TO 4
1230 IN=5*I+J
1240 FI=IN+1
1250 LINE (X(IN),Y(IN))-(X(FI),Y(FI))
1260 NEXT J
1270 NEXT I
1280 FOR I= 1 TO 5
1290 LINE (X(I),Y(I))-(X(I+5),Y(I+5))
1300 LINE (X(I+5),Y(I+5))-(X(I+10),Y(I+10))
1310 NEXT I
1320 RETURN
```



Il programma riportato qui sopra e commentato nel testo permette di effettuare l'anamorfosi conica e l'anamorfosi cilindrica (o inversione circolare) di una griglia ortogonale. Nei disegni a destra: in alto, la griglia; al centro, la trasformazione cilindrica; in basso, l'inversione circolare.

Ritratto anamorfico di Edoardo VI, 1546, conservato a Londra, alla National Portrait Gallery.

pratica le formule che si applicano si basano ancora sulle proprietà delle coordinate polari.
Il punto trasformato in coordinate cartesiane è:

$$x' = RO' * COS(THETA)$$
$$Y' = RO' * SIN(THETA)$$

Nell'inversione circolare ciò che cambia è solo la componente RO', la quale non è altro che il segmento OP'; per quanto riguarda la componente angolare THETA, basta calcolarsi il seno e il coseno, che risultano identici a quelli del punto originario, dato appunto che l'inversione non modifica gli angoli, ma opera solo lungo la direzione radiale.
Le relazioni che legano seno e coseno sono perciò:

$$SIN(THETA) = X/OP$$
$$COS(THETA) = Y/OP$$

Dove X,Y sono le coordinate originarie del punto e $OP = (X^2 + Y^2)^{0.5}$
Nel programma applichiamo queste formule nelle istruzioni dalla 530 alla 560. Anche in questo programma occorre ricordarsi di traslare la griglia nell'origine del sistema di riferimento, e dopo la trasformazione risistemare le cose. Ciò viene fatto prima nella istruzione 525, mentre la traslazione contraria è effettuata nelle istruzioni 570 e 580. Anche in questo caso si può usare un fattore di scala per ingrandire o ridurre il disegno (istruzione 4; SC=1 lascia le dimensioni invariate). Chi prova il programma può quindi provare a modificare i valori di S1 per le anamorfosi cilindriche e di S2 per quelle coniche. Infine, nell'istruzione 510 viene fissato il valore del raggio del cerchio di inversione (RQ) immediatamente elevato al quadrato.
Chi vuole provare differenti valori può modificare questa istruzione. Tenete comunque conto che è molto facile che i valori che si ottengono escano dal campo rappresentabile sullo schermo dell' M10.
Infine un'ultima nota: nell'istruzione 542 si modifica il valore di VET quando è pari a 0 per evitare errori di calcolo della divisione per zero, un trucco che approssima il risultato ma permette di trattare il caso in cui un valore da invertire coincida proprio con l'origine del cerchio di inversione.
Come si è detto, provando più volte l'inversione si ottengono alternativamente due configurazioni: la trasformazione è invertibile (a meno della singolarità dell'origine !). Inoltre tutti i segmenti di retta che passano per l'origine vengono trasformati in segmenti di retta, tutti i segmenti che non passano per l'origine vengono trasformati in archi di cerchio.

Anche in leasing con Olivetti Leasing

LA FAMIGLIA DEI PERSONAL COMPUTER OLIVETTI

FRIENDLY & COMPATIBLE

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

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

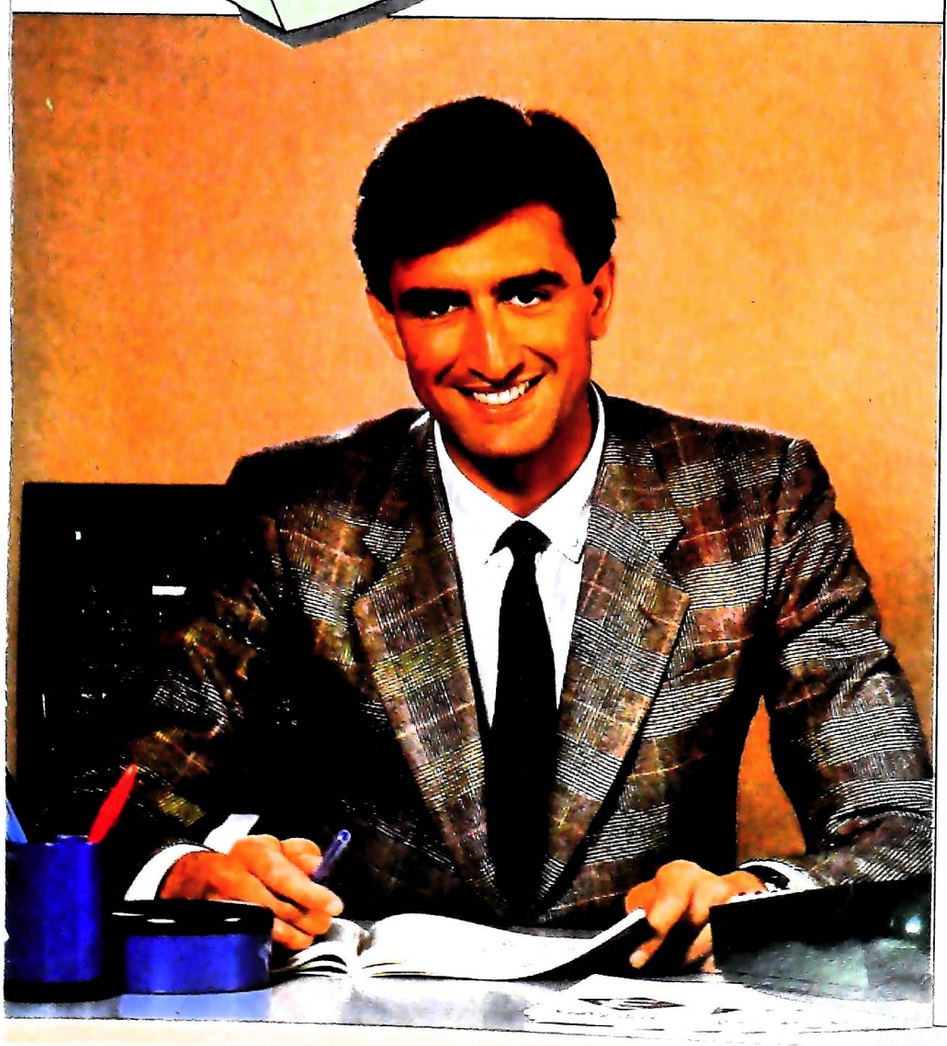
olivetti

Per maggiori informazioni inviare il coupon a: Olivetti, Divisione Personal Computer, Via Meravigli 12, 20123 Milano.

NOME .....
INDIRIZZO .....
CITTÀ .....
TELEFONO .....

UN NUOVO MODO DI USARE LA BANCA.

Consulenza



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

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

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

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

Veniteci a trovare, ci conosceremo meglio.

BANCO DI ROMA
CONOSCIAMOCI MEGLIO.