VIC 20

COLOUR COMPUTER

INTRODUZIONE AL BASIC. PARTE 2

COD. 2502

UN COMPLETO CORSO IN ITALIANO DI AUTOAPPRENDIMENTO DEL BASIC CON IL VIC 20

by Andrew Colin

commodore COMPUTER

INDICE

Questo corso è la Parte 2 di una serie intesa ad aiutare a conoscere ogni aspetto di programmazione del computer Commodore VIC 20. Esso parte dai principi illustrati nella Parte 1 per fornire tutta la conoscenza necessaria per scrivere programmi BASIC ben strutturati sul computer VIC. Il corso è costituito da due parti:

1. Un testo autodidattico diviso in 10 lezioni o "Unità", ciascuna delle quali tratta un importante aspetto della programmazione.

2. Due cassette di nastro che contengono una raccolta di programmi VIC, che aiuteranno a studiare le unità.

INDICE

INTRODUZIONE

Benvenuti alla secondo parte del corso VIC BASIC. La struttura del libro e i nastri saranno già noti dato che il corso è una continuazione diretta del VIC BASIC Parte 1. Le Unità sono state numerate consecutivamente dal 16 in avanti per sottolineare questa continuità.
Innanzitutto si troveranno le Unità abbastanza simili a quelle già studiate ma procedendo nel manuale diventeranno sempre più lunghe e probabilmente un pochino più difficili. Ci si occuperà delle applicazioni avanzate del BASIC, comprese le matrici, la manipolazione di stringhe di caratteri, di giochi animati e dell'uso di cassette di nastro come memoria di supporto. Questo è il momento giusto per pensare ai metodi di studio e revisionarli, se il caso. Ricordarsi le regole d'oro:

1) Leggere ogni unità completamente, dall'inizio alla fine prima di iniziare a studiarla in dettaglio.
2) Svolgere *tutti* i problemi pratici che sono stati scelti per illustrare i punti essenziali del BASIC.
3) Non passare alla successiva Unità fino a che non si è completamente sicuri della prima. Se per caso ci si blocca, cercare di ritornare indietro di un paio di Unità e ripetere il lavoro.
4) Non correre. Le ultime Unità nel corso richiederanno quattro o cinque giorni ciascuna per poter essere assorbite a fondo.
   Buona fortuna!

UNITA': 16

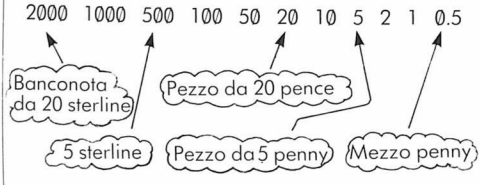
LE FUNZIONI DATA, READ E RESTORE

Questa Unità introduce un importante e utile gruppo di comandi che usano le parole chiave DATA, READ e RESTORE.
I programmi spesso devono fare riferimento a liste di parole o di frasi o serie di numeri che non seguono qualsiasi profilo aritmetico fisso. Per esempio, un programma che calcola e visualizza il calendario per un dato anno deve conoscere i nomi dei giorni della settimana e dei mesi dell'anno. Esso ha inoltre bisogno di una sequenza del numero di giorni di ciascun mese e cioè:

31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31

ANALISI DELLE MONETE

Un'altra sequenza comune è la serie di valori delle monete e delle banconote, ad esempio in valuta britannica. Per evitare i decimali, che possono dar luogo a problemi, ci si limiterà al pence:



Le persone incaricate di pagare i salari nelle grandi organizzazioni, devono pianificare le operazioni accuratamente ogni settimana. Uno speciale problema sta nell'assicurarsi che ci sia la corretta quantità di cambio per comporre ciascuna busta paga esattamente, usando il minor numero di banconote e di monete.
Si tratta di un utile processo detto "analisi delle monete" che parte da una somma di denaro e la suddivide nel più piccolo numero possibile di banconote e di monete. Per esempio:

```
158,37 =  7 da 2000P (e cioè 7 pezzi
                da 20 sterline = 2000 pence)
          1 da 1000P
          1 da 500P
          3 da 100P
          0 da 50P
          1 da 20P
          1 da 10P
          1 da 5P
          1 da 2P
          0 da 1P
          0 da 0.5P (e cioè 1/2 penny)
```

Questo processo, usato su tutti i diversi salari da pagare, aiuta i cassieri addetti a decidere quante banconote e quante monete di ciascun tipo richiedere alla banca.
Si studierà ora un programma per questo tipo di analisi, che potrebbe risultare tanto semplice da non richiedere uno schema di flusso.
Si inizia chiedendo all'utente di fornire una cifra per il salario da suddividere. La cifra iniziale, in pence è memorizzata in W:

```
INPUT "PAGA IN PENCE"; W
```

Successivamente si estrae il numero di banconote da 20 sterline necessarie e lo si inserisce nella variabile. Il numero corretto è il risultato quando W è diviso per 2000, ignorando qualsiasi resto o parte frazionaria. Un comando appropriato è:

```
T = INT(W/2000)
```

Notare che a meno che W non sia una somma di denaro che può essere pagata esattamente in banconote da 20 sterline (ad esempio 80 st.), W/2000 sarà un numero con una parte decimale, ad esempio 7.9185. INT e le parentesi assicurano che venga usata soltanto la parte numerica intera di questa espressione (ad esempio 7) e il resto venga scartato.
Successivamente si visualizzerà il numero di banconote da 20 sterline:

```
PRINT T; "DA 2000P"
```

È così contata una parte sostanziale della busta paga dato che la maggior parte di essa in effetti potrebbe essere pagata interamente in banconote da 20 st. ciascuna. Per continuare il processo di analisi, dobbiamo togliere questo importo dall'importo totale, lasciando solo il resto, che deve essere meno di 20 sterline ossia 2000 pence. L'importo è 2000★T, cosicchè occorre inserire:

```
W = W - 2000★T
```

(nell'esempio W sarebbe ora 15837 – 2000★7 = 1837).
Per la fase successiva, si calcola e si visualizza il numero di banconote da 10 sterline e si diminuisce W di conseguenza. È possibile usare di nuovo la variabile T dato che il suo valore originale (n. di banconote da 20 st.) non è più interessante:

```
T = INT(W/1000)
PRINT T; "DA 1000P"
W = W - 1000★T
```

Le fasi seguenti fino al mezzo penny finale, sono tutte molto simili. Si ottiene:

```
 10 INPUT "SALARI IN PENCE"; W
 20 T = INT(W/2000)                    da 20 st.
 30 PRINT T; "DA 2000P."
 40 W = W -2000★T
 50 T = INT(W/1000)
 60 PRINT T; "DA 1000P."               da 10 st.
 70 W = W - 1000★T
...
...
140 T = INT(W/50)
150 PRINT T;"DA 50P."                  da 50 penny
160 W = W -50★T
...
...
320 T = INT(W/0.5)
330 PRINT T;"DA 0.5P."                 da 1/2 penny
340 W = W - 0.5★T
350 STOP
```

Questo programma appare estremamente ripetitivo. Se si potessero accogliere in qualche modo i successivi valori delle banconote e delle monete in una variabile — ad esempio V, l'intero programma di 35 righe, potrebbe essere condensato in una singola iterazione con tre comandi:

```
T = INT(W/V)
PRINT T; "DA"; V; "P."
W = W – V★T
```

Questa iterazione verrebbe eseguita 11 volte; una volta per V=2000, una per V=1000 e così via fino a V=0.5.

USO DI UNA ITERAZIONE

Sarebbe comodo e bello usare un comando FOR, ma ciò non risolverebbe il problema in quanto i valori di V non seguono un profilo determinato. Occorre trovare un altro modo per inserire i valori delle banconote e delle monete in V, quantunque si possa sempre usare un FOR per eseguire l'iterazione 11 volte.
Si consideri una soluzione "stupida". Si sa che un modo per ottenere i numeri in un programma è di fare in modo che l'utente li batta sulla tastiera. Si potrebbe sempre inserire un comando INPUT V nell'iterazione e costringere l'utente a fornire la sequenza di numeri 2000 1000 500 100 . . . 0.5. Il programma apparirebbe pertanto così:

```
10 INPUT "PAGA IN PENCE"; W
20 FOR J=1 TO 11
30 INPUT "SUCCESSIVO VALORE"; V
40 T=INT(W/V)
50 PRINT T; "DA"; V; "P."
60 W=W–T★V
70 NEXT J
80 STOP
```

Impostare questo programma e provare ad eseguirlo. Si otterrà una visualizzazione come questa:

```
PAGA IN PENCE? 9472
SUCCESSIVO VALORE? 2000
4 DA 2000 P.
SUCCESSIVO VALORE? 1000
1 DA 1000 P.
SUCCESSIVO VALORE? 500
0 DA 500 P.
. . .
```

LE ISTRUZIONI DATA E READ

Naturalmente ci sono molti motivi che rendono questo programma poco pratico... È un lavoro molto duro per l'utente e se si verifica un singolo errore nella battitura delle sequenze 2000, 1000, 500, 1000... l'intera analisi viene distrutta e deve essere ripresa da capo. I progettisti del BASIC hanno superato questo problema in un modo ingegnoso ed elegante. Si supponga che il VIC possa battere propri numeri man mano che procede. Si potrebbe usare una tastiera "fantasma" in modo che i numeri non compaiano sullo schermo, e la battitura verrebbe fatta istantaneamente consentendo alla macchina di operare alla sua massima velocità. Cosa dire dei numeri da battere sulla tastiera fantasma? Questi verrebbero immessi in anticipo, sotto forma di istruzioni DATA.
Cambiare ora la riga 30 del programma nel VIC e aggiungere alcune righe DATA per ottenere

```
10 INPUT "PAGA IN PENCE"; W
20 FOR J= 1 TO 11
30 READ V                    cambiata questa riga
40 T=INT(W/V)
50 PRINT T; "DA"; V; "P."
60 W=W–V★T
70 NEXT J
80 STOP
1000 DATA 2000               da qui nuove righe
1010 DATA 1000
1020 DATA 500
1030 DATA 100
1040 DATA 50
1050 DATA 20
1060 DATA 10
1070 DATA 5
1080 DATA 2
1090 DATA 1
1100 DATA 0.5
```

Se si esegue questa nuova versione del programma, si ha un'analisi del denaro per qualsiasi cifra immessa. Occorre soltanto indicare la cifra da analizzare. Il programma ha 8 righe di codice e quindi un'istruzione DATA per ciascun valore della banconota o della moneta. Questo è chiaramente un miglioramento rispetto alla versione originale che aveva 35 comandi.
Esaminiamo ora il programma ulteriormente. Il comando READ nella riga 30 è molto simile a INPUT. La parola chiave READ può essere seguita dai nomi di una o più delle variabili, che possono essere numeri o stringhe. La differenza importante è che il comando ottiene le sue informazioni da un'istruzione DATA incorporata nel programma anzichè dall'utente. Se lo si desidera, è possibile immaginare un diavoletto che vive all'interno del VIC che ha una propria tastiera privata. Ogni volta che il VIC obbedisce ad un comando READ, il demonietto trova un'istruzione DATA e rapidamente batte i suoi contenuti su questa tastiera. Egli ricorda quali istruzioni ha usato e le elabora nell'ordine dei rispettivi numeri di label. Pertanto, quando la macchina esegue il comando READ (alla 30) per la prima volta, il diavoletto trova la prima istruzione DATA e batte il numero che essa contiene: 2000. Questo valore viene assegnato a V. La seconda volta il valore usato è 1000 e così via.
Vediamo ora come il comando READ può manipolare stringhe nonchè numeri. Si supponga di voler che il programma di analisi del denaro usi nomi descrittivi per le banconote e le monete ad esempio

1 PEZZO da 5 PENNY

invece dei simboli

1 DA 5 P

I nomi che ci servono possono essere inclusi nelle istruzioni DATA unitamente ai valori stessi. Il comando READ imposterà ora due variabili: il valore V e il corrispondente nome N$. Il programma modificato con le modifiche nelle righe 30, 50 e le istruzioni DATA è:

```
 10 INPUT "SALARI IN PENCE"; W
 20 FOR J = 1 TO 11
 30 READ V, N$
 40 T = INT(W/V)
 50 PRINT T; N$
 60 W = W-V*T
 70 NEXT J
 80 STOP
1000 DATA 2000, BANCONOTA/E DA
     20 STERLINE
1010 DATA 1000, BANCONOTA/E DA
     10 STERLINE
1020 DATA  500, BANCONOTA/E DA
     5 STERLINE
1030 DATA  100, BANCONOTA/E DA
     1 STERLINA
1040 DATA   50, PEZZO/I DA 50 PENNY
1050 DATA   20, PEZZO/I DA 20 PENNY
1060 DATA   10, PEZZO/I DA 10 PENNY
1070 DATA    5, PEZZO/I DA  5 PENNY
1080 DATA    2, PEZZO/I DA  2 PENCE
1090 DATA    1, PEZZO/I DA  1 PENNY
1100 DATA  0.5, PEZZO/I DA ½ PENNY
```


ESPERIMENTO 16·1

Modificare il programma dell'analisi del denaro in modo che funzioni per il sistema monetario degli Stati Uniti d'America. I valori e i rispettivi nomi sono:

Banconota/e da 50$
Banconota/e da 10$
Banconota/e da 5$
Dollaro/i
Quarto/i di dollaro $ 0.25
DIME 1/10 di dollaro $ 0.10
NICKEL 5 cents $ 0.05
PENNY centesimo $ 0.01

Esperimento 16.1 completato	

FORMATO DELLE ISTRUZIONI DATA

Ci sono poche e semplici regole che occorre conoscere a proposito delle istruzioni DATA. Innanzitutto le istruzioni DATA possono contenere stringhe e numeri misti in qualsiasi ordine. La lunghezza massima di un'istruzione è di 4 righe di schermo (88 caratteri). Se un'istruzione DATA comprende più di un elemento, gli elementi sono separati da virgole. Non occorre inserire virgolette davanti e dietro a una stringa a meno che la stringa non comprenda una virgola o un carattere di controllo dello schermo ad esempio SHIFT e CLR/HOME. Per esempio, l'istruzione DATA

```
DATA 21, QUEEN ST.
```

contiene due elementi (un numero e una stringa) mentre

```
DATA "21, QUEEN ST"
```

contiene soltanto un elemento: la stringa 21, QUEEN ST.
Secondo, il numero di elementi in ciascuna istruzione DATA non deve corrispondere al numero di variabili nel comando READ. Ciascun READ prende tanti elementi quanti ne occorrono, e se ciò usa solo una parte di riga, non ha importanza; il successivo READ inizia da dove è terminato il primo. Analogamente un'istruzione DATA che contiene troppi elementi, farà semplicemente sì che il VIC vada alla successiva istruzione DATA non appena è necessario.
Per illustrare questo punto, il secondo programma di analisi del denaro funzionerà ancora correttamente se i dati sono ridisposti come segue:

```
1000 DATA 2000
1010 DATA BANCONOTE DA 20 STERLINE
1030 DATA 1000
1040 DATA BANCONOTE DA 10 STERLINE
```

oppure

```
1000 DATA 2000, BANCONOTE DA 20
STERLINE, 1000, BANCONOTE DA 10
STERLINE, 500, BANCONOTE DA 5 STERLINE,
100,...
```

o ancora

```
1000 DATA 2000
1010 DATA BANCONOTE DA 20 STERLINE, 1000
1020 DATA BANCONOTE DA 10 STERLINE, 500,
BANCONOTE DA 5 STERLINE, 100
1030 DATA BANCONOTE DA 1 STERLINA
```

In altre parole, il diavoletto VIC tratta rapidamente con i contenuti delle istruzioni DATA sulla tastiera senza essere troppo preoccupato di dove l'istruzione termina o dove inizia la successiva.
Terzo, le istruzioni DATA non fanno parte del programma allo stesso modo dei vari comandi. Ogniqualvolta si batte RUN, le istruzioni DATA vengono effettivamente selezionate e inserite in una pila diversa prima che il primo comando venga eseguito. Ciò significa che quando un programma viene impostato, le istruzioni DATA possono essere davanti ad esso, dopo di esso o tra i comandi. Per esempio, il programma dell'analisi del denaro potrebbe essere stato scritto con un'istruzione DATA a righe alterne. In ogni caso, questo sarebbe stato un esempio di cattiva prassi di programmazione. Ammucchiare il programma in questa maniera ne avrebbe reso difficile la lettura e non è cosa consigliata. Un'utile convenzione consiste nel porre tutte le istruzioni DATA insieme al termine o all'inizio del programma, e nel dare ad esse numeri di label che sono immediatamente riconoscibili.

ERRORI DELLE ISTRUZIONI DATA E READ

Possono verificarsi due tipi di errori con le istruzioni DATA e comandi READ. Se si dà un comando READ quando tutte le istruzioni DATA sono già state usate (o se non ce n'è), si ottiene un errore OUT OF DATA. Ciò spiega cosa succede se si batte RETURN quando il cursore è in una riga con READY.
Il VIC pensa che si estenda READ Y. Se READ specifica una variabile numerica e la successiva voce nell'istruzione DATA non è un numero, si ottiene un errore di sintassi nell'istruzione DATA (non nel comando READ). Ad es. se si immette:

```
10 READ A
...
...
100 DATA HELLO
```

si ottiene

```
? SYNTAX
ERROR IN 100
```

Ciò può essere motivo di confusione dato che l'errore reale è più probabilmente nel comando READ; si voleva probabilmente inserire:

```
10 READ A$
```

Vale la pena di notare che non c'è corrispondente difetto nell'altro modo: se si legge un numero in una variabile stringa questo viene considerato come stringa di cifre senza segnalare un errore. Perchè no? Una stringa può essere qualsiasi sequenza di caratteri, compresa una sequenza di cifre.

IL COMANDO RESTORE

Infine citeremo il comando RESTORE. Questo comando prende il VIC e lo riporta all'inizio del mucchio d'istruzioni DATA cosicchè queste possono essere lette da capo. RESTORE può essere usato in qualsiasi momento anche se le istruzioni DATA non sono state utilizzate per nulla.

ESPERIMENTO 16·2

a) Scrivere un programma per visualizzare i mesi dell'anno, uno per riga. Le ultime righe del programma dovrebbero essere:

```
1000 DATA GENNAIO, FEBBRAIO, MARZO,
     APRILE
1010 DATA MAGGIO, GIUGNO, LUGLIO,
     AGOSTO
1030 DATA SETTEMBRE, OTTOBRE,
     NOVEMBRE, DICEMBRE
```

b) Scrivere un programma per leggere una data (nella forma giorno-mese-anno) e visualizzare la data e l'anno in cifre, ma il mese in lettere. Per esempio un input di

22,6,1936

dovrebbe dare 22 GIUGNO 1936

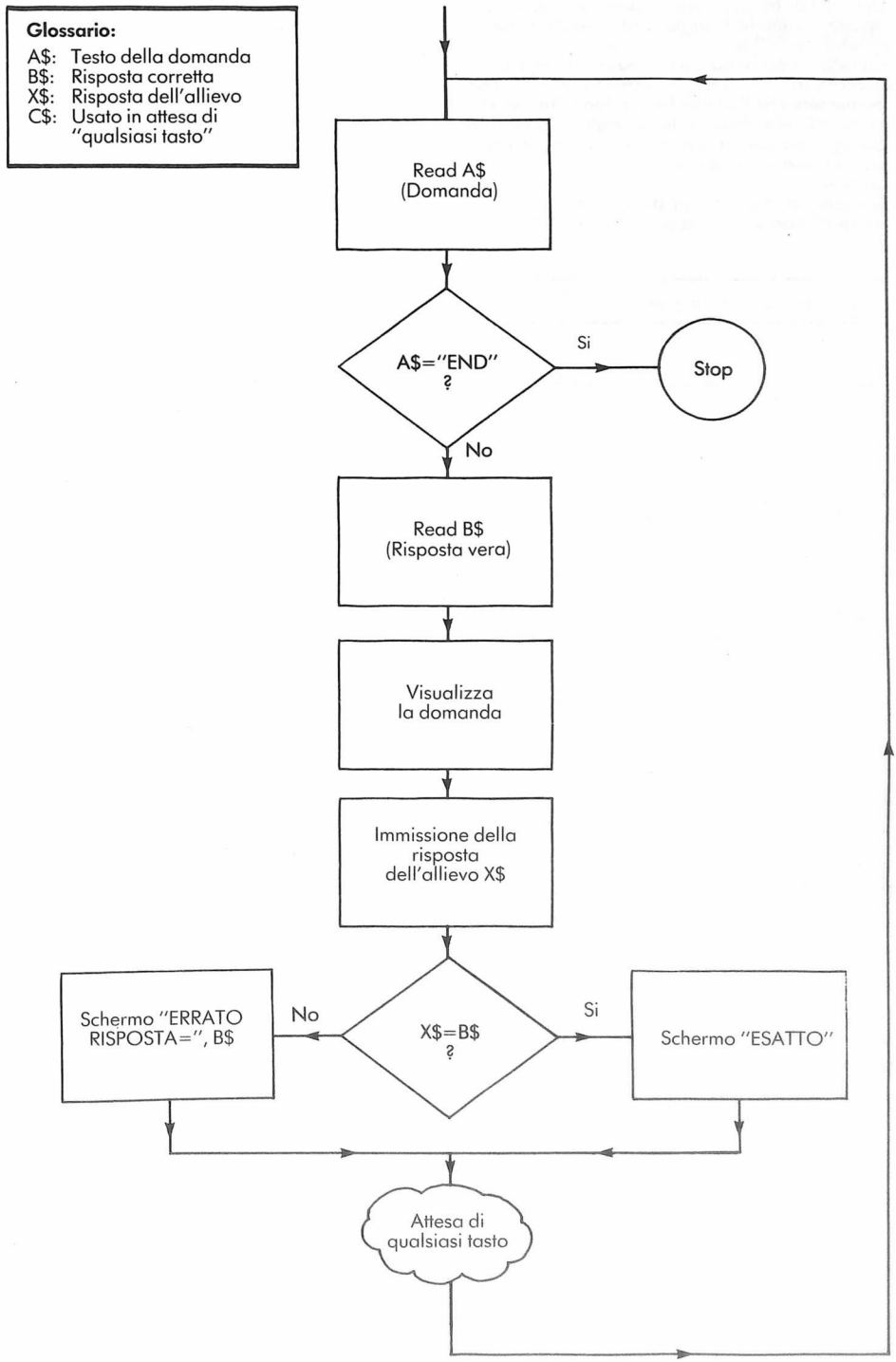
Usare le stesse istruzioni DATA come nel precedente programma. Scrivere il programma nella forma di un'iterazione includendo RESTORE in modo che quando viene visualizzata una data, ne viene immessa un'altra.

Esperimento 16.2 completato	

ESPERIMENTO 16·3

L'istruzione DATA è preziosissima nei programmi che presentano quiz o domande. Studiare il seguente programma, immetterlo sul VIC e provarlo.

```
  10 READ A$
  20 IF A$ = "FINE" THEN 190
  30 READ B$
  40 PRINT "  SHIFT  e  CLR HOME  "
  50 PRINT A$
  60 PRINT
  70 INPUT X$
  80 PRINT
  90 IF X$ = B$ THEN 130
 100 PRINT "SBAGLIATO. LA RISPOSTA È"
 110 PRINT B$
 120 GOTO 140
 130 PRINT "ESATTO"
 140 PRINT
 150 PRINT "PREMI QUALSIASI TASTO"
 160 GET C$
 170 IF C$ = "" THEN 160
 180 GOTO 10
 190 STOP
 500 DATA CAPITALE DELLA FRANCIA,
     PARIGI
 510 DATA TOKYO È CAPITALE
     DEL, GIAPPONE
 520 DATA PAESE CON MAGG. NUM.
     ABITANTI, CINA
 530 DATA PAESE A SUD DEGLI USA,
     MESSICO
1000 DATA END
```


Glossario:
A$: Testo della domanda
B$: Risposta corretta
X$: Risposta dell'allievo
C$: Usato in attesa di "qualsiasi tasto"
Read A$ (Domanda)
A$="END" ?
Si
Stop
No
Read B$ (Risposta vera)
Visualizza la domanda
Immissione della risposta dell'allievo X$
X$=B$?
No
Schermo "ERRATO RISPOSTA=", B$
Si
Schermo "ESATTO"
Attesa di qualsiasi tasto

Una volta fatto girare il programma, creare alcune domande e aggiungerle usando i numeri di label da 540 in poi.
Questo programma quiz di base può essere migliorato in vari mod: per esempio, si potrebbe permettere che l'allievo faccia due o tre tentativi prima di visualizzare la risposta esatta e si potrebbe contare il numero di risposte corrette e visualizzare una percentuale al termine della lezione.
Scrivere un programma di quiz migliorato per porre domande su un argomento preferito.

Esperimento 16.3 completato	

Il quiz di auto-test per questa Unità è detto UNIT16QUIZ

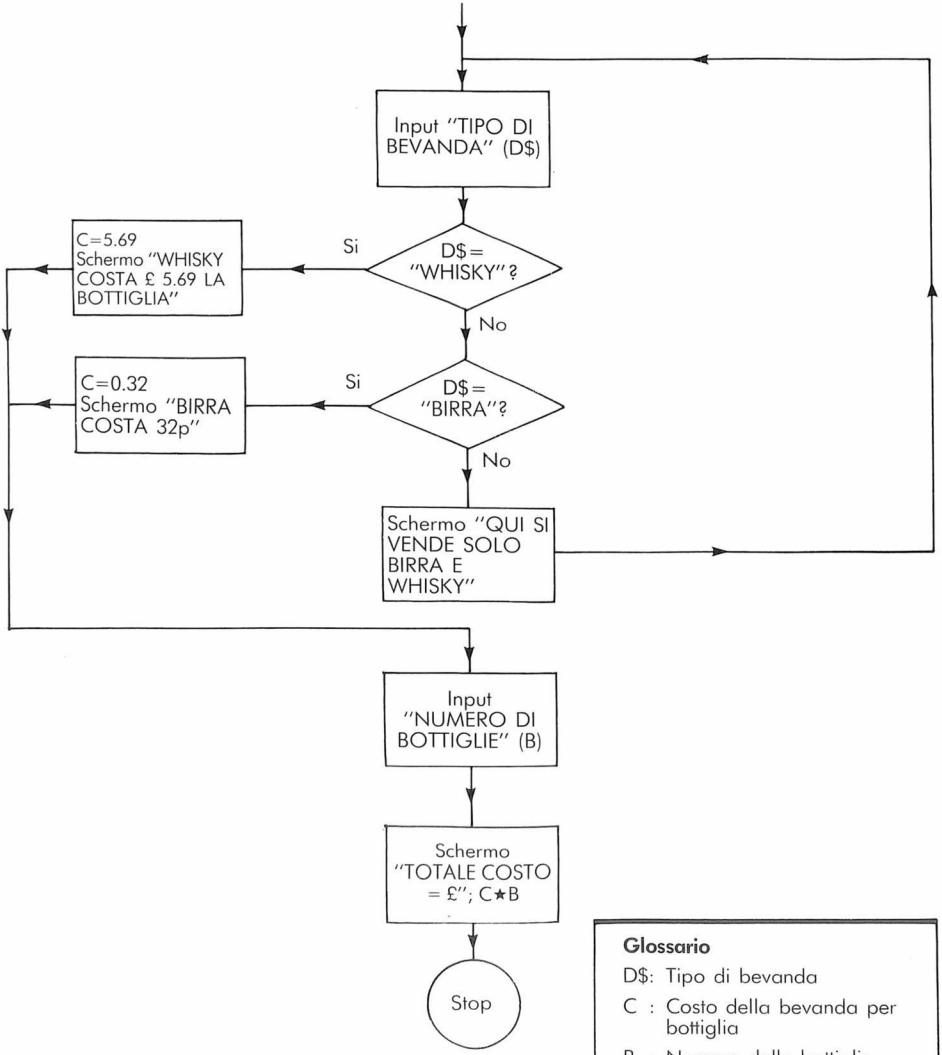
UNITA':17

COME AFFRONTARE LA COMPLESSITÀ DEI PROGRAMMI

Ora incontreremo il più grande problema del programmatore — il controllo della complessità. Molte persone comprendono i principi della programmazione e possono scrivere brevi e semplici programmi con facilità, ma quando applicano gli stessi metodi ai problemi più complessi ottengono scarso successo. Questo sembra essere un limite alla quantità di dettagli che è possibile tenere in mente in qualsiasi momento. Quando questo limite viene superato, il risultato è confusione, quantità di gravi errori e programmi che sistematicamente danno risposte sbagliate. Nessuno dei sussidi descritti nella Parte 1 (ad esempio il controllo nell'Unità 8 e lo schema di flusso nell'Unità 11), danno molto aiuto se usati isolatamente. Tutto è ancor troppo complicato.

In questa unità si considereranno alcuni dei modi per ridurre la complessità di un programma, senza deviare dal lavoro che si suppone esso debba fare. Chi ambisce diventare un programmatore, deve studiare questi metodi e usarli sistematicamente in tutto il lavoro. Essi saranno di aiuto dando soluzioni invece che bloccare su un problema, cercando di risolverlo con modifiche casuali, per vedere se per fortuna è possibile imbattersi in quella che sembra funzionare. Uno dei principali vantaggi degli schemi di flusso nel disegno dei programmi è che essi indicano la struttura di un programma molto meglio che non segmenti di codice BASIC.

Si confronti:



Glossario

D$: Tipo di bevanda

C : Costo della bevanda per bottiglia

B : Numero delle bottiglie

con:

```
10 INPUT "TIPO DI BEVANDA"; D$
20 IF D$ = "WHISKY" THEN 70
30 IF D$ = "BIRRA" THEN 100
40 PRINT "SI VENDE SOLO BIRRA"
50 PRINT "E WHISKY"
60 GOTO 10
70 C = 5.69
80 PRINT "WHISKY COSTA 5.69"
90 GOTO 120
100 C = 0.32
110 PRINT "BIRRA COSTA 32P."
120 INPUT "QUANTE BOTTIGLIE"; B
130 PRINT "TOTALE COSTO = £"; C*B
140 STOP
```

L'effetto di tradurre lo schema di flusso in BASIC è ovvio; una chiara serie di istruzioni è stata spiacciccata in una lista unidimensionale senza forma di comandi che devono essere disaggrovigliati prima di avere un senso.

USO DEL DUE PUNTI

Il Microsoft BASIC, la versione del linguaggio usato sul VIC, ha alcune utili caratteristiche che gli consentono di conservare la maggior parte della struttura di uno schema di flusso.
Un'intera sequenza di comandi può essere raggruppata riunendo i comandi stessi dopo lo stesso numero di label e separandoli con il simbolo del due punti ":" senza un RETURN. L'effetto è lo stesso che se il comando fosse stato scritto su righe separate salvo che è impossibile saltare uno dei comandi intermedi mediante un GOTO o un IF. Per esempio, la sequenza

```
10 INPUT "COME TI CHIAMI"; N$
20 PRINT "HELLO"; N$
30 PRINT
40 S = 0
50 Q = 100
```

potrebbe essere sostituita da

```
10 INPUT"COME TI CHIAMI";N$:
   PRINT"HELLO";N$:PRINT:S=0:Q=100
```

posto che nessun'altra parte del programma debba fare un salto ad uno qualsiasi dei comandi originariamente numerati da 20 a 50. Il numero limite dei comandi che possono essere raggruppati è definito dalla massima lunghezza per istruzione (88 caratteri, 4 righe di schermo).

ALTRI MODI DI USARE LE ISTRUZIONI IF-THEN

Il comando THEN in un IF-THEN non deve sempre essere seguito da un numero di label ma può anche essere seguito da un comando (o da un gruppo di comandi) che vengono eseguiti soltanto se la condizione è vera. Ciò significa che è possibile sostituire

```
10 IF X=0 THEN 30
20 GOTO 40
30 PRINT "X=0"
40 ———
```

con

```
10 IF X=0 THEN PRINT "X=0"
20 ———
```

A questo punto è necessaria un po' di attenzione. Se la condizione in un comando IF-THEN è falsa, il VIC trasferisce sempre il controllo all'istruzione con label successiva. Ne segue che se un comando IF-THEN fa parte di un gruppo di comandi separati da due punti qualsiasi comando che lo segue sarà inevitabilmente saltato se la condizione è falsa. Ciò potrebbe non essere quello che si intende fare! Per illustrare questo punto si consideri la sequenza

```
10 IF X=0 THEN 20: Y=5: GOTO 30
20 Y=7
30 PRINT Y
```

Osservando il programma ci si può domandare cosa significa: il programmatore voleva che Y venisse definito a 7 se X era 0 o a 5 se non era 0. Sfortunatamente ciò non è quello che effettivamente succede. Si consideri il comando sulla riga 10:

```
IF X=THEN 20
```

Se la condizione è vera (e cioè X=0), il VIC salta al comando 20, esattamente come ci si aspetterebbe. Se la condizione è falsa, la macchina segue la regola e trasferisce il controllo all'istruzione con label successiva che risulta essere 20! I comandi

```
Y=5: GOTO 30
```

non verranno mai eseguiti in nessun caso. Questa trappola viene evitata seguendo una regola semplice: se un comando IF-THEN coinvolge un salto ad una label, questo deve essere seguito da un RETURN — non da un due punti.
Usando questa nuova funzione, il programma del prezzo delle bevande precedentemente discusso può essere abbreviato e chiarito:

```
10 INPUT "TIPO DI BEVANDA"; D$
20 IF D$="WHISKY"THENC=5.69:PRINT
   "WHISKY COSTA £ 5.69.":GOTO 50
30 IF D$ ="BIRRA"THEN C=0.32: PRINT
   "BIRRA COSTA 32P":GOTO 50
40 PRINT "VENDITA SOLO BIRRA E":PRINT
   "WHISKY":GOTO 10
50 INPUT "QUANTE BOTTIGLIE";B
60 PRINT "TOTALE COSTO= £";C*B
70 STOP
```

Confrontare le due versioni e notare che la seconda si conforma molto più da vicino alla struttura dello schema di flusso originale.

ESPERIMENTO 17·1

Riscrivere i seguenti programmi usando il minor numero possibile di comandi con label:

a)
```
10 INPUT "QUANTI MINUTI"; M
20 R = TI+ M*3600
30 IF TI <R THEN 30
40 PRINT "TEMPO SCADUTO"
50 STOP
```

b)
```
10 PRINT "USA 1000000 PER
20 PRINT "TERMINARE INPUT"
30 S=0
40 N=0
50 INPUT "NUMERO SUCCESSIVO"; X
60 IF X = 1000000 THEN 100
70 S=S+X
80 N=N+1
90 GOTO 50
100 PRINT "MEDIA="; S/N
110 STOP
```

c) Caricare il programma intitolato UNIT 17 PROG e listarlo. Si vedrà che si suppone che esso riconosca i nostri JIM, BOB, KATE e PENNY e dica di che cosa mancano. Sfortunatamente il programma non funziona. Correggerlo.

Esperimento 17.1 completato	

OPERATORI LOGICI

Ulteriore aiuto nella semplificazione dei programmi è dato dagli operatori logici AND, OR e NOT. Queste parole hanno significati speciali in BASIC e sono usate per concatenare semplici condizioni nei comandi IF-THEN cosicchè si possono prendere decisioni più complesse.

L'OPERATORE "AND"

Iniziamo con AND, l'operatore logico più frequentemente usato. Esso generalmente si trova tra due condizioni come in questo caso:

```
IF A> 18 AND M$ = "Q" THEN . . .
```


La risultante condizione composta (che è tutto ciò che si trova tra IF e THEN) è *vera* soltanto se entrambe le condizioni semplici sono a loro volta *vere*. Se una o l'altra (o entrambe) sono *false*, la condizione composta è *falsa*.
L'operatore AND generalmente consente di sostituire due o più comandi IF-THEN con uno solo. Si consideri un programma che esamini le domande per entrare in una società di polizia privata. Le regole dicono che i candidati devono avere almeno 18 anni di età ed essere alti almeno 160 cm. Lo schema di flusso probabilmente comprenderà una sezione di questo tipo:

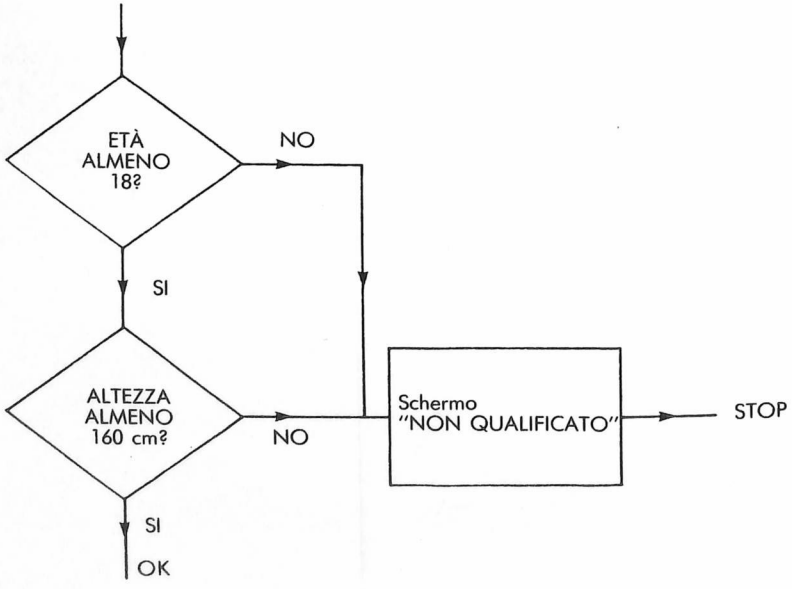


Figure 17.2

Se fossero ammesse solo condizioni semplici, la sequenza potrebbe essere codificata nella forma:

```
100 IF A > = 18 THEN 130
110 PRINT "NON QUALIFICATO"
120 STOP
130 IF H < 160 THEN 110
140 REM ETÀ E ALTEZZA ACCETTABILI
150...
```

Usando una condizione composta, che applica le prove per l'età e l'altezza contemporaneamente, la sezione può essere scritta in maniera molto più compatta e con maggior eleganza. Il suo significato è immediatamente chiaro:

```
100 IF A > = 18 AND H > = 160 THEN
120
110 PRINT "NON QUALIFICATO": STOP
120 REM ETA E ALTEZZA ACCETTABILI
```

AND è un operatore piuttosto simile a ★, salvo che usa condizioni anzichè numeri. Ciò significa che le condizioni possono essere concatenate indefinitamente, fino al limite della lunghezza interna di 88 caratteri. Si potrebbe inserire:

```
IF A=5 AND B<7 AND X$<>"GIUSEPPE"
AND... THEN 1260
```

La risultante istruzione composta è vera soltanto se *tutte* le condizioni semplici sono a loro volta vere.
Un uso speciale di AND è di decidere se una variabile cade in un campo specifico. È possibile provare se il valore della variabile X cade ad esempio tra 7 e 12 compresi. Un matematico esprimerebbe quest'idea scrivendo:

```
7< = X< = 12
```

ma non è possibile inserire tale "condizione" in un comando IF-THEN — non è in linguaggio BASIC. Per contro, si usano due semplici condizioni concatenate con AND:

```
IF X>=7 AND X< = 12 THEN...
```

L'OPERATORE "OR"

L'operatore OR è usato più o meno allo stesso modo di AND, ma produce una condizione vera se una o l'altra o entrambe le due condizioni coinvolte sono vere. Un uso comune di OR è per controllare che la risposta ad una domanda sia una di quelle previste. Per esempio:

```
10 PRINT"SEI IN GAMBA? RISPONDI"
20 INPUT "SI O NO"; A$
30 IF A$="SI" OR A$="NO" THEN 50
40 PRINT "RISPONDI ALLA DOMANDA!";:
   GOTO 10
50 REM RICEVUTA RISPOSTA VALIDA
```

La condizione composta può essere ampliata per includere parecchi OR come in:

```
IF H$="NERO"OR H$="MARRONE"OR
H$="ROSSO"ORH$="ONESTO"THEN
30
```

Notare che una forma che non è possibile usare (dato che non è in linguaggio BASIC corretto), è:

```
IF H$="NERO" OR"MARRONE"OR"ROSSO"
OR "ONESTO" THEN 30
```

COMBINAZIONI DI OPERATORI LOGICI

È spesso comodo usare condizioni composte che sfruttano più di un tipo di operatore logico. Per esempio, le regole per emettere le patenti di guida, possono dire che il richiedente deve avere un'età superiore ai 16 anni per guidare un motociclo, superiore ai 18 anni per guidare un'automobile e superiore ai 21 anni per guidare un autobus. È possibile porre:

```
IF V$="MOTOCICLO" AND A >=16
OR V$="AUTO" AND A >=18
OR V$="BUS" AND A>=21
THEN PRINT "OK"
```

Quando il VIC viene ad elaborare questa condizione composta, lo fa in un ordine particolare; prima di tutto le condizioni semplici, quindi gli AND e infine gli OR. In questo esempio, il processo dà esattamente il risultato richiesto.
In pratica, le condizioni composte potrebbero non sempre essere così facili da scrivere. Si consideri l'esempio di un'azienda che cerca un programmatore con tre anni di esperienza nel linguaggio di programmazione BASIC o COBOL. Se si ponesse:

```
IF E>=3 AND L$="BASIC" OR
          L$="COBOL"
```

(E è il numero di anni di esperienza, L$ il linguaggio)

le regole di valutazione sceglierebbero:
a) Un programmatore BASIC con almeno 3 anni di esperienza
o
b) Un programmatore COBOL con al limite nessuna esperienza.

Ciò in quanto viene usato per primo AND e si associa E>=3 con "BASIC" ma non non "COBOL". Il modo per evitare questo problema consiste nell'usare la parentesi. Esattamente come nell'algebra ordinaria, tutto ciò che si trova all'interno di parentesi viene elaborato prima di qualsiasi cosa che si trovi all'esterno. Scrivendo

```
IF E>=3 AND (L$="BASIC' OR L$=
"COBOL")
```

si ottiene l'ordine corretto e pertanto la risposta sarà corretta.

IL COMANDO "NOT"

NOT è il terzo operatore logico. Esso viene applicato ad una condizione (semplice o composta) e ne inverte il senso. Per esempio, se X>5 è vero, NOT X >5 è falso e viceversa.
Non è mai necessario usare NOT con condizioni semplici dato che può essere usata tranquillamente la relazione "opposta". Pertanto

NOT X=5 è lo stesso che scrivere X<>5
NOT X<>5 è lo stesso che scrivere X=5
NOT X<5 è lo stesso che scrivere X>=5
NOT X>5 è lo stesso che scrivere X<=5
NOT X<=5 è lo stesso che scrivere X>5
NOT X>=5 è lo stesso che scrivere X<5

NOT entra in scena con proprie condizioni composte, come si vedrà più avanti.
La regole del BASIC specificano che in assenza di parentesi, NOT debba essere usato prima di qualsiasi altro operatore logico. Si è visto che è inutile usarlo per invertire condizioni semplici. Per far sì che esso affronti un'intera condizione composta, la condizione deve essere racchiusa tra parentesi come questa:

```
IF NOT (X=5 AND Y=7) THEN...
```

Una condizione composta con una NOT potrebbe essere usata per rilevare e rifiutare talune risposte vietate. Ciò è illustrato dalla sequenza:

```
10 PRINT "COSA PENSI"
20 INPUT "DI QUESTO"; R$
30 IF NOT (R$="FLAGELLO" OR R$=
   "DISGRAZIA" OR R$="BESTEMMIA")
   THEN 50
40 PRINT "BADA A COME PARLI!"
   GOTO 10
50 REM RISPOSTA NON VERA
```

Le condizioni composte precedute da NOT possono essere spesso semplificate. Se ogni operatore logico all'interno delle parentesi è un OR, il NOT e le parentesi possono essere tolti posto che:

a) Ciascuna condizione semplice sia invertita
b) Ogni OR sia cambiato in un AND.
   Pertanto la riga 30 nel nostro esempio, potrebbe essere scritta:

```
30 IFR$<>"FLAGELLO"AND R$<>"DISGRAZIA"
   AND R$<>"BESTEMMIA" THEN 50
```

Una regola simile si applica alle condizioni composte invertite in cui ogni operatore è un AND: gli AND diventano OR, le condizioni semplici sono invertite e il NOT e le parentesi vengono tolti. Queste due regole sono dette "leggi di De Morgan" da chi le ha scoperte. La maggior parte dei manuali di testo e di programmazione, raccomandano di evitare le condizioni NOT ogniqualvolta possibile ed è solitamente più facile procedere in questo modo.

ESPERIMENTO 17·2

a) Le norme doganali dicono che è possibile importare senza pagare dogana: "un litro di alcool e due litri di vino liquoroso o quattro litri di vino semplice".
S, F e W sono variabili che denotano le quantità di alcool, vino liquoroso e vino semplice rispettivamente. La norma stessa è ambigua; occorre pertanto scrivere due condizioni composte ciascuna delle quali deve essere vera se deve essere pagata dogana. La prima delle risposte dovrebbe interpretare la regola nel senso più generoso per il viaggiatore; la seconda nel senso più restrittivo.

b) Usare le leggi di De Morgan per esprimere le seguenti condizioni composte senza usare il NOT:

```
NOT (N$="BIANCHI" OR N$="ROSSI"
      OR N$="MAURI")
NOT (X< = 15 AND X >=4)
```

c) Si supponga che un programma abbia misurato un tempo di reazione (in secondi) e lo abbia inserito nella variabile T. Scrivere una sezione di codice che visualizza uno dei seguenti commenti a seconda dei casi:

Valore di T	Commento
T < 0.1	FANTASTICO!!
0.1 < = T <0.15	ECCEZIONALE
0.15 < = T < 0.2	MOLTO BUONO
0.2 < = T < 0.25	BUONO
0.25 < = T < 0.28	DISCRETO
0.28 < = T < 0.33	MOLTO LENTO
0.33 < = T < 0.4	SVEGLIA!
0.4 < T	PROVA ANCORA QUANDO SEI SOBRIO!!

d) C'è un'enciclopedia in quattro volumi:
   1: Da ABRAHAM a FRANCE
   2: Da FRANCHISE a LEVANTE
   3: Da LEVITAZIONE a QUOTA
   4: Da QUOZIENTE a XILOFONO

Scrivere un programma che immetta qualsiasi parola e mi dice in quale volume cercarla. Il programma dovrebbe anche dire se la parola non è inclusa (ad esempio "QUOTO").
Suggerimento: ricordare che gli operatori <, > = e altri del genere, possono essere usati come stringhe e danno risultati secondo il loro ordine alfabetico.

Esperimento 17.2 completato	

UNITA': 18

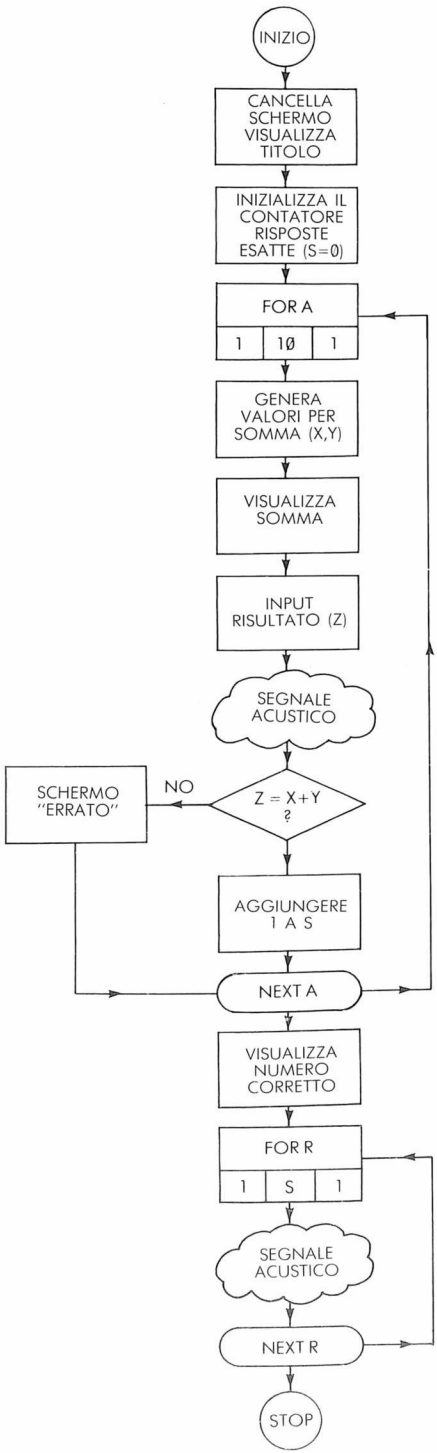
INTRODUZIONE ALLE SUBROUTINE

Fino a questo punto nel corso, si è fatta abbastanza pratica scrivendo piccoli programmi — ad esempio fino a 30 comandi o giù di lì. Presto o tardi si sarà portati a scoprire che i programmi più interessanti devono essere molto più lunghi e che occorrono strumenti e tecniche specializzati per aiutare a costruirli correttamente.

Un sussidio vitale nello scrivere grandi programmi è la possibilità di ricorrere alle subroutine. Nello schema di flusso (Unità 11 della Parte 1), si è già parlato della possibilità di inserire azioni complicate all'interno di "nuvolette" e di rimandarne la codifica ad un tempo successivo. È questo un metodo utile per affrontare le complessità in quanto consente di trascurare una gran parte di dettagli e di concentrarsi sui risultati principali del problema.

Le subroutine sono come le nuvolette. La subroutine si compone di un gruppo di comandi che collaborano per eseguire un compito particolare e ben definito. Quando si disegna un programma, si può pensare a questo compito come ad una singola fase, per quanto complicato possa essere.

Inizieremo con una subroutine semplicissima. Si consideri un programma che ponga dei quiz sulle somme.



Glossario

S: Contatore del numero di risposte corrette
X, Y: Valori da sommare
Z: Somma (risultati)
A: Contatore del numero di domande
R: Contatore dell'iterazione di premio

Il compito che abbiamo scelto da usare come una subroutine è la nuvoletta contrassegnata

"Make Sound"

Il codice per questa azione sarebbe normalmente qualcosa del genere:

```
10 POKE 36878,15:POKE 36876,245
20 FOR M=1 TO 100:NEXT M
30 POKE 36878,0
40 FOR M=1 TO 800: NEXT M
```

Per trasformare queste istruzioni in una subroutine occorre "vestirle" in modo che si inseriscano correttamente nel programma principale. In linguaggio tecnico occorre cioè fornire una *interfaccia.*
Innanzitutto si sceglie una serie di numeri di label elevata per non entrare in conflitto con il programma principale o con qualsiasi altra subroutine. Il numero di label più elevato ammesso è 63999.
Secondo si aggiunge un commento (REM) descrittivo all'inizio della subroutine. Ciò non è assolutamente necessario, ma è un segno di programmazione competente ed è molto comodo dato che molte subroutine possono essere usate in altri programmi.
Terzo, si assegna un nome distintivo alla variabile usata dalla subroutine, ad esempio una doppia lettera. Anche in questo caso non è obbligatorio ma segue un'utile convenzione che verrà spiegata più avanti.
Infine, si terminerà la subroutine con il comando speciale:

RETURN

Questo comando, che viene usato al termine di ogni subroutine, deve essere battuto correttamente e interamente (6 lettere) e seguito come al solito da RETURN. Non confondere il comando RETURN e il tasto RETURN — essi sono totalmente diversi. Usando questa convenzione, le istruzioni per la subroutine potrebbero essere:

```
1000 REM SUBROUTINE PER CREARE
     SUONO PIP
1010 POKE 36878,15:POKE 36876,245
1020 FOR MM=1 TO 100:NEXT MM
1030 POKE 36878,0
1040 FOR MM=1 TO 800: NEXT MM
1050 RETURN
```

Ora può essere scritto il programma principale. Ogniqualvolta si ha una nuvoletta con l'istruzione "CREA SUONO" (segnale acustico), questa può essere tradotta dal comando di richiamo della subroutine:

GOSUB 1000

solitamente seguito da un REM sulla stessa riga per spiegare cio che viene fatto. L'intero programma (compresa la subroutine) si trasforma come segue: SHIFT e CLR HOME

```
10 PRINT "
   PROVA ARITMETICA"
20 PRINT "CALCOLA SOMME
   SEGUENTI"
30 S=0
40 FOR A=1 TO 10
50 X = INT(10*RND(0)+1)
60 Y = INT(10*RND(0)+1)
70 PRINT X; "+"; Y; "=";
80 INPUT Z
90 GOSUB 1000:REM CREA SUONO
100 IF Z=X+Y THEN 130
110 PRINT "ERRATO"
120 GOTO 150
130 PRINT "ESATTO"
140 S=S+1
150 NEXT A
155 FOR T=1 TO 500: NEXT T
160 PRINT "ESATTO";S;"SU 10"
170 FOR R = 1 TO S
180 GOSUB 1000:REM CREA SUONO
190 NEXT R
200 STOP
1000 REM SUBROUTINE PER CREARE
     SUONO PIP
1010 POKE 36878,15:POKE 36876,245
1020 FOR MM=1 TO 100:NEXT MM
1030 POKE 36878,0
1040 FOR M=1 TO 800: NEXT MM
1050 RETURN
```

Battere questo programma e verificare che funzioni come ci si aspetta.

COME FUNZIONA GOSUB

Eseguiamo ora attentamente il programma. Il GOSUB è più o meno come GOTO ma con una differenza importante. Quando il VIC salta alla subroutine, ricorda la label del successivo comando del programma principale e memorizza questa informazione in una parte speciale della memoria denominata catasta operativa. Il RETURN al termine della subroutine assomiglia a sua volta ad un GOTO ma la destinazione è sempre il numero di label memorizzata nella catasta! Ciò fornisce un meccanismo automatico che assicura che, ogniqualvolta il VIC finisce l'esecuzione di una subroutine, ritorni sempre indietro al punto corretto nel programma principale. Il numero di label memorizzato nella catasta, mentre il VIC esegue la subroutine, è detto *concatenamento* o *indirizzo di ritorno.*
Il nostro programma inizia eseguendo i comandi nella riga 10 nel modo solito. Il comando 90 dice GOSUB 1000; così il computer salta a 1000, ma per strada nota la label del comando che segue GOSUB (che risulta essere 100) e inserisce "100" nella catasta. In questo caso "100" è il concatenamento.
Una volta raggiunta la subroutine, la macchina esegue i comandi 1010-1050 di cui l'ultima riga è RETURN. Dato che la catasta operativa contiene "100" il RETURN è equivalente a un "GOTO 100 e così la macchina ritorna nel programma principale nel punto giusto.
Quando a tutte le somme è stata data risposta, il

programma arriva ad un altro richiamo di subroutine.
Esso salta di nuovo alla riga 1000 ma stavolta il concatenamento disposto nella catasta operativa è "190" invece di "100". Quando viene eseguito RETURN una seconda volta, il controllo ritorna alla riga 190 e non alla riga 100 come precedentemente. La subroutine qui fa parte di un'iterazione. La variabile di controllo per questa iterazione è S, il numero giusto, cosicchè la subroutine verrà richiamata una volta per ciascuna somma cui si è data una risposta corretta.
Questo semplice esempio mostra come è possibile separare uno dei lavori che costituiscono un programma e trattarlo a parte. Chiaramente, più complessa è la funzione che si deve separare e maggiormente si semplifica il disegno generale del programma. L'esempio mostra anche come è possibile richiamare una subroutine da più di un punto senza doverla scrivere ogni volta. Ciò può essere vero, ma bisogna fare attenzione a coloro che dicono che l'importanza principale delle subroutine consiste nell'abbreviare i programmi. Ciò è falso e ingannevole. Il punto realmente importante della subroutine è la possibilità di semplificare la struttura dei programmi separando sezioni compresse e considerandole isolatamente. Ciò sarà più ovvio nelle illustrazioni successive.

ESPERIMENTO 18·1

a) Modificare il programma a pagina 172 in modo che il margine diventi nero quando una risposta è sbagliata e diventi porpora quando è corretta. Non dimenticarsi di ripristinare il colore iniziale quando viene visualizzata la somma successiva.
b) Usare ora le stesse subroutine per scrivere un programma completamente diverso. Si immagini un bambino a cui viene insegnato a contare. Per ogni domanda il programma emette una serie di pip (tra 1 e 9). Ci si aspetta che l'allievo conti i pip e che batta i numeri adatti. Per esempio, ...pip-pip-pip... ha la risposta corretta "3" e qualsiasi altra cosa è sbagliata.

Esperimento 18.1 completato	

SUBROUTINE CON COMPITI VARIABILI

L'esperimento 18.1 mostra che le subroutine possono essere abbastanza indipendenti dal programma in cui risiedono. Esse possono essere spostate da programma a programma e possono essere scritte da persone diverse. (Il lettore ha appena usato le subroutine scritte dall'autore di questo manuale nell'ultimo programma). È possibile effettivamente comprare librerie di subroutine per eseguire le varie funzioni e i vari lavori e ciò può risparmiare un tempo notevole nella costruzione di un programma.

Il programma con subroutine è paragonabile ad un ufficio con un capo (il programma principale) e parecchi assistenti (le subroutine). Ciascun assistente è specializzato nel fare solo un lavoro, ad esempio prelevare il documento in cima ad un armadio di archivio o fare il caffè. Il capo ha un telefono e può chiamare un assistente in qualsiasi momento e dirgli di fare il suo lavoro speciale. Quindi (almeno in BASIC), il capo attende fino a che l'assistente lo richiama e dice "fatto".

Le subroutine che abbiamo già esaminato erano molto limitate. Ciascuna di esse poteva svolgere soltanto un lavoro preciso, ad esempio cambiare il colore del margine o eseguire un particolare tipo di rumore. In un ufficio dove gli assistenti sono ugualmente inflessibili su ciò che essi possono fare, il solo comando che il capo può dare è "fallo!". Ciò fa sì che l'assistente incaricato del caffè inizi a preparare una tazza di caffè, che è la sola cosa che egli sa fare. Se il capo ha dei visitatori e vuole cinque tazze di caffè, deve chiamare l'assistente del caffè per cinque volte. Gli assistenti in questo ufficio, sarebbero molto più utili se il capo potesse in qualche modo qualificare il lavoro che ad essi assegna. Per esempio, si risparmierebbe tempo se chi fa il caffè fosse in grado di contare e il capo potesse dirgli quante tazze preparare. Sarebbe anche utile se il capo potesse dire all'archivista quale documento prelevare dall'archivio. Gli assistenti sarebbero sempre limitati ad un lavoro ma potrebbero eseguirlo in un modo più flessibile.

Nello stesso modo generale, una subroutine in un programma diventa molto più utile se lo si può chiedere di fare uno qualsiasi di un'intera famiglia di compiti collegati. Per esempio, potrebbe essere comodo per un programma usare una subroutine che emetta qualsiasi numero di "pip" secondo le istruzioni pervenute dal programma principale.

Quest'idea solleva l'interessante questione della comunicazione. Ci si aspetterebbe che i messaggi che vengono scambiati tra il capo e i suoi nuovi versatili assistenti siano più complicati dato che egli deve ora indicare un numero o il titolo di un documento.

Gli assistenti che hanno l'incarico speciale di trovare l'informazione per il capo, possono trasmettergliela quando gli dicono "fatto". Nell'ufficio tutto ciò è abbastanza semplice in quanto c'è un sistema telefonico, ma cosa succede nei programmi?

TRASMISSIONE DI PARAMETRI ALLE/DALLE SUBROUTINE

Molti linguaggi di programmazione tipo PASCAL o ADA, dispongono di speciali meccanismi per la comunicazione tra il programma principale e le subroutine, ma il BASIC, fa le cose in modo ancor più semplice. Le informazioni sono passate in *variabili* che sono usate in comune tra il programma principale e le sue subroutine. Queste variabili hanno un nome speciale: *parametri*. Qualsiasi variabile si comporta come un parametro, ma noi adotteremo una convenzione speciale: ogni nome di parametro dovrà essere costituito da una lettera seguita dalla cifra 1, seguita dal segno $ se il parametro è una stringa. Ecco alcuni esempi:

```
A1        X1        C1$        G1
```

Qui c'è un esempio di una subroutine per visualizzare una riga con qualsiasi numero di ★, come segue:

```
★★★
```

o

```
★★★★★★★★★★★
```

```
3000 REM VISUALIZZA NUMERO DI ★
     DATI IN X1 SU UNA RIGA
3010 FOR JJ = 1 TO X1
3020 PRINT "★";
3030 NEXT JJ
3040 PRINT
3050 RETURN
```

Si esamini da vicino questa routine. I comandi da 3010 a 3030formano un'iterazione che è ripetuta X1 volte. Ogni volta, viene visualizzato un ★ sulla stessa riga degli ★ precedenti. X1 è il *parametro* o la variabile che dice alla subroutine quanti ★ occorrono. JJ è una variabile locale; cioè è usata all'interno della subroutine, ma il suo valore all'esterno della subroutine non ha alcun interesse. La subroutine non è di alcuna utilità a meno che non venga richiamata. Ciò richiede un paio di comandi: uno per definire X1 a un valore appropriato e uno per eseguire il richiamo effettivo. Per ottenere una riga di 17 ★ , il programma potrebbe comprendere:

```
X1=17
GOSUB 3000
```

Il valore di un parametro può essere impostato in parecchi modi, ad esempio dai comandi READ, INPUT o FOR nonchè da una semplice assegnazione. Per visualizzare un tronco di piramide, scriveremo:

```
10 FOR X1=1 TO 18
20 GOSUB 3000
30 NEXT X1
40 STOP
(seguito dalla subroutine che visualizza
gli ★).
```

Impostare questo programma (con la subroutine) e controllare che funzioni come previsto.

LE DECISIONI NEL PROGETTARE LE SUBROUTINE

Esaminiamo ora alcune delle decisioni prese mentre questo programma veniva scritto. Durante il disegno iniziale, il programmatore aveva scoperto che gli occorrevano subroutine per visualizzare un numero variabile di ★ su una riga. Egli aveva quindi deciso senza alcun motivo particolare di inserire la subroutine a 3000 e di usare X1 come parametro. In questa fase, avrebbe potuto ugualmente inserire la subroutine ovunque (ad esempio a 4500) e avrebbe potuto scegliere una diversa variabile (ad esempio N1) come parametro.

Una volta presa la decisione, comunque, le possibilità erano molto più ristrette. La subroutine ora doveva partire


```
3000REM .....
```

I richiami dovevano usare X1 come parametro ed essere scritti nella forma:

```
GOSUB 3000
```

Questa è una buona illustrazione di un argomento generale: quando si inizia a disegnare un programma, c'è ampia libertà per svolgere le cose in maniere diverse; ma una volta presa una decisione, la libertà si riduce sempre più fino a che al termine risulta esserci una sola via percorribile.

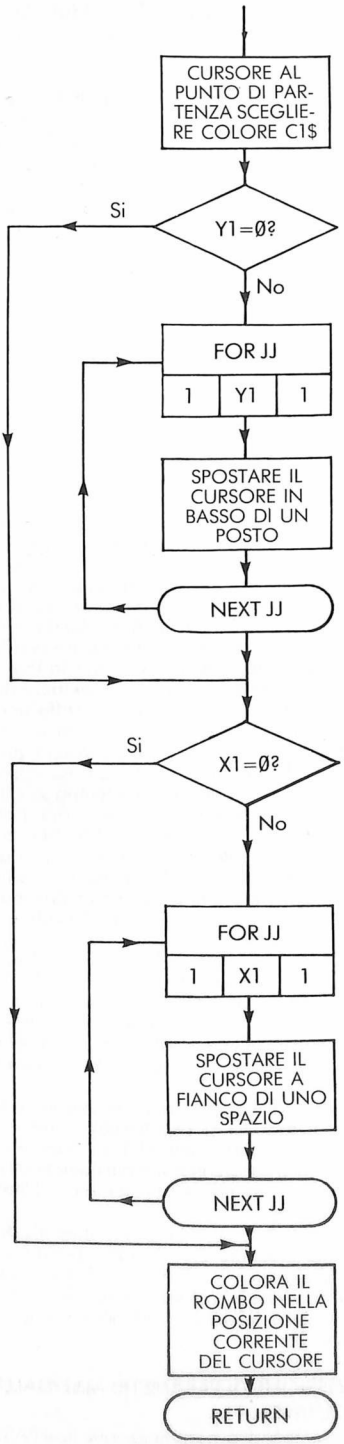
USO DI PIÙ DI UN PARAMETRO

Le subroutine non sono limitate ad un parametro, ma possono usare qualsiasi numero (ragionevole) di parametri. Qui per esempio c'è una subroutine che visualizza un rombo colorato in qualsiasi posizione sullo schermo. I parametri sono:

X1: Numero degli spostamenti a destra
Y1: Numero degli spostamenti in basso
C1$: Colore del rombo

Glossario

C1$: Colore del rombo
X1, Y1: Posizione del rombo
JJ: Usato per contare i movimenti del cursore



La corrispondente codifica è:

```
2000 REM VISUALIZZA ROMBO
     COLORATO C1$ IN X1 SPAZI
     IN ORIZZONTALE SULLO SCHERMO
     E IN Y1 SPAZI IN VERTICALE
2010 PRINT " CLR HOME ";C1$;
2020 IF Y1=0 THEN 2060
2030 FOR JJ=1 TO Y1
2040 PRINT " CRSR ↓ ";
2050 NEXT JJ
2060 IF X1 = 0 THEN 2100
2070 FOR JJ=1 TO X1
2080 PRINT " CRSR → ";
2090 NEXT JJ
2100 PRINT " CTRL e RVS ON
     CTRL e RVS OFF CRSR ↑
     SHIFT e CRSR ← SHIFT e CRSR ←
     ";
2110 RETURN
```

Questa subroutine usa i comandi del cursore in stringhe per spostare il cursore sullo schermo. La 2100 colora un rombo. Sembra spaventosa quando viene scritta completamente, ma la stringa comprende soltanto nove caratteri: Reverse on (negativo attivato), reverse off (negativo disattivato) ◤◢ e ◥◣ (due volte ciascuno) e tre movimenti del cursore per arrivare dalla fine della prima riga di grafici all'inizio della seconda. I caratteri nella stringa sono esattamente quelli che si userebbero se si dovesse disegnare un rombo direttamente dalla tastiera.
Si ricorderà che nel VIC BASIC i comandi FOR seguono l'iterazione almeno una volta anche se il valore finale è meno del valore iniziale. La prova per Y1=0 è inclusa in modo che l'iterazione FOR nelle righe 2030-2050 può essere saltata del tutto se necessario. La prova per X1=0 c'è per un motivo analogo.
Per provare la subroutine ecco un programma che riempie lo schermo con rombi verdi:

```
10 PRINT" SHIFT e CLR HOME ";
20 C1$= " CTRL e GRN "
30 FOR X1=1 TO 19 STEP 3
40 FOR Y1=0 TO 21 STEP 4
50 GOSUB 2000: REM DISEGNA
     ROMBO COLORATO –C$ IN X1, Y1
60 NEXT Y1
70 NEXT X1
80 GOTO 80: REM STOP ITERAZIONE
```

ESPERIMENTO 18·2

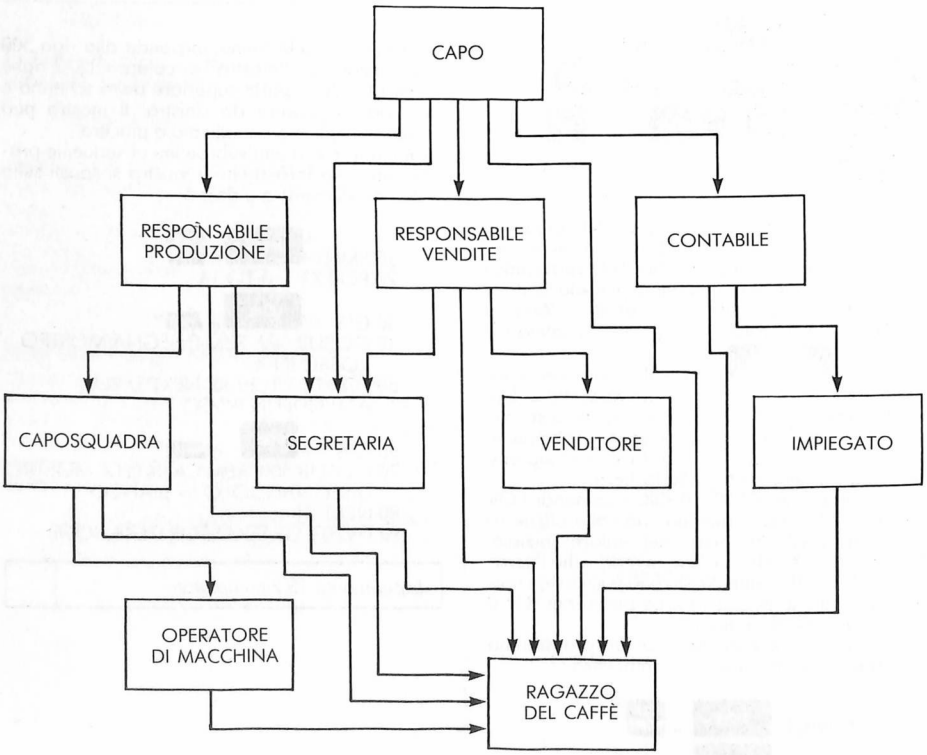
Scrivere una subroutine, iniziando alla riga 500 che disegna un "mostro" di colore C1$, 2 righe al disotto della parte superiore dello schermo e X1 spazi a partire da sinistra. Il mostro può essere semplice o complicato a piacere.
Aggiungere ora una subroutine al seguente programma che farà sì che il mostro si sposti sullo schermo da sinistra a destra:

```
10 PRINT" SHIFT e CLR HOME ";
20 FOR X1 = 0 TO 16
30 C1$ =" CTRL e RED "
40 GOSUB 500: REM DISEGNA MOSTRO
     ROSSO IN X1
50 FOR T=1 TO 150: NEXT T: REM
     ATTENDI UN POCO
60 C1$ =" CTRL e WHT "
70 GOSUB 500: REM CANCELLA MOSTRO
     DISEGNANDOLO IN BIANCO
80 NEXT X1
90 GOTO 90: REM STOP ITERAZIONE
```

Esperimento 18.2 completato	

SUBROUTINE PIÙ COMPLESSE

La maggior parte degli uffici non sono così semplici come quello che abbiamo descritto precedentemente in questa unità. Generalmente parlando, il capo è aiutato da parecchi "dirigenti", ciascuno dei quali può avere uno o più assistenti personali. Questi assistenti possono a loro volta avere propri aiutanti e così via giù giù lungo la catena di comando. Alcune persone possono eseguire il loro particolare lavoro per chiunque lo richieda; per esempio, il ragazzo del caffè deve servire chiunque. Il capo può chiedere il caffè per sè oppure può chiedere alla sua segretaria di chiederlo a suo nome. Per chiarire la struttura dei comandi, l'ufficio può avere un organigramma più o meno come questo:

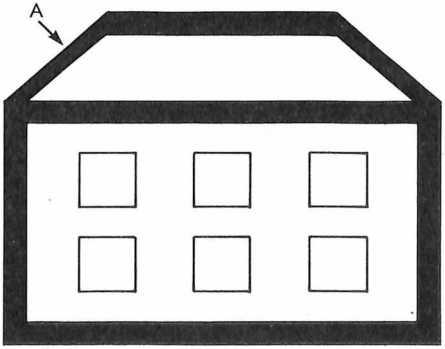


Questo tipo di struttura può essere riflesso in un programma BASIC. Una subroutine può essere richiamata dal programma principale o può essere richiamata da qualsiasi altra subroutine a qualsiasi "livello". Il computer non si confonde in quanto prevede arrangiamenti speciali per memorizzare tutti i concatenamenti che gli servono per ritornare al punto giusto nel programma principale. La catasta in cui i concatenamenti sono memorizzati, non è semplicemente un richiamo di memoria ma prevede una posizione separata per ciascun "livello" di richiamo.

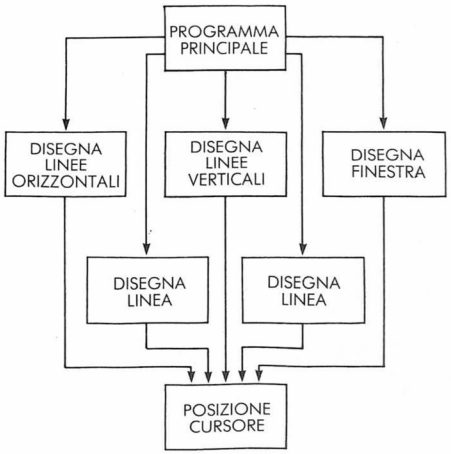
ESPERIMENTO 18·3

Caricare ed eseguire il programma denominato PICTURE dalla cassetta di nastro. Esso genera un disegno grezzo, ma riconoscibile di una casa. Il disegno è realizzato richiamando una serie di subroutine, una per ciascuna fila di finestre nell'immagine. Pertanto la subroutine alla riga 1000 disegna una riga orizzontale da sinistra a destra. La riga è lunga N1 unità e inizia alla locazione X1 spazi in orizzontale sullo schermo e Y1 righe verso il basso. Analogamente, la subroutine in 2000 disegna le righe verso il basso, la subroutine in 3000 diagonalmente verso l'alto da sinistra a destra e quella a 4000 diagonalmente verso il basso da sinistra a destra. Pertanto, per disegnare la linea contrassegnata A nell'immagine, la sequenza di richiamo è

X1=1: Y1=8: N1=5: GOSUB 3000



Listare ed esaminare il codice nelle varie subroutine. Esse iniziano tutte con un compito comune: inserire il cursore nella posizione indicata da X1. Dato che si tratta di un lavoro ben definito, è comodo trasformarla in una subroutine a sè. La "struttura di potere" del programma è pertanto:



Ora cancellare i comandi da 10 a 140 e usare le subroutine per disegnare un'immagine a scelta. Potrebbe trattarsi di un castello o di una fabbrica con una ciminiera. È inoltre possibile definire una nuova subroutine per disegnare finestre con archi e disegnare una chiesa.

Esperimento 18.3 completato

Le subroutine sono un argomento importante e se ne continuerà la discussione nella succesiva unità.

UNITA':19

ALTRO SULLE SUBROUTINE

In questa unità si continuerà lo studio delle subroutine. Il segreto per scrivere robusti e utili programmi e per farli lavorare rapidamente, è una raccolta di tecniche dette "software engineering". Queste tecniche non sono solitamente citate nei manuali introduttivi in quanto sono generalmente considerate strumenti per professionisti. Non vale quindi la pena di imparare a programmare scorrettamente quando è altrettanto facile farlo nella maniera corretta.

SPECIFICHE DELLE SUBROUTINE

Un'idea vitale nel software engineering è la *specifica delle subroutine*. Si tratta di una descrizione esatta di cioè che una subroutine fa e di come (e cioè attraverso quali parametri) essa comunica con il programma principale. La specifica della subroutine non dice nulla sul meccanismo interno della subroutine stessa o sul modo in cui essa svolge il compito che è predisposta per fare.

Le specifiche delle subroutine servono a due scopi diversi. Innanzitutto possono essere stampate in un catalogo di subroutine in modo che altri programmatori possano scegliere utili subroutine per i rispettivi programmi ed effettuare tutti gli arrangiamenti pratici per richiamarle. Secondo, una specifica di subroutine dà un saldo punto di partenza al programmatore che scrive la subroutine stessa. Egli può scriverla in qualsiasi modo a suo piacere purchè esegua esattamente ciò che la specifica dice. Notare l'ordine delle cose: la specifica prima del programma. La sola eccezione è che talune voci possono dover essere aggiunte alla specifica dopo che la subroutine è stata scritta.

Vediamo un esempio. Si supponga di scrivere un programma per aiutare i bambini ad eseguire le somme con le frazioni tipo "1/6 + 1/32" o "5/8 × 2/3". Se le somme sono generate casualmente, in qualche punto il programma dovrà elaborare le proprie risposte alle domande che esso pone. Si ricorderà che lavorando con le frazioni era possibile ottenere risposte nei termini inferiori. Ad esempio si poteva ottenere:

$$\text{"1/6 + 1/3 = }\frac{1+2}{6}\text{=3/6"o"5/8 × 2/3 = 10/24"}$$

Le frazioni "3/6" e "10/24" non sono sbagliate ma devono essere semplificate prima di poter essere accettate come completamente corrette.

Il lavoro per semplificare le frazioni è un compito autonomo, chiaramente adatto per farne una subroutine. Questa subroutine sarà diversa da quelle nell'unità 18 per una cosa molto importante: essa prenderà i parametri dal programma principale, eseguirà un calcolo e darà i risultati al *programma principale*, non visualizzerà cioè nulla sullo schermo (salvo eventualmente in caso di emergenza), nè richiederà alcun input dall'utente. Per quanto riguarda la subroutine, il programma principale è il mondo esterno.

Iniziamo identificando e denominando i parametri. Per fare questo lavoro la subroutine ha bisogno di una *coppia* di numeri (ad esempio 3 e 6 o 10 e 24) che sono rispettivamente il "numeratore" e il "denominatore" della frazione che si sta cercando di semplificare. Sceglieremo A1 e B1 per rappresentare i valori originali. A1 e B1 sono detti *parametri di input*, quantunque l'input avvenga nel programma principale e non (almeno direttamente), dall'utente.

Il risultato della subroutine è un'altra coppia di numeri tipo 1 e 2 o 5 e 12. Faremo in maniera che la subroutine ritorni a questi valori in C1 e D1. Come ci si potrebbe aspettare, C1 e D1 sono detti parametri di *output*, quantunque non venga eseguita alcuna stampa (PRINT) ad opera della subroutine.

Infine, decideremo a caso di inserire il primo comando della subroutine in 5500. Questo numero non entra in conflitto con qualsiasi altra subroutine nella nostra raccolta.

Possiamo ora scrivere una specifica formale:

Specifica di subroutine provvisoria

Scopo: Semplificare le frazioni ai loro minimi termini

Numeri di riga: da 5500 a ☐

Parametri:
Input A1 (numeratore frazione)
B1 (denominatore frazione)
Output C1 (numerat. fraz. semplificata)
D1 (denomin. fraz. semplificata)

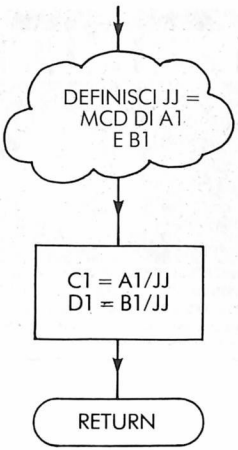
Variabili locali: ☐

Notare che la specifica è provvisoria e contiene due caselle vuote. Una è prevista per l'ultima riga della subroutine e l'altra è intesa per una lista delle variabili usate dalla subroutine stessa. Queste caselle non possono essere riempite fino a che la subroutine non è stata scritta.

SUBROUTINE PER SEMPLIFICARE LE FRAZIONI

Ora torniamo alla subroutine. Per semplificare una frazione occorre per prima cosa trovare il cosiddetto "massimo comun divisore" e quindi dividere per esso numeratore e denominatore. Per esempio, il MCD di 10, 24 è 2 e 10/24 in termini ridotti è pertanto 5/12.

Questo processo viene facilmente riportato su uno schema di flusso. Non conosciamo ancora come funziona l'MCD dei due numeri cosicchè useremo una nuvoletta:



Glossario

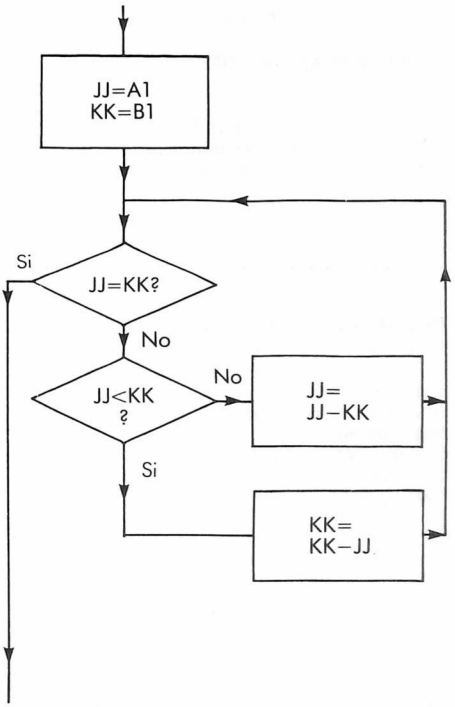
A1/B1: Frazione da ridurre ai minimi termini
C1/D1: Risultato

Un modo semplice per trovare l'MCD di due numeri è detto algoritmo di Euclide. Iniziando con i due numeri sottrarre il più piccolo dal più grande fino che i due sono identici: questo valore è il cosiddetto MCD. Iniziando con 10 e 24 si ottiene:

24	10	
		Togliere 10 da 24
14	10	
		Togliere 10 da 14
4	10	
		Togliere 4 da 10
4	6	
		Togliere 4 da 6
4	2	
		Togliere 2 da 4
2	2	
		2=2, cosicchè l'MCD di 10 e 24 è 2

Questo processo può essere riportato su uno schema di flusso come segue:

Nota: Usiamo variabili locali JJ e KK in modo da non danneggiare i valori di A1 e di B1.



Glossario

A1, B1: numeri di cui si vuole il massimo comun divisore
JJ,KK: variabili locali (risultato in JJ)

La subroutine potrebbe apparire come segue:

```
5500 REM RIDURRE FRAZIONE A1/B1 AI
     SUOI MINIMI TERMINI
5510 REM RISULTATO IN C1/D1 LOCALI
     =JJ,KK
5520 JJ=A1 : KK=B1
5530 IE JJ = KK THEN 5560
5540 IF JJ < KK THEN KK=KK−JJ: GOTO
     5530
5550 JJ=JJ−KK : GOTO 5530
5560C1=A1/JJ : D1=B1/JJ
5570 RETURN
```

Possono ora essere inseriti gli ultimi dettagli rimanenti della specifica.

Specifica della subroutine

Scopo: Semplificare le frazioni ai loro minimi termini

Numeri di riga: da 5500 a 5570

Parametri:
Input A1 (numeratore frazione)
B1 (denominatore frazione)
Output C1 (numerat. fraz. semplificata)
D1 (denomin. fraz. semplificata)

Variabili locali: JJ, KK

PROGRAMMA DI GESTIONE

Per controllare la subroutine abbiamo bisogno di un programma "di gestione". Questo è il "programma principale" più semplice che possiamo costruire che prova la subroutine in ogni modo ragionevole.
La specifica della subroutine risulta estremamente utile per scrivere il programma di gestione. Essa ci dice:
(a) di inserire i parametri di input in A1 e B1
(b) di chiamare la subroutine in 5500
(c) di osservare i risultati in C1 e D1
(d) di evitare di usare le righe da 5500 a 5570 per qualsiasi altro scopo
(f) di non usare JJ e KK nel programma principale.

In generale, se una specifica non comprende le informazioni che occorrono per scrivere un programma di gestione, la specifica è incompleta.
Un adatto programma di gestione è il seguente:

```
10 INPUT "FRAZIONE"; A1,B1
20 GOSUB 5500
30 PRINT "RISULTATO="; C1;"/"; D1
40 GOTO 10
```

Alcune prove producono quanto segue

```
RUN
FRAZIONE? 33,67
RISULTATO = 33/67
FRAZIONE? 33, 69
RISULTATO = 11/23
FRAZIONE? 12345, 23456
RISULTATO = 12345/23456
FRAZIONE? 10, 24
RISULTATO = 5/12
FRAZIONE? 3, 6
RISULTATO = 1/2
. . . .
```

Questi risultati sembrerebbero esatti e per il momento accetteremo la subroutine come corretta.

ESPERIMENTO 19·1

Scrivere un programma che consenta all'utente di battere due frazioni (ad esempio p/q e s/t) e di visualizzare il risultato semplificato della loro somma. Per esempio:

```
PRIMA FRAZIONE? 3, 8          (significa 3/8)
SECONDA FRAZIONE? 5,12        (significa 5/12)
SOMMA = 19/24
```

SUGGERIMENTO: $p/q + s/t = \dfrac{p\star t + q\star s}{q\star t}$

Porre A1= P★T+Q★S,B1=Q★T e quindi richiamare la subroutine di semplificazione.

Esperimento 19.1 completato	

Robustezza delle subroutine

È interessante confrontare i programmi scritti da professionisti con quelli prodotti da amatori inesperti. Un programma di un professionista ha due caratteristiche essenziali:

(a) È *robusto*. Esso non dà mai risposte sbagliate e non "si rompe" mai visualizzando errori non previsti o bloccandosi in un'iterazione se l'utente fornisce l'informazione scorretta.

(b) È *adattabile*. Il programma è costruito e documentato con schemi di flusso, glossari, specifiche di subroutine e commenti (REM) in modo che qualsiasi programmatore competente possa facilmente modificarlo per adattarlo ad una nuova esigenza.

Per contro, il programma di un dilettante è *fragile*. Esso solitamente funziona finchè viene usato da colui che lo scrive, pronto ad intervenire se viene immesso qualsiasi input scorretto. Esso non è per nulla documentato e probabilmente non ha alcuna struttura discernibile. Pochi mesi dopo averlo scritto il programmatore dimentica come funziona e nessuno riesce a capirci più nulla. In queste condizioni, la maggior parte dei tentativi volti a correggere eventuali errori o migliorare il programma non fa che peggiorare le cose. La robustezza di una catena si misura dal suo anello più debole.

Allo stesso modo un programma è affidabile nella misura in cui lo è la subroutine meno affidabile. Uno dei concetti base del software engineering è che ogni subroutine dovrebbe essere *perfetta*, o almeno tanto perfetta quanto è possibile farla. Dovrebbe essere *impossibile* per una subroutine fare qualcosa di sbagliato senza almeno darne una segnalazione.

LIMITI DEL CAMPO DI UN PARAMETRO

A pagina 174 c'era una semplice subroutine con un unico parametro X1, che visualizzava un numero di asterischi su una riga. In quell'unità si assumeva con ottimismo che tutto fosse corretto; ma esaminiamolo ora più da vicino. Eccolo qui di nuovo unitamente ad un programma di gestione:

```
  10 UNPUT"NUMERO ASTERISCHI"; X1
  20 GOSUB 3000
  30 GOTO 10
3000 REM VISUALIZZA NUMERO ★
       DATI IN X1 SU UNA RIGA
3010 FOR JJ = 1 TO X1
3020 PRINT "★";
3030 NEXT JJ
3040 PRINT
3050 RETURN
```

Innanzitutto notiamo che la variabile locale JJ non è citata nel commento (REM) alla riga 3000 ed accettiamo ciò come un errore piccolo ma genuino. Ora iniziamo la prova. La subroutine sembra funzionare bene per X1=1,3,6 e così via. Con fiducia proviamo altri numeri:

21: Funziona correttamente.

22: Dà 22 asterischi e una riga vuota in più! Se si usasse la subroutine per disegnare un grafico, ciò distruggerebbe il suo aspetto.

23: Ciò non *dà* una riga con 23 asterischi; visualizza una riga di 22 e una seconda riga con un asterisco.

0: Ci si aspetta una riga vuota; per contro si ottiene una riga con un asterisco.

−3: Questo è un valore privo di significato cosicchè ci si dovrebbe aspettare che la subroutine dia un avvertimento del tipo "cosa significa?". Per contro dà una riga con un asterisco, esattamente come se si fosse impostato X1=1.

Sta diventando penosamente chiaro che il programma non è conforme alla sua specifica. Abbiamo bisogno di qualche modifica.

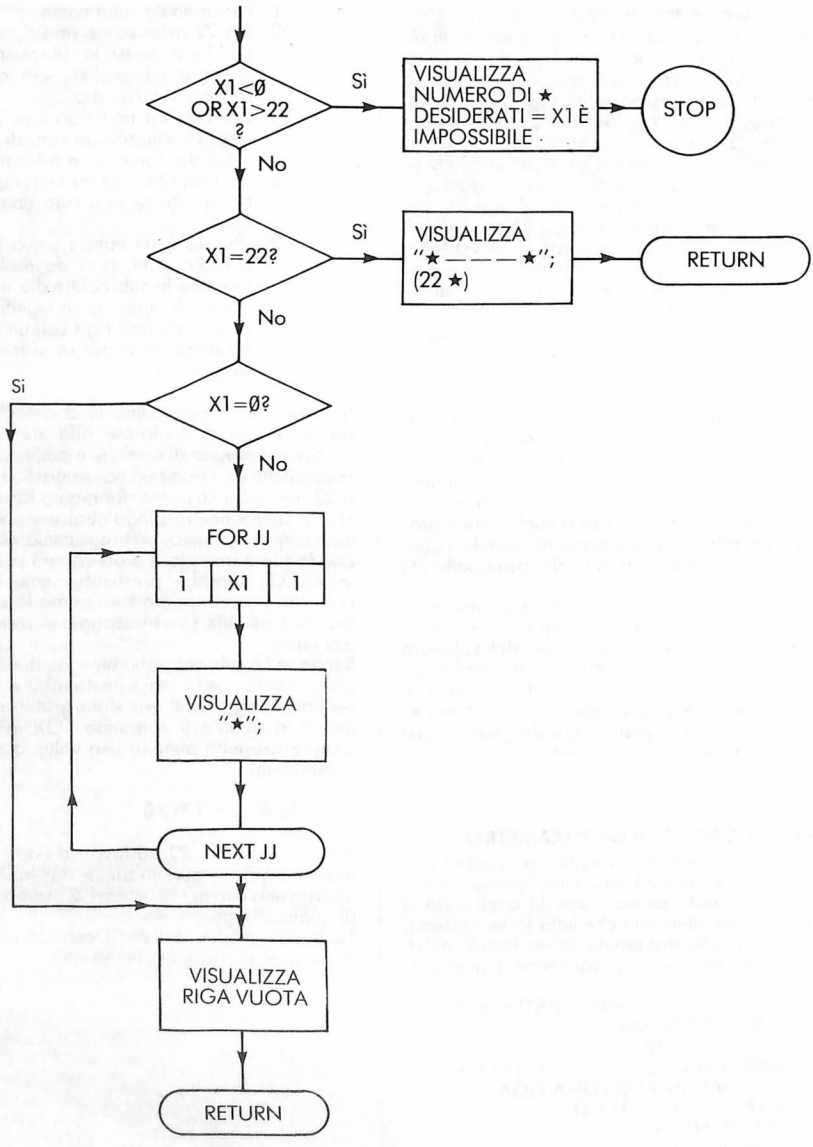
Innanzitutto una riga può contenere soltanto tra 0 e 22 asterischi cosicchè dobbiamo fare in modo che la subroutine *respinga* qualsiasi valore aldifuori di questo campo. Un programma richiamante che fornisse un valore fuori campo per il parametro X1, sarebbe presumibilmente in errore cosicchè è appropriato fare in modo che la subroutine visualizzi un messaggio di avvertimento e si fermi.

Secondo, la subroutine ha bisogno di predisposizioni speciali per i valori estremi 0 e 22. Nella versione originale 0 era stato gestito scorrettamente in quanto il comando FOR in BASIC è sempre eseguito almeno una volta, *anche* per i comandi tipo

```
FOR JJ = 1 TO 0
```

All'altro estremo, 22 portava ad una riga extra indesiderata, in quanto viene *automaticamente* inserita una nuova riga dopo il 22esimo carattere di qualsiasi riga.

Tenendo nota di questi punti, ecco uno schema di flusso e un programma revisionati:



Glossario

X1: Il numero di asterischi da visualizzare
JJ: Contatore asterischi

```
3000 REM VISUALIZZA ASTERISCHI DATI
     IN X1 SU UNA RIGA. RANGE 0-22.
     VARIABILE = JJ
3010 IF X1 <0 OR X1 >22 THEN PRINT
     NUMERO ★ DESIDERATI="; X1:
     PRINT "IMPOSSIBILE": STOP
3020 IF X1 = 22 THEN PRINT
     "★★★★★★★★★★★★★★★★★★★★
     ★★";:PRINT:RETURN
3030 IF X1 = 0 THEN 3070
3040 FOR JJ = 1 TO X1
3050 PRINT "★";
3060 NEXT JJ
3070 PRINT: PRINT
3080 RETURN
```

Questo illustra tre fatti importanti:

(a) Dove i parametri di una subroutine possono assumere soltanto un range limitato di valori, il buon software engineering richiede che la subroutine debba controllare che ogni valore sia entro il range e segnalare qualsiasi discordanza.

(b) Quando una subroutine viene testata, è particolarmente importante provare i valori estremi ammessi (ad esempio 0 e 22) dato che è qui dove spesso si annidano gli errori.

(c) Le subroutine che sono correttamente strutturate e funzionano sicuramente in tutti i casi, sono solitamente più lunghe delle loro controparti più semplici.

ESPERIMENTO 19·2

(a) Il programma che segue si propone di visualizzare le registrazioni di 11 giocatori di cricket nella forma di un "istogramma" o grafico a barre, dove ciascuna fila corrisponde ad un giocatore e ciascuna stella ad una porta. Eseguire il programma con la vecchia e la nuova versione della subroutine iniziando alla riga 3000 e osservare la differenza:

```
 10 REM ISTOGRAMMA BATTUTE
 20 PRINT" SHIFT e CLR HOME "
 30 FOR J = 1 TO 11
 40 READ X1: GOSUB 3000
 50 NEXT J
 60 STOP
100 DATA 3,0,15,22,5,0,4,1,0,22,5
```

(b) Tornare alla subroutine indicata a pagina 182 e provarla di nuovo più a fondo. Cosa succede se A1 o B1 sono 0 o negativi (ad esempio −5)?
A1 o B1 sono decimali (ad esempio 3.143)?

Queste prove convinceranno (se non lo si sapeva già) che l'algoritmo di Euclide funziona soltanto per numeri interi positivi.

Disegnare una versione correttamente strutturata della subroutine ricordando che:

(a) Non è sensato per A1 e B1 avere valori frazionari (quantunque ciò potrà accadere). Se un numero X è un *intero,* l'espressione X=INT(X) è vera.

(b) Non è sensato per B1 avere qualsiasi valore minore di 1.

(c) È sensato che A1 sia 0 o un numero negativo; ciò potrebbe avvenire durante la sottrazione di due frazioni. Se A1=0, il valore del risultato dovrebbe essere 0/1, indipendentemente da B1. Se A1 è negativo, la subroutine dovrebbe ricordare il fatto, usare un numero positivo nell'algoritmo di Euclide e cambiare il segno di C1 immediatamente prima che il risultato venga fornito.

Esperimento 19.2 completato	

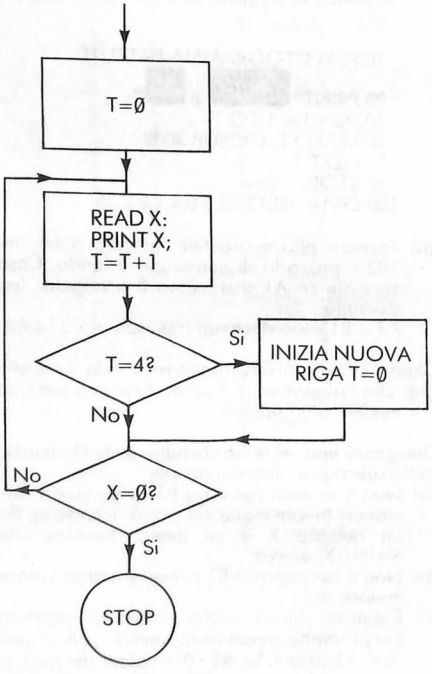
CONVENZIONI DI DENOMINAZIONI NELLE SUBROUTINE

Nelle discussioni sulle subroutine abbiamo introdotto una convenzione di denominazione: A1, B1, ...Z1 per i parametri e AA...ZZ per le variabili locali. Non ci sono regole fisse su questi nomi in BASIC e si troveranno numerosi programmi che non osservano la convenzione, ma vale la pena di rispettarla in quanto protegge contro alcuni degli errori più sottili che possono verificarsi nei grandi programmi.

In pratica è abbastanza comune che i programmi non riescano a funzionare in quanto il valore di qualche variabile importante è stato danneggiato da una subroutine. Ecco un semplicissimo esempio.

Si consideri un programma che abbia una lista di numeri nei suoi comandi di dati. Si suppone ne legga e ne visualizzi quattro per riga. L'ultimo numero, che agisce da terminatore, è 0.

Per organizzare la struttura si usa una variabile T per contare la quantità di numeri già su una riga. Ogni volta che viene visualizzato un numero, aumentiamo T di 1. Quando esso raggiunge 4, iniziamo una nuova riga e riportiamo T a 0. Lo schema di flusso generale e il programma sono:



Glossario

X: Numero corrente
T: Numeri di elementi sulla riga corrente

```
10 T=0
20 READ X
30 PRINT X;
40 T=T+1
50 IF T=4 THEN PRINT:T=0
60 IF X <> 0 THEN 20
70 STOP
80 DATA 15,23,40,11,37,51,99
90 DATA 33,12,89,53,17,20,0
```

Se si imposta questo programma, si troverà che funziona perfettamente.

Si supponga ora che un anno dopo, quando non si ricordano più i dettagli del programma, si decida di fare una modifica: si vuole che il computer emetta un segnale acustico ogniqualvolta visualizza un nuovo numero. Nel catalogo della subroutine si trova:

Specifica della subroutine

Scopo: Creare un segnale acustico seguito da mezzo secondo di silenzio

Righe: 2000-2050

Parametri: Nessuno

Questa subroutine sembra interamente adatta. Si aggiunga il testo al programma:

```
2000 REM CREA UN PIP
2010 POKE 36878,15: POKE 36876,245
2020 FOR T=1 TO 100: NEXT T
2030 POKE 36878,0: POKE 36876,0
2040 FOR T=1 TO 500: NEXT T
2050 RETURN
```

e si inserisca un nuovo comando:

```
25  GOSUB 2000
```

Sfortunatamente il programma non funziona più; i segnali acustici si verificano come si prevedeva ma lo schema è andato tutto all'aria. Il motivo è che la variabile di controllo dello schema T è stata modificata dalla subroutine. Ciò non sarebbe accaduto se il creatore della subroutine avesse seguito almeno in parte le convenzioni. Se avesse chiamato la sua variabile locale TT (invece di T), non la si sarebbe usata nel programma principale; oppure se avesse citato 'T' come variabile locale nella specifica di subroutine, ci sarebbe stato l'avvertimento.

In questo esempio, il fatto che il programma modificato fosse difettoso, era immediatamente ovvio. Talvolta l'errore è così facile da scoprire; esso semplicemente porta a delle risposte sbagliate. Si consideri questo programma che immette una serie di numeri e ne visualizza il totale.

```
10 INPUT "QUANTI NUMERI"; N
20 T=0
30 FOR J=1 TO N
40 INPUT X
50 T=T+X
60 NEXT J
70 PRINT "TOTALI"; T
80 STOP
```

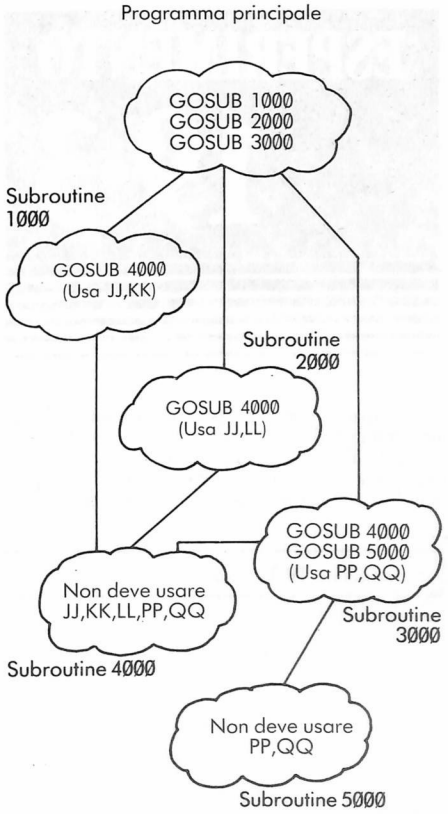
Glossario

N: Numero dei numeri
T: Totale mobile
X: Numero successivo da leggere
J: Conteggio dell'immissione di numeri

Questo funziona bene. Si supponga ora che il programmatore decida di migliorare il tutto facendo in modo che il programma emetta un segnale acustico ogni volta che accetta un numero. Egli aggiunge la subroutine del catalogo e la suddivide in due comandi extra:

```
15 GOSUB 2000
e 45 GOSUB 2000
```

Il programma è ora molto più soddisfacente da usare: esso suona come un moderno registratore di cassa. Sfortunatamente ora asserisce che sommando 247,37,12,93,52 e 39 si ottiene 540. Questa risposta sembra ragionevole ma effettivamente è sbagliata! Se non si notasse l'errore e si usasse il programma nella propria azienda, si potrebbe terminare con un "fallimento assistito dal computer". L'errore è abbastanza facile da notare una volta che si sa dov'è, ma il fatto triste è che ci sono numerosi errori analoghi nascosti ovunque nei programmi, completamente insospettati fino a che non fanno crollare ponti, non fanno morire i pazienti e non fanno precipitare i razzi in mare.
Se si scelgono le convenzioni di denominazione, è possibile solitamente evitare questo tipo di errore. Il programma principale non deve *mai* usare variabili "con la doppia lettera" che sono riservate per le variabili locali nelle subroutine e si deve usare soltanto la forma "lettera −1" tipo A1 o B1 per i parametri.
Se il programma usa più di una subroutine, occorre fare in modo che esse si adattino l'una all'altra. Chiaramente, tutte le subroutine devono usare diversi numeri di riga e se necessario occorrerà variarne di conseguenza uno o più.
Quando due o più subroutine vengono richiamate "allo stesso livello" (ad esempio potrebbero essere entrambe richiamate nel programma principale), esse possono usare sicuramente gli stessi parametri e le variabili locali. Se le subroutine sono a livelli diversi e una di esse "richiama" l'altra, devono usare variabili locali e parametri diversi. Tutto ciò è reso chiaro dal seguente diagramma:



Le unità che seguono faranno ampio uso delle subroutine di tutti i tipi. Munirsi quindi di un quaderno a fogli mobili e iniziare la propria libreria di subroutine. Ciascuna voce dovrebbe avere quattro elementi di documentazione:

Specifica
Schema di flusso
Glossario
Testo origine (e cioè il programma stesso)

Se si dispone del sistema base VIC, occorre battere manualmente nei programmi il testo delle subroutine. Sistemi più avanzati consentono di conservare la subroutine su una cassetta di nastro o su un floppy disk e di copiarla automaticamente.

ESPERIMENTO 19·3

Disegnare, scrivere e documentare una subroutine che prende tre numeri come parametri e fornisce il valore del maggiore come risultato. Scrivere una adatto programma di gestione e provare la subroutine a fondo il più possibile.

Esperimento 19.3 completato	

ESPERIMENTO 19·4

Il file denominato "BIGLETTERS" sulla cassetta di nastro è una subroutine che consente all'utente di battere una lettera o un carattere e di visualizzarlo quattro volte più grande.
Qui di seguito sono indicate le specifiche dell'intera subroutine. Studiarla e scrivere un programma che crei una riga di testo a caratteri cubitali.

Specifiche della subroutine

Scopo: Visualizzare i caratteri VIC quattro volte più grandi

Numeri di riga: da 8000 a 8200

Parametri: Input: La subroutine deve essere richiamata una volta per ciascun carattere. Il carattere da visualizzare deve essere fornito sotto forma di stringa di 1 carattere di nome A1$.
Output: A1$ (convertito in quattro volte il suo corpo normale).

Variabili locali: AA,BB,JJ,KK,LL,MM,NN,QQ

Note: (i) QQ non deve essere usato aldifuori della subroutine per qualsiasi motivo.
Note: (ii) La subroutine gestisce tutti i caratteri stampabili nelle serie "non shiftata" "shiftata" e "Commodore". Essa accetta anche e interpreta BLK, WHT, READ, CYN, PUR, GRN, BLU, YEL, RVS ON, RVS OFF, CLR HOME e RETURN. Altri tasti tipo DEL e i tasti di controllo del cursore sono ignorati.

Esperimento 19.4 completato	

UNITA': 20

MATRICI

Una tipica classe scolastica potrebbe essere composta dai seguenti nomi:

ADAMI
BONINI
CARLETTI
FINETTI
MAGNANI
MONETTI
SILVESTRI
TOMMASI
VALENTI
ZILIOLI

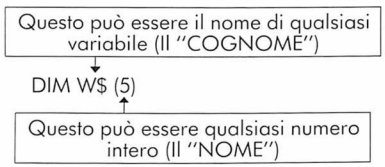
I fortunati sono quelli che hanno il cognome che inizia con una delle prime lettere dell'alfabeto. Ciò significa che sono i primi ai quali viene offerta la scelta di un tavolo, ecc. e non devono aspettare a lungo ad essere interrogati. Al povero Zilioli non viene data alcuna scelta ed è sempre l'ultimo in ogni coda. Talvolta, l'insegnante inizia la lettura dei nomi alla rovescia e Zilioli sarà il primo, ma ciò non risolve il problema di Zilioli.
Pensiamo di scrivere un programma per capovolgere una lista di nomi. Si vuole che l'utente batta l'elenco dei nomi di una classe, un nome per ogni riga e quindi lo legga alla rovescia dallo schermo in ordine invertito. Così a prima vista non sembrerebbe un compito molto difficile rispetto a quelli che sono stati già programmati e tuttavia la soluzione è elusiva. Il meglio che si può fare è di trovare in anticipo quanti nomi ci saranno e quindi scrivere un lungo e complicato programma usando una diversa variabile per ciascun nome. Per esempio, se ci sono quattro nomi nella lista, scegliamo le variabili A$, B$, C$ e D$ e scriviamo quanto segue:

```
 10 PRINT "IMMETTI NOMI ALLIEVI"
 20 INPUT A$
 30 INPUT B$
 40 INPUT C$
 50 INPUT D$
 60 PRINT "ORDINE INVERTITO="
 70 PRINT D$
 80 PRINT C$
 90 PRINT B$
100 PRINT A$
110 STOP
```

Questo programma darà una lista esattamente di quattro nomi, ma non può essere adattato per funzionare con qualsiasi altro numero di nomi, senza aggiungere (o cancellare) alcun comando. Se la classe avesse — ad esempio — 30 allievi, occorrerebbe scrivere un programma speciale con 30 variabili e 63 comandi. La scrittura del programma sarebbe come un castigo scolastico e converrebbe scrivere l'elenco manualmente.
Fortunatamente il BASIC ha un importante meccanismo che aiuta a superare questa difficoltà: si tratta della funzione *matrice*.

LE ISTRUZIONI DI DIMENSIONE

Una matrice è una *famiglia* di variabili che usa in comune il "cognome" ma che ha singoli "nomi" detti indici. Se si vuole usare tale famiglia, lo si dice normalmente al computer in uno speciale comando detto istruzione "DIM" (dimensione) perciò:



Ciò dice al VIC di accantonare spazio per una famiglia di variabili stringa denominata W$. Ci sono sei di queste variabili e i loro nomi completi sono:

W$(0)
W$(1)
W$(2)
W$(3)
W$(4)
W$(5)

L'indice è il numero tra parentesi che segue il nome della famiglia della matrice. La prima variabile ha un indice di 0, cosicchè il numero di variabile nella famiglia è sempre *uno in più* del numero nell'istruzione DIM. È talvolta comodo dimenticarsi della presenza della variabile con indice 0 e usare soltanto le variabili che hanno indici che iniziano con 1.

USO DELLE VARIABILI MATRICE

Nella maggior parte dei casi i membri di una famiglia di variabili sono come le normali variabili. È possibile includerli nelle espressioni, stamparli, leggerli e assegnare loro dei valori. Se il nome della famiglia termina con $, ciascun membro può contenere una stringa; altrimenti ciascun membro contiene un numero.
Per illustrare questi punti ecco alcuni comandi BASIC leciti. Sia chiaro che essi non intendono formare un programma sensato!

```
DIM N(20):REM DICHIARA UNA MATRICE
     DI 21 ELEMENTI DA N(0) a N(20)
N(5) = N(3) + 5
PRINT (N1); N(2); N(3)
INPUT N(2)
IF N(12) = N(17) THEN 150
```

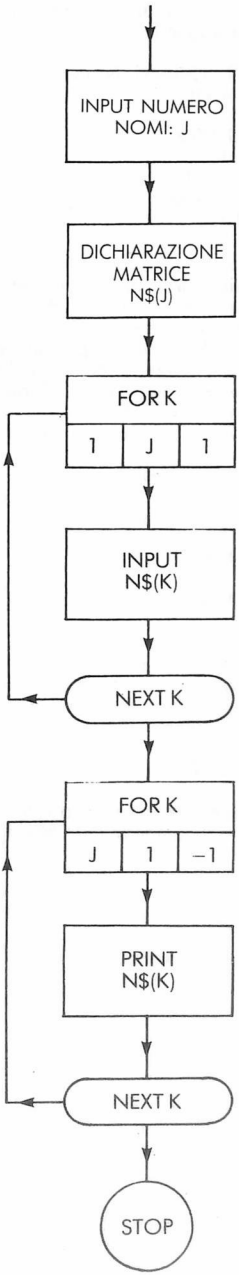
Notare che una cosa che non è possibile fare è di usare un membro di una famiglia, un "elemento di matrice" come è spesso chiamato, come variabile controllata in un comando FOR.

```
FOR N(5) = 1 TO 17
...                    } non è ammesso
NEXT N(5)
```

Finora non abbiamo detto nulla che sembra aiutarci con il problema di invertire l'elenco dei nomi. Ecco il punto chiave:

> L'indice di un elemento di matrice può essere un'espressione che viene elaborata quando il programma viene eseguito.

Si pensi a qust'idea per un momento e si veda se è possibile trarre alcune implicazioni prima di leggere. Si consideri un comando che fa parte di un'iterazione per cui viene eseguito parecchie volte ripetutamente. Se il comando include un riferimento ad un elemento di matrice, è possibile scegliere un'espressione indice in modo che venga usato un elemento diverso ogni volta che viene eseguita l'iterazione!
È possibile ora scrivere una soluzione molto più soddisfacente al problema originale. Il solo requisito extra è di chiedere all'utente di iniziare dando il numero dei nomi della lista.



Glossario

J: Numero dei nomi
Da N$(1) a N$(N): Lista dei nomi
K: Nome del contatore e indice a N$

Le corrispondenti istruzioni possono essere scritte nel modo seguente:

```
10 INPUT "QUANTI NOMI"; J
20 DIM N$(J)
30 FOR K = 1 TO J
40 INPUT N$(K)
50 NEXT K
60 FOR K = J TO 1 STEP -1
70 PRINT N$(K)
80 NEXT K
90 STOP
```

Questo semplice programma ha ottenuto la generalizzazione che era assente dai precedenti tentativi. Esso funzionerà per qualsiasi numero di nomi da 1 in su. Provare a impostarlo e a eseguirlo. Notare che la matrice N$ non è dichiarata fino a che il programma non "sa" quanti elementi deve avere. Quindi (dimenticandosi di N$(0)), gli viene attribuito esattamente il numero corretto di elementi.

Una cosa che non si deve mai fare è dichiarare la stessa matrice più di una volta. Occorre evitare cioè sequenze del tipo:

```
30 DIM A(50)
...
70 DIM A(50)
```

ma ci si deve inoltre assicurare di non inserire la dichiarazione di matrice all'interno di un'iterazione. Per esempio, se si tentasse di trasformare il semplice programma per invertire la lista in un'iterazione, inserendo

```
90 GOTO 10
```

si otterrebbe un errore la seconda volta che esso tenta di eseguire il comando DIM alla riga 20.

ESPERIMENTO 20·1

Si immagini che la dimensione della classe sia troppo grande e che di conseguenza l'insegnante non possa contare il numero correttamente. Scrivere una versione del programma di inversione della lista, che cerchi lo speciale terminatore "ZZZZ" all'estremità, invece di chiedere il numero all'inizio. Per esempio, se l'input fosse

```
CRANACH
DURER
MICHELANGELO
TURNER
ZZZZ
```

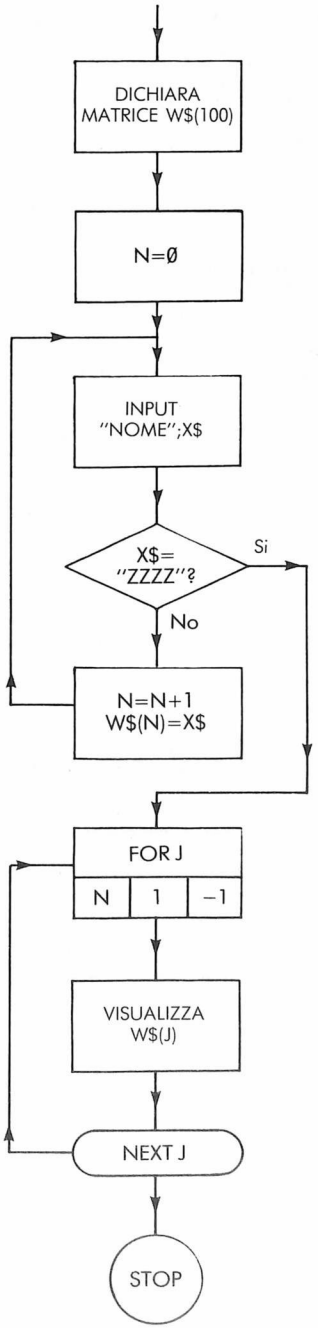
l'output sarebbe

```
TURNER
MICHELANGELO
DURER
CRANACH
```

Suggerimenti: usare il seguente schema di flusso:

Glossario

W$(100): Matrice per i nomi (massimo 100)
N: Contatore dei nomi
X$: Nome corrente
J: Indice di W$



Esperimento 20.1 completato

ULTERIORI USI DELLE MATRICI

Come si è visto da questo esempio, uno dei principali vantaggi della matrice è la possibilità di avere una tabella di stringhe (o di numeri) e di fare riferimento ai suoi elementi in qualsiasi momento e in qualsiasi ordine. Ciò è spesso utile. S'immagini di scrivere un programma che debba visualizzare i suoi risultati in parole (ad esempio "OTTO" o "DICIASSETTE") anzichè in cifre tipo 8 o 17. Si supporrà che tutti i risultati siano noti e siano compresi tra Ø e 2Ø. È possibile impostare una tabella — la si chiamerà T$ — per tradurre le cifre in parole. Si dispone in modo che ciascun elemento contenga il nome del proprio indice, cosicchè T$(Ø) = "ZERO", T$(1) = "UNO" e così via fino a T$(2Ø) = "VENTI". Quindi per visualizzare qualsiasi numero X, basta immettere

```
PRINT T$(X)
```

Per esempio se X=8, il comando visualizza T$(8) che è la stringa "OTTO".
Naturalmente occorre fare un certo lavoro all'inizio del programma per ottenere l'impostazione della tabella. È possibile sempre scrivere una lunga lista di 21 comandi tipo:

```
T$(Ø)="ZERO"
T$(1)="UNO"
T$(2)="DUE"
...
T$(2Ø)="VENTI"
```

ma è molto meno fastidioso inserire i nomi dei numeri in un'istruzione DATA e leggerli (READ) con un'iterazione FOR. Il programma dovrebbe iniziare come segue:

```
1Ø DIM T$(2Ø)
2Ø FOR J=Ø TO 2Ø
3Ø READ T$(J)
4Ø NEXT J
5Ø DATA ZERO,UNO,DUE,TRE,QUATTRO
   CINQUE
6Ø DATA SEI,SETTE,OTTO,NOVE,DIECI
7Ø DATA UNDICI,DODICI,TREDICI,
   QUATTORDICI

8Ø DATA QUINDICI,SEDICI,DICIASSETTE
9Ø DATA DICIOTTO,DICIANNOVE,VENTI
```

Useremo ora la matrice T$ per visualizzare una tabellina per moltiplicazioni in parole. Un semplice programma che forma il punto d'inizio del nostro disegno è:

```
1ØØ FOR J=Ø TO 1Ø
11Ø PRINT "2★";J;"=";2★J
12Ø NEXT J
```

Ora modifichiamo il comando PRINT facendogli visualizzare l'appropriata entrata di tabella invece di ciascun numero.

```
"2"     diventa T$(2)
"J"     diventa T$(J)
"2★J"   diventa T$(2★J)
```

Si ottiene

```
100 FOR J= 0 TO 10
110 PRINT T$(2);"★"; T$(J);"=";
    T$(2★J)
120 NEXT J
```

Se si impostano queste istruzioni dopo quelle contrassegnate da 10 a 90, si può provare il programma.
Una proprietà base di una matrice è che se si conosce l'indice di un elemento, è possibile scegliere l'elemento ed estrarlo immediatamente. Talvolta è possibile procedere in un altro modo: si sa il valore dell'elemento e si vuole trovare dove lo si incontra (se mai) nella matrice. Quest'operazione è più difficile dato che occorre far sì che il computer cerchi nella matrice, entrata per entrata, fino a che non trova quella che corrisponde all'elemento o al limite non raggiunge la fine della matrice.
Facciamo un semplice esempio. Si vuole scrivere un programma che immette due numeri, li somma e visualizza la loro somma, ma comunica con l'utente interamente con *parole*. Per esempio, un tipico dialogo potrebbe essere

```
DATI DUE NUMERI
?OTTO,CINQUE
LA SOMMA È TREDICI
```

Entrambe le parole devono essere convertite in numeri prima che vengano sommate. La conversione da parole a numeri si verifica due volte ed è una chiara indicazione per la subroutine. La specifica e il codice per la subroutine potrebbero essere scritti abbastanza facilmente. Esse sono:

Specifiche per la subroutine

Scopo: Convertire una parola in un numero da 0 a 20

Numeri di riga: 1000-1060

Parametri: Input: Parola da convertire
Output B1: Valore del numero

Locali: JJ

Riferimento esterno: T$(0-20):
Nomi dei numeri

```
1000 REM CONVERTI PAROLA A1$ IN
     NUMERO B1
1010 FOR JJ=0 TO 20
1020 IF A1$ = T$(JJ) THEN 1050
1030 NEXT JJ
1040 PRINT "NON TROVATA
     ENTRATA": STOP
1050 B1 = JJ
1060 RETURN
```

Notare che la subroutine imposta un'iterazione FOR per cercare nella lista T$. Essa abbina la parola data in A1$ con T$(0), T$(1) e così via. Quando trova una corrispondente entrata esce dall'iterazione saltando al comando 1050. Se cerca nella lista e non trova un abbinamento esatto, stampa una segnalazione e si ferma.
Il programma principale è molto lineare. Eccolo, con qualche commento extra:

```
 10 DIM T$(20)
 20 FOR J=0 TO 20
 30 READ T$(J)
 40 NEXT J
 50 DATA ZERO,UNO,DUE,TRE,
    QUATTRO,CINQUE
 60 DATA SEI,SETTE,OTTO,NOVE,DIECI
 70 DATA UNDICI,DODICI,TREDICI,
    QUATTORDICI
 80 DATA QUINDICI,SEDICI,DICIASSETTE
 90 DATA DICIOTTO,DICIANNOVE,VENTI
100 PRINT"INDICA DUE NUMERI"
110 INPUT X$,Y$
120 REM IMPOSTA PARAMETRI E
    RICHIAMA SUBROUTINE PER
    CONVERTIRE X$ IN X
125 A1$=X$: GOSUB 1000:X=B1
130 REM COME PER Y
135 A1$=Y$: GOSUB 1000:Y=B1
140 Z=X+Y: REM SOMMA I DUE NUMERI
150 IF Z >20 THEN PRINT"RISULTATO
    NON IN LISTA":STOP
160 PRINT"SOMMA=";T$(Z)
170 STOP
```

Vale la pena di notare che abbiamo mantenuto tutti i dettagli di ciascun richiamo di subroutine su una riga. Ciò comprende la messa a punto del parametro di input, del comando effettivo di richiamo e dell'estrazione del risultato dal parametro di output.
Il comando 150 è chiaro in quanto il programma non può visualizzare qualsiasi numero superiore a 20. Se il comando non esistesse e l'utente battesse ad esempio, DODICI e QUINDICI, la macchina tenterebbe di accedere a T$(27). Questo elemento non esiste e visualizzerebbe:

```
? BAD SUBSCRIPT
  ERROR IN 160
```

nella nostra versione la macchina non dà alcuna risposta esatta, ma almeno il commento è un poco più informativo.

ESPERIMENTO 20·2

a) Modificare il programma nell'ultima sezione in modo che lavori con numeri ROMANI fino a XL(40)
b) Le istruzioni DATA di un programma contengono 20 nomi e numeri telefonici, disposti come segue:

```
DATA MAXWELL, 3398123
DATA BOHR, 558
DATA EINSTEIN, 4073189
DATA VON NEUMANN, 777000
DATA NEWTON, 3074
DATA ZUSE, 222
DATA PLANCK, 1237543
DATA BOYLE, 146543
DATA BABBAGE, 03474
DATA LAPLACE, 5674
DATA PTOLEMY, 54863
DATA ARISTOTELE, 66543
DATA MCCARTHY, 47
DATA DIJKSTRA, 645
DATA BERZELIUS, 777
DATA CHARLES, 5543
DATA MENDELEEV, 645634
DATA TSIOLKOVSKY, 645332
DATA ARCHIMEDES, 2
DATA HOYLE, 21352
```

Disegnare e scrivere un programma che inviti l'utente a battere un nome, quindi a cercare il nome nell'elenco e visualizzare il corrispondente numero telefonico, se lo trova. Se non lo trova, il programma dovrebbe visualizzare un adatto messaggio. Due tipiche passate potrebbero essere:

```
NOME? NEWTON
TELEFONO NEWTON = 3074

NOME? FREUD
FREUD NON HA
TELEFONO
```

battuto dall'utente

Esperimento 20.2 completato	

Il quiz di autotest per questa unità è intitolato UNIT20QUIZ.

UNITA':21

LE FUNZIONI STRINGA

La manipolazione delle stringhe è una caratteristica essenziale del linguaggio BASIC che gli conferisce i poteri di risolvere i problemi pressochè in qualsiasi settore della vita quotidiana. Finora comunque i programmi che abbiamo considerato, hanno tutti interpretato le stringhe come oggetti completi e indivisibili. Ogni stringa è stata memorizzata, spostata e visualizzata esattamente nella stessa forma in cui era stata originariamente immessa nel VIC. In questa unità osserveremo qualche funzione speciale che consente di suddividere le stringhe in sezioni più piccole e anche in singoli caratteri. Queste funzioni aiuteranno a risolvere tutti i tipi di problemi che risulterebbero altrimenti difficili o impossibili da superare. Per esempio, saremo in grado di estrarre i cognomi da cognome e nome di più persone e impareremo a fare in modo che il VIC visualizzi lunghe frasi senza spezzare le parole.
Le funzioni coinvolte sono dette "funzioni stringa" in quanto usano o generano stringhe anzichè numeri. Qualsiasi funzione che come risultato genera una stringa ha un $ come ultimo carattere del suo nome, ad esempio MID$, SFR$ e così via.

La funzione stringa "LEN"

Le funzioni stringa sono costruite in BASIC in modo da poterle usare direttamente anche senza includerle in un programma. Proviamone qualcuna. Accendere la macchina e battere le seguenti righe, terminando ciascuna riga con il tasto RETURN

```
PRINT LEN("VIC")
PRINT LEN ("COMMODORE")
Z$="STRING"
PRINT LEN(Z$)
```

In ciascun caso il VIC visualizza la lunghezza (LENgth) della stringa coinvolta. In generale, la funzione LEN indica il numero di caratteri nella stringa argomento. In questo contesto "argomento" è una parola tecnica che significa l'oggetto sul quale una funzione opera.
Notare il modo in cui LEN è scritto

```
LEN (stringa argomento)
```

L'argomento deve essere racchiuso fra parentesi. Può essere una stringa esplicita racchiusa tra virgolette o il nome di una variabile stringa o qualsiasi espressione che produce una stringa come risultato. La funzione LEN produce un *numero* cosicchè l'intera costruzione può essere usata ovunque occorra un numero. Per esempio si potrebbe vedere

```
      X=LEN(Q$)
oppure FORJ=1 TO LEN (P$)
oppure PRINT LEN ("COMPRA" + S$ +
      "PANE")
```

La funzione stringa "MID$"

Un'altra funzione vitale è MID$. Questa funzione sceglie una *porzione* di qualsiasi stringa come suo argomento. Battere il comando

```
PRINT MID$ ("ABCDEFG",2,4)
```

Il risultato mostra come funziona MID$. In questo caso esso visualizza una stringa di quattro caratteri iniziando con il secondo carattere di "ABCDEFG".
In termini formali, la funzione MID$ prende tre argomenti che sono separati da virgole e racchiusi tra parentesi. Gli argomenti sono i seguenti:
Il primo è la stringa da usare.
Il secondo è un numero che specifica la posizione del primo carattere nel risultato. Il terzo è un altro numero che dà la lunghezza del risultato.
Come ci si potrebbe aspettare, gli argomenti possono essere variabili del tipo appropriato. La lunghezza del risultato può essere qualsiasi, da 0 (detta la stringa "nulla") fino all'intera lunghezza del primo argomento. In pratica, è spesso di un carattere.
Ecco un semplice programma per immettere una parola e visualizzarla alla rovescia. Studiarlo accuratamente e notare come vengono usate le funzioni LEN e MID$:

```
10 INPUT "BATTI UNA PAROLA"; X$
20 PRINT "PAROLA ALLA ROVESCIA="
30 FOR J = LEN(X$) TO 1 STEP -1
40 PRINT MID$(X$,J,1);
50 NEXT J
60 STOP
```

Impostare il programma e controllare il risultato; provare con parole di 1, 2 o più caratteri.

Estrazione dei cognomi

Passiamo ora ad estrarre porzioni di stringhe più ampie. Se si chiede a qualcuno di battere il proprio cognome e nome, si potrebbero avere risposte di questo genere:

```
      J.P. JONES
oppure JANET BLOGGS
oppure GEORGEP O'HAGAN
oppure ALFRED HENRY FFOUKES-SMYTHE
```

Se si vuole estrarre il cognome da una tale stringa, non è bene iniziare dal principio in quanto il computer non può individuare un cognome da un secondo nome o addirittura da una stringa di iniziali. In ogni caso il cognome viene sempre per ultimo e ciò suggerisce un modo per individuarlo. Si esamini ciascun carattere iniziando dal termine della stringa, fino a che non si arriva ad uno che non può far parte di un cognome. La successiva posizione alla destra deve essere il punto in cui inizia il cognome. Se ogni carattere nella stringa fa parte del cognome, il suo proprietario viene chiaramente da un paese tipo Afghanistan, dove la gente ha soltanto un nome.

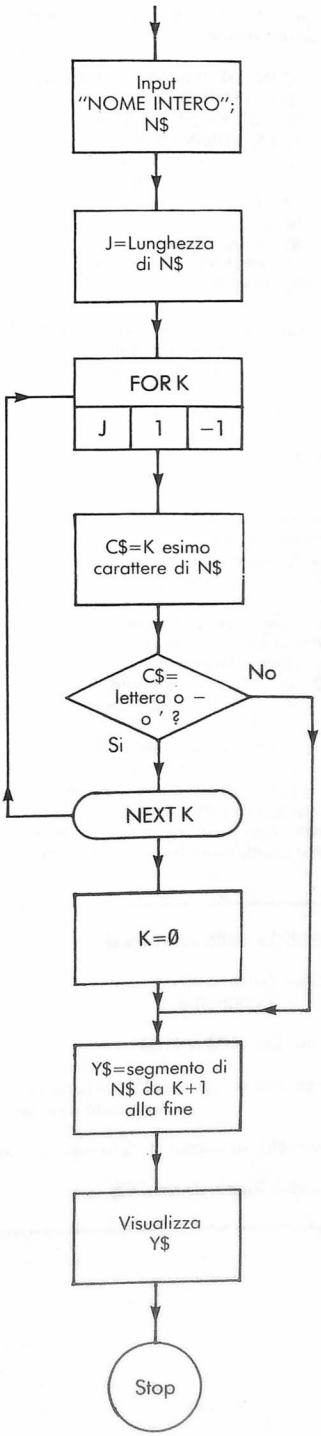
Osservare questa stringa che ha le posizioni dei caratteri contrassegnati:

```
J . P . J O N E S
1 2 3 4 5 6 7 8 9
```

Se si inizia la ricerca da destra, il primo carattere che s'incontra che non può far parte di un cognome è il punto nella posizione 4. Il cognome pertanto inizia alla posizione 5 e può essere estratto da una funzione MID$:

MID$ (N$,5,5) (dove N$="J.P.JONES")
che dà "JONES".

In generale, se la lunghezza dell'intera stringa è J e la posizione del primo carattere che non fa parte del cognome è K, il cognome stesso inizierà al carattere (K+1) e la sua lunghezza sarà (K−J).
Quali simboli può comprendere un cognome? Gli esempi suggeriscono che lettere, trattini e apostrofi sono i soli caratteri che occorre aspettarsi.
Così sono state raccolte sufficienti idee per schizzare uno schema di flusso. Esso apparirà come segue:



Glossario

N$: Stringa col nome intero
J: Lunghezza di N$
K: È usato per analizzare all'indietro la stringa
Y$: Risultato: cognome in N$

Ora possiamo provare l'algoritmo in un breve programma e cioè:

```
10 INPUT "NOME E COGNOME"; N$
20 J=LEN(N$)
30 FOR K=J TO 1 STEP -1
40 C$=MID$(N$,K,1)
50 IF NOT(C$ >="A" AND C$ <="Z"
   OR C$="-" OR C$=" ' ") THEN 80
60 NEXT K
70 K=0
80 Y$=MID$(N$,K+1,J-K)
90 PRINT Y$
100 GOTO 10
```

Commenti: C$ è usato per contenere il carattere K-esimo dell'intero nome. Esso fa parte di un cognome se:

(a) è una lettera (e cioè risiede nel campo da A a Z),
oppure (b) è un trattino,
oppure (c) è un apostrofo.

L'istruzione condizionale salta a 80 se C$ *non* fa parte di un cognome.
La riga 70 viene eseguita soltanto per coloro che hanno un solo nome. Quando il comando FOR termina, la variabile controllata (K, nell'esempio), è "indefinita" (il che significa che può avere qualsiasi valore) e non necessariamente 0. Pertanto K deve essere impostato in modo che la riga 80 possa essere eseguita.
Il comando nella riga 80 estrae il cognome e lo inserisce in Y$.
Quando questo programma viene battuto, sembra funzionare correttamente su tutti gli esempi forniti. Il programma soddisfa una funzione utile e generica cosicchè lo trasformeremo in una *subroutine* con le seguenti specifiche e istruzioni. Notare il cambiamento nei nomi delle variabili:

Specifiche della subroutine

Scopo: Estrarre un cognome da un nome e cognome

Righe: Da 4100 a 4180

Parametro di input: N1$ contiene un cognome e nome

Parametro di output: Y1$ fornisce il cognome

Variabili locali: JJ,KK,CC$

```
4100 REM ESTRAI COGNOME DA N1$
     E INDICALO IN Y1$
4110 JJ=LEN (N1$)
4120 FOR KK=JJ TO 1 STEP -1
4130 CC$=MID$(N1$,KK,1)
4140 IF NOT (CC$ >="A" AND CC$ <=
     "Z" OR CC$="-" OR CC$="'")
     THEN 4170
4150 NEXT KK
4160 KK=0
4170 Y1$=MID$(N1$,KK+1,JJ-KK)
4180 RETURN
```

Un programma di gestione per provare questa subroutine potrebbe essere:

```
10 INPUT"NOME E COGNOME
   PREGO"; N1$
20 GOSUB 4100
30 PRINT "COGNOME="; Y1$
40 GOTO 10
```

USO DI MID$ PER CORREGGERE UNA STRINGA

Un'osservazione finale sulla funzione MID$: non è possibile usarla sul lato sinistro di un comando di assegnazione. Per esempio, se si vuole cambiare il quarto carattere della stringa X$ in una "U", *non è possibile* scrivere:

```
MID$(X$,4,1) = "U"
```

non è BASIC

In ogni caso è possibile effettuare la stessa cosa suddividendo la stringa in tre porzioni e ricombinandola con l'operazione +:

```
X$=MID$(X$,1,3)+"U"+MID$(X$,5,LEN(X$)-4)
```

Primi 3 caratteri di X$

Fine di X$ per i caratteri da 5 in avanti

ESPERIMENTO 21·1

Questo esperimento è in tre parti:

(a) Scrivere un programma che immette una stringa dalla tastiera e la visualizza, dopo aver per prima cosa cambiato ogni "E" in una "O". Il risultato potrebbe essere:

Quosta idoa ó stata prosa da un programma TV prosontato da Miko Bongiorno.

(b) È possibile applicare la funzione MID$ alla variabile "time" TI$ in modo da estrarre le ore, i minuti e i secondi (sotto forma di stringa) separati ciascuno da una barra. Scrivere un breve programma che visualizza l'ora corrente così:

13/23/57

(c) La subroutine per l'estrazione del cognome possiede un errore che non abbiamo trovato nelle prove originali: se qualcuno batte un punto o uno spazio dopo il cognome, la subroutine dà una stringa nulla. Disegnare un'adatta modifica per la subroutine.

Esperimento 21.1 completato	

LEFT$ E RIGHT$

Due funzioni stringa che sono spesso utili sono LEFT$ e RIGHT$. Come si può dedurre dal nome, LEFT$ estrae il lato sinistro di una stringa e RIGHT$ il lato destro. Ciascuna funzione ha due argomenti; il primo (come MID$) è la stringa da dividere e il secondo è la lunghezza del risultato. Pertanto:

PRINT LEFT$("ABCDEFG",3) dà ABC
e PRINT RIGHT$("ABCDEFG",2) dà FG

Si sarà notato che nessuna di queste due funzioni ottiene nulla che non potrebbe essere fatto con MID$ ma esse sono tuttavia talvolta molto più utili e comode da usare.

POSIZIONAMENTO DEL CURSORE

Una particolare applicazione di LEFT$ è per posizionare il cursore in qualsiasi punto dello schermo. Inizieremo impostando due variabili stringa:

Y$ come "HOME" (posizione di partenza) seguito da numerosi "caratteri di spostamento verso il basso del cursore"

X$ come numerosi caratteri di "cursore a sinistra":

10 Y$="[CLR HOME][CRSR ↓]<22 volte>[CRSR ↓]"

20 X$="[CRSR →]<21 volte>[CRSR →]"

Per spostare il cursore ad una posizione Y righe verso il basso a partire dalla parte superiore dello schermo, potremmo stampare (PRINT) i primi (Y+1) caratteri di Y$: un "home" (posizione di partenza) e Y volte cursore verso il basso. Analogamente spostiamo X posti orizzontalmente stampando (PRINT) i primi X caratteri di X$. Questi possono essere combinati in una singola istruzione:

100 PRINT LEFT$(Y$,Y+1); LEFT$(X$,X);

-PERMUTAZIONI — n!

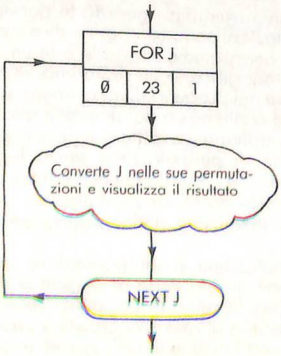
Il successivo esempio riguarda le permutazioni. Le permutazioni sono utili per risolvere problemi e trovare anagrammi di parole e in un contesto più serio essi giocano un ruolo importante nelle statistiche e nel progetto di esperimenti scientifici. La sezione contiene un po' di matematica, ma se si trova la matematica difficile è possibile usare il programma di permutazione senza leggere la spiegazione. Se l'intero concetto è troppo ostico, saltare questa parte. Le permutazioni non sono una parte essenziale della programmazione in BASIC.

Una *permutazione* è un particolare ordine di disposizione di una serie di oggetti o di eventi. Per esempio, l'ordine in cui viene fatto suonare uno scampanìo di otto campanelli è una permutazione e così l'ordine in cui i cavalli in una gara

tagliano il traguardo. Quante permutazioni è possibile ottenere? Ciò dipende dal numero degli oggetti. In una gara con un solo cavallo può esserci soltanto un vincitore. Se ci sono due cavalli denominati A e B uno solo può vincere, ma l'altro deve essere per forza secondo: ci sono quindi due permutazioni e cioè AB e BA. Con tre cavalli, possono esserci tre differenti vincitori e in ciascun caso uno dei due cavalli rimanenti può essere secondo. Ci sono pertanto sei permutazioni: ABC, ACB, BAC, BCA, CAB e CBA. Con 4 cavalli possono esserci 4x3x2 ossia 24 differenti risultati. La tabella mostra il modo in cui si producono queste cifre.

N. di oggetti	*N. di permutazioni*
1	1
2	2 = 2
3	3x2 = 6
4	4x3x2 = 24
5	5x4x3x2 = 120
6	6x5x4x3x2 = 720
7	7x6x5x4x3x2 = 5040
8	8x7x6x5x4x3x2 = 40320

Come si può vedere il numero di permutazioni cresce molto rapidamente e supera rapidamente i tre milioni per 10 oggetti. Il numero di permutazioni di "n" oggetti è denominato "fattoriale di n ", una frase che matematicamente si scrive talvolta nella forma "n!", per indicare il prodotto di tutti i numeri da 1 a n. Risolveremo ora un programma che legge qualsiasi stringa e ne visualizza tutte le permutazioni. Per esempio, se l'input è "TEA" l'output dovrebbe comprendere TEA, TAE, ATE, AET, EAT e ETA.
Si supponga che la stringa iniziale sia lunga n lettere. Sappiamo che ci saranno n! (fattoriale di n) diverse permutazioni, ma come è possibile far sì che il computer le elabori senza ripetersi o mancandone qualcuna?
Un modo per affrontare questo problema consiste nell'inventare un metodo per convertire i numeri 0,1,2,3... in permutazioni in modo che ciascuna sia diversa da tutte le altre. Quindi se n è — ad esempio 4 — (una stringa di quattro lettere) è possibile produrre tutte le 24 permutazioni di quattro lettere convertendo ciascuno dei numeri da 0 a 23 nella corrispondente permutazione e visualizzando i risultati. Lo schema di flusso per tale programma sarebbe:



In questo schema di flusso possiamo chiamare J come "numero di permutazioni". Abbiamo ancora bisogno di trovare un modo per convertire il numero di permutazioni nella corrispondente permutazione di lettere. Questo problema non è facile, ma potremmo ottenere qualche indicazione osservando le risposte in alcuni semplici casi. Si supponga che gli oggetti da permutare siano le lettera A B C e così via.

1 Oggetto: 1 Permutazione

A

2 Oggetti: 2 Permutazioni

AB
BA

3 Oggetti: 6 Permutazioni

ABC ACB
BAC BCA
CAB CBA

4 Oggetti: 24 Permutazioni

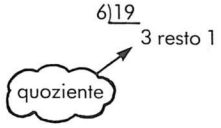
ABCD ABDC ACBD ACDB ADBC ADCB
BACD BADC BCAD BCDA BDAC BDCA
CABD CADB CBAD CBDA CDAB CDBA
DABC DACB DBAC DBCA DCAB DCBA

Si vedrà che se ciascuna permutazione viene presa come una "parola" tutte le parole della stessa dimensione sono scritte nell'ordine del dizionario.
Quando si studiano queste liste, emerge un profilo ben definito. Si supponga di dividere i membri in ciascuna serie secondo le rispettive prime lettere e quindi le permutazioni di tre lettere in tre gruppi di due ciascuno:

ABC	BAC	CAB
ACB	BCA	CBA

In ciascun gruppo la lettera iniziale è seguita da tutte le permutazioni delle altre due. Così A è seguito e BC e CB "subpermutazioni".
È possibile vedere questo profilo comparire anche nelle permutazioni di quattro lettere. Ci sono quattro gruppi, ciascuno con sei membri. Ciascuna lettera iniziale è seguita da sei subpermutazioni delle altre tre.
Per qualsiasi permutazione possiamo definire un "numero di gruppo" e un "numero di subpermutazione". Il numero di gruppo indicherà la prima lettera (secondo il codice A=0, B=1, C=2 e così via) e il numero di subpermutazione sarà la posizione all'interno del gruppo (sempre iniziando da 0). Per esempio si consideri la permutazione BCDA. Questa ha un numero di gruppo di 1 (in quanto inizia con un B) e il numero di subpermutazione è 3, come è possibile controllare dalla tabella che precede.
C'è ora un importante suggerimento circa il metodo di convertire qualsiasi numero di permutazione nella corrispondente permutazione. Troviamo per prima cosa il numero di gruppo, che definisce la prima lettera; quindi troviamo il numero di subpermutazione e ricaviamo la corrispondente permutazione dalle lettere rimanenti!

Per trovare il numero di gruppo e di supermutazione tutto ciò che occorre fare è di dividere il numero di permutazioni per la dimensione del gruppo. Il quoziente dà il numero di gruppo e pertanto la prima lettera e il resto specifica il numero di subpermutazione.
Per dare un esempio, si consideri il numero di permutazione 19 di quattro lettere.



La corrispondente permutazione inizia con D (D=3) ed è seguita dal numero di subpermutazione 1 delle lettere A B C. La subpermutazione può essere ricavata esattamente con lo stesso processo della permutazione. Ci sono soltanto 2 importanti modifiche:

1) La lettera usata all'inizio della permutazione principale deve essere rimossa dalla lista di lettere in modo che non venga utilizzata di nuovo

2) La dimensione del gruppo deve essere aggiustata (ad esempio da 6 a 2 o da 2 a 1).

Qui c'è uno specifico esempio, che prende la permutazione 9 di quattro lettere. Iniziamo contrassegnando le lettere A=Ø, B=1, C=2, D=3.

1) Dividiamo 9 per 6 ottenendo il quoziente = 1, resto = 3. La prima lettera della permutazione è pertanto B. Rimuoviamo la B dalla lista delle lettere e ricontrassegnamo le altre: A=Ø, C=1, D=2.

2) Troviamo ora la subpermutazione 3 dalle lettere A, C e D. La dimensione di gruppo è 2. Dividendo 3 per 2 otteniamo il quoziente = 1, il resto = 1. La successiva lettera della permutazione è pertanto C. Rimuoviamo la C dalla lista e ricontrassegnamo le altre lettere A=Ø, D=1.

3) Troviamo ora la subpermutazione 1 dalle lettere A e D. La dimensione del gruppo è 1. Dividendo 1 per 1, si ottiene il quoziente 1, resto = Ø. La successiva lettera della permutazione è D. Rimuoviamo questa lettera dalla lista. Ciò lascia una sola lettera, una A contrassegnata Ø.

4) Troviamo infine la subpermutazione Ø dalla lettera A. Ovviamente è la A, ma possiamo ancora usare lo stesso processo usato in precedenza: la dimensione del gruppo è 1 e Ødiviso 1 dà come quoziente = Ø, come resto = Ø. Come previsto, la lettera finale è A e la permutazione nel suo complesso è BCDA.

Per verificare la comprensione di questo processo, provare a convertire qualche numero tra Ø e 23 e assicurarsi che le risposte risultino identiche alla tabella che precede. (Ricordarsi che la prima permutazione ABCD corrisponde a Ø non a 1).
Ora convertiremo il metodo in un programma. Le lettere da permutare non sono necessariamente ABCD ma possono essere qualsiasi cosa l'utente batte. Analogamente la lunghezza della stringa è arbitraria quantunque possiamo aspettarci che il programma giri per un tempo lunghissimo se ci sono più di 6 o sette lettere. Innanzitutto dobbiamo gestire un "lotto" di lettere e assicurarci che siano selezionate correttamente. Il modo più semplice per far ciò consiste nell'inserirle in una stringa (ad esempio Y$). Dato che sono quindi numerate automaticamente, la lettera con la label "notazionale" Q, può essere scelta nella forma:

```
MID$(Y$,Q + 1,1)
```

Il "+1" deve essere incluso in quanto lo schema di numerazione automatica inizia a 1, mentre il nostro metodo produce numeri di gruppi che iniziano da Ø.

RIMOZIONE DELLE LETTERE DA UNA STRINGA

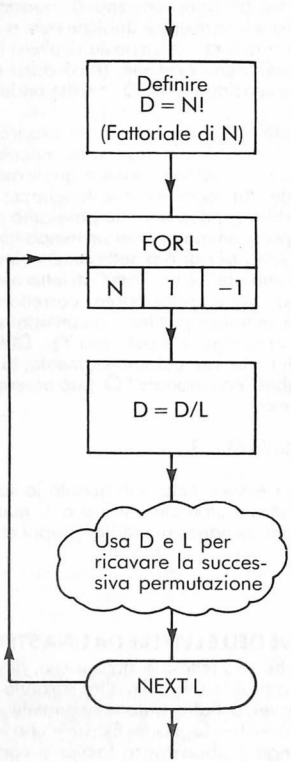
Una volta che una lettera è stata usata, deve essere rimossa dalla stringa. Le altre saranno quindi spostate verso l'alto automaticamente il che equivale a ricontrassegnarle. Estrarre una lettera da una stringa è abbastanza facile: si concatenano (si uniscono) la porzione della stringa alla sinistra della lettera utilizzata e la porzione alla destra.

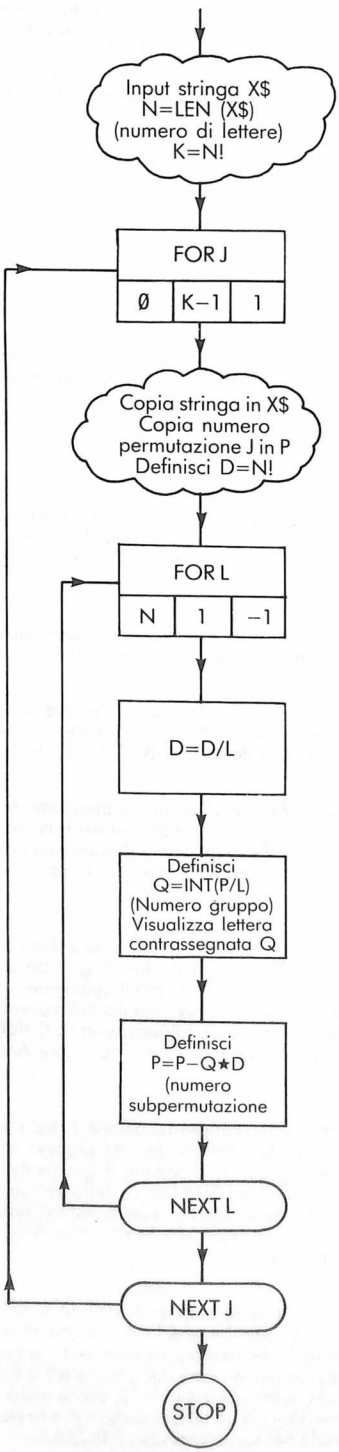
Risultato: HORRBLE

Se il numero di label della lettera risultato è Q, la porzione di sinistra avrà Q lettere e la porzione di destra (LEN(Y$) −Q1). (Ricordarsi che le label vanno da Q a Q). Il comando richiesto è:

```
Y$=LEFT$(Y$,Q) + RIGHT$(Y$,LEN(Y$)
−Q−1)
```

Secondo, il numero di lettere di ciascuna subpermutazione diventa 4 3 2 1 e la corrispondente dimensione del gruppo . . . 6 2 1 1. Questi ultimi sono i valori di n! per diversi valori di n e possono essere prodotti da un programma tipo questo:





In questo schema di flusso L e D prenderanno le corrette sequenze di valori. Per esempio se N=6, L diventa 6,5,4,3,2,1 e D (nel momento in cui viene usato) sarà 120,24,6,2,1 e 1.
Possiamo ora riunire tutte queste idee in un programma. Il processo per convertire una stringa in una permutazione gradualmente la distrugge cosicchè dobbiamo tenere una copia base dell'originale e sostituire la "copia di lavoro" per ciascuna permutazione separata. Lo schema di flusso e il programma sono:

Il Glossario è

X$:	Stringa da permutare
N:	Lunghezza della stringa da permutare
K:	Fattoriale di N (calcolato nella riga 30)
J:	Numero di permutazione
Y$:	Copia di lavoro di X$
D:	Dimensione corrente del gruppo
L:	Numero delle lettere nella subpermutazione corrente
P:	Numero di permutazione corrente
Q:	Numero di gruppo della subpermutazione corrente
S:	Variabile per tutti i numeri da 1 a N

Dopo questa complessa analisi, il programma risulta sorprendentemente corto. Esso è:

```
 10 INPUT X$
 20 N=LEN(X$)
 30 K=1:FOR S=1 TO N:K=K*S:NEXT S
 40 FOR J=0 TO K -1
 50 Y$=X$: D=K: P=J
 60 FOR L=N TO 1 STEP -1
 70 D=D/L
 80 Q=INT(P/D): P=P-D*Q
 90 PRINT MID$(Y$,Q+1,1)
100 Y$=LEFT(Y$,Q)+RIGHT$(Y$,LEN(Y$)
    -Q-1)
110 NEXT L
120 PRINT
130 NEXT J
140 STOP
```

Il quoziente e il resto sono calcolati nella riga 80. Ricordarsi che INT scarta qualsiasi frazione; ciò fa sì che il comando funzioni correttamente. Per esempio se P=17 e D=6, allora

$$Q=INT(17/6) = INT(2.8333333) = 2$$
$$P=17 - 6*2 = 17 - 12 = 5$$

Impostare questo programma e provarlo con i propri dati. Vedere se è possibile modificarli in modo che visualizzi più di una permutazione sulla stessa riga.

CONVERSIONE DI STRINGHE IN NUMERI - VAL

Due altre funzioni stringa aiutano a convertire stringhe in numeri e viceversa.

VAL ("una stringa")

prende una stringa di cifre decimali (eventualmente preceduta da + o – e contenente un punto decimale) e la converte nel corrispondente valore numerico.
VAL è utile per ottenere un input valido da utenti molto ingenui. Si consideri un programma che chiede di battere un numero, mediante un'istruzione del tipo

INPUT X

Se l'utente batte effettivamente qualcosa che non è un numero ad esempio "PARDON", il sistema BASIC dice semplicemente

REDO FROM START

Ciò non è molto utile e l'utente può non rendersi conto di che cosa ci si aspettava da lui.
Come alternativa, tutto ciò che l'utente batte può essere letto come stringa, quindi non c'è rischio di un messaggio REDO FROM START e il programma può analizzare la risposta dell'utente, carattere per carattere emettendo adatti messaggi di errore se rileva qualche sbaglio.

Infine, se la stringa risulta composta da simboli che costituiscono un numero accettabile, il valore può essere estratto con VAL. Qui c'è una specifica, uno schema di flusso e una subroutine per input "tollerante" di numeri.

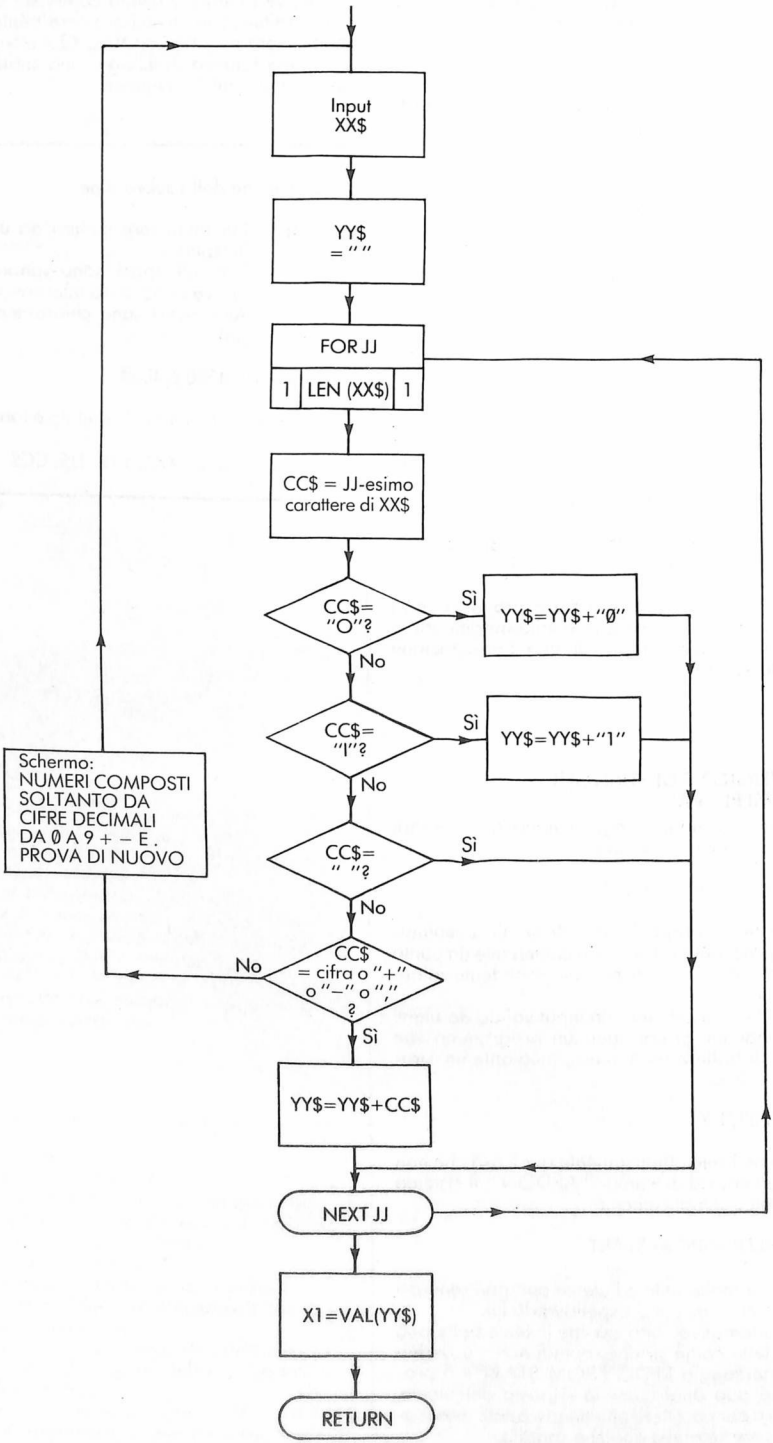
Specifiche della subroutine

Scopo: Far immettere numeri da un utente inesperto.
Tutti gli spazi sono ignorati e le lettere l e O sono interpretate 1 e 0. Altri errori sono chiaramente spiegati.

Righe: da 4500 e 4660

Parametro di output: Il risultato è fornito in X1

Variabili locali: XX$, YY$, JJ$, CC$


Input XX$
YY$ = ""
FOR JJ
1 LEN (XX$) 1
CC$ = JJ-esimo carattere di XX$
CC$= "O"?
Sì
YY$=YY$+"0"
No
CC$= "I"?
Sì
YY$=YY$+"1"
No
CC$= " "?
Sì
No
CC$ = cifra o "+" o "–" o "."?
No
Sì
Schermo: NUMERI COMPOSTI SOLTANTO DA CIFRE DECIMALI DA 0 A 9 + – E . PROVA DI NUOVO
YY$=YY$+CC$
NEXT JJ
X1=VAL(YY$)
RETURN

```
4500 REM INPUT TOLLERANTE DI NUMERI
4510 INPUT XX$
4520 YY$=""
4530 FOR JJ = 1 TO LEN (XX$)
4540 CC$=MID$(XX$,JJ,1)
4550 IF CC$="O" THEN YY$=YY$+"0":
     GOTO 4600
4560 IF CC$="l" THEN YY$=YY$+"1":
     GOTO 4600
4570 IF CC$="" THEN 4600
4580 IF NOT(CC$<="9" AND CC$>="0"
     OR CC$="+" OR CC$="-"OR
     CC$=".") THEN 4620
4590 YY$=YY$+CC$
4600 NEXT JJ
4610 X1=VAL(YY$):RETURN
4620 PRINT "UN NUMERO È COSTITUITO
     SOLO DA"
4630 PRINT "CIFRE 0-9,"
4640 PRINT "+,-,."
4650 PRINT "PREGO TENTA ANCORA"
4660 GOTO 4510
```

CONVERSIONE DI NUMERI IN STRINGHE— STR$

STR$ (un numero) funziona all'opposto di VAL. Esso prende un numero come argomento e fornisce una stringa di simboli, gli stessi che sarebbero stati visualizzati se fosse stato usato PRINT. STR$ è una funzione preziosa per ottenere sullo schermo un tracciato nitido di numeri.
Il problema principale con il comando PRINT è che non è mai possibile sapere quale sarà il tracciato esatto. Per illustrare questo punto, impostare il seguente programma:

```
 5 PRINT "NUMERO", "QUADRATO"
10 FOR J = 1 TO 7 STEP 0.1
20 PRINT J, J*J
30 NEXT J
40 STOP
```

Eseguire questo programma lentamente tenendo abbassato il tasto CTRL. All'inizio tutto sembra andare bene. Lo schermo visualizza la tabella dei quadrati che vi si aspettava. Si ottiene.

```
1          1
1.1        1.21
1.2        1.44
```

e così via.
Per contro, l'entrata 2.8 sembra peculiare; essa dice

```
2.8        7.8399999
```

ed è seguita da una riga vuota. Si sa perfettamente che il quadrato di 2.8 è 7.84, e non 7.83999999 come compare sullo schermo.

Dopo 3.6 la tabella impazzisce. Essa indica:

```
3.6                    12.96
3.69999999
13.69
3.79999999
14.44
3.89999999
15.21
3.99999999
```

e così via.
La difficoltà è dovuta ai due effetti che interagiscono l'uno con l'altro.
Il primo problema è quello "dell'errore di troncamento". Dato che il VIC — come la maggior parte dei computer — lavora sul sistema binario, non può gestire esattamente numeri tipo 0.1. C'è sempre un piccolo errore. Nel nostro programma il valore di J inizia con 1 e cresce verso 7 mediante ripetute aggiunte di 0.1; al limite gli errori si accumulano e compaiono in risultati che sono pressochè esatti ma non abbastanza quanto ci si aspettava. Pertanto, la differenza tra

```
7.84        e        7.83999999
```

è soltanto 0.00000001, ma ciò è sufficiente per mettere scompiglio nel tracciato; il numero sembra abbastanza diverso e viene inserita forzatamente una riga vuota in quanto l'ultima cifra del numero va oltre l'ultima colonna dello schermo.
Il secondo emerge quando l'errore di troncamento interessa il valore stampato dalla J. "3.7" è trasformato in "3.6999999" e questo numero è così lungo che finisce nella parte dello schermo dove dovrebbe comparire normalmente il secondo numero. Il risultato costringe il VIC ad iniziare una nuova riga per il secondo numero nell'istruzione PRINT e così distrugge l'intero aspetto della tabella.
STR$ consente un controllo migliore sul tracciato dei numeri decimali. Esso prende un argomento numerico e produce una stringa in cifre decimali, spazi, ecc. che è la stessa che sarebbe stata visualizzata dall'istruzione PRINT. La differenza è che il risultato è *interno* e può essere manipolato e corretto prima di venire visualizzato. Per illustrare questo punto ecco un breve programma che visualizza un numero scritto alla rovescia:

```
10 INPUT "DATO UN NUMERO"; X
20 X$=STR$(X)
30 FOR J=LEN(X$) TO 1 STEP -1
40 PRINT MID$ (X$,J,1);
50 NEXT J
60 PRINT
70 GOTO 10
```

ARROTONDAMENTO

La prima tecnica che esamineremo è quella dell'*arrotondamento*. Il VIC generalmente visualizza frazioni fino a 8 cifre decimali salvo che esclude gli zeri di coda. Solitamente, 3 o 4 cifre decimali sono un'accuratezza sufficiente per i risultati. Quando un decimale è accorciato mediante arrotondamento, esso solitamente aggiunge 1

all'ultima cifra conservata se la parte scartata inizia con 5 o più. Per esempio se il valore corretto di un numero è 3.14159, il valore arrotondato (a tre cifre decimali) è 3.142. Per contro, il valore arrotondato di 2.71828 è 2.718.
La meccanica dell'arrotondamento è abbastanza lineare. Per arrotondare un numero positivo, a tre cifre, aggiungiamo 0.0005 e quindi scartiamo la quarta e le successive cifre. Questi esempi mostrano il processo al lavoro:

3.13159	2.71828
0.0005 +	0.0005 +
3.14209	2.71878
(scartare 09)	(scartare 78)
3.142	2.178

È chiaro che l'arrotondamento si libera degli errori di troncamento introdotti dal VIC (dato che 3.69999999 arrotondato a tre cifre diventa 3.700) ed evita inoltre le variazioni imbarazzanti nel numero dei caratteri visualizzati.
Il processo per stampare i numeri arrotondati a 3 cifre decimali è il seguente:

(1) Aggiunta di 0.0005

(2) Uso di STR$ per convertire in forma di stringa (ad esempio) NN$

(3) Individuazione della posizione del punto decimale (ad esempio) PP

(4) Visualizzazione della parte di sinistra di NN$ fino a 3 cifre dopo il punto decimale.

Potrebbe darsi che non ci sia punto decimale in NN$. Ciò succederebbe se il valore originale terminasse in ".9995" (ad esempio 2.9995"). Quindi NN$ comparirebbe come "3" senza punto decimale. Ciò deve essere considerato un caso speciale.
Possiamo introdurre questo algoritmo in un'utile subroutine.

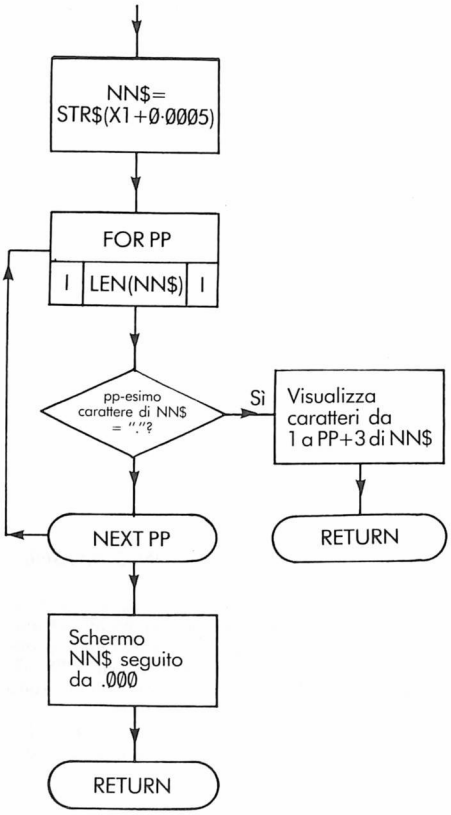
Specifiche della subroutine

scopo: Visualizzare un numero positivo arrotondato a 3 cifre decimali

Righe: da 5000 a 5050

Parametro di input: X1 ha il valore di numero

Variabili locali: NN$, PP



La corrispondente codifica è:

```
5000 REM VISUALIZZA X1 A 3 CIFRE
     DECIMALI
5010 NN$=STR$(X1 + 0.0005)
5020 FOR PP=1 TO LEN(NN$)
5030 IF MID$(NN$,PP,1)="."THEN
     PRINT LEFT$(NN$,PP+3);:RETURN
5040 NEXT PP
5050 PRINT NN$;".000";:RETURN:REM
     NIENTE DECIMALI IN NN$
```

Un'adatta routine di gestione è una versione modificata del programma che ci dava tutti quei fastidi originariamente:

```
10 FOR J=1 TO 7 STEP 0.1
20 X1=J: GOSUB 5000
30 X1=J*J: GOSUB 5000
40 PRINT
50 NEXT J
60 STOP
```

Se si esegue questo programma, si vedrà che tutte le difficoltà scompaiono completamente!

ESPERIMENTO 21·2

(a) Modificare la subroutine della visualizzazione discusso precedentemente in modo che il programma principale possa scegliere il numero di cifre decimali da usare. Questo numero sarà fornito come *parametro* in Y1. Suggerimenti: la costante da aggiungere può essere scritta nella forma:

```
0.5*10↑-Y1
```

Provare la subroutine a fondo e usarla per visualizzare alcune nuove tabelle.

Esperimento 21.2 completato	

COME EVITARE CHE LE PAROLE SUPERINO LA CAPACITÀ DELLO SCHERMO

Il VIC è una superba macchina sotto molti aspetti ma anche i suoi progettisti ammettono che ha uno schermo piuttosto limitato. I computer più costosi generalmente consentono la comparsa di 40 o 80 caratteri su ciascuna riga. Lo schermo stretto non è un inconveniente serio nella programmazione ma richiede cura per visualizzare messaggi in modo che le parole non vengano spezzate. Se si usa una serie di comandi PRINT, occorre osservare le seguenti regole:

(1) Nessuna riga deve comprendere più di 22 caratteri

(2) Se la riga ha esattamente 22 caratteri, il comando PRINT deve seguire il testo con un punto e virgola per impedire che venga introdotta forzatamente una riga vuota. Ciò in quanto un carattere nella 22esima posizione provoca sempre l'inizio di una nuova riga.

Per terminare questa unità, descriveremo una subroutine, che dispone automaticamente il testo in modo da evitare questo problema.
Si supponga di avere una stringa di parole separate da spazi. La stringa può avere qualsiasi lunghezza fino ad un massimo di 255 caratteri. Se semplicemente la stampiamo (PRINT), questa verrà tagliata a righe di 22 caratteri senza qualsiasi riguardo alle posizioni delle parole e degli spazi. Dobbiamo trovare un metodo migliore per dividerla in righe.
Se la stringa è di 22 caratteri o meno, può essere visualizzata così come è. Altrimenti dobbiamo esaminare la stringa e trovare il segmento più ampio (iniziando dal principio), che può essere visualizzato senza tagliare una parola in due. Visualizziamo quel segmento, lo rimuoviamo dalla parte anteriore della stringa e iniziamo di nuovo il processo su ciò che rimane. Per trovare il segmento più ampio, cerchiamo uno spazio che inizia dal 23esimo carattere e cerchiamo all'indietro. Per vedere, si consideri la stringa

FRIENDS, ROMANS, COUNTRYMEN,
LEND ME YOU YOUR EARS

Il 23esimo carattere è una R; così cerchiamo all'indietro fino a che arriviamo allo spazio, carattere 17. Visualizziamo la riga di 16 caratteri

FRIENDS, ROMANS,

e rimuoviamo 17 caratteri dalla parte anteriore della stringa, lasciando

COUNTRYMENT, LEND ME YOUR EARS

Il 23esimo carattere è ora una U. La riga successiva da visualizzare comparirebbe nella forma

COUNTRYMEN, LEND ME

e la riga finale dato che è più corta di 22 caratteri, rimarrebbe

YOUR EARS

Una specifica di subroutine, lo schema di flusso e la codifica per questo processo, sono tutti indicati qui di seguito.

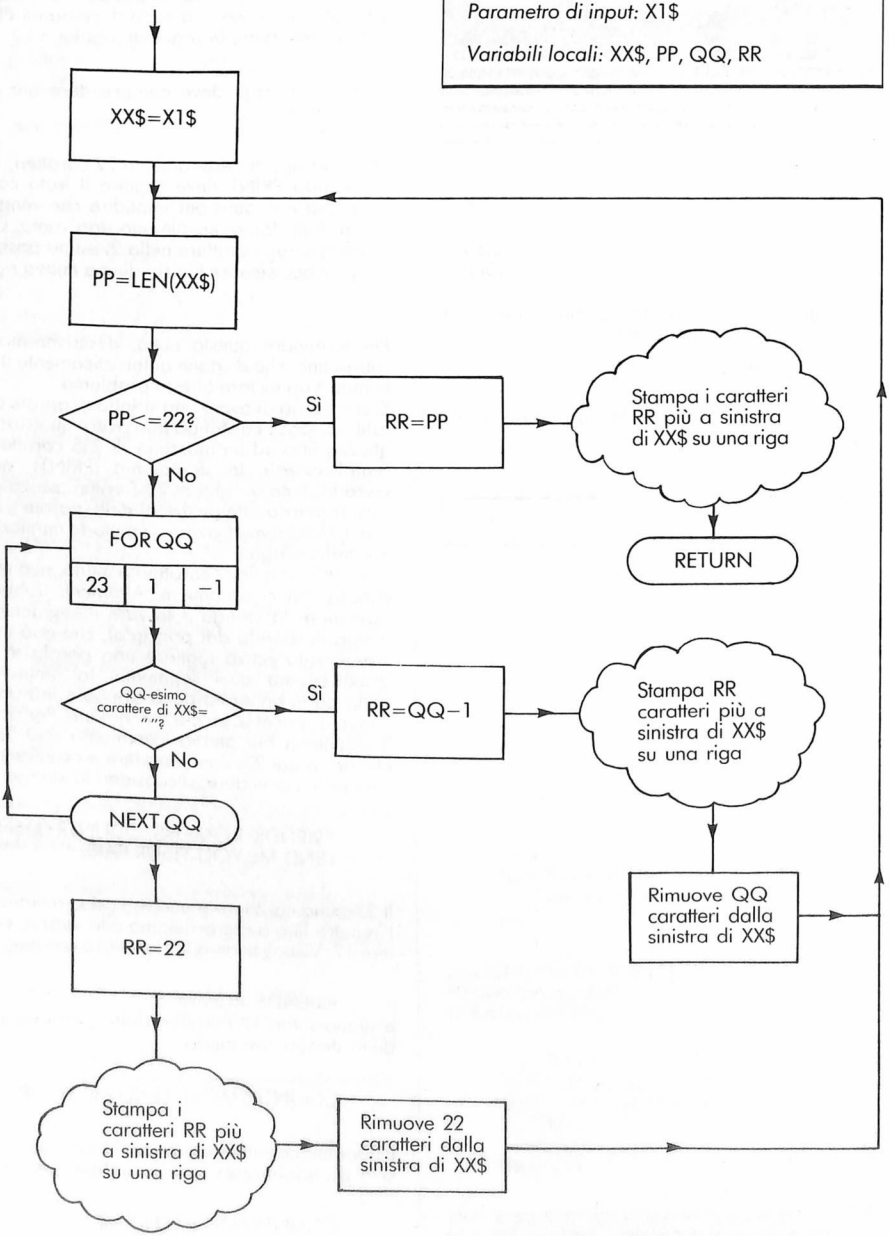
Specifiche della subroutine

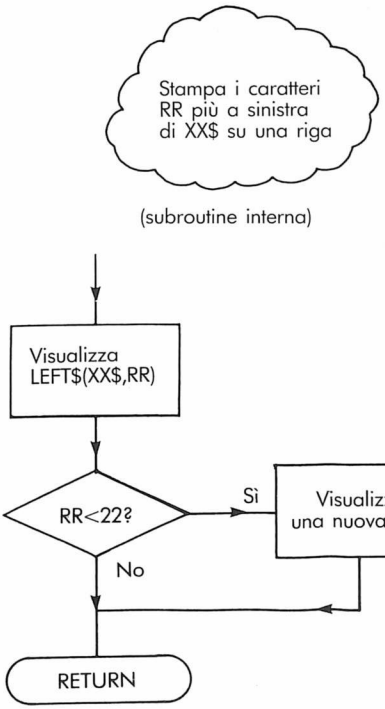
Scopo: Visualizzare la stringa X1 in modo che le parole non vengano spezzate su due righe

Righe: da 5500 a 5600

Parametro di input: X1$

Variabili locali: XX$, PP, QQ, RR






```
5500 REM VISUALIZZA X1$ SENZA
     DIVIDERE LE PAROLE
5510 XX$=X1$
5510 PP=LEN(XX$)
5530 IF PP<=22 THEN RR=PP:GOSUB
     5580:RETURN
5540 FOR QQ=23 TO 1 STEP -1
5550 IF MID$(XX$,QQ,1)=" "THEN
     RR=QQ -1:GOSUB 5580:
     XX$=RIGHT$(XX$,PP-QQ):GOTO
     5520
5560 NEXT QQ
5570 RR=22:GOSUB 5580:XX$=RIGHT$
     (XX$,PP-22):GOTO 5520
5580 REM SUBROUTINE INTERNA
5590 PRINT LEFT$(XX$,RR):IF RR<22
     THEN PRINT
5600 RETURN
```

Un adatto programma di gestione per questa subroutine è:

```
10 X1$="BATTI QUALSIASI STRINGA
   LUNGA FINO A TRE RIGHE PER
   PROVARE SUBROUTINE TRACCIATO"
20 GOSUB 5500
30 INPUT X1$: GOSUB 5500
40 GOTO 10
```

ESPERIMENTO 21·3

(a) L'utente di un programma batte una stringa che contiene un numero ed eventualmente una parola (ma non necessariamente) separati da 1 o più spazi. La stringa di input potrebbe essere

```
        3 APPLES
oppure 174 PETS
oppure   1 QUEUE
```

Scrivere un programma che estrae la parola e il numero e li visualizza nell'ordine opposto con il numero raddoppiato cioè:

```
APPLES   6
PETS   348
QUEUE    2
```

SUGGERIMENTO: Usare MID$ e VAL per estrarre i numeri.

(b) In risposta ad una domanda un utente potrebbe scrivere una stringa tipo questa:

```
IO VOGLIO 6 ARANCE 17 MELE
2 POMPELMI 157 NOCI
E 15 MELONI
```

Scrivere una subroutine che analizza tale stringa e imposta le variabili come segue:

Matrice N1$:	I *nomi* delle varie voci richieste
Matrice Q1:	Le quantità delle varie voci
Variabili X:	Numero delle diverse voci richieste

Per esempio, la frase suddetta dovrebbe dare:

```
N1$(1) = "ARANCE"      Q1(1) = 6
N1$(2) = "MELE"        Q1(2) = 17
N1$(3) = "POMPELMI"    Q1(3) = 2
N1$(4) = "NOCI"        Q1(4) = 157
N1$(5) = "MELONI"      Q1(5) = 15

X = 5
```

La subroutine dovrebbe ignorare le parole IO, VOGLIO, VORREI, E.
SUGGERIMENTO: analizzare la stringa con un puntatore e usare MID$ per estrarre sequenze di lettere o cifre ciascuna terminante con uno spazio.

Esperimento 21.3 completato	

Il quiz di autotest per questa unità è intitolato UNIT21QUIZ.

UNITA':22

ALTRI USI DELLE MATRICI — Ricerca e riordino

Questa unità introduce ulteriormente allo studio delle matrici e del modo in cui sono usate. Si esamineranno la *ricerca* e il *riordino*, due tecniche che sono estremamente importanti per molte moderne applicazioni di computer.

L'esperimento al termine dell'Unità 20 chiederà di scrivere un programma per cercare in una lista di nomi contenuti in una matrice. Se ci sono solo pochi numeri questo processo non è difficile. Si inizia dall'alto e si lavora procedendo verso il basso, fermandosi soltanto quando si trova un esatto accoppiamento oppure quando si raggiunge il fondo della lista e si esauriscono i nomi da ricercare. Un frammento di codice che comporta tale richiesta, potrebbe essere il seguente:

```
120 FOR J= 1 TO 12
130 IF X$=A$(J) THEN 170
140 NEXT J
150 PRINT "NESSUN ABBINAMENTO"
160 STOP
170 PRINT "ABBINAMENTO IN"; J
180 STOP
```

Glossario

X$: Nome da cercare
A$(1-12): Matrice di nomi da cercare
J: Puntatore all'elemento corrente in A$

Nel discutere i metodi di ricerca il nome (o numero) che viene cercato è detto "bersaglio" e l'atto di abbinarlo a fronte di un'entrata nella lista è detto un "confronto". Nell'esempio, il bersaglio è in X$ e il confronto si verifica nella riga 130.

In pratica, le liste di nomi fra i quali cercare sono spesso molto più lunghe. L'elenco del telefono di Londra, per esempio, contiene all'incirca 2 milioni di nomi. Se il programma dovesse cercare nell'intero elenco da cima a fondo, occorrerebbe effettuare 2 milioni di confronti. Ciò richiederebbe un tempo lunghissimo anche alle estreme velocità dei computer.

Fortunatamente ci sono delle scorciatoie per eseguire queste elaborazioni. Si supponga che i nomi della lista siano "selezionati" o riordinati in ordine alfabetico crescente. È possibile usare questo fatto per organizzare la ricerca. Per esempio, è possibile iniziare confrontando il bersaglio con un nome prossimo al centro della lista. Si potrebbe essere abbastanza fortunati e scoprire un abbinamento; ma se ciò non succede, si verifica uno dei due risultati seguenti:

(a) La parola bersaglio è *minore* (e cioè più vicina all'inizio della lista) della parola centrale

oppure

(b) La parola bersaglio è *maggiore* (e cioè più vicina al termine della lista) della parola centrale

Nel primo caso è possibile essere sicuri che se il bersaglio si trova nella lista, si troverà nella prima metà. Analogamente il secondo caso dice che il bersaglio può essere soltanto nella seconda metà. In entrambi i casi si è fatto in modo di eliminare metà della lista con due confronti — uno di uguaglianza e uno di ordine relativo.

Una volta conosciuta quale metà della guida usare, è possibile applicare lo stesso processo a quella metà e identificare un *quarto* della lista originale e quindi un ottavo e così via.

Illustriamo ora il processo. Si supponga che la lista di nomi sia:

ANDREW
ANTONIA
BEATRICE
CHRIS
FRANCES
HENRY
JIM
JOAN
JULIA
OLIVE
PETER
SUSAN
TIMOTHY
TOM
WILLIAM

Si userà TOM come parola bersaglio. Per cominciare, lo confrontiamo con il nome intermedio della lista che è JOAN. Ora TOM<>JOAN, cosicchè non registriamo alcun abbinamento. Inoltre, TOM>JOAN, così possiamo eliminare tutta la lista da JOAN in su e concentrare la ricerca nella parte da JULIA fino alla fine.

La parola centrale in questa parte è SUSAN. TOM>SUSAN, cosicchè scartiamo di nuovo tutta la lista salvo il pezzettino tra TIMOTHY e WILLIAM.

La parola centrale della rimanente sezione è TOM, che dà un abbinamento diretto. Se la parola bersaglio non compare nella lista, tutto diventa rapidamente ovvio in quanto la dimensione della lista da cercare si riduce a nulla. Per esempio, se si prende il bersaglio GEORGE:

fase 1: GEORGE<JOAN, cosicchè usiamo la lista ANDREW–JIM (7 nomi)

fase 2: GEORGE>CHRIS, cosicchè usiamo la lista FRANCES–JIM (3 nomi)

fase 3: GEORGE<HENRY, cosicchè usiamo lista FRANCES–FRANCES (1 nome)

fase 4: GEORGE>FRANCES. Non è possibile alcuna ulteriore suddivisione per cui GEORGE non può essere nella lista.

A questo punto scegliere alcuni nomi, alcuni nella lista altri no e ripetere il processo di ricerca seguendo il metodo appena descritto.

LA RICERCA DICOTOMICA

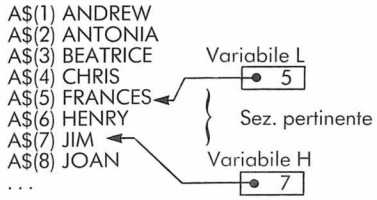
Se ci si pensa, si vedrà che ad ogni fase, la dimensione della lista da cercare viene all'incirca dimezzata. Ne segue che se si raddoppia la dimensione della lista, si aggiunge soltanto uno stadio al processo di ricerca. Quando ci si muove in liste più grandi, si inizia ad acquistare un enorme vantaggio rispetto ai metodi che si basano sulla ricerca dall'alto al basso, osservando ogni nome. Il metodo "veloce" richiede all'incirca 12 confronti per una lista con 1000 nomi o 21 per una lista con 1 milione. Dato che esso si basa sulla divisione della distinta, il metodo è detto "ricerca dicotomica".

Codifichiamo ora il metodo in BASIC. Supponiamo che la lista da esaminare contenga 100 voci e si possa trovare negli elementi da A$(1) a A$(100) della matrice A$. La parola bersaglio è X$. Per organizzare il processo occorre identificare la parte della lista in cui deve essere svolta la ricerca. Per far ciò, si useranno due variabili come *puntatori*.

H 'punta' alla parte superiore della parte pertinente (che è cioè l'elemento con l'indice più elevato)

L 'punta' alla base della pertinente sezione (l'elemento con l'indice più basso)

La frase "punta a" significa "contiene l'indice di". Ciò è illustrato qui di seguito:



Pertanto la parte "interessante" della lista inizia da A$(L) e termina ad A$(H). Se al limite si trova che H è minore di L, la dimensione della lista è 0 e la ricerca è fallita.

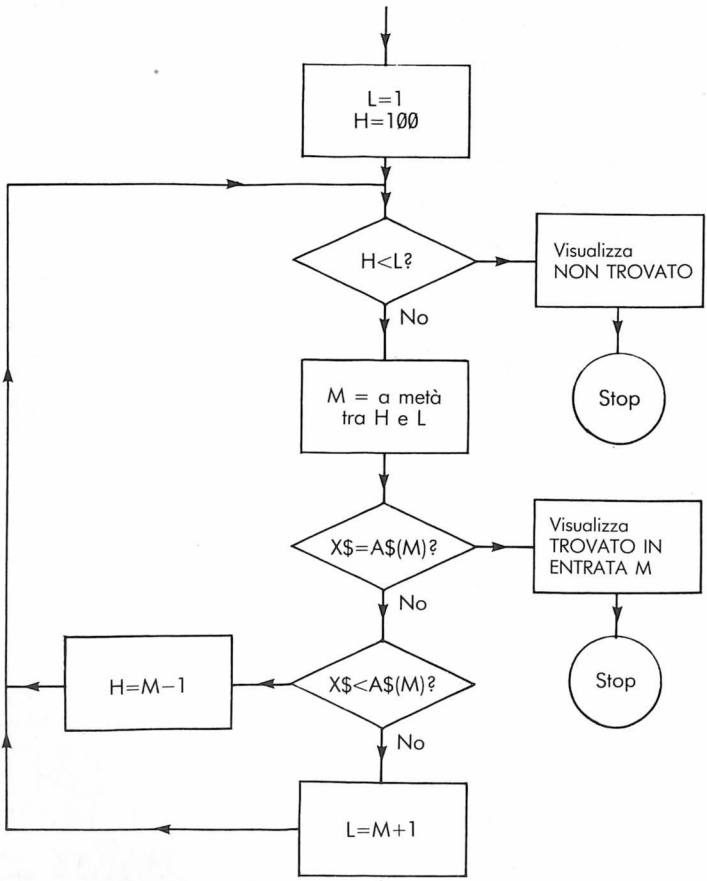
Trovare la parola centrale della parte interessante è abbastanza facile. Il suo indice è la "media" di H e L, ridotto a un numero intero se necessario. L'appropriata espressione è

```
INT(0.5*(H+L))
```

È come assegnare questo valore ad una variabile M.

Nella pianificazione dell'algoritmo dobbiamo pensare accuratamente alla modifica dei valori di H e L. Si supponga di trovare che la parola bersaglio è maggiore della parola centrale A$(M). Ciò ci fornisce un nuovo limite inferiore di L=M+1, ma non cambia per nulla il limite superiore H. Analogamente se la parola bersaglio è minore di A$(M) il nuovo limite superiore H sarà M−1, ma L non dovrà essere cambiato.

È possibile riprodurre queste idee in uno schema di flusso:



Glossario

X$: Parola bersaglio
A$(1-100): Lista di parole in cui cercare (in ordine alfabetico)

L,H: Puntatori alla parte attiva della lista A$
M: Punto intermedio della parte attiva della lista

E un corrispondente frammento di codice potrebbe essere:

```
230 L=1: H=100
240 IF H<L THEN PRINT X$; "NON
    TROVATO":STOP
250 M=INT(0.5*(H+L))
260 IF X$=A$(M) THEN PRINT X$;
    "TROVATO IN ENTRATA";M:STOP
270 IF X$<A$(M) THEN H=M-1:GOTO
    240
280 L=M+1:GOTO 240
```

ESPERIMENTO 22·1

Trasformare il codice di ricerca in una subroutine con le seguenti specifiche:

Specifiche della subroutine

Scopo: Cercare nella lista ordinata A$ tra le voci H1 e L1, l'entrata X$.

Righe: 6000-6100
Parametri di input: H1: Limite superiore di ricerca
L1: Limite inferiore di ricerca
X$: Parola bersaglio
Output: Se si trova una copia di X$ in A$, M1 è il suo indice. Se non si trova una copia M1=1

Provare la subroutine con il seguente programma di gestione:

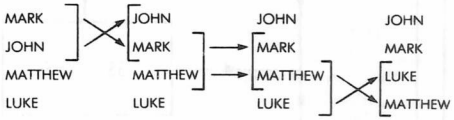
```
10 DATA BAIN, BEAVIS, BOWEY, BURNS,
   CLARK, FLEMING
20 DATA GORDON, GREEN, HOOD,
   KIDD, MACCABE, MALLY
30 DATA MARSHALL, MILLER, NORTH,
   PACK, PERKINS, REED, ROSE
40 DATA ROSS, SIMPSON, SMITH,
   SYKES, TEDFORD, WEBSTER, WOOD
50 DIM A$(26)
60 FOR J=1 TO 26:READ A$(J):NEXT J
70 INPUT"BATTI UN NOME"; X$
80 L1=1:H1=26:GOSUB 6000
90 IF M1=-1 THEN PRINT X$; "NON
   TROVATO":GOTO 70
100 PRINT X$; "TROVATO ALLA
   POSIZIONE"; M1
110 GOTO 70
```

Esperimento 22.1 completato

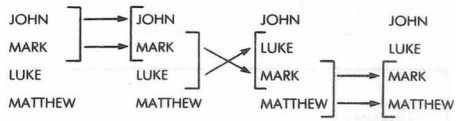
METODO DI ORDINAMENTO "BUBBLE SORT"

Finora in tutti gli esempi i nomi erano comodamente in ordine alfabetico quando il programma iniziava. Si supponga ora che i nomi vengano forniti in ordine casuale. Occorre pertanto ridisporli o riordinarli.
In un elenco riordinato di nomi, è possibile prendere qualsiasi coppia: quella con l'indice maggiore sarà alfabeticamente maggiore o uguale a quella con l'indice minore. Questo fatto è la base del metodo di ordinamento "Bubble sort" che scambia una coppia di numeri fuori sequenza sostituendo l'uno con l'altro.

Si inizia con una lista di nomi in ordine casuale, la si esamina e si confrontano le successive coppie di nomi (1 e 2, 2 e 3 e così via). Se si trova qualsiasi coppia fuori sequenza, questi nomi vengono scambiati; ciascuno viene spostato nello spazio precedentemente occupato dall'altro. Ecco un esempio di tale spostamento:



Questa operazione porterà sempre il nome più grande in basso ma non lascerà necessariamente l'intera lista in ordine. Occorre riesaminarla di nuovo ripetutamente fino a che non si devono più effettuare altri spostamenti. In questo caso il secondo esame darebbe:



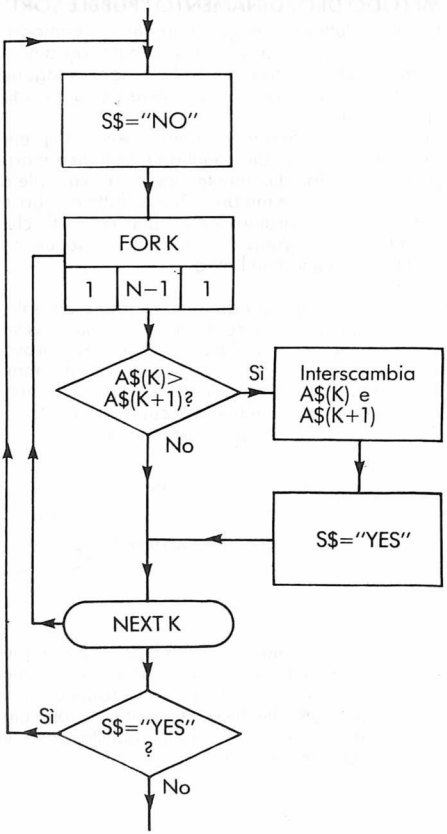
E il terzo esame non darebbe alcun interscambio indicando così che la lista è in ordine.
L'interscambio di due variabili non è altrettanto semplice quanto sembra. Se si cerca di scambiare i valori di X e Y scrivendo

```
X=Y : Y=X
```

la cosa non funziona; il primo comando distrugge il valore iniziale di X ed entrambe le variabili terminano con il valore originale di Y. Occorre una terza variabile temporanea — ad esempio D — per contenere il valore di X fino a che occorre:

```
D=X : X=Y : Y=D
```

E lo schema di flusso per il metodo di ordinamento "Bubble sort" è il seguente:



Glossario

S$: Segnalatore per interscambi
A$(N-1): Matrice di parole da riordinare
N: Numero di parole da riordinare
K: Puntatore a A$

La codifica corrispondente:

```
....
130 S$="NO"
140 FOR K=1 TO N–1
150 IF A$(K)>A$(K+1) THEN D$=A$(K):
    A$(K)=A$(K+1):A$(K+1)=D$:
    S$="YES"
160 NEXT K
170 IF S$ = "YES" THEN 130
....
```

ESPERIMENTO 22·2

Trasformare il metodo di ordinamento "Bubble sort" in una subroutine con la seguente specifica:

Specifiche di subroutine

Scopo: Riordinare campi in ordine alfabetico

Numeri di riga: 6500-6580

Parametri: *Input:* Lista di voci da riordinare in A1$(1) fino a A1$(N1)
Output: La lista riordinata compare da A1$(1) ad A1$(N1)

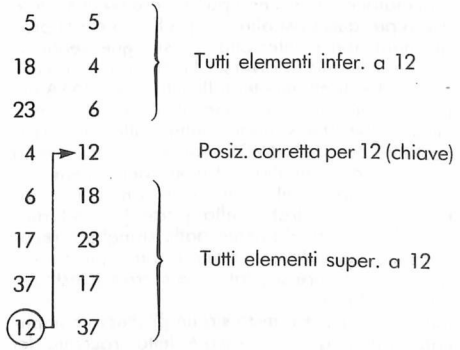
Variabili locali: KK, SS$, DD$

Provare con propri dati.

Esperimento 22.2 completato	

QUICKSORT

Il metodo di ordinamento "Bubble sort" è interessante e semplice se la lista è abbastanza corta (ad esempio 10 voci o meno) ma come la lista cresce, ogni esame diventa sempre più lungo e occorrono sempre ulteriori passate, cosicchè il tempo richiesto per il lavoro cresce col quadrato del numero di voci da riordinare. Ciò significa che una lista di 50 voci richiederà circa 25 volte il tempo necessario per selezionare una di 10.
Esistono parecchi metodi di selezione o di riordino che sono molto più veloci di questo. Uno di essi è chiamato "Quicksort" è stato inventato da C.A.R. Hoare. In termini grossolani, il tempo che esso richiede cresce soltanto con il numero di elementi da riordinare. Quicksort usa una tecnica di programmazione detta *recorsione* in cui una subroutine richiama se stessa per svolgere una parte del proprio lavoro. Molti trovano il metodo duro da comprendere particolarmente se è espresso in BASIC che non è stato progettato pensando a programmi recorsivi. Per usare efficacemente Quicksort non occorre comprenderlo; ciononostante qui c'è una breve spiegazione che si riferisce alla codifica che viene data.
Il metodo inizia con una lista di voci che non sia ordinata in alcun modo speciale. Esso prende quello inferiore, lo chiama elemento "chiave" e lo dispone nel suo posto finale corretto nella lista, assicurandosi che tutti gli elementi al di sopra di esso gli siano inferiori e tutti gli elementi al disotto gli siano superiori. Ciò avviene interscambiando le voci o gli elementi se necessario. Per esempio, qui di seguito è indicata la prima fase nel riordino di una lista di 8 elementi:



La seconda fase consiste nel riordinare tutti gli elementi al disopra dell'elemento chiave e la terza fase selezionare tutti quelli aldisotto dell'elemento chiave. Per entrambe queste fasi, la subroutine richiama se stessa recorsivamente, per cui il riordino di una parte di una lista non diventa altro che il problema di riordinarla tutta. Una buona parte della subroutine che segue è provvista dal meccanismo della recorsività. La matrice SS% e la variabile puntatore PP sono usate per ricordare esattamente ciò che sta accadendo a qualsiasi "livello" di controllo cosicchè tutte le chiamate e i ritorni avvengano in maniera ordinata. Il comando 6170 è l'equivalente di un RETURN.

Nella subroutine, le righe da 6040 a 6090 eseguono la fase 1; quelle da 6100 a 6130 si occupano della fase 2 (che può essere saltata se la "lista" al disopra dell'elemento chiave comprende meno di 2 elementi). Le righe da 6140 a 6160 agiscono alla fine della fase 3 e le righe 6010, 6020, 6110, 6130, 6150 e 6170 sono tutto ciò che occorre per la recorsione. La subroutine comprende due aspetti non familiari: un nome di matrice che termina con % e un comando con la parola chiave ON. Entrambi verranno discussi più avanti.

Specifiche della subroutine

Scopo: Riordinare gli elementi in ordine numerico, usando l'algoritmo Quicksort di Hoare

Numeri di riga: da 6000 a 6180

Parametri: *Input:* Lista di numeri da riordinare in A1(1) fino a A1(N1). Numero degli elementi in N1
Output: La lista riordinata compare da A1(1) a A1(N)

Variabili locali: SS, SS%, AA, BB, XX, YY, ZZ, DD, PP

NOTE: (i) SS% non deve essere usato altrove nel programma se la subroutine è richiamata più di una volta.
(ii) La subroutine può essere usata per riordinare stringhe invece di numeri se vengono effettuate al suo interno le seguenti sostituzioni:

A1$ per A1; ZZ$ per ZZ; DD$ per DD

```
6000 REM QUICKSORT:RIORDINA N1
     ELEMENTI DI A1
6010 IF SS=1 THEN 6030
6020 DIM SS%(N1):SS=1:REM DICHIARA
     CATASTA
6030 AA=1:BB=N1:SS%(0)=1:PP=1
6040 XX=AA: YY=BB: ZZ=A1(BB)
6050 IF XX>=YY THEN 6090
6060 IF A1(XX)<=ZZ THEN XX=XX+1:
     GOTO 6050
6070 IF A1(YY) >=ZZ THEN YY=YY-1:
     GOTO 6050
6080 DD=A1(YY):A1 (YY) = A1 (XX):
     A1(XX)=DD: GOTO 6050
6090 A1(BB) = A1(XX):A1(XX)=ZZ
6100 IF XX-AA <= 1 THEN 6140
6110 SS%(PP)= XX: SS%(PP+1)=BB:
     SS%(PP+2)=2: PP=PP+3
6120 BB=XX-1: GOTO 6040
6130 PP=PP-3: XX=SS%(PP):
     BB=SS%(PP+1)
```

```
6140 IF BB-XX<=1 THEN 6170
6150 SS%(PP)=3: PP=PP+1: AA=XX+1:
     GOTO 6040
6160 PP=PP-1
6170 ON SS%(PP-1) GOTO 6180, 6130,
     6160
6180 RETURN
```

Questa completa subroutine intitolata "QUICK-SORT" può essere trovata sulla cassetta di nastro.

CONFRONTO DEI TEMPI DI RIORDINO

Quicksort è così più complicato di Bubble sort che ci si potrebbe domandare se vale la pena di usarlo. È possibile giudicare da soli da questa tabella che mostra i tempi necessari per riordinare matrici di varie dimensioni. Le cifre sono state trovate eseguendo entrambi i tipi di riordino su un VIC e cronometrandoli.

Dimens. matrice	*Tempo (Quicksort)*	*Tempo (Bubble sort)*
20	2	5
40	5	22
60	8	47
80	14	93
100	17	138
120	20	192
140	24	282
160	27	357
180	31	445
200	37	569

LA CAPACITÀ DI MEMORIA DEL VIC

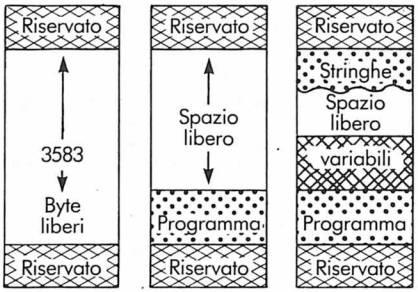
Quando si inizia ad usare le matrici, subito si presenta il problema di uno spazio nella memoria del VIC. Ciò in quanto le matrici "arraffano" una grande quantità di spazio molto rapidamente; ciascun elemento di una matrice numerica usa fino a 5 byte di memoria e ogni elemento stringa usa 3 byte, più lo spazio necessario per la stringa stessa. Ci sono poi piccoli sovraccarichi addizionali per ciascuna matrice.
In questa sezione osserveremo il modo in cui la memoria del VIC è organizzata. Un indicatore utile è la funzione incorporata FRE(0) che dice in qualsiasi momento quanti byte rimangono inutilizzati. Quando si accende il VIC, compare il messaggio:

```
3583 BYTES FREE (di più se è montata una
     espansione di RAM)
```

Ora se si batte

```
PRINT FRE(0)
```

la macchina risponde 3581, in quanto 2 byte sono stati usati per obbedire alla funzione FRE.
La situazione generale è indicata nella parte (i) del programma che segue; dei 5120 byte nel VIC, 1537 sono riservati per vari scopi e i restanti sono ancora liberi.



(i) (ii) (iii)

Successivamente si potrebbe battere un programma o caricarne uno da una cassetta. Il programma è inserito nella parte inferiore della sezione libera della memoria, occupando all'incirca 1 byte per ogni carattere. Il risultato è indicato nel diagramma parte (ii).
Ora si inizia il programma. La macchina inizia a rispondere ai comandi e non appena incontra qualsiasi variabile cui si fa riferimento per la prima volta, alloca lo spazio necessario nell'area immediatamente adiacente al programma stesso. Ai comandi DIM è attribuito uno spazio nella stessa area e lo spazio può esaurirsi rapidamente; un comando apparentemente innocente tipo

```
DIM A(200)
```

costerà oltre 1000 byte.
Una volta allocato lo spazio per una variabile o una matrice, questo non può essere recuperato e usato per qualsiasi altro scopo fino a che il programma non è interrotto. Le stringhe vengono gestite in maniera diversa. Le stringhe che vengono lette direttamente dalle istruzioni DATA nel programma non occupano alcun spazio in più. Le stringhe che vengono lette dalla tastiera o costruite con "+", MID$ e altre funzioni stringa sono disposte all'altra estremità della memoria, lasciando spazio libero tra le stringhe e le variabili. Ciò è illustrato nella parte (iii) del diagramma. Lo spazio usato dalle stringhe è recuperabile; quando una stringa non è più necessaria, può essere scartata e lo spazio è restituito all'area libera.
(Se si pensa che questo sia un processo complicato, si è nel giusto — esso è detto "raccolta dei rifiuti". Fortunatamente è completamente automatico e non occorre saperne nulla). A parte il limite della dimensione, la memoria del VIC non è partizionata in alcun modo. È possibile avere programmi, variabili e stringhe a piacere posto che lo spazio totale non superi quello libero disponibile.
Impostare ed eseguire i seguenti programmi e pensare ai risultati alla luce di quanto detto:

```
10 PRINT "SPAZIO LIB",
   "DOPO RIGA"
20 PRINT FRE(0),10
30 X=0
40 PRINT FRE(0), 30
50 DIM A(20)
```

```
 60 PRINT FRE(0),50
 70 DIM N$(5)
 80 PRINT FRE(0),70
 90 C$="UNA STRINGA"
100 PRINT FRE(0),90
110 D$="ALTRA" + "STRINGA"
120 PRINT FRE(0),110
130 D$=""
140 PRINT FRE(0),130
150 C$=""
160 PRINT FRE(0),150
170 STOP
```

ESAURIMENTO DELLO SPAZIO DI MEMORIA DEL VIC

È possibile ora spiegare i vari modi per cui si esaurisce lo spazio.

(a) Se il programma è troppo lungo, si tocca il limite di spazio prima di finire di immetterlo. Ciò è molto più probabile che succeda se si cerca di caricare il programma che è stato sviluppato su un VIC con una memoria più ampia.

(b) Se ci sono troppe variabili o se le matrici sono troppo lunghe, si ottiene un messaggio

OUT OF MEMORY (ESAURITA MEMORIA)

quando la macchina cerca di allocare una nuova variabile o di eseguire un comando DIM.

(c) Se si cerca di produrre e memorizzare troppe stringhe contemporaneamente, si ottiene ancora una volta un messaggio OUT OF MEMORY.

L'esaurimento dello spazio in memoria è sempre un'esperienza frustrante. Si ha la sensazione che se solo la macchina avesse un poco più di memoria, il programma funzionerebbe perfettamente. Ecco alcuni modi per superare la difficoltà:

1. Ridurre la dimensione delle matrici al minimo. Non cercare di indovinare il numero di voci che l'utente cercherà di fornire — fare in maniera che il programma trovi le dimensioni delle sue matrici di conseguenza. Per esempio è generalmente meglio scrivere:

```
10 INPUT N
20 DIM X(N),Y$(N)
```

che

```
10 DIM X(100), Y$(100)
20 INPUT N
```

2. Se ci sono matrici con numeri e si sa — per certo — che ogni elemento è destinato ad essere un numero intero nel campo da −32767 a +32767, è possibile usare *matrici intere*. Le matrici intere hanno nomi che terminano in % (come A1% e JJ%) e usano soltanto 2 byte per elemento anzichè i soliti 5. Per modificare un programma per usare matrici intere, occorre cambiare ad esempio

```
10 DIM N(500)
```

in

```
10 DIM N%(500)
```

e quindi ogni menzione di un elemento di matrice tipo N(J) in N%(J).

3. Man mano che il programma viene eseguito, è bene fare in modo di scartare tutte le stringhe che non servono più. Per far ciò, si assegna la stringa nulla all'appropriata variabile:

```
X$=M$+"SPOSATO" + "F$"
.....
..... (X$ non serve più)
X$=""
```

4. Esaminare il programma accuratamente e vedere se è possibile trovare un algoritmo che richiede meno spazio di quello che si sta usando. Occorre realmente memorizzare tutti gli elementi di una matrice oppure è possibile calcolare e usarli uno per uno?

5. Se si usa un numero lungo o una costante stringa in parecchi punti, è bene assegnarla ad una variabile e quindi usare il nome della variabile in sua vece. Per esempio si consideri:

```
 90 PRINT "RISULTATO CONFERMATO
    COME";
100 PRINT X/2.718281828
110 GOTO 150
120 Y=Z/2.718281828
130 Q=Y+2.718281828
140 PRINT "RISULTATO CONFERMATO
    COME"; Q+2
150 ......
```

Questo programma occuperà un po' meno spazio se lo si scrive nella forma:

```
 10 L$="RISULTATO CONFERMATO
    COME": E=2.718281828
.....
 90 PRINT L$;
100 PRINT X/E
110 GOTO 150
120 Y=Z/E
130 Q=Y+E
140 PRINT L$;Q+2
150 .....
```

Ricordarsi che il carattere π (1 byte) sta per 3.14159165... questo è possibile provando a battere PRINT π.

6. Acquistare cartucce di memoria extra dal rivenditore Commodore che danno 3K, 8K o 16K RAM in più.

Se si è realmente alla ricerca di spazio, ecco alcune altre cose che è possibile fare per restringere i programmi in uno spazio limitato. Si tratta di cose non realmente consigliate, che rendono il programma più difficile da comprendere e possono introdurre degli errori.

1. Rimuovere i messaggi visualizzati sullo schermo e basarsi sulle spiegazioni scritte.

2. Accertarsi se è possibile usare in comune variabili per più di uno scopo.

3. Esaminare il programma e rimuovere qualsiasi spazio (salvo che nelle stringhe). Per esempio si risparmieranno 5 byte cambiando

   IF A<5 AND B>7 THEN 400
   in IFA<5A AND B>7 THEN 400

   Si risparmierà inoltre un po' di spazio inserendo il maggior numero possibile di comandi in ciascuna riga.

4. Come azione finale, togliere i commenti (REM) al programma, imitando l'eroe Phileas Fogg di Giulio Verne, che per arrivare a Liverpool in tempo per vincere la scommessa distrusse il ponte della sua nave e lo bruciò nella caldaia. Egli vinse la gara, ma rovinò in tal modo la nave.

MATRICI BIDIMENSIONALI

Come è possibile vedere, le matrici sono utili nei problemi in cui il programma deve gestire contemporaneamente molte variabili diverse. In qualche problema è naturale disporre queste variabili in una tabella quadrata o rettangolare anzichè in una semplice lista ordinata. Si consideri un programma per giocare a scacchi. Esso deve "conoscere" quale pezzo occupa eventualmente ciascuna casella della tastiera. Ogni quadrato o casella può essere ripetuto da una variabile il cui valore riflette il pezzo di quella casella. Le 64 variabili sono disposte in una tabella con 8 file e 8 colonne, che formano il profilo della scacchiera. Il BASIC consente l'uso di matrici bidimensionali (e anche tridimensionali o più). Una tipica dichiarazione di una matrice bidimensionale sarebbe la seguente:

```
DIM X(5,7)
```

Questo comando imposta una matrice denominata X in cui ogni elemento è un numero. La matrice ha (5+1) ossia 6 file e (7+1) ossia 8 colonne: 48 elementi in tutto. Ecco un'immagine di tale matrice:

	Ø	1	2	3	4	5	6	7
Ø	X(Ø,Ø)	X(Ø,1)	X(Ø,2)	X(Ø,3)	X(Ø,4)	X(Ø,5)	X(Ø,6)	X(Ø,7)
1	X(1,Ø)	X(1,1)	X(1,2)	X(1,3)	X(1,4)	X(1,5)	X(1,6)	X(1,7)
2	X(2,Ø)	X(2,1)	X(2,2)	X(2,3)	X(2,4)	X(2,5)	X(2,6)	X(2,7)
3	X(3,Ø)	X(3,1)	X(3,2)	X(3,3)	X(3.4)	X(3,5)	X(3,6)	X(3,7)
4	X(4,Ø)	X(4,1)	X(4,2)	X(4,3)	X(4,4)	X(4,5)	X(4,6)	X(4,7)
5	X(5,Ø)	X(5,1)	X(5,2)	X(5,3)	X(5,4)	X(5,5)	X(5,6)	X(5,7)

Ciascun elemento della matrice ha due indici: un numero di fila e uno di colonna. Per esempio X(3,4) è nella fila 3 e nella colonna 4. A parte questa fondamentale differenza, tutto ciò che si conosce a proposito delle matrici unidimensionali, può essere esteso a quelle bidimensionali. Passiamo ad un'*illustrazione*. Si supponga di aver eseguito un'indagine e scoperto il prezzo di alcune merci base in ciascuno dei cinque negozi della propria zona. Si potrebbe esprimere i risultati in una tabella di questo genere

	FINE FARE	ASDA	SAINS-BURYS	CO-OP	FRASERS
FARINA	29	31	27	26	32
PATATE	15	12	13	24	33
BURRO	47	49	40	45	39
ZUCCHERO	22	20	19	27	29
FORMAGGIO	94	80	103	107	99
MELE	32	18	22	27	21
	(Tutti i prezzi sono espressi in Lire all'ettogrammo)				

Il problema da risolvere è, data una particolare lista di acquisti, quale è il negozio più a buon mercato da visitare? Un'immagine "dell'utente" del programma potrebbe essere:

```
INDICA FABBISOGNO
IN ETTI.

FARINA?          3
PATATE?         14
BURRO?           1   <-- Battuto dall'utente
ZUCCHERO?        Ø
FORMAGGIO?       2
MELE?            3

MEGLIO COMPRARE DA ASDA
DOVE PAGHI
3 ETTI FARINA       : 93
14 ETTI PATATE      :168
1 ETTO BURRO        : 49
Ø ETTI ZUCCHERO     : Ø
2 ETTI FORMAGGIO    :160
3 ETTI MELE         : 54
   TOTALE           :524
```

L'algoritmo base è lineare:

Iniziamo a scegliere alcune variabili e i loro nomi. Certamente avremo necessità di memorizzare il nome dei negozi e i vari articoli delle merci. Le variabili adatte sono:

da F$(1) a F$(6) per i cibi
e da S$(1) a S$(5) per i negozi

Ora, abbiamo bisogno di matrici per mostrare la quantità di ciascun cibo necessario e il corrispondente importo pagato (o totale) presso ciascun negozio. Le variabili adatte sono:

da F(1) a F(6) per le quantità e da T(1) a T(5) per i totali.

Infine, useremo una matrice bidimensionale per contenere la tavola dei prezzi. Una dichiarazione del tipo

DIM P(6,5) potrebbe andare bene.

Notare che abbiamo sistematicamente ignorato elementi con indici di 0. Ciò è comune nei piccoli problemi.
Il codice effettivo è abbastanza semplice, anche se un pochino lungo. Eccolo:

```
 10 DATA FARINA, PATATE, BURRO,
    ZUCCHERO, FORMAGGIO, MELE
 20 DATA FINE FARE, ASDA,
    SAINSBURYS, COOP, FRASERS
 30 DATA 29, 31, 27, 26, 32
 40 DATA 15, 12, 13, 24, 33
 50 DATA 47, 49, 40, 45, 39
 60 DATA 22, 20, 19, 27, 29
 70 DATA 94, 80, 103, 107, 99
 80 DATA 32, 18, 22, 27, 21
 90 DIM F$(6),S$(5), F(6), T(5), P(6,5)
100 FOR K=1 TO 6: READ F$(K): NEXT K
110 FOR J=1 TO 5: READ S$(J): NEXT J
120 FOR K= 1 TO 6: FOR J=1 TO 5
130 READ P(K,J)
140 NEXT J,K
150 PRINT "[SHIFT] e [CLR HOME] PREGO
    INDICARE FABBISOGNO"
160 PRINT "IN ETTI"
170 FOR K=1 TO 6
180 PRINT F$(K);: INPUT F(K)
190 NEXT K
200 FOR J=TO 5
210 FOR K=1 TO 6
220 T(J)=T(J)+F(K)*P(K,J)
230 NEXT K,J
240 M=T(1): N=1
250 FOR J=2 TO 5
260 IF T(J)<M THEN M=T(J): N=J
270 NEXT J
280 PRINT "MEGLIO COMPRARE"; S$(N)
290 PRINT "DOVE PAGHI": PRINT
300 FOR K=1 TO 6
310 PRINT F(K);"ETTI";
320 IF F(K)<>1 THEN PRINT "[SHIFT]
    e [CRSR] [SHIFT] e [CRSR] S.";
330 PRINT F$(K); TAB(15); F(K)*P(K,N)
340 NEXT K
350 PRINT: PRINT"TOTALE=";
    TAB(15);M;"LIRE"
360 STOP
```

Devono essere chiariti uno o due punti di scarsa importanza.

(a) Tutte le matrici sono dichiarate insieme in un comando. Ciò è più breve che scrivere

```
 90 DIM F$(6)
100 DIM S$(5)
```

e così via.
Il limite al numero di matrici che possono essere dichiarate è definito dalla lunghezza massima di riga: 88 caratteri.

(b) La sequenza dei comandi

```
NEXT J
NEXT K
```

può essere compressa in

```
NEXT J,K
```

Ciò si applica altrettanto bene a qualsiasi variabile di controllo e a qualsiasi numero di esse (quantunque sia raro il caso di trovarne più di due).

(c) La frase "TAB (15)" nel comando 350 fa sì che la macchina muova il suo cursore interno alla 15esima colonna nello schermo (se il cursore non vi si trova già). È usato per allineare l'importo pagato per ciascun articolo alimentare.
In generale, le parentesi possono contenere qualsiasi espressione. Pertanto il programma

```
10 FOR J=1 TO 20
20 PRINT TAB(J);"\"
30 NEXT J
40 STOP
```

visualizzerà una linea diagonale attraverso lo schermo.

Notare che J è sistematicamente usato per il numero di negozio e K per il numero di tipo di cibo. P(K,J) è pertanto il prezzo del cibo numero K presso il negozio numero J.

ESPERIMENTO 22·3

Questo esperimento è in due parti:

(a) Una classe con molti studenti dà un esame competitivo. L'insegnante produce una serie di punteggi del tipo

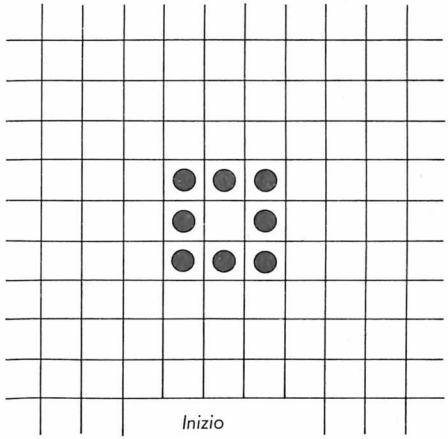
```
ADAMS    27
BRIGGS   66
CHILVERS 89
DALE     38
.....
```

e così via.
Le regole dicono soltanto il primo 25% (un quarto) degli studenti può passare l'esame.
Scrivere un programma che può leggere nella lista dei punteggi originali e visualizzare i nomi degli studenti che passano. Si supponga che non ci siano più di 100 studenti e che il voto dell'ultimo studente sia seguito dal nome fittizio "ZZZZ".
SUGGERIMENTO: riordinare una copia dei punteggi usando la subroutine QUICKSORT sulla cassetta di nastro e trovare il punteggio minimo abilitante a un quarto di strada lungo l'elenco riordinato. Usarlo per individuare gli studenti che superano l'esame.

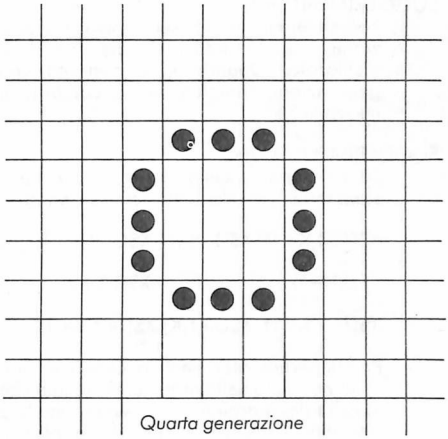
(b) Il gioco della "vita" è stato inventato da R. Convay, un matematico inglese. Esso riguarda la storia della vita di una colonia di piccoli insetti che vivono in un'area rettangolare, uno per ogni cella.
La colonia vive da generazione in generazione. Il fato di ciascun insetto è determinato dalle seguenti regole:

1. Se un insetto ha 1 o meno vicini immediati, muore di solitudine.
2. Se ha 4 o più vicini, muore di sovraffollamento.
3. Se ha 2 o 3 vicini, sopravvive alla successiva generazione.

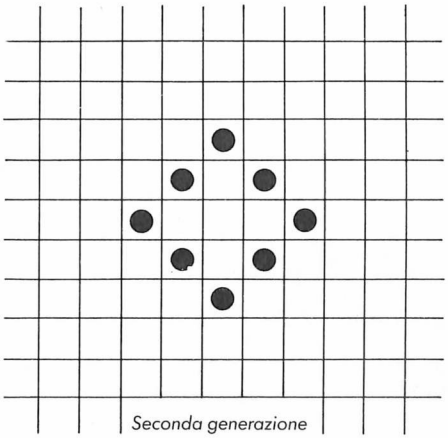
Inoltre, se una cella vuota ha esattamente 3 insetti nelle celle vicine, nasce un nuovo insetto in quella cella.
Per dare un esempio, si consideri



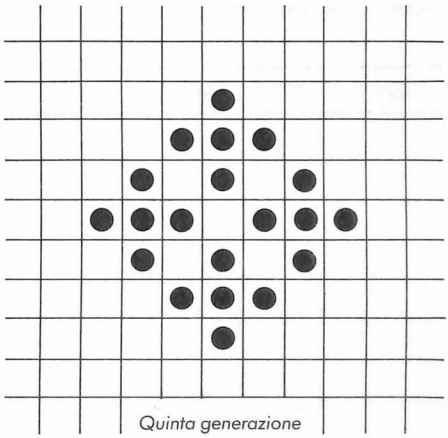
Inizio



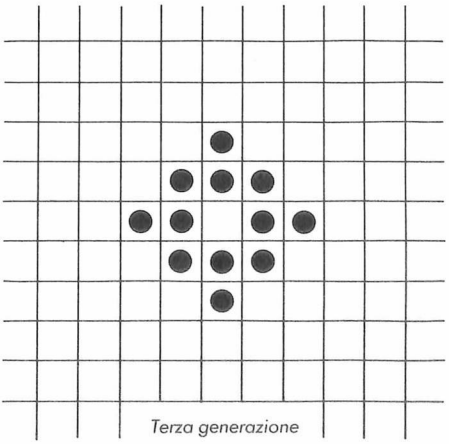
Quarta generazione



Seconda generazione



Quinta generazione



Terza generazione

Scrivere un programma per svolgere il gioco "della vita" su una matrice 9x9. Ciascun elemento dovrebbe essere una stringa che contiene un carattere (ad esempio "★" o uno spazio). Il programma dovrebbe leggere una "posizione di partenza" quindi visualizzare le successive generazioni fino a che non viene interrotto.
Si tratta di un esperimento sfidante che consente di usare il programma parziale "LIFERSTAR" sulla cassetta di nastro. Il programma legge una posizione di partenza nella matrice X$(9,9) (righe 10-180) quindi la visualizza (righe 190-260). Il programma parziale dichiara anche una matrice Y$(9,9) che si troverà utile per passare da una generazione alla successiva.

SUGGERIMENTO (1):

Per determinare la successiva generazione, usare la matrice Y$, che è già stata dichiarata. Quando si è costruita una generazione completa in Y$, copiarla di nuovo in X$.

SUGGERIMENTO (2):

Se si osserva la cella X$ (J,K) (dove J e K sono indici), le otto celle vicine saranno:

X$(J−1,K−1), X$(J−1,K), X$(J−1,K+1)

X$(J,K−1) X$(J,K+1)

X$(J+1,K−1), X$(J+1,K), X$(J+1,K+1)

Per impedire riferimenti a celle che non sono per nulla nella matrice, si assuma che le celle del margine siano sempre vuote e si limitino le operazioni alle 7 file e colonne "interne".

Controllare ora la risposta a fronte di quella indicata nell'Appendice C.

Esperimento 22.3 completato	

UNITA':23

UNO SGUARDO PIÙ RAVVICINATO ALL'INTERNO DEL VIC

Un computer è un dispositivo estremamente complicato. Se si cerca di spiegare tutto quanto lo riguarda ad un principiante, in una sola volta, lo si lascia frastornato e inevitabilmente confuso prima che egli possa fare qualcosa d'interessante o di utile. Per contro, è possibile trattare la macchina come un pacchetto ad un party di ragazzi: qualcosa con numerosi involucri di carta che si possono strappare uno alla volta. Se si nascondono i dettagli non necessari, è sempre possibile far sì che lo strato più esterno appaia abbastanza semplice. Per esempio, molte persone penseranno sempre al VIC esattamente come ad una macchina che esegue giochi che vengono forniti su una cassetta di nastro o su cartucce a innesto. Per coloro che non conoscono nulla di programmazione ciò è perfettamente ragionevole e un utile livello di conoscenza.

Alcune persone desiderano andare più a fondo. Il lettore avrà già capito che il VIC è una macchina che memorizza ed esegue programmi BASIC. Anche questo è un importante livello di conoscenza in quanto consente di usare la macchina in numerosi modi originali e interessanti, ma esclude dettagli sul modo in cui le informazioni sono memorizzate, sul modo in cui si esegue un programma e sul modo in cui effettivamente funziona.

In questa Unità si andrà un po' più in profondità verso il meccanismo interno del VIC. Si troverà che la descrizione della memoria del VIC sembra diversa dall'immagine presentata nelle unità precedenti. Ciò in quanto si sta osservando la memoria da un punto di vista nuovo e più ravvicinato. Entrambe le descrizioni sono vere e ciascuna è appropriata al livello al quale il sistema viene descritto.

Questa Unità esplora i misteriosi comandi PEEK e POKE. Occorre iniziare con due avvertenze:

1. A differenza del resto del manuale, il materiale in questa unità si applica soltanto al VIC e non può essere usato con qualsiasi altro computer. La maggior parte dei personal computer ammettono i comandi PEEK e POKE che però eseguono cose diverse, trattandosi di macchine diverse.

2. PEEK e POKE sono comandi che consentono di introdursi ulteriormente nel funzionamento interno del VIC più di qualsiasi altro comando BASIC. Ciò significa che viene escluso un livello di protezione. Un programma con errori può danneggiare il software del VIC e fare in modo che si comporti in maniera molto strana.
   Per esempio, la tastiera potrebbe risultare totalmente inattiva o sullo schermo potrebbero essere visualizzati sciami di caratteri strani. Inoltre il computer potrebbe rifiutarsi di obbedire a semplici comandi tipo LIST o RUN. Se ciò succede, è sempre possibile riguadagnare il controllo spegnendo la macchina per 30 secondi. Dato che ciò cancella il programma, è doppiamente importante memorizzare frequentemente il programma su una cassetta se si usano i comandi PEEK e POKE.
   La corruzione del software è un effetto provvisorio. È assolutamente impossibile provocare al VIC un danno permanente eseguendo qualsiasi programma, per quanto possa essere pieno di errori.

Per conoscere il PEEK e il POKE, occorre per prima cosa imparare un po' di più sul computer VIC vero e proprio.

Un diagramma del VIC (al livello appropriato a questo capitolo) è illustrato alla pagina successiva.

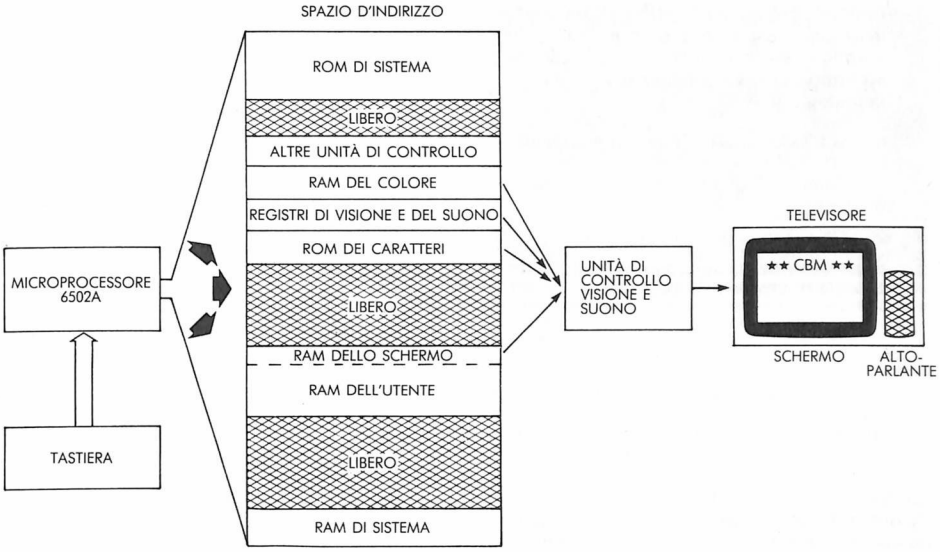


Figura 23.1

La macchina si compone di parecchie parti:

(a) Uno "spazio di indirizzo" che contiene moduli di memoria di diversi tipi.

(b) Un microprocessore che prende le istruzioni dalla memoria e le esegue. La maggior parte delle istruzioni si traducono in modifiche apportate ai contenuti della memoria. Le istruzioni sono simili ai comandi, ma più semplici che in un programma BASIC.

(c) Una tastiera che è controllata dal microprocessore.

(d) Una "unità di controllo visione e suono" che produce l'immagine e gli effetti sonori prodotti dal televisore.

Lo spazio di indirizzo è come una rastrelliera nella quale è possibile inserire separati "segmenti" di memoria. In tutto c'è spazio per 65536 byte di memoria e le varie fessure sono numerate da 0 a 65535. La maggior parte dei segmenti di memoria contengono 1024 byte (o multipli di 1024 ad esempio 2048 o 4096 o 8192) cosicchè il numero 1024 è detto un "kilo" o "K" per brevità. La capacità compresa nello spazio di indirizzo è esattamente 64 Kilobyte.
Lo spazio di indirizzo è riempito solo parzialmente e i segmenti di memoria che esso contiene sono di tre tipi diversi:

(a) RAM sta per "memoria ad eccesso casuale". Il contenuto di ciascun byte può essere letto e alterato quante volte si desidera.

(b) ROM è "memoria di sola lettura". I contenuti di ciascun byte sono fissi per sempre quando la ROM viene costruita. Una volta che la ROM è nella macchina, il VIC può leggere il byte ma non può alterarlo.

(c) I registratori e le unità di controllo sono dispositivi speciali che fanno lavori particolari come aiutare a produrre suoni e immagini. Anch'essi sono strutturati come la memoria cosicchè il microprocessore può leggere e cambiarne i loro contenuti.

L'ORGANIZZAZIONE DELLA MEMORIA DEL VIC:

Quando si compra un VIC modello base, si troverà lo spazio di indirizzo abitato come segue:

Indirizzi da 0 a 1023: 1K di RAM riservato dal VIC per proprio uso. È qui che la macchina tiene nota dell'ora, un record per la posizione del cursore e così via.

Indirizzi da 1024 a 4095; non occupati. (Qui è dove risiede la cartuccia RAM da 3K).

Indirizzi da 4096 a 8191: 4K di "RAM dell'utente" che serve a molti scopi: contiene il programma BASIC, i dati (variabili e stringhe) e 506 byte vicini alla parte superiore usati per controllare lo schermo del televisore in un modo che verrà descritto successivamente.

Indirizzi da 8192 a 32767: Liberi. (Qui è dove risiedono le cartucce a innesto RAM da 8 o 16K).

Indirizzi da 32768 a 36863: 4K di ROM che contiene una descrizione del "profilo" di ogni carattere che può essere visualizzato sullo schermo. Su questo punto si parlerà più in dettaglio più avanti.

Indirizzi da 36864 a 37887: questa area contiene i registri del suono e della visione e qualche altra unità di controllo che non occorre descrivere.

Indirizzi a 37888 a 38912: 1K di RAM dove sono usate 506 parole per controllare il colore di ciascun carattere sullo schermo.

Indirizzi da 38912 a 49151: vuoti.

Indirizzo da 49152 a 65535: 16K di ROM che contiene il software del VIC: i programmi che organizzano l'editing dello schermo, il caricamento e salvataggio di programmi, esecuzioni di programmi dell'utente.

Come è possibile vedere, c'è una sostanziale quantità di spazio libero — 37K in tutto. Questo spazio può essere riempito da moduli RAM che aumentato la quantità di RAM dell'utente o da moduli ROM che contengono programmi pronti. Ad esempio, la cartuccia RAM da 3K VIC 1210 che si inserisce a innesto sul dorso della macchina, occupa esattamente lo spazio di 3K tra 1024 e 4095.

COS'È UN BYTE?

Ora osserveremo il modo in cui ogni singolo byte è disposto. Esso si compone di 8 elementi separati detti bit, ciascuno dei quali può avere due possibili valori. In un byte ci sono 256 possibili combinazioni di bit. Se chiamiamo i valori di un bit 0 e 1, alcune delle combinazioni sono:

00000000 01011100 10101011

Il significato di un byte dipende interamente dal contesto in cui è usato. Nel VIC potrebbe essere:

(a) Un codice per un carattere specifico (ad esempio "A" potrebbe essere 00000001).

(b) Un profilo di un massimo di 8 punti sullo schermo.

(c) Un'istruzione per il microprocessore per eseguire qualche azione.

(d) Un numero, dove i bit sono interpretati da una regola matematica detta "sistema binario".

(e) Un byte può anche essere uno di un gruppo di 5 byte che costituiscono una variabile numerica o uno di molti in una variabile stringa.

Per valutare i byte è di aiuto conoscere il sistema binario. Fortunatamente non è molto difficile. Un byte da 8 bit può essere convertito in un numero normale con la seguente regola:

Il bit più a sinistra equivale a 128 se 1, zero se 0
Il successivo bit equivale a 64 se 1, zero se 0
Il successivo bit equivale a 32 se 1, zero se 0
Il successivo bit equivale a 16 se 1, zero se 0
Il successivo bit equivale a 8 se 1, zero se 0
Il successivo bit equivale a 4 se 1, zero se 0
Il successivo bit equivale a 2 se 1, zero se 0
Il bit più a destra equivale a 1 se 1, zero se 0

Per dare un esempio prendiamo il byte 10110110. I bit equivalgono a 128, 32, 16, 4 e 2 cosicchè il numero corrispondente è 128+32+16+4+2 ossia 182.
Talvolta occorre convertire un numero nel suo equivalente in byte. Il numero deve essere minore di 256 (e non minore di 0), perchè la conversione possa avvenire. Il metodo è:

1. Se è possibile sottrarre 128, farlo e scrivere un 1. Altrimenti scrivere uno 0.

2. Se è possibile sottrarre 64, farlo e scrivere un 1 alla destra del precedente simbolo. Altrimenti scrivere uno 0.

3-8. Procedere analogamente per 32, 16, 8, 4, 2 e 1

Come esempio si prenda il numero 201. Il processo dà:

(1) 201−128=73 per cui 1
(2) 73−64=9 per cui 11
(3) 9−32 non funziona, per cui 110
(4) 9−16 non funziona, per cui 1100
(5) 9−8=1 per cui 11001
(6) 1−4 non funziona, per cui 110010
(7) 1−2 non funziona, per cui 1100100
(8) 1−1=0 per cui 11001001

IL COMANDO "PEEK"

La funzione "PEEK" prende un indirizzo come argomento e fornisce il contenuto di quell'indirizzo sotto forma di un numero. Per esempio se si ripristina la macchina e si batte

PRINT PEEK(36879)

la macchina risponde con "27" poichè questo è l'equivalente del profilo di bit nella cella di memoria 36879. Come si ricorderà la cella 36879 è usata per controllare il colore del margine e del fondo. È possibile variarne il contenuto mediante un comando POKE dopodichè il PEEK darà il nuovo valore.
È possibile applicare la funzione PEEK a uno qualsiasi dei 54536 diversi indirizzi.
Se si sceglie un indirizzo che non è occupato (ad esempio 10000), la risposta è priva di significato.
Se si sceglie un'area con ROM, è possibile trovare quale profilo è stato inserito al momento della fabbricazione.
Per dare un altro sguardo al modo in cui i byte vengono usati, si osservano i contenuti delle celle (da 32776 a 32783). (Questa è una parte della ROM dei caratteri). Se si esegue una serie di PEEK, si ottiene:

```
32776 :  24
32777 :  36
32778 :  66
32779 : 126
32780 :  66
32781 :  66
32782 :  66
32783 :   0
```

Ciò sembra abbastanza privo di significato ma si provi ad osservare cosa succede quando si adopera la conversione in forma binaria:

```
 24 : 00011000          11
 36 : 00100100         1  1
 66 : 01000010        1    1
126 : 01111110        111111
 66 : 01000010        1    1
 66 : 01000010        1    1
 66 : 01000010        1    1
  0 : 00000000
```

È ora chiaro il profilo di una lettera "A".
Quando il VIC visualizza una A sullo schermo, usa i contenuti di questi 8 byte per ottenere il profilo corretto della lettera.

ESPERIMENTO 23·1

(a) Usando PEEK, trovare quale carattere è memorizzato nelle locazioni da 33192 a 33199.

(b) Usare la subroutine qui di seguito indicata in un programma che esplora la ROM dei caratteri e mostra come sono costruiti i "profili". Ricordarsi che è possibile inserire un PEEK in un comando con label esattamente come con qualsiasi altra funzione.

```
1000 REM DATA UNA LOCAZIONE IN X1,
     CALCOLARNE IL PROFILO BINARIO
     E VISUALIZZARLO
1010 YY=256: FOR K=1 TO 8
1015 YY=YY/2
1020 IF X1>=YY THEN X1=X1-YY:
     PRINT"★";:GOTO 1040
1030 PRINT" ";
1040 NEXT K
1050 PRINT:RETURN
```

Esperimento 23.1 completato	

IL COMANDO POKE

Il comando POKE usa due argomenti: un indirizzo (nel campo da 0 a 65535) e un numero (nel campo da 0 a 255). L'effetto è di memorizzare il profilo binario di quel numero nell'indirizzo selezionato. Naturalmente ciò funziona soltanto se l'indirizzo è in una RAM o in un'unità di controllo e può avere effetti strani se si sceglie l'indirizzo sbagliato.
Un uso importante del comando POKE è di pilotare l'unità di controllo del suono e delle immagini. Se si osserva la Fig. 23.1, si vedrà che l'unità di controllo è collegata a quattro moduli dello spazio d'indirizzo:

(a) I registri di suono e visione
(b) La RAM dello schermo
(c) La RAM del colore
(d) La ROM dei caratteri

L'immagine che si vede sullo schermo è disegnata 50 volte ogni secondo. Ogni volta il generatore osserva la cella 36879 dipinge i colori del margine e del fondo di conseguenza. Quindi dipinge ciascuno dei 506 caratteri sullo schermo (si considera lo spazio come un carattere), iniziando dall'angolo superiore sinistro e procedendo da sinistra a destra e dall'alto verso il basso. Il processo usato per produrre ciascuno dei caratteri è abbastanza complesso. Per colorare il primo carattere ecco cosa succede:
Innanzitutto il generatore cerca nell'indirizzo 7680, che è il primo indirizzo della RAM dello schermo, dove trova un codice dello schermo che gli dice quale carattere usare. Il codice è illustrato in Fig. 23.2, dove occorre ignorare la colonna contrassegnata "SET2", per il momento. Usando questo foglio di codifica, si vedrà che una "M" è rappresentata da 13 e un ♥ da 83. La tabella va solo fino a 127 in quanto i codici da 128 a 255 rappresentano gli stessi caratteri ma in negativo. Per esempio il codice per un segno $ in negativo è 36+128 ossia 164.
Successivamente il generatore va alla ROM dei caratteri per trovare quale profilo disegnare. Per trovare il profilo adatto, esso moltiplica il codice dello schermo per 8 aggiunge 32768 ed estrae 8 byte iniziando con l'indirizzo calcolato. Per esempio, si ottiene il profilo della M dagli 8 byte che iniziano a 32768+13*8 ossia 32872.
Successivamente il generatore va alla RAM del colore ed estrae il byte che si trova in 38400. Ciò dice il colore del primo carattere secondo il codice

0	1	2	3	4	5	6	7
Nero	Bianco	Rosso	Bluverde	Porpora	Verde	Blu	Giallo

Ora il generatore ha sufficienti informazioni per dipingere il primo carattere nella forma e nel colore corretti. Ciò fatto, il generatore visualizza il secondo carattere allo stesso modo, ma stavolta usa la cella 7681 (e non la 7680) per accedere ad un codice dello schermo e 38401 (e non 38400) per il codice del colore.
Il generatore continua fino a che non ha elaborato tutti i 506 byte della RAM dello schermo e della RAM del colore e non ha dipinto l'intera immagine, poi ritorna da capo. Il sistema sembra complicato ma è molto flessibile. Il generatore opera in continuo e abbastanza indipendentemente dal microprocessore. Per visualizzare qualsiasi informazione basta registrare i codici appropriati nelle RAM dello schermo e del colore: dopo 1/50 di sec. sullo schermo compare il nuovo carattere.
Si supponga di voler visualizzare un rombo rosso nel 14esimo carattere della 7ª riga. Ogni riga intera conta 22 caratteri cosicchè questa posizione è visualizzata 6*22+13 ossia 145 caratteri dopo quella in alto a sinistra. Le corrispondenti celle sono

7680+145 = 7825 (RAM dello schermo)
e 38400+145 = 38545 (RAM del colore)

Secondo la Fig. 23.2, il codice per un rombo è 90, e "rosso" è 2 cosicchè la seguente coppia dei comandi dovrebbe inserire un rombo rosso nel posto giusto:

```
POKE 7825,90:POKE 38545,2
```

I diagrammi nell Fig. 23.3 aiuteranno a ricavare l'esatto indirizzo RAM per ogni posizione sullo schermo. È talvolta più comodo disegnare immagini usando POKE che non i comandi PRINT. Ad esempio ecco un programma che disegna una linea rossa diagonale sullo schermo:

```
10 FOR J=21 TO 462 STEP 21
20 POKE 7680+J,78:POKE 38400+J,2
30 NEXT J
40 GOTO 40 : REM STOP ITERAZIONE
```

INTRODUZIONE ALL'ANIMAZIONE

POKE diventa realmente utile nei programmi di animazione. Si supponga di voler visualizzare un cerchio in movimento. Fondamentalmente il metodo consiste nel disegnare il cerchio in una posizione, quindi cancellarlo e disegnarlo nella successiva e così via. Se si usassero le istruzioni PRINT con il movimento del cursore, ciò sarebbe terribilmente laborioso, mentre con il POKE è abbastanza facile. Per esempio c'è un breve programma, che sposta un cerchio a caso, senza però farlo mai uscire dal margine dello schermo. Notare che X e Y rappresentano le posizioni del cerchio rispettivamente in senso orizzontale e verticale:

CODICI DELLO SCHERMO

SET 1	SET 2	POKE
@		0
A	a	1
B	b	2
C	c	3
D	d	4
E	e	5
F	f	6
G	g	7
H	h	8
I	i	9
J	j	10
K	k	11
L	l	12
M	m	13
N	n	14
O	o	15
P	p	16
Q	q	17
R	r	18
S	s	19
T	t	20

SET 1	SET 2	POKE
U	u	21
V	v	22
W	w	23
X	x	24
Y	y	25
Z	z	26
[		27
£		28
]		29
↑		30
←		31
SPACE		32
!		33
"		34
#		35
$		36
%		37
&		38
'		39
(		40
)		41

SET 1	SET 2	POKE
★		42
+		43
,		44
—		45
.		46
/		47
0		48
1		49
2		50
3		51
4		52
5		53
6		54
7		55
8		56
9		57
:		58
;		59
<		60
=		61
>		62

Figura 23.2

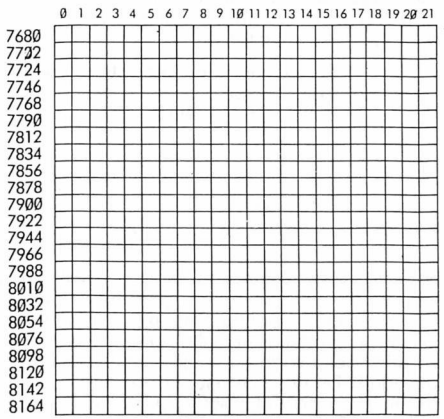
SET 1	SET 2	POKE
?		63
		64
♠	A	65
	B	66
	C	67
	D	68
	E	69
	F	70
	G	71
	H	72
	I	73
	J	74
	K	75
	L	76
	M	77
	N	78
	O	79
	P	80
●	Q	81
	R	82
♥	S	83

SET 1	SET 2	POKE
	T	84
	U	85
	V	86
	W	87
♣	X	88
	Y	89
♦	Z	90
		91
		92
		93
π		94
		95
SPACE		96
		97
		98
		99
		100
		101
		102
		103
		104
		105

SET 1	SET 2	POKE
		106
		107
		108
		109
		110
		111
		112
		113
		114
		115
		116
		117
		118
		119
		120
		121
	√	122
		123
		124
		125
		126
		127

```
10 PRINT " SHIFT e CLR HOME "
20 X=12 : Y = 10
30 XN=X+INT(3*RND (0))-1
40 IF XN<0 OR XN > 21 THEN 30
50 YN=Y+INT(3*RND(0))-1
60 IF YN<0 OR YN>22 THEN 50
70 POKE 7680+22*Y+X, 32
80 Y=YN: X=XN
90 POKE 7680+22*Y+X, 87
100 POKE 38400+22*Y+X, 0
110 GOTO 30
```

MAPPA DELLA RAM DELLO SCHERMO



MAPPA DELLA RAM DEL COLORE

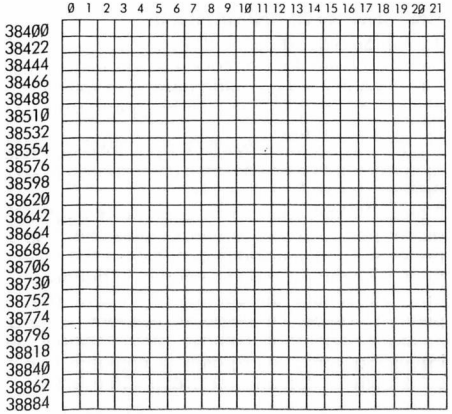


Figura 23.3

ESPERIMENTO 23·2

Ecco un programma leggermente Più lungo che usa il POKE per creare un effetto "artistico". Provarlo e quindi fare del proprio meglio per migliorarlo.

```
10 PRINT " SHIFT e CLR HOME "
20 FOR K=0 TO 7
30 FOR J=K TO 9
40 Y1=2+J
50 FOR X1=2+J TO 20-J
60 GOSUB 1000
70 NEXT X1
80 X1=20-J
90 FOR Y1=2+J TO 20-J
100 GOSUB 1000
110 NEXT Y1
120 Y1=20-J
130 FOR X1=20-J TO 2+J STEP-1
140 GOSUB 1000
150 NEXT X1
160 X1=2+J
170 FOR Y1=20-J TO 2+J STEP-1
180 GOSUB 1000
190 NEXT Y1
200 NEXT J
210 NEXT K
220 GOTO 10
1000 REM SUBROUTINE PER DISEGNARE
     UN * IN X1, Y1 DI COLORE K
1010 ZZ=X1+22*Y1
1020 POKE 7680+ZZ, K
1030 POKE 38400+ZZ, K
1040 RETURN
```

Esperimento 23.2 completato	

ALTRO A PROPOSITO DEI PEEK E DEI POKE

Un modo comune per usare i PEEK consiste nell'esaminare cosa c'è sullo schermo. Se si dà ad un PEEK l'indirizzo di una cella nella RAM dello schermo, il risultato sarà il codice del carattere dello schermo in quella cella. Questa funzione è utile se si vuole disegnare un'immagine sullo schermo usando i comandi del colore e del cursore e quindi analizzare o registrare l'immagine con un programma.

Per consentire di disegnare l'immagine senza interferenze, scrivere comandi per cancellare lo schermo e immettere X (o qualsiasi altra variabile). Quando compare il punto di domanda, è possibile usare i comandi del cursore per disegnare qualsiasi immagine a piacere; è possibile anche cancellare il comando di input?. Per contro, non occorre usare RETURN fino a che l'immagine non è terminata; quando si preme RETURN il cursore deve essere in qualche posto in una riga completamente vuota, preferibilmente al disopra o al disotto dell'immagine.

Tutto ciò è illustrato nel seguente programma che conta e visualizza sullo schermo i numeri di caratteri diversi dallo spazio. Impostarlo, avviarlo, e quindi usare il controllo del cursore per distribuire alcuni simboli sullo schemo. Quindi battere RETURN.

```
10 INPUT "[SHIFT] e [CLR HOME]";X:REM X
   VARIABILE FITTIZIA
20 S = 0
30 FOR J=0 TO 505 : REM ANALIZZA
   SCHERMO
40 IF PEEK(7680+J)<>32 THEN S=S+1 :
   REM 32=CODICE SCHERMO
   PER SPAZIO
50 NEXT J
60 PRINT"NUMERO SIMBOLI="; S
70 STOP
```

Ci sono talune altre locazioni che possono essere comodamente usate con POKE per cambiare il comportamento del VIC in modi utili e prevedibili. L'indirizzo 36869 controlla la scelta dei profili dei caratteri. Il comando

```
POKE 36869,242
```

fa passare al SET 2 nella Fig. 23.2. Qui ci sono le lettere maiuscole e minuscole ma molti segni grafici in meno. Questa serie è utile per presentare prospetti leggibili e in altri tipi di data processing gestionale. È possibile mostrare i caratteri SET 1 e SET 2 sullo schermo contemporaneamente: ciò in quanto molti caratteri sono duplicati. Se si vuole scrivere un programma che visualizza i suoi risultati nel SET 2, occorre pianificarlo con molto anticipo. Prima di iniziare a battere il programma, premere i tasti [SHIFT] e [C=] contemporaneamente. Tutto ciò che si batte comparirà nel SET 2; le lettere saranno visualizzate in minuscole a meno che non si tiene abbassato il tasto SHIFT. I comandi del programma BASIC e i REM devono essere tutti in minuscolo, ma è possibile, se necessario, inserire lettere in maiuscolo nelle stringhe. Ecco un esempio:

(premere [SHIFT] e [C=])

```
10 poke 36869,242: rem definito output
su set 2
20 print "Questo è uno schermo serie 2"
30 stop
```

Per tornare al SET 1 (cosa che è possibile fare in qualsiasi momento), impartire il comando

```
poke 36869,240
```

oppure premere [SHIFT] e [C=] di nuovo.

C'è un gruppo di locazioni che controlla il comportamento della tastiera.

★ Tasti con ripetizione: quando il VIC è acceso, i soli tasti che si ripetono se li si tiene abbassati sono lo spazio e i tasti di controllo del cursore. Ciò è controllato dal contenuto dell'indirizzo 650, che è interpretato come segue:

```
  0: Si ripetono solo i tasti "standard"
127: Non c'è ripetizione di tasti
128: Tutti i tasti si ripetono
```

Se si impartisce il comando

```
POKE 650,128
```

si troverà che ora tutti i tasti si ripetono.

★ Coda della tastiera: se si battono caratteri più velocemente di quanto il programma non possa accettarli, essi verranno posti in una coda e forniti al programma uno alla volta. Il numero di caratteri nella coda può essere ispezionato eseguendo PEEK 198. I caratteri nella coda possono anche essere scartati inserendo (POKE) 0 nell'indirizzo 198. Ciò è spesso utile nei giochi dove lo scopo è di fare in modo che il computer reagisca a ciò che il giocatore sta facendo e non a ciò che ha fatto erroneamente qualche secondo prima.

UN ESEMPIO DI ANIMAZIONE

Terminiamo questa unità discutendo il disegno di un gioco animato. Caricare il programma intitolato "WASPS" ed eseguirlo parecchie volte fino a che non si è acquisita una certa esperienza come uccisore di vespe.

L'intero programma WASP è riprodotto alla fine di questa unita e ora daremo uno sguardo ai suoi principi generali di disegno.

Il gioco WASP, come la maggior parte degli altri giochi per computer, è una simulazione o se si preferisce una imitazione di qualcosa che si suppone succeda nel mondo esterno. In questo programma simula un cacciatore in una stanza piena di vespe. Le vespe si muovono a caso, mentre il cacciatore si muove, gira e lancia il suo insetticida spray in risposta ai comandi battuti sulla tastiera. Possono verificarsi vari eventi:

(a) Una vespa può essere uccisa

(b) Il cacciatore può essere punto

(c) Il cacciatore può finire l'insetticida spray.

Lo scheletro del programma è un modello, o una serie di variabili, che descrive completamente la posizione in qualsiasi istante. Una volta che questo modello è stato disegnato, è possibile scrivere vari pezzi di codice (prevalentemente subroutine) che operano sul modello e lo cambiano in conformità con gli eventi che si suppone accadano nel mondo esterno.

Ecco dunque una descrizione del modello del gioco WASP:

N: Numero di vespe all'inizio

NA: Numero di vespe rimaste in qualsiasi momento

TT: Tempo di inizio della caccia (in jiffies)

BU: Numero di colpi d'insetticida rimasti

ST: Numero di volte che il cacciatore è rimasto punto

IP: Altezza del ronzio della vespa (l'altezza del suono prodotto dalle vespe)

A,B: La posizione attuale del cacciatore. A è il numero di colonna dello schermo da sinistra e B è il numero di righe dall'alto.

C: L'attuale direzione del cacciatore:
1=Nord
2=Nord-Est
3=Est
4=Sud-Est
5=Sud
6=Sud-Ovest
7=Ovest
8=Nord-Ovest

La posizione di ciascuna vespa è registrata sotto forma di due elementi nella matrice W%(N,2). Pertanto la posizione di colonna della prima vespa è W%(1,1) e la sua posizione di riga è W%(1,2). La seconda vespa occupa W%(2,1) e W%(2,2) e così via.
La posizione dell'ultima vespa attiva è memorizzata in W%(NA,1) e W%(NA,2). Se una vespa (diversa dall'ultima vespa attiva) viene uccisa, il record di tutte quelle al di sotto di essa viene spostato verso l'alto per riempire lo spazio rimasto vuoto.
Per esempio:

NA=6

W%

1	17
3	12
18	2
10	7
7	9
4	17

← Questa è uccisa

dà

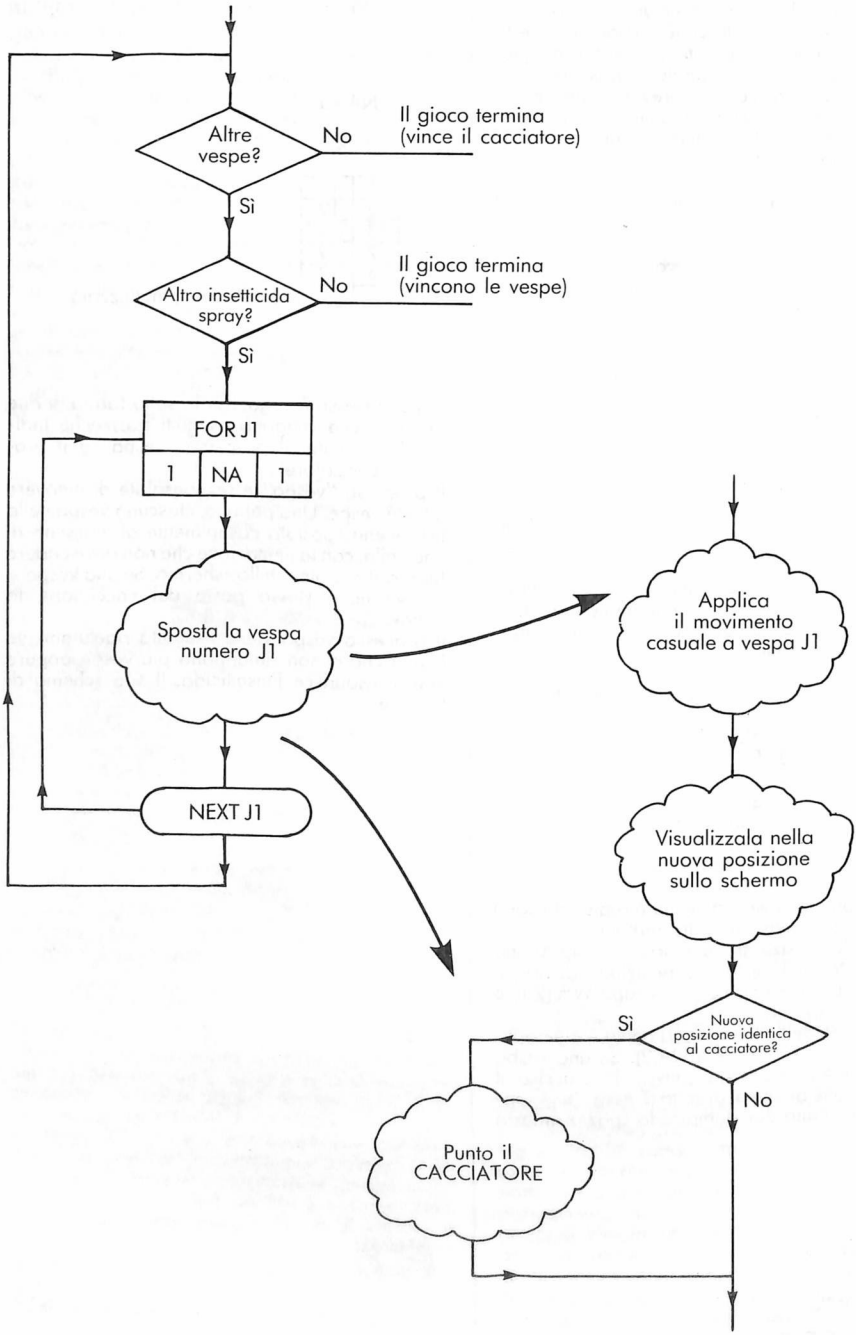
NA=5

W%

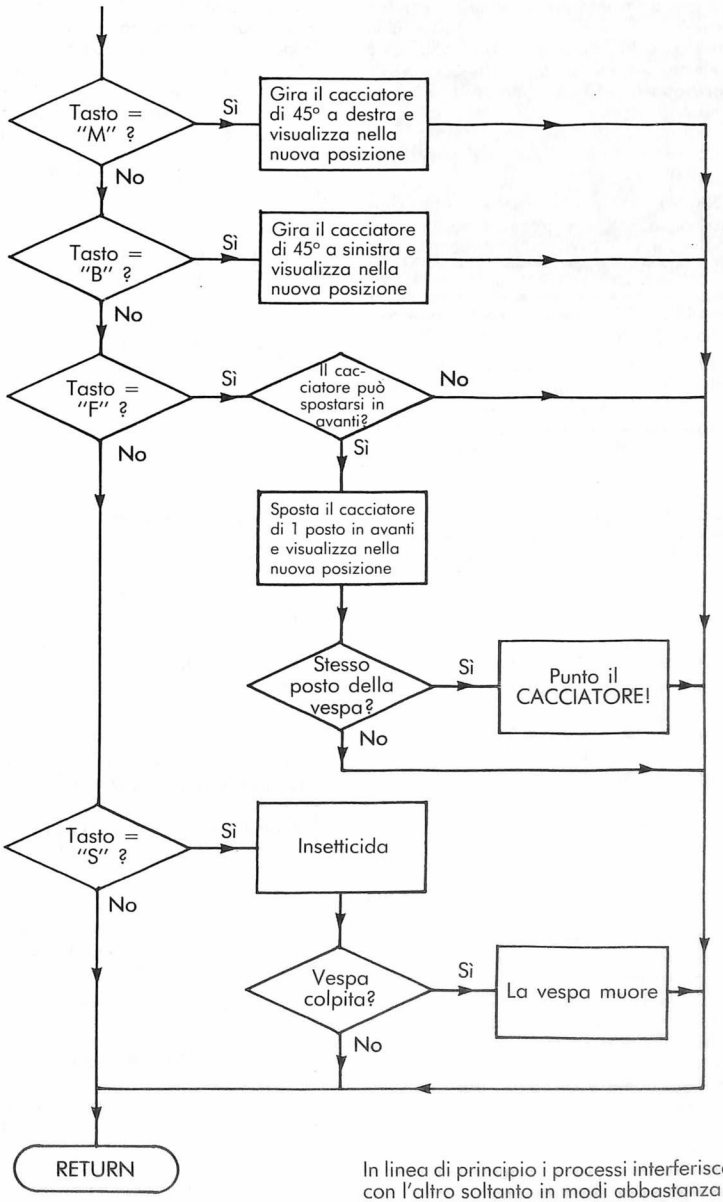
1	17
3	12
18	2
7	9
4	17
—	—

Questa fila è inutilizzata

Il gioco stesso è organizzato sotto forma di due processi che vengono eseguiti pressochè indipendentemente: il processo "vespa" e il processo "cacciatore".
Il processo "vespa" è responsabile di muovere tutte le vespe. Una per una, ciascuna vespa nella lista viene spostata casualmente al massimo di una cella, con la limitazione che non deve cadere fuori dal margine dello schermo. Se una vespa si muove nello stesso posto del cacciatore, lo punge.
Il processo vespa viene eseguito ripetutamente fino a che o non rimangono più vespe oppure non si esaurisce l'insetticida. Il suo schema di flusso è:



Il processo "cacciatore" è richiamato ogniqualvolta viene premuto un tasto. Il suo schema di flusso è:



In linea di principio i processi interferiscono l'uno con l'altro soltanto in modi abbastanza specifici. Per esempio, le attività del cacciatore riducono gradualmente il numero di vespe attive e la quantità di insetticida rimanente, al limite provocando l'interruzione del processo.
In pratica, occorre fare in modo che entrambi i processi si svolgano contemporaneamente (più o meno). Un modo semplice per far ciò è di dare al cacciatore una possibilità di fare qualcosa dopo che ciascuna vespa si è mossa. Viene estratto un carattere dalla tastiera e se viene premuto un tasto, è possibile dare inizio al "processo del cacciatore".

È possibile ora dare una descrizione dettagliata del programma. Esso si adatta comodamente nella memoria del VIC da 3,5K con circa 500 byte a disposizione per modifiche e miglioramenti. Per ottenere questo spazio, sono state adottate alcune misure di risparmio.

Le righe da 10 a 90 visualizzano una serie di istruzioni dell'utente.

La riga 100 imposta taluni numeri in forma simbolica. Cioè aiuta successivamente a risparmiare spazio. Per esempio ciascun riferimento "PE" sarà di 3 byte più corto di un riferimento a "36879".

Le righe da 120 a 130 determinano il numero di vespe con cui iniziare.

La riga 140 dichiara quattro matrici. Esse sono tutte le matrici intere per risparmiare spazio. W% è, come già si sa, usato per contenere la posizione di ciascuna vespa attiva. V%, U% e D% aiutano tutti a visualizzare e a spostare il cacciatore.

Si ricorderà che i quadrati nella RAM dello schermo sono numerati da sinistra a destra iniziando con la fila superiore e procedendo verso il basso.

	7930		
7951	7952	7953	7954
7973	7974	7975	7976
7995	7996	7997	
	8018		

Qui c'è una parte dello schermo dove ciascuna cella è contrassegnata con il proprio indirizzo. Si supponga che il cacciatore si trovi in qualche cella con la mira puntata a nord. Come si può vedere, il contenitore di insetticida occuperà un quadrato di numero 22 meno quello occupato da se stesso. Analogamente, se il cacciatore punta a Nord-Est, la cella di mira avrà un numero inferiore di 21 e così via. La matrice V% contiene la "differenza del numero di cella" per ciascuna delle 8 possibili direzioni, iniziando con V%(1) (Nord) e procedendo fino a V%(8) (Nord-Est). In effetti, (tenendo presente che la variabile C contiene la direzione verso la quale è rivolto il cacciatore, l'appropriata differenza del numero di cella è sempre V%(C).
La matrice V% contiene i codici dello schermo dei simboli usati per rappresentare la mira del cacciatore. Essi variano secondo la direzione:

N	NE	E	SE	S	SW	W	NW
|	/	—	\	|	/	—	\

Usando queste tabelle il cacciatore può facilmente venir visualizzato in qualsiasi posizione e in qualsiasi direzione. Osservare le righe 2000-2020 ricordando che la posizione del cacciatore è registrata nelle variabili A e B. SR è la costante 7680, che è l'indirizzo della prima cella nella RAM dello schermo. La matrice D% contiene i movimenti Est e Sud che corrispondono ad una mossa in una qualsiasi delle 8 direzioni.
I valori sono:

N	0	−1
NE	1	−1
E	1	0
SE	1	1
S	0	1
SW	−1	1
W	−1	0
NW	−1	−1

Ciò rende facile spostare il cacciatore. Per esempio, per andare in un quadrato in direzione 6 (Sud-Ovest), si aggiunge D%(6,1) a A e D%(6,2) a B.
Le righe da 150 a 210 impostano gli appropriati valori per V%, U% e D%.
La riga 220 imposta la disponibilità iniziale di insetticida. La formula assicura una scala scorrevole come questa.

N. delle vespe	1	2	3	4	5	6	7	8
N. degli spruzzi	7	9	12	14	15	17	18	19

Ciò tiene conto del fatto che le prime vespe, quando ne sono presenti molte, sono molto più facili da colpire che non l'ultima.

La riga 240 imposta l'intero colore dello schermo a "nero".

Le righe da 250 a 290 elaborano le posizioni iniziali per tutte le vespe e le visualizzano sullo schermo. Innanzitutto tutte le vespe sono inserite nella metà superiore dello schermo (righe da 1 a 12).

La riga 320 fissa la posizione iniziale del cacciatore e lo visualizza in questa posizione.

Le righe da 330 a 420 formano il cuore della simulazione; esse eseguono il processo delle vespe e richiamano il processo del cacciatore ogniqualvolta viene premuto un tasto. 340 e 350 visualizzano la situazione corrente. Notare che la subroutine in 1000 muove la vespa J1 e quella in 3000 attiva il cacciatore

Le righe da 500 a 570 visualizzano i messaggi finali di congratulazioni o di consolazione a seconda dei casi.

La subroutine che muove le vespe è alle righe da 1000 a 1090. Il metodo base è di ottenere la posizione precedente della vespa nelle variabili locali XN e YN. Ciascuno di questi numeri viene quindi "perturbato" da una quantità casuale che può essere +1,0 o −1. Se il risultato è al di fuori del campo dello schermo è respinto e il processo viene ripetuto.
Quando è stata determinata una nuova posizione, la vespa nella vecchia posizione viene cancellata (riga 1040).
Viene disegnata una vespa nella nuova posizione a meno che la posizione non sia già occupata da qualche altra cosa. Quindi la nuova posizione viene registrata nella tabella in W%(J1,1) e W% (J1,2).
A questo punto viene generato il ronzio della vespa. L'altezza corrente del ronzio è memorizzata nella variabile IP. Il valore di IP è perturbato da un'unità casuale, con una trappola per impedirgli di allontanarsi troppo dal suo campo normale circa 192. Quindi la variabile viene usata per iniziare una nota ronzante che continua fino a quando non viene cambiato IP.
Infine, la nuova posizione della vespa è confrontata con quella del cacciatore e se corrispondono, viene richiamata la subroutine di "puntura" a 4000.

Le righe da 2000 a 2020 visualizzano il cacciatore nella sua nuova posizione.

Le righe da 2500 a 2520 cancellano il cacciatore da una vecchia posizione disegnando degli spazi nei punti appropriati.

Le righe da 3000 a 3270 sorvegliano il cacciatore. Le parti di questa subroutine sono le seguenti:
da 3020 a 3030 rotazione verso destra. Notare che il Nord (C=1) deve seguire Nord-Est (C=8).
Da 3050 a 3060 rotazione a sinistra. Notare che Nord-Est (C=8) deve seguire Nord (C=1).
Da 3080 a 3140 spostamento in avanti.
Viene prodotta una nuova posizione orientativa in AA e BB ed è trasferita soltanto a A e B se non è troppo vicina al margine dello schermo. Quando viene fatta la mossa viene esaminata la lista delle posizioni delle vespe per vedere se il cacciatore si è seduto su una vespa; in questo caso è punto. Ciò accade nelle righe da 3110 a 3130.
Da 3160 a 3270 spruzzo. PP e QQ sono le posizioni dell'area bersaglio. Le righe 3170 e da 3190 a 3240 sono interessate agli effetti (suono e visione) dello spruzzo. Le righe da 3250 a 3270 esaminano la lista delle vespe per vedere se ne è stata colpita una. In questo caso viene richiamata la subroutine in 5000.
Le righe da 4000 a 4080 rappresentano la subroutine richiamata quando il cacciatore viene punto. Essa è principalmente costituita da effetti audiovisivi, ma ci sono anche predisposizioni perchè il cacciatore salti ad una nuova posizione casuale.

Le righe da 5000 a 5070 riguardano la morte della vespa. A parte gli effetti soliti, il record della vespa morta è rimosso dalla lista e i record vengono spostati verso l'alto, se necessario.

ESPERIMENTO 23·3

Disegnare e programmare un proprio gioco animato. Le aree possibili di interesse, comprendono:

Colpire alieni provenienti dallo spazio esterno

Ricerca della strada per uscire da un labirinto inseguiti da un mostro.

Cattura di palle lanciate casualmente.

Esperimento 23.3 completato	

```
10 REM WASPSHOOTER COPYRIGHT
   (C) ANDREW COLIN 1981
20 PRINT" [SHIFT] e [CLR HOME] [CRSR ↓]
   W A S P S H O O T E R":PRINT:PRINT
30 PRINT"UCCIDI TUTTE LE VESPE"
40 PRINT"PRIMA DI TERMINARE"
50 PRINT"L'INSETTICIDA"
60 PRINT: PRINT"M PER GIRARE A DESTRA"
70 PRINT"B PER GIRARE A SINISTRA"
80 PRINT"F PER ANDARE AVANTI"
90 PRINT"S PER SPRUZZARE"
100 PA=36875:PB=36876:PC=36877:
    PD=36878:PE=36879:SR=7680
110 POKE PB,0:POKE PC,0:POKE PD,15
120 INPUT"QUANTE VESPE"; N
130 IF N<1 OR N>20 THEN PRINT"
    DA 1 A 20 PREGO":GOTO 120
140 DIM V%(8),U%(8),D%(8,2),W%(N,2)
150 FOR J = 1 TO 8
160 READ V%(J),U%(J),D%(J,1),D%(J,2)
170 NEXT J
180 DATA -22,93,0,-1,-21,78,1,-1
190 DATA 1,67,1,0,23,77,1,1
200 DATA 22,93,0,1,21,78,-1,1
210 DATA -1,67, -1,0,-23,77,-1,-1
220 BU=INT(7*SQR(N)): SQ=0
230 PRINT " [SHIFT] e [CLR HOME] "
240 FOR J=0 TO 505: POKE 38400+J,0:
    NEXT J
250 FOR J = 1 TO N
260 W%(J,1)=INT(22*RND(0))
270 W%(J,2)=INT(12*RND(0))+1
280 POKE SR+22*W%(J,2)+W%(J,1),35
290 NEXT J
300 NA = N
310 TS = TI
320 A=3:B=18:C=2: GOSUB 2000
330 IF NA=0 THEN 500
335 IF BU = 0 THEN 600
340 PRINT" [CLR HOME] "
350 PRINT" [CLR HOME] VESPE";NA;"TEMPO";
    INT((TI-TS)/60)
360 PRINT"SPRUZZI 5 spazi [SHIFT] e
    [CRSR ↓] [SHIFT] e [CRSR ↓] [SHIFT] e
    [CRSR ↓] [SHIFT] e [CRSR ↓] [SHIFT]
    e [CRSR ↓] "; BU; "4 spazi"
370 FOR J1 = 1 TO NA
380 IF W%(J1,1)>=0 THEN GOSUB 1000
390 GET A$: IF A$ = " " THEN 410
400 GOSUB 3000
410 NEXT J1
420 GOSUB 2000:GOTO 330

500 REM VINCE
510 PRINT" [SHIFT] e [CLR HOME] [CRSR ↓]
    [CRSR ↓] 3 spazi BENE!":PRINT
520 PRINT"HAI UCCISO"; N-NA:
    PRINT
530 PRINT"VESPE IN"; INT((TI=TS)/60);
    "SECONDI":PRINT
540 PRINT"SEI STATO PUNTO":PRINT
550 PRINT SQ; "VOLTE"
560 POKE PD,0: POKE PC,0:POKE PB,0:
    POKE PA,0
570 STOP

600 REM INSETTICIDA TERMINATO
610 PRINT" [SHIFT] e [CLR HOME]
    [CRSR ↓] [CRSR ↓] SPIACENTE
    INSETTICIDA FINITO":PRINT
620 PRINT"MANCATO":PRINT
630 GOTO 520

1000 REM SPOSTA J-esima VESPA A CASO
1010 XX=W%(J1,1): YY=W%(J1,2)
1020 XN=XX+INT(3*RND(0))-1: IF XN<0
     OR XN>21 THEN 1020
1030 YN=YY+INT(3*RND(0))-1: IF YN<1
     OR YN>22 THEN 1030
1040 POKE SR+22* YY+XX,32
1050 ZZ=SR+22*YN+XN:IF PEEK (ZZ)=32
     THEN POKE ZZ,35
1060 W%(J1,1)=XN:W%(J1,2)=YN
1070 IP=IP+INT(3*RND(0))-1: IF IP<180
     OR IP>205 THEN IP=192
1080 IF XN = A AND YN = B THEN GOSUB
     4000
1090 POKE PA, IP: RETURN

2000 REM VISUALIZZA CACCIATORE
2010 XX=SR +22*B+A: YY=XX+V%(C)
2020 POKE XX,81: POKE YY,U%(C):
     RETURN

2500 CANCELLA CACCIATORE
2510 XX=SR+22*B+A:YY=XX+V%(C)
2520 POKE XX,32: POKE YY,32: RETURN

3000 REM SPOSTA CACCIATORE O SPRUZZA
3010 IF A$<>"M" THEN 3040
3020 GOSUB 2500: C=C+1: IF C=9 THEN
     C=1
3030 GOSUB 2000: RETURN
3040 IF A$ <>"B" THEN 3070
3050 GOSUB 2500:C=C-1: IF C=0 THEN
     C=8
3060 GOSUB 2000: RETURN
3070 IF A$<>"F" THEN 3150
3080 GOSUB 2500: AA=A+D%(C,1):
     BB=B+D%(C,2)
3090 IF AA>2 AND AA <19 AND BB>2
     AND BB<19 THEN A=AA:B=BB
3100 GOSUB 2000
3110 FOR JJ= 1 TO NA
3120 IF A=W%(JJ,1) AND B=W%(JJ,2)
     THEN GOSUB 4000
3130 NEXT JJ
3140 RETURN
3150 IF A$<>"S" THEN RETURN
3160 PP=A+2*D%(C,1):QQ=B+2*D% (C,2)
```

```
3170 POKE PC,252
3180 BU=BU-1
3190 RR=SR + 22*QQ+PP
3200 FOR KK = 1 TO 5
3210 POKE RR,102: FOR TT= 1 TO 30:
     NEXT TT
3220 POKE RR,32: FOR TT= 1 TO 50:
     NEXT TT
3230 NEXT KK
3240 POKE PC,0
3250 FOR JJ = 1 TO NA
3260 IF PP=W%(JJ,1) AND QQ=W%(JJ,2)
     THEN J2=JJ: GOSUB 5000
3270 NEXT JJ : RETURN

4000 REM CACCIATORE PUNTO

4010 PRINT" [CLR HOME] PUNTO!".
4020 GOSUB 2500
4030 A=INT(3+16*RND(0)):
     B=INT(3+16*RND(0)):
     C=INT(1+8*RND(0))
4040 POKE PD,15: GOSUB 2000:
     SQ=SQ + 1
4050 FOR JJ=1 TO 20: POKE PB,240-JJ:
     POKE PE,250-JJ
4060 FOR TT = 1 TO 50: NEXT TT
4070 NEXT JJ
4080 POKE PB,0: POKE PE,27: RETURN

5000 REM VESPA UCCISA

5010 PRINT" [CLR HOME] UNA VESPA MORDE
     LA POLVERE!"
5020 POKE PD,15: POKE PE,123
5030 FOR JJ = 1 TO 20: POKE PB,255-JJ
5040 FOR TT= 1 TO 10: NEXT TT
5050 POKE PB,0: FOR TT = 1 TO 40: NEXT TT
5060 NEXT JJ: POKE PE,27
5070 IF J2 = NA THEN NA=NA-1: RETURN
5080 W%(J2,1) = W%(J2+1,1)
5090 W%(J2,2) = W%(J2+1,2)
5100 J2=J2+1: GOTO 5070
```

UNITA': 24

ALTRO A PROPOSITO DEGLI OPERATORI LOGICI

In questa unità completeremo il nostro studio del VIC BASIC considerando alcuni argomenti vari. L'Unità 17 esaminava l'uso degli operatori logici AND, OR e NOT usati per costruire condizioni composte. Gli stessi operatori possono anche essere usati in un contesto completamente diverso per manipolare le cifre binarie in numeri e in altre variabili.
Battere il comando

```
PRINT 13 AND 17
```

Il risultato, 5, è completamente misterioso fino a che non si osserva la rappresentazione binaria dei numeri coinvolti:

```
13 = ....0001101
 7 = ....0000111
-------------------
 5 = ....0000101
```

Come è possibile vedere, il risultato ha un "1" soltanto nelle colonne in cui entrambi i numeri originali avevano "1". Possiamo spiegare l'operazione AND con una "tavola di verità" che si applica indipendentemente a ciascuna colonna:

```
0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1
```

Usando questa tavola, è possibile prevedere il risultato del comando

```
PRINT 27 AND 6
```

```
27 = ...0011011
 6 = ...0000110

   ...00000 = 2
```

Per assicurarsi di aver compreso il funzionamento di AND, provare a calcolare in anticipo i risultati di quanto segue:

```
PRINT 15 AND 12
PRINT 21 AND 10
PRINT 11 AND 7
```

Controllare il calcolo sul VIC in ciascun caso.
L'operatore OR è molto simile a AND con la differenza che dà un "1" in qualsiasi colonna dove uno o l'altro o entrambi i numeri originali erano "1". La sua tavola di verità è:

```
0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1
```

Usando questa tavola non si dovrebbero avere problemi nel prevedere il risultato del comando.

```
PRINT 7 OR 10
```

L'operatore NOT si limita a prendere un singolo numero e a cambiare ogni suo bit in quello contrario. Si troverà che

```
NOT (....0001010) = ....1110101
```

Nel VIC (e in conseguenza nella maggior parte degli altri computer) un numero che si compone interamente di uni rappresenta −1 (1 negativo). Come ci si sarebbe aspettato, il risultato del comando

```
PRINT NOT 0
```

è −1.
Le operazioni AND, OR e NOT sono utili nel lavorare con quantità dove le singole cifre binarie hanno significati speciali. Per esempio, se si usa il VIC per controllare le luci di casa attraverso la porta utente (user port), è abbastanza probabile che le 8 posizioni dei singoli interruttori siano rappresentate come le 8 cifre binarie di un singolo numero. Per scoprire se, ad esempio, il quinto interruttore da destra è acceso, occorrerebbe eseguire un'operazione AND tra il carattere e il numero binario . .0010000. Il risultato sarebbe diverso da 0 soltanto se il numero avesse un "1" in quella posizione — in altre parole, se il quinto interruttore fosse acceso.
L'equivalente di . .0010000 è 16 cosicchè il programma di controllo potrebbe contenere una riga del tipo

```
360 IF (S AND 16)<>0 THEN 590
```

COME IL VIC VALUTA LE CONDIZIONI

Ci si potrebbe domandare come questi apparenti nuovi significati degli operatori si colleghino con quelli convenzionali usati nelle condizioni composte. Per scoprire la risposta occorre osservare un po' più in profondità i meccanismi del VIC. Quando la macchina elabora una semplice condizione (ad esempio X=5 o A$<>"YES" o 5=4), essa produce sempre un valore di verità che è −1 se la condizione è vera e 0 se è falsa. Provare i seguenti comandi (quantunque sembrino strani) e spiegare perchè essi producano i risultati che in effetti producono:

```
PRINT 5=4
PRINT 6<9
PRINT 9>6
PRINT (1=1)*(1<2)
```

Il comando IF comprende sempre un'espressione tra IF e THEN. Questa è solitamente una condizione, ma non deve esserlo necessariamente; forme del tipo

```
IF X −3 THEN
```

sono abbastanza accettabili. Il comando (o gruppo di comandi) che segue THEN è eseguito se l'espressione dopo IF ha un qualsiasi valore salvo lo 0. Eseguire il seguente programma e spiegarne i risultati:

```
10 FOR X = 1 TO 5
20 IF X -3 THEN PRINT X
30 NEXT X
40 STOP
```

Anche quando l'espressione è una condizione, il modo in cui la macchina lavora è sempre lo stesso. Si consideri il comando

```
IF "PAPERINO"<"TOPOLINO" THEN
PRINT "PLUTO"
```

Si vedrà che la condizione è vera e ci si aspetta che la macchina visualizzi di conseguenza la stringa "PLUTO". Il VIC passa in effetti attraverso una fase intermedia. Esso per prima cosa valuta la condizione a "1" quindi, esegue il comando PRINT in quanto −1 non è lo stesso di 0.
Per completare la spiegazione delle condizioni composte, tutto ciò che si deve dire è che gli operatori logici possono essere applicati ai valori di verità prodotti da condizioni semplici. Si supponga che X$="D". Di conseguenza la condizione composta all'interno dell'espressione

```
IF NOT(X$="C" OR X$="D")
```

si traduce in

```
   NOT (0 OR −1)
= NOT (....00000 OR ....11111)
= NOT (....11111)
= ....00000
```

per cui la condizione è falsa.

I CODICI ASCII DEL CBM

Il prossimo argomento è la rappresentazione interna dei caratteri. Si sa già che una stringa, quando memorizzata internamente, richiede un byte per ciascun carattere che essa contiene. È talvolta utile conoscere esattamente come viene rappresentato ciascun carattere.
Nell'Unità 23 abbiamo introdotto l'idea di un "codice dello schermo" e abbiamo dato una tabella che dimostrava come ciascun carattere che poteva essere visualizzato sullo schermo, avesse un proprio codice speciale. Anche all'interno del VIC i caratteri sono rappresentati da un codice, ma questo è diverso dal codice dello schermo! È possibile vedere che deve essere diverso in quanto il codice deve essere in grado di gestire ogni carattere prodotto premento un tasto sulla tastiera. Ciò comprende tasti tipo RETURN o i movimenti del cursore che non corrispondono a qualsiasi simbolo visualizzabile.
Il codice usato è una modifica dell'American Standard Code for Information Interchange (abbreviato, "ASCII"). Questo codice consente di trasmette informazioni su linee telefoniche tra macchine di vari tipi.

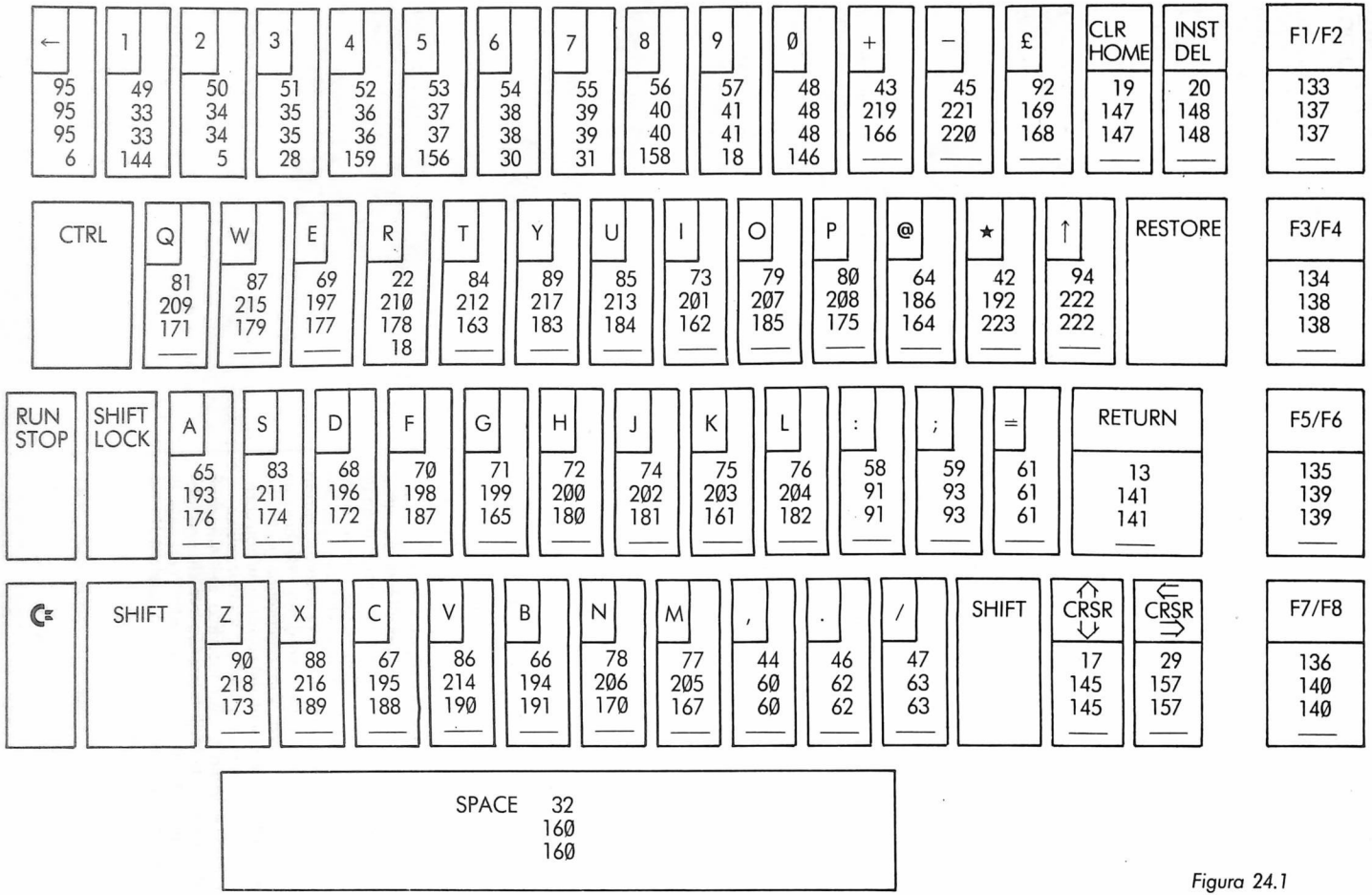

← 95 95 95 6
1 49 33 33 144
2 50 34 34 5
3 51 35 35 28
4 52 36 36 159
5 53 37 37 156
6 54 38 38 30
7 55 39 39 31
8 56 40 40 158
9 57 41 41 18
Ø 48 48 48 146
+ 43 219 166 —
– 45 221 220 —
£ 92 169 168 —
CLR HOME 19 147 147 —
INST DEL 20 148 148 —
F1/F2 133 137 137 —
CTRL
Q 81 209 171 —
W 87 215 179 —
E 69 197 177 —
R 22 210 178 18
T 84 212 163 —
Y 89 217 183 —
U 85 213 184 —
I 73 201 162 —
O 79 207 185 —
P 80 208 175 —
@ 64 186 164 —
★ 42 192 223 —
↑ 94 222 222 —
RESTORE
F3/F4 134 138 138 —
RUN STOP
SHIFT LOCK
A 65 193 176 —
S 83 211 174 —
D 68 196 172 —
F 70 198 187 —
G 71 199 165 —
H 72 200 180 —
J 74 202 181 —
K 75 203 161 —
L 76 204 182 —
: 58 91 91 —
; 59 93 93 —
= 61 61 61 —
RETURN 13 141 141 —
F5/F6 135 139 139 —
C=
SHIFT
Z 90 218 173 —
X 88 216 189 —
C 67 195 188 —
V 86 214 190 —
B 66 194 191 —
N 78 206 170 —
M 77 205 167 —
, 44 60 60 —
. 46 62 62 —
/ 47 63 63 —
SHIFT
CRSR ↑↓ 17 145 145 —
CRSR ←→ 29 157 157 —
F7/F8 136 140 140 —
SPACE 32 160 160

Figura 24.1

I dettagli del codice ASCII del CBM sono indicati nella Fig. 24.1. Si tratta di un disegno ingrandito della tastiera che mostra i codici generati quando vengono battuti i vari tasti. Ciascun tasto prevede quattro (o tre) numeri. Essi corrispondono a:

(a) Il carattere "non shiftato" (cioè non preceduto da shift)

(b) Il carattere normalmente shiftato (tasto premuto tenendo abbassato SHIFT)

(c) Il carattere "shift Commodore" (tasto premuto tenendo abbassato C=)

(d) Il carattere "Shift Control" (tasto premuto contemporaneamente al tasto CTRL)

Solo alcuni tasti rispondono quando CTRL viene tenuto abbassato. Quelli che non rispondono sono contrassegnati con un ——— al disotto degli altri tre numeri.

(NOTA: alcuni valori di questo diagramma sono diversi da quelli indicati nella tabella al termine del manuale fornita con il VIC. Quelli qui indicati sono quelli generati in ASCII. Entrambe le serie di valori possono essere usate con il comando CHR$).

Il diagramma mostra che, ad esempio quando viene battuto D con il tasto C= tenuto abbassato, il codice ASCII del CBM del carattere prodotto è 172. La stringa "COMMODORE" verrebbe memorizzata come una sequenza di 9 byte con i valori 67, 79, 77, 77, 79, 68, 79, 82, 69. Il diagramma chiarisce che, i tasti di controllo del cursore e i tasti speciali di funzione alla destra della tastiera producono codici ASCII CBM, quantunque essi non corrispondano ad alcun carattere stampato.
La funzione standard che fornisce il codice ASCII CBM di qualsiasi carattere è ASC, che prende una stringa come argomento e produce il codice ASCII CBM del primo carattere. Così

```
PRINT ASC("X")
```

dà 88 e

```
PRINT ASC("123456")
```

dà 49.

Per ovvii motivi, ASC non può essere applicato ad una stringa nulla (""). Se ci si prova, si ottiene un messaggio

```
?ILLEGAL QUANTITY ERROR.
```

Il programma usato per inserire i numeri nella Figura 24.1 era fondamentalmente questo:

```
10 GET A$:IF A$= " " THEN 10
20 PRINT A$;ASC(A$)
30 GOTO 10
```

Impostare questo programma ed eseguirlo per controllare alcuni dei valori nella Figura 24.1. Occorrerà una certa fantasia per gestire tutti i caratteri di controllo; si potrebbe per esempio rimuovere A$; dalla riga 20.

USO DI ASC - CONTEGGIO DELLA COMPARSA DELLE LETTERE

La funzione ASC è particolarmente utile in due campi: il primo quando si vogliono tradurre singoli caratteri in numeri. Per esempio, è possibile voler violare un codice segreto analizzando un messaggio cifrato e contando il numero delle volte che ogni lettera viene usata. Chiaramente si potrebbe scrivere un programma con numerose istruzioni del tipo.


```
...
IF A$="J" THEN AJ=AJ+1
IF A$="K" THEN AK=AK+1
```

e successivamente

```
...
PRINT "J",AJ
PRINT "K",AK
```

Con la funzione ASC è possibile fare molto meglio. Il diagramma mostra che i codici ASCII CBM per lettere iniziano a 65 per A e procedono fino a 90 per Z. È possibile usare il codice ASCII CBM per ciascuna lettera con un indice per una matrice dove ciascun elemento corrisponde ad una lettere e tenere nota del numero di volte che quella lettera è usata.
Nel programma che segue, ★ è usato come carattere di terminazione. Altri caratteri che non sono lettere, sono visualizzati sullo schermo, ma ignorati. Il programma consente di battere un messaggio e quindi di visualizzare la frequenza di ciascuna lettera:

```
10 DIM T(26)
20 GET X$:IF X$=""THEN 20
30 IF X$="★" THEN 90
40 PRINT X$;
50 IF X$ < "A" OR X$ > "Z" THEN 20
60 P = ASC(X$) - 64
70 T(P) = T(P) + 1
80 GOTO 20
90 PRINT
100 FOR P = 1 TO 26
110 PRINT T(P);
120 NEXT P
130 STOP
```

Glossario

T(26): Matrice di contatori T(1) per A, T(2) per le B e così via fino a T(26) per le Z.
X$: Carattere corrente
P: Codice ASCII del carattere corrente meno 64.
È usato come indice per T.

In questa forma il programma produrrà una serie piuttosto disordinata di numeri. È possibile migliorare l'output usando la funzione CHR$ che è l'opposto della funzione ASC: essa converte un numero in codice ASCII nella corrispondente stringa di un carattere. Per esempio, il comando

```
PRINT CHR$(68)
```

dà D.
Possiamo cambiare le ultime righe del programma in modo che risulti

```
100 FOR P=1 TO 26 STEP 2
110 PRINT CHR$(P+64);T(P),
    CHR$(P+65);T(P+1)
120 NEXT P
130 STOP
```


Il programma visualizza ora una tabella ordinata, lunga 13 righe, con due entrate per riga. L'esempio che segue fornisce una visualizzazione tipica:

```
SIBELIUS
WAS VERY REBELIUS
WHEN SCORING FOR THE TIMPANI
IN HIS FIRST SYMPANI

A 3     B 2
C 1     D 0
E 6     F 2
G 1     H 3
I 10    J 0
K 0     L 2
M 2     N 5
O 2     P 2
Q 0     R 5
S 8     T 3
U 2     V 1
W 2     X 0
Y 2     Z 0
```

In questo programma la funzione CHR$ è usata per convertire la sequenza numerica 1,2,3 . . .26 nelle stringhe "A", "B", "C", . . ."Z".

USO DI ASC - PER IGNORARE L'INPUT ILLECITO

La seconda area in cui ASC è utile è nel gestire dati che per qualsivoglia ragione non possono essere trattati dal normale comando INPUT. Per dare un semplice esempio, si pensi a disegnare una interfaccia per gli utenti così ingenui — o così goffi —da non riuscire a scrivere un numero senza battere almeno uno o più tasti sbagliati. Si vuole rendere la cosa più facile facendo in modo che la macchina ignori tutti i tasti salvo le dieci cifre decimali da 0 a 9, il tasto DEL per cancellare gli errori e il tasto RETURN per terminare il numero. Se un tasto viene totalmente ignorato non viene neppure visualizzato sullo schermo, cosicchè l'utente non deve nemmeno rendersi conto di che cosa ha effettivamente battuto.
Una adatta specifica, schema di flusso e subroutine sono indicati qui di seguito.
Notare che tutti i caratteri significativi compresi tra DEL e RETURN, sono rilevati dai rispettivi codici ASCII. Il tasto DEL toglie il carattere più a destra della stringa che viene assemblata. Esso inoltre sostituisce il simbolo visualizzato sullo schermo con uno spazio quindi sposta il cursore all'indietro in modo che il successivo carattere battuto appaia nel posto giusto. Così la cancellazione di un carattere richiede tre caratteri: cursore a sinistra, spazio e cursore a sinistra.

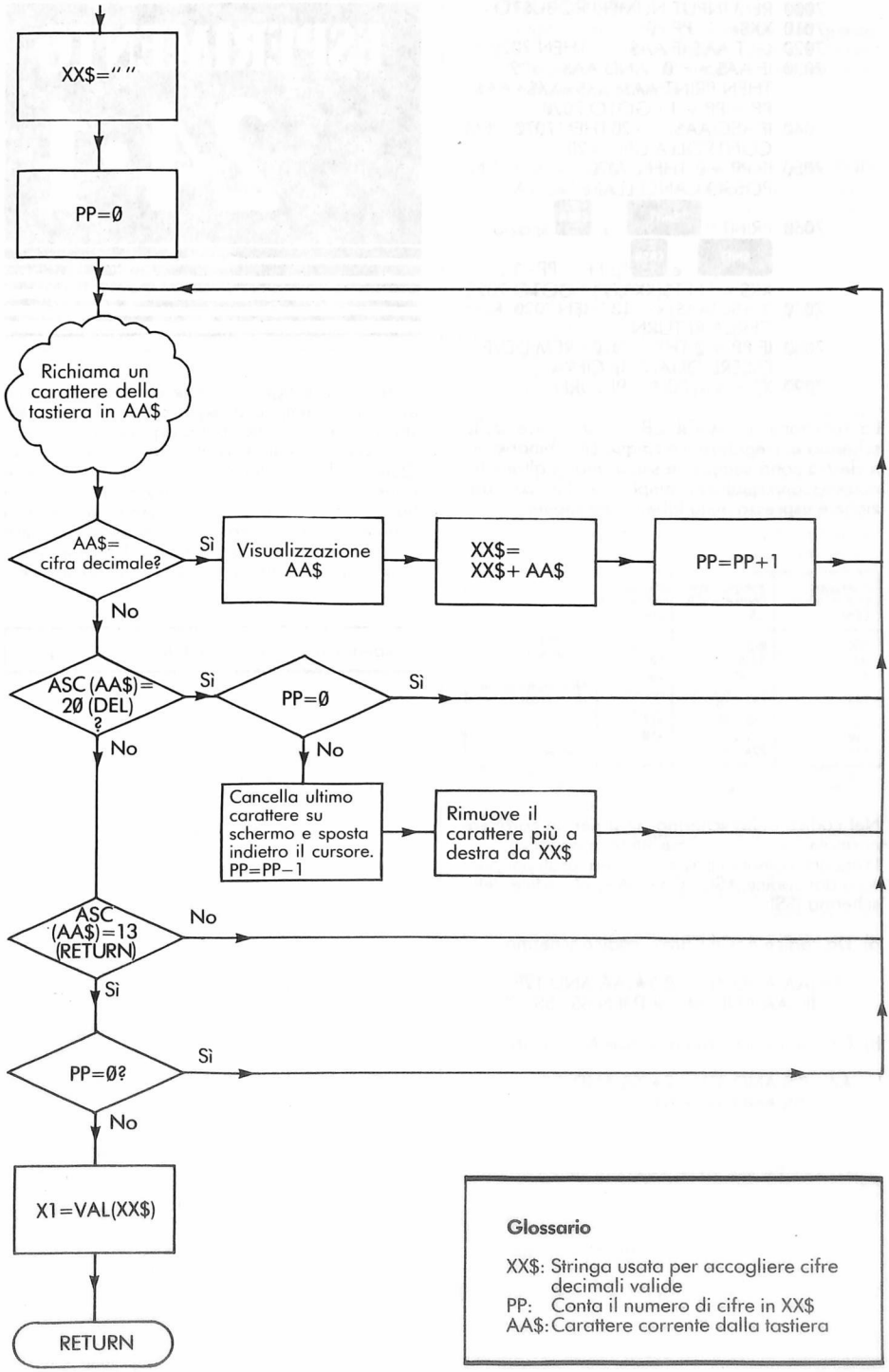
Specifica della subroutine

Scopo: Leggere un numero dalla tastiera, ignorando tutti gli altri caratteri privi di significato

Righe: da 7000 a 7090

Parametri: Output: Risultato fornito in X1

Locali: PP, AA$, XX$


XX$=""
PP=Ø
Richiama un carattere della tastiera in AA$
AA$= cifra decimale?
Sì
Visualizzazione AA$
XX$= XX$+ AA$
PP=PP+1
No
ASC (AA$)= 2Ø (DEL) ?
Sì
PP=Ø
Sì
No
No
Cancella ultimo carattere su schermo e sposta indietro il cursore. PP=PP−1
Rimuove il carattere più a destra da XX$
ASC (AA$)=13 (RETURN)
No
Sì
PP=Ø?
Sì
No
X1=VAL(XX$)
RETURN
Glossario
XX$: Stringa usata per accogliere cifre decimali valide
PP: Conta il numero di cifre in XX$
AA$: Carattere corrente dalla tastiera

```
7000 REM INPUT NUMERI ROBUSTO
7010 XX$="": PP=0
7020 GET AA$:IF AA$="" THEN 7020
7030 IF AA$>="0" AND AA$<="9"
     THEN PRINT AA$;:XX$=XX$+AA$:
     PP = PP + 1 : GOTO 7020
7040 IF ASC(AA$) <> 20 THEN 7070 : REM
     CONTROLLA DEL (=20)
7050 IF PP = 0 THEN 7020 : REM NON
     POSSO CANCELLARE NULLA
7060 PRINT " SHIFT e CRSR spazio
     SHIFT e CRSR ";: PP = PP-1 :
     XX$ = LEFT$(XX$,PP) : GOTO 7020
7070 IF ASC(AA$) <> 13 THEN 7020 : REM
     CERCA RETURN
7080 IF PP = 0 THEN 7020 : REM DEVE
     ESSERE QUALCHE CIFRA
7090 X1 = VAL(XX$) : RETURN
```


La relazione tra ASCII CBM e il codice dello schermo è irregolare. Le cinque cifre binarie più a destra sono sempre le stesse, ma le altre cifre non seguono qualsiasi semplice profilo. La situazione è espressa nella tabella che segue:

3 bit super. codice ASCII CBM	Range numer. codice ASCII CBM	3 bit super. codice dello schermo	Commenti
000	0-31	——	Caratteri di controllo
001	32-63	x01	
010	64-95	x00	
011	96-127	——	Non usati caratteri di controllo
100	128-159	——	
101	160-191	x11	
110	192-223	x10	
111	224-255	——	Non usati

Nel codice dello schermo, x=0 per un carattere normale e x=1 per un carattere in negativo.
I seguenti comandi possono essere usati per passare dal codice ASCII CBM (AA) al codice dello schermo (SS):

a) Da codice ASCII CBM a codice schermo:

```
SS=(AA AND 31) + 0.5*(AA AND 128):
    IF (AA AND 64)=0 THEN SS=SS+32
```

b) Da codice schermo a codice ASCII CBM:

```
AA=(SS AND 31) + 2*(SS AND 64) -
    (SS AND 32) + 64
```

ESPERIMENTO 24·1

Scrivere un programma che consenta all'utente di eseguire semplici disegni con una linea piuttosto spessa. Inizialmente il programma visualizza uno spazio in negativo al centro dello schermo. Questo è l'inizio di una linea continua che viene estesa di uno spazio vero l'alto, quando l'utente batte il tasto di funzione F1. Analogamente la linea è estesa a destra, verso il basso o a sinistra rispettivamente in risposta ai tasti F3, F5 e F7.
Usare il programma per disegnare una spirale.

Esperimento 24.1 completato	

IL COMANDO "ON"

Un'altra funzione che talvolta è utile è il comando ON. Questo comando consente al programma di saltare in una qualsiasi di parecchie direzioni in relazione al valore di una variabile o espressione.

Il VIC prende il valore della variabile (o espressione) e lo usa per scegliere uno dei numeri di label nella lista. Se il valore è 1, prende il primo, se è 2 prende il secondo e così via. Se il valore è minore di 1 o più alto del numero di label nella lista, non c'è alcun salto.
L'esempio suddetto è equivalente a:

```
IF A = 1 THEN 100
IF A = 2 THEN 150
IF A = 3 THEN 180
IF A = 4 THEN 195
IF A = 5 THEN 230
```

Se A è minore di 1 o maggiore di 6, verrà eseguita la riga che segue ON.
Nel BASIC VIC c'è anche una versione del comando ON che usa GOSUB invece di GOTO. Un uso del comando ON potrebbe essere in un programma che presenta all'utente un "menù" di opzioni come questo:

```
 10 PRINT"DESIDERI CONSIGLI PER"
 20 PRINT"MEMORIZZARE PROGRAMMI(1)"
 30 PRINT"USARE RND(2)"
 40 PRINT"DISEGNARE IMMAGINI(3)"
 50 PRINT"IL CODICE ASCII(4)"
 60 PRINT"PRODURRE SUONI(5)"
 70 INPUT"BATTI 1-5"; X
 80 ON X GOSUB 300, 400, 500, 600, 700
 90 GOTO 10
...
300 REM FORNISCE CONSIGLI PER
    MEMORIZZARE PROGRAMMI
...
390 RETURN
400 REM FORNISCE CONSIGLI PER USARE RND
...
490 RETURN
...
...
700 FORNISCE CONSIGLI PER PRODURRE
    SUONI
...
790 RETURN
```

IL COMANDO "END"

La maggior parte dei programmi in questo manuale hanno usato STOP per riportare il controllo alla tastiera al termine di un programma. Un comando alternativo è

```
END
```

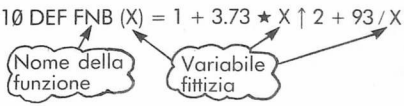
La differenza è che quando viene eseguito, END non dice in quale riga l'interruzione si è verificata; esso segnala soltanto "READY". È possibile usare STOP o END a piacere.

IL COMANDO "DEF"

La successiva funzione da descrivere, denominata DEF, è francamente una delle parti meno utili del BASIC VIC. Si suggerisce che, a meno che non si abbia una profonda conoscenza matematica e si sia particolarmente interessati alle formule, si salti direttamente alla successiva sezione che riguarda l'uso delle cassette di nastro.
La parola chiave DEF consente di denominare una formula e quindi di farvi riferimento per nome invece che scriverla completamente ogni volta. La definizione è scritta in termini di una variabile "fittizia" che è sostituita da un valore effettivo ogniqualvolta la formula viene usata. Il nome della formula deve contenere le lettere FN seguite da una o due lettere o da una lettera e da una cifra.

```
FNA o FNX o FNQC o FNG1
```

sono tutti nomi di formule appropriati.
Una definizione di formula potrebbe essere la seguente:



Una volta che una funzione è stata inserita in un programma, può essere usata scrivendone il nome con un adatto argomento. Pertanto:

```
20 Q = FNB(77)
```

servirà invece di

```
20 Q = 1 + 3.73 ★ 77 ↑ 2 + 93/77
```

o

```
30 PRINT FNB(S)
```

può essere usato per

```
30 PRINT 1 + 3.73 ★ S ↑ 2 + 93 / S
```

o di nuovo

```
40 ZZ = FNB(P-Q)
```

è ora un modo valido per scrivere

```
40 ZZ = 1 + 3.73 ★ (P-Q) ↑ 2 + 93 / (P-Q)
```

Notare che in ogni caso la variabile fittizia X è sostituita dall'argomento di FNB.
DEF soffre di parecchie limitazioni che ne riducono l'utilità. Tre delle più importanti sono:

a) Non è possibile avere più di una variabile fittizia in una definizione, ad esempio una formula comprendente SQR(X↑2+Y↑2) non è ammessa come parte di una definizione di funzione.

b) Non è possibile definire formule stringa vale a dire non può esistere FNB$ nel BASIC del VIC.

c) Non è possibile avere una subroutine (contrapposta ad un'espressione) per elaborare il valore. Si è limitati ad una formula, quantunque si possa pensare che le subroutine possano andare bene.

ESPERIMENTO 24·2

(Solo per i matematici!)

a) Definire una funzione FNA per elaborare la formula

X↑3 + (X+7)↑2 − 100

Usarla per tabulare il valore di $x^3(x+7)^2-100$ per valori di x compresi fra 2 e 3, procedendo ad intervalli di 0.1. Stimare anche il valore di x per il quale.

$x^3 + (x-7)^2 - 100 = 0$

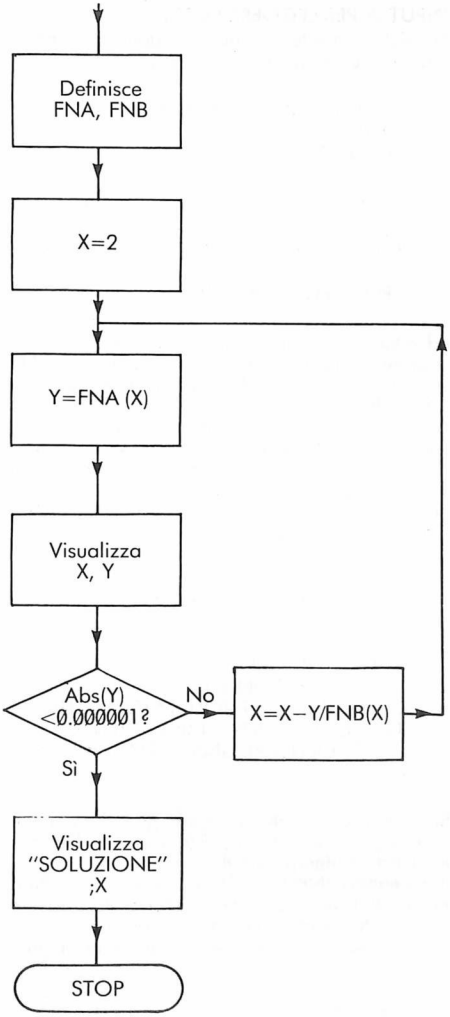
b) Definire una seconda funzione FNB per la formula

3 ★ X↑2 + 2 ★ (X + 7)

(Il calcolo denota che si tratta della derivata della prima funzione rispetto a X).
Scrivere ora un programma per calcolare una soluzione approssimata all'equazione

$x^3 + (x - 7)^2 - 100 = 0$

usando il metodo di Newton-Raphson. Uno schema di flusso appropriato è:



Glossario

X: Valore corrente di x
Y: Valore corrente di $x^3 + (x-7)^2-100$

Esperimento 24.2 completato

MEMORIZZAZIONE E RICHIAMO DI DATI SULLA CASSETTA

I lettori non esperti di matematica ritornino qui! Ora osserveremo i comandi per memorizzare e richiamare i dati (invece che i programmi) sulle cassette di nastro. Per esempio, si potrebbe avere una grande raccolta di osservazioni scientifiche o risposte ad un questionario che si vuole analizzare con parecchi programmi diversi. Chiaramente vale la pena di tenere questi dati in forma leggibile dalla macchina in modo da non doverli ribattere ripetutamente.
L'unità base di memoria su una cassetta di nastro è il "file". Esso è costituito da tre sezioni:

CODA	CORPO DEL FILE	TESTATA
marcatore di fine file	caratteri	nome del file

Direzione del movimento del nastro →

Questo diagramma mostra un segmento di nastro "svolto" dalla cassetta. La testata viene per prima e identifica il file contenendone il nome. Il nome può essere qualsiasi stringa di caratteri di una ragionevole lunghezza, ad esempio "DATI SATELLITE" o "INDIRIZZI CLUB TENNIS".
Successivamente viene il CORPO DEL FILE. Esso contiene una sequenza di caratteri e può essere lungo a piacere fino ad un'intera cassetta (90 minuti). Ciascun minuto di registrazione contiene circa 1200 caratteri.
Il corpo del file è diviso in "blocchi" di dimensioni uguali ciascuno dei quali contiene 256 caratteri. Ciascun blocco è seguito da uno spazio che consente al meccanismo del nastro di fermare e far ripartire il nastro fra i blocchi. Infine la CODA contiene un gruppo speciale di simboli che contrassegnano la fine del file. In pratica non occorre sapere molto a proposito dei dettagli in quanto il sistema funziona automaticamente.

PRINT # - PER SCRIVERE I DATI

Una singola cassetta di nastro può contenere parecchi file diversi registrati uno dopo l'altro. Il solo possibile inconveniente con questa configurazione è che per leggere i file che si trovano in fondo alla coda occorre far passare per prima cosa quelli che si trovano all'inizio.
Per scrivere un file su una cassetta, si usano tre nuovi comandi:

```
OPEN 1,1,2,"nome file"
PRINT # 1,
CLOSE1
```

Per iniziare un file, il programma deve dare un comando OPEN nella forma sopra indicata. Il nome del file può essere scelto liberamente ma i numeri 1,1 e 2 devono essere scritti precisamente come indicato.
Quando viene eseguito OPEN, il VIC fa comparire un messaggio sullo schermo che dice

PRESS RECORD AND PLAY ON TAPE

Caricare una cassetta vergine nel registratore (o se non è vergine una che è stata riavvolta oltre i file o i programmi che verranno ancora usati) e premere i tasti di controllo corretti. Il nastro deve essere sufficientemente lungo in relazione al file che si vuole scrivere, dato che non c'è modo di cambiare i nastri a metà del file. Ricordarsi di premere RECORD in modo che rimanga abbassato. Se ci si dimentica di farlo, il VIC esegue tutti i movimenti per scrivere un file ma non inserisce effettivamente alcuna informazione sul nastro. Fare quindi attenzione. Una volta che il nastro è caricato, il VIC scriverà una testata con il nome del file. Quindi è possibile iniziare a trasmettere i dati con il comando


```
PRINT # 1,
```

Notare che tutti gli 8 caratteri in questa parola chiave sono inseparabili e devono essere battuti esattamente come indicato. In particolare la virgola è essenziale e non è possibile usare il punto interrogativo invece di PRINT.
La parola chiave deve essere seguita dai nomi delle variabili che si vogliono scrivere. Se ce n'è più di una nel comando, i nomi devono essere separati dalla sequenza; ",";. Le variabili possono essere numeri, stringhe od una combinazione di entrambi. Ecco alcuni esempi:

```
PRINT # 1,X
PRINT # 1,P$
PRINT # 1,Q$(J);",";X(J);",";R$(J+1)
```

È possibile avere un numero di variabili a piacere nel comando PRINT # 1, posto che

a) La lunghezza del comando non superi gli 88 caratteri (questo è il limite normale che si applica a tutti i comandi)

b) Il numero totale di caratteri inviato al nastro da un qualsiasi comando sia inferiore a 80.

Se non si viola la seconda regola nulla sembra andare storto quando si scrivono i dati ma non è più possibile leggerli successivamente. Far quindi attenzione! È possibile, naturalmente, usare PRINT#1 ripetutamente all'interno di una iterazione per scrivere tutte le informazioni desiderate. Se si scrivono più di poche variabili si noterà che il nastro si muove a scatti e ciò in quanto il VIC ha un serbatoio interno di informazioni che agisce come "buffer" ossia come una memoria di transito tra il programma e il nastro. Quando la macchina esegue i comandi PRINT#1, i dati usati vengono per prima cosa raccolti nel buffer. Quando il buffer è pieno i suoi contenuti vengono inviati alla cassetta in una singola "scarica" per scrivere un blocco. Quindi il nastro si ferma, il buffer viene svuotato e il processo riparte da capo. Quando sono state scritte tutte le informazioni che si vogliono registrare, impartire un comando CLOSE1. Ciò forza il VIC a scrivere un altro blocco (quantunque il buffer possa essere soltanto parzialmente pieno) e una coda con marcatore di fine file.

INPUT # PER LEGGERE I DATI

Per richiamare le informazioni dalla cassetta di nastro, occorrono le tre istruzioni:

```
OPEN 1,1,Ø, "NOME FILE"
INPUT # 1,
CLOSE1
```

Il comando OPEN con lo Ø davanti al nome del file (invece di 2) fa sì che il VIC apra un file per la *sola lettura*. La macchina visualizza il messaggio

```
PRESS PLAY ON TAPE
```

ed attende che s'inserisca la cassetta nel registratore e si prema PLAY. *Non premere* RECORD se non si vogliono perdere dati preziosi. Quando il VIC rileva che il nastro è caricato, inizia a cercare sul nastro un file con un nome che corrisponde a quello nel comando OPEN. Il processo di abbinamento richiede soltanto che la stringa nel comando OPEN sia la stessa dell'*inizio* del nome del file. Se il nome effettivo del file è "DATI GIOVEDI" il file sarà aperto da uno qualsiasi dei seguenti comandi:

```
OPEN 1,1,Ø,"DATI GIOVEDI"
o OPEN 1,1,Ø,"DATI"
o OPEN 1,1,Ø,"D"
o OPEN 1,1,Ø,"":REM LA STRINGA NULLA
        APRE OGNI FILE
o OPEN 1,1,Ø:REM PUOI OMETTERE TITOLO
o X$= "DAT": OPEN 1,1,Ø, X$ : REM
        PUOI USARE UNA VARIABILE
```

Se il titolo è indicato come stringa nulla o viene omesso, il comando aprirà il primo file che incontra, qualunque sia il suo nome.
Il comando INPUT # 1, è come il comando PRINT # 1 ma alla rovescia. La parola chiave è seguita dai nomi delle variabili da leggere dal nastro, separati da virgole, se ce n'è più d'uno. Ne sono esempi:

```
INPUT # 1,Z
INPUT # 1,P$
INPUT # 1,R$,Q,T$
```

Notare che il numero e il tipo delle variabili che seguono INPUT # 1, deve essere indentico a quello usato per inserire i valori sul nastro all'inizio. Il comando

```
INPUT # 1,A$,B$,C:REM DUE STRINGHE
E UN NUMERO
```

potrebbe essere usato per estrarre i dati originariamente scritti da

```
PRINT # 1,A$;",";B$;",";C :REM DUE
        STRINGHE E UN NUMERO
o PRINT # 1,Z$(Q);",";"PORTA A";",";
        X:REM DUE STRINGHE E UN
        NUMERO
```

ma se i dati fossero stati scritti nella forma Y;",";P$ il suddetto INPUT non funzionerebbe. Quando il sistema legge da una cassetta di nastro, possono succedere varie cose impreviste. Per tener conto di questa difficoltà, il VIC riserva una variabile speciale denominata ST (per "Status=Stato") e la usa per dare un rapporto codificato ogni volta che viene eseguito il comando INPUT=1. Il valore 0 significa che tutto va bene. 64 segnala che si è raggiunto la fine del file e altri valori implicano che qualcosa è andato male: il nastro si è corrotto, o probabilmente non era stato correttamente registrato all'inizio. Per illustrare l'azione del registratore a cassetta, ecco un paio di programmi. Il primo consente di disegnare una figura sullo schermo usando il cursore e i comandi del colore e quindi registrare questa immagine su un file "prendendo" (PEEK) i valori dello schermo e le RAM dei colori e scrivendoli come numeri. Il secondo programma legge il file e ricostruisce l'immagine. Studiare entrambi i programmi attentamente e notare il modo in cui ST è stato usato. Quindi, impostarli uno per uno e provarli.

```
 10 OPEN 1,1,2,"IMMAGINE SCHERMO"
 20 PRINT [SHIFT] e [CLR HOME] [CRSR]
    DISEGNA IMMAGINE A PIACERE"
 30 PRINT"USANDO I COMANDI DEL
    CURSORE"
 40 PRINT"E DEL COLORE."
 50 PRINT"LASCIA CURSORE SU RIGA"
 60 PRINT"SUPERIORE CHE
 70 PRINT"DEVE ESSERE VUOTA."
 80 PRINT"QUINDI PREMI RETURN"
 85 FOR S=1 TO 5000: NEXT S
 90 INPUT" [SHIFT] e [CLR HOME] ";X$:
    REM UTENTE DISEGNA IMMAGINE
100 FOR J=0 TO 505: REM ESAMINA RAM
    SCHERMO E COLORE
110 PRINT # 1,PEEK(7680+J);",";PEEK
    (38400+J)
120 NEXT J
130 CLOSE1
140 STOP
```

ora riavvolgere il nastro e battere il seguente programma:

```
 10 OPEN1,1,0,"IMMAGINE SCHERMO"
 20 J=0
 30 INPUT # 1,X,Y:REM RICAVA CODICI
    COLORE E SCHERMO
 40 IF ST<>0 THEN 70
 50 POKE 7680+J,X: POKE 38400+J,Y
 60 J = J+1:GOTO 30
 70 IF ST = 64 THEN 100:REM CONTROLLA
    FINE FILE
 80 PRINT"DIFETTO NASTRO"
 90 STOP

100 CLOSE
110 GOTO 110:REM ARRESTA ITERAZIONE
    SE TUTTO OK
```

GET #

Un altro comando che talvolta è usato con le cassette di nastro è

```
GET # 1,
```

Questo comando è piuttosto simile a PRINT # 1, salvo che trasferisce singoli caratteri. Esso apparirebbe in sequenze tipo

```
...
100 GET A$:IF A$=""THEN 100:REM
    OTTIENI UN CAR. DA TASTIERA
110 PRINT A$:REM VISUALIZZALO
120 PRINT #:1,A$:=REM INVIALO A
    CASSETTA DI NASTRO
...
```

e

```
...
200 GET # 1,X$:REM OTTIENI UN
    CARATTERE DA NASTRO
210 IF ST<>0 THEN 300:REM SALTA SE
    FINE FILE O ERRORE
220 PRINT X$;
...
```

PROBLEMI

Come si è visto, il VIC fornisce le funzioni base per scrivere dati su una cassetta e per richiamarli successivamente. Queste funzioni sono primitive e hanno taluni inconvenienti:

1) Lettura e scrittura sono lente:

2) L'affidabilità del sistema a cassetta non è perfetta. Le cassette possono essere già danneggiate al momento dell'acquisto oppure possono essere danneggiate dall'umidità, da una manipolazione scorretta, dell'eccessivo caldo o freddo o da forti campi magnetici. Tutte queste circostanze possono produrre errori nei file. Il tasso di errore nel memorizzare i dati è molto più alto di quello che si ha nei programmi, in quanto

   a) I file di dati sono generalmente più lunghi.

   b) Non c'è modo di verificare i file di dati come è possibile fare con i programmi

   c) Il comando SAVE nel VIC può registrare effettivamente ciascun programma due volte, su parti diverse del nastro. Il sistema può pertanto avere due possibilità di leggere il programma correttamente. I dati per contro, sono registrati soltanto una volta e un singolo errore può essere fatale.

3) Il VIC può gestire soltanto una cassetta. Ciò significa che non è possibile correggere un file o aggiungervi dati a meno che non sia sufficientemente corto da poterlo inserire completamente nella memoria del VIC.

Chi è seriamente interessato a memorizzare grandi quantità di dati deve comprare un'unità a floppy disk. La Commodore dispone di una buonissima unità del genere specificamente studiata per il VIC. Se si decidesse di procedere con le cassette, usare nastro della migliore qualità che è possibile trovare, tenere il registratore a cassetta pulito e in perfette condizioni e soprattutto prepararsi spiritualmente a qualsiasi inconveniente occasionale.

ESPERIMENTO 24·3

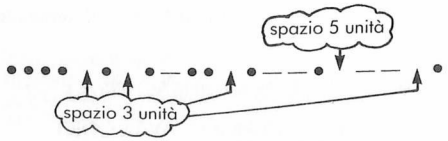
In questo esperimento si disegnerà e si costruirà un insegnante meccanico del Codice Morse.
Il Codice Morse è usato per trasmettere informazioni per radio. Ciascuna lettera dell'alfabeto ha un codice che si compone di punti e di linee. Il codice completo è

A	•—	J	•———	S	•••
B	—•••	K	—•—	T	—
C	—•—•	L	•—••	U	••—
D	—••	M	——	V	•••—
E	•	N	—•	W	•——
F	••—•	O	———	X	—••—
G	——•	P	•——•	Y	—•——
H	••••	Q	——•—	Z	——••
I	••	R	•—•		

L'unità base di tempo è il punto e altri intervalli di tempo sono definiti come segue:

Linea	3 punti
Distanza tra punti e linee della stessa lettera	1 punto
Distanza tra lettere	3 punti
Distanza tra parole	5 punti

Per esempio il messaggio HELP ME, verrebbe trasmesso nella forma:



Nel Codice Morse la punteggiatura è ignorata. Per iniziare, scriveremo una subroutine costituita da due parametri:

a) Una stringa contenente una frase

b) Un numero che dà la velocità desiderata di trasmissione in millesimi di secondo per punto.

La subroutine converte la frase in Morse e la trasmette sul generatore di suoni del VIC alla velocità richiesta.

SUGGERIMENTO: Impostare una subroutine che emetta un suono — o una pausa — di un'adatta lunghezza. Gestirla con programma che usa una tabella bidimensionale come questa:

1	3	0	0	0	(A)
3	1	1	1	0	(B)
3	1	3	1	0	(C)
3	3	1	1	0	(Z)

La tabella contiene il codice per ciascuna delle lettere da A a Z e deve essere impostata usando le istruzioni READ e DATA.
Una volta soddisfatti della subroutine, passare alla seconda parte dell'esperimento. Questo comporta due programmi:

a) un programma per immettere un testo (un'intera serie di frasi) tramite tastiera e la loro registrazione in un file su cassetta. Usare la frase fittizia "ZZZZ" per terminare l'input.

b) Un programma per leggere le frasi e trasmetterle in Codice Morse a qualsiasi velocità desiderata.

Quando i programmi funzionano, possono essere usati per preregistrare messaggi e trasmetterli ad altissima velocità risparmiando tempo e aumentando la capacità di un canale radio. Essi sono anche utili se si vuole far pratica nel ricevere segnali Morse, dato che è possibile iniziare lentamente e gradualmente accelerare la velocità. Per ottenere i migliori risultati occorre servirsi di qualcuno che batta le frasi da trasmettere in modo da non sapere cosa aspettarsi in anticipo.

Esperimento 24.3 completato	

ESPERIMENTO 24·4

Questo esperimento invita a disegnare e a scrivere un programma di "appuntamenti" col computer.
Caricare il programma "MAKENAMES" dalla cassetta di nastro. Inserire una cassetta vergine nel registratore ed eseguire il programma. Essa produrrà una lista di 100 persone e scriverà i rispettivi dettagli personali sulla cassetta in un file denominato "COMPUTER DATA".
Il record per ciascuna persona si compone delle seguenti voci (nell'ordine in cui sono registrate):

NAME (nome) (stringa)

ADDRESS (indirizzo) (stringa)

TOWN (città) (stringa). Una fra EDINBURGH, GLASGOW, DUNDEE o ABERDEEN

SEX (sesso) (stringa). Una fra F o M (femmina o maschio)

AGE (età) (numero)

HEIGHT (altezza) (numero). Altezza in pollici

MAIN HOBBY (hobby principale (stringa)	Entrambi ricavati dalla seguente lista: FOOTBALL, TENNIS, HILLWALKING (passeggiata in montagna), OPERA (opera), JAZZ (Jazz), ROCK (rock) THEATRE (teatro), READING (lettura), POLITICS (politica), STUDYING (studio), CHESS (scacchi), GAMBLING (giochi d'azzardo), HORSE-RACING (corse dei cavalli), CARS (auto), MOTOR-BIKES (moto), CYCLING (bicicletta), MEETING PEOPLE (incontrare persone)
SECOND HOBBY (hobby secondario (stringa)	

POLITICS (politica) (stringa)	Uno fra CONSERVATIVE (conservatore), LABOUR (laburista), LIBERAL (liberale), SDP, OTHER (altri), NONE (nessuno)

Una volta ottenuto il file delle persone, iniziare scrivendo il programma più semplice che apre il file, legge e visualizza i record uno alla volta. Successivamente disegnare e scrivere un programma di "appuntamenti" che chiede i particolari personali di un "cliente" e quindi cerca il file e preleva la "persona più adatta.
Si supponga che la persona scelta debba vivere nella stessa città. Le persone che soddisfano questi vincoli segnano punti di "bonus" sulla scala arbitraria seguente:

Compatibilità di età: se la ragazza ha la stessa età o non è più giovane di 4 anni dell'uomo:	3 punti
Compatibilità di altezza: se la ragazza ha la stessa altezza o non è più piccola dell'uomo di 4 pollici:	2 punti
Hobbies: per ciascuna hobby condiviso:	5 punti
Politica: per un punto di vista politico comune:	3 punti
se uno vota per i laburisti e l'altro per i conservatori: *meno*	4 punti

L'accoppiamento "migliore" è quello con il punteggio più elevato.

Attenzione: Se si trova qualcuno che realmente piace, è bene rinunciare subito a mettersi in contatto: le persone sul file non sono reali.

Esperimento 24.4 completato	

UNITA': 25

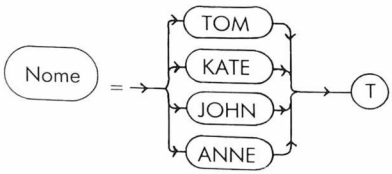
DISEGNO DI PROGRAMMI - CASI PRATICI

L'argomento del disegno di programmi è stato un tema ricorrente nel corso di questo manuale. Questa unità finale comprende alcuni casi pratici ciascuno dei quali mostra come un problema apparentemente complesso può essere risolto rapidamente e facilmente esaminandone la struttura e trasferendo questa struttura al programma stesso.

FRASI CASUALI - LA GRAMMATICA DELLA LINEA DEL TRAM

Il primo dei nostri esempi pratici riguarda la produzione di frasi casuali come quelle nell'Unità 6. Chiaramente queste frasi non possono rappresentare semplici raccolte di parole messe insieme in qualsiasi ordine; se devono avere un senso, devono seguire le regole della grammatica.

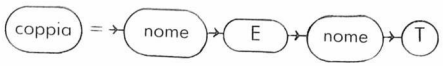
La notazione spesso usata per queste regole è detta una grammatica "a linea del tram". Si supponga che in un certo punto in una frase che viene costruita occorra scegliere il nome di una persona dalla lista di TOM, KATE, JOHN, ANNE, possiamo scrivere questa parte della grammatica con l'aiuto del diagramma



Si immagini un tram che entri nel diagramma da sinistra (nella direzione della freccia). Quando il guidatore arriva ad uno scambio, la direzione che egli prende, viene decisa a caso. Il tram al limite può arrivare al capolinea alla destra, ma può farlo attraverso uno qualsiasi di quattro percorsi: TOM, KATE, JOHN o ANNE.

In questo diagramma ciascun ovale contiene una parola che è una possibile candidata per parte della frase che viene costruita. Gli ovali nei diagrammi della linea tranviaria possono anche contenere i nomi di altri diagrammi. La differenza è sempre chiara in quanto i nomi dei diagrammi sono scritti in minuscolo.

Osservare il seguente diagramma a linee tranviarie:

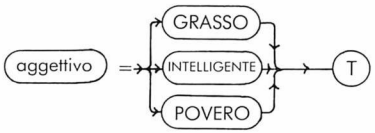


(dove "nome" è il diagramma con TOM, KATE, JOHN e ANNE).

Per il guidatore del tram, il primo nome è un tipo di subroutine. Nel momento in cui il tram ha trovato la sua strada attraverso il diagramma delle coppie, può essere arrivato con una delle seguenti frasi:

TOM E JOHN
ANNE E TON
o addirittura KATE E KATE

Se si vuole definire una frase con un numero variabile di parole, è possibile inserire una derivazione nel diagramma. Se si definisce:

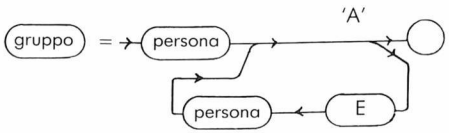


la grammatica

darà

JOHN POVERO
ANNA GRASSA
KATE
o TOM POVERO

poichè il guidatore del tram può scegliere a caso se seguire la strada dell'aggettivo o no. Le grammatiche a linea tranviaria possono contenere delle iterazioni. Si consideri il diagramma



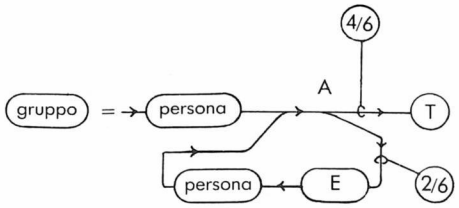
Ricordarsi che il guidatore ha la libera scelta quando arriva al punto A nel diagramma. Se va diritto, raggiungerà il capolinea e terminerà la frase. Se gira a destra aggiungerà un'altra persona. Alcune delle frasi che egli può produrre, sono

TOM
TOM INTELLIGENTE E KATE
TOM GRASSO E ANNA POVERA
E JOHN INTELLIGENTE
TOM E JOHN E KATE POVERA
oppure TOM GRASSO E TOM GRASSO
E TOM GRASSO

Sotto alcuni aspetti il diagramma a linee tranviarie è come uno schema di flusso. La differenza essenziale è che nei punti di scambio tipo A, la scelta del binario avviene casualmente e non in risposta ad una specifica domanda. L'elemento casuale è essenziale altrimenti il diagramma produrrebbe la stessa frase ogni volta.

PROBABILITÀ

Quantunque la scelta del percorso non sia determinata in anticipo, si vuole sempre mantenere tuttavia un certo tipo di controllo su di essa, altrimenti il guidatore potrebbe decidere di continuare l'iterazione all'infinito. È possibile fare ciò aggiungendo una probabilità o una possibilità a ciascuno dei percorsi possibili.
Un modo per far ciò consiste nel dare al guidatore un dado a sei facce e istruirlo a lanciarlo ogniqualvolta deve compiere una scelta. Nel punto A, per esempio, gli diremo di girare a destra se lanciando ottiene un 5 o un 6, ma di procedere fino al capolinea se ottiene un 1, un 2, un 3 o un 4. Ciò significa, nel lungo termine, che egli può prevedere di girare a destra due volte ogni sei volte che passa per A e procedere per quattro volte. Possiamo indicare questo sul diagramma aggiungendo dei marcatori di probabilità come questi:



Notare che le probabilità dei percorsi che traggono origine da un punto tipo A devono dare per somma 1, ossia la certezza, in quanto il guidatore del tram è costretto a prenderne in ogni caso uno. Se si considerano i marcatori di probabilità come frazioni, la loro somma deve dare 1. Nella definizione di (gruppo) essi corrispondono a:
4/6 + 2/6 = 6/6 = 1.
Si consideri ora come il VIC può essere messo in condizioni di produrre frasi casuali.
Le regole di base sono le seguenti:

1) La frase che viene costruita viene contenuta nella variabile stringa X1$. La variabile inizia come stringa nulla e le parole sono aggiunte una alla volta. Ciascuna parola è preceduta da uno spazio.

2) Ogni diagramma di grammatica separato è rappresentato da una subroutine. Uno dei suoi parametri, sia per l'input che per l'output, è X1$.

Osserviamo ora qualche operazione elementare. Per aggiungere un nome noto alla frase, dobbiamo semplicemente concatenarlo come segue:

```
X1$=X1$ + " E"
```

(notare lo spazio che precede)

Per scegliere una parola casuale da una lista, il metodo più semplice è di assicurarsi che tutte le parole candidate si trovino in una matrice. Si supponga che ci siano J di esse in elementi consecutivi iniziando da N$(K) per terminare in N$(K+J−1). Quindi, il seguente comando ne preleverà una a caso e la collegherà a X1$:

```
X1$=X1$+" "+N$(K+INT(J★RND(Ø)))
```

Ciò funziona in quanto l'espressione indice K+INT(J★RND(Ø)) ha la stessa probabilità di uscire con qualsiasi numero tra K e K+J−1: esattamente ciò che ci serve.
Per eseguire un percorso di linea tranviaria con una data probabilità, possiamo usare la condizione

```
RND(Ø)<p/q
(dove p/q è il marcatore di probabilità)
```

Se poniamo

```
RND(Ø)<4/6
```

la condizione sarà vera quattro volte su sei e falsa le altre due volte. Ciò significa che l'altro marcatore di probabilità (2/6) non deve essere scritto nel programma.
Possiamo ora costruire un programma per produrre frasi di gruppi così come sono definite dalla nostra grammatica. Iniziamo impostando una matrice con i nomi e gli aggettivi e inizializziamo X1$ ad un valore nullo. Ciò occupa le righe da 1Ø a 4Ø nel programma che segue.
Successivamente, scriviamo una subroutine per ciascuno dei diagrammi a linea tranviaria. Quello che inizia alla riga 1ØØØ è per un (nome). Le costanti nell'espressione indice sono 1 e 4 in quanto ci sono 4 nomi possibili che iniziano in N$(1). Analogamente, la subroutine che inizia in 11ØØ produce un (aggettivo) e le appropriate costanti nelle espressioni di indice sono 5 e 3.

La subroutine in 12ØØ dà una (persona). Stabiliamo la probabilità di usare un (aggettivo) 3/6 — il che indica che la probabilità di non averne una è pure 3/6: probabilità pari.
A 13ØØ troveremo la subroutine per il diagramma (gruppo). Notare con quanta precisione segue le linee tranviarie:

1310 sceglie la (persona) di partenza

1320 prende una decisione casuale se terminare la frase

1330 inserire la parola (E)

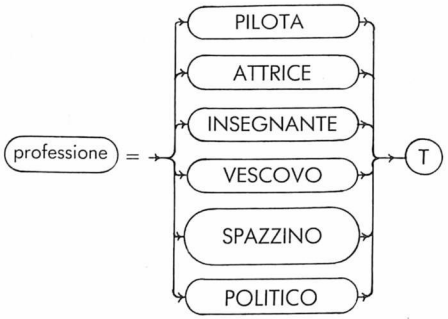
1340 sceglie un'altra (persona)

1350 riporta la subroutine al punto in cui decide se fermarsi oppure procedere di nuovo.

Infine, forniamo alcuni comandi per il guidatore nelle righe 40, 50 e 60.

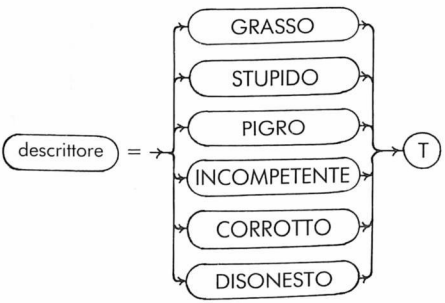
```
 10 DIM N$(7)
 20 N$(1)="TOM":N$(2)="KATE" :
    N$(3)="JOHN":N$(4)="ANNE"
 30 N$(5)="GRASSO":N$(6)=
    "INTELLIGENTE":N$(7)="POVERO"
 40 X1$=" "
 50 GOSUB 1300
 60 PRINT X1$
 70 GOTO 40

1000 REM NOME
1010 X1$=X1$+" "+N$(1+INT(4*RND(0)))
1020 RETURN


1100 REM AGGETTIVO
1110 X1$=X1$+" "+N$(5+INT(3*RND(0)))
1120 RETURN

1200 REM PERSONA
1210 IF RND(0) < 3/6 THEN GOSUB 1100:
     REM RICHIAMA AGGETTIVO
1220 GOSUB 1000 : REM RICHIAMA PERSONA
1230 RETURN

1300 REM GRUPPO
1310 GOSUB 1200 : REM RICHIAMA PERSONA
1320 IF RND(0) < 4/6 THEN RETURN : REM
     PUNTO "A" IN DIAGRAMMA
1330 X1$=X1$+"E"
1340 GOSUB 1200 : REM RICHIAMA ALTRA
     PERSONA
1350 GOTO 1320
```

Impostare questo programma ed eseguirlo. Provare quindi l'effetto di probabilità nelle righe 1210 e 1320.
Una volta che i principi della costruzione di frasi casuali sono stati stabiliti, possono facilmente essere estesi alle frasi complete. Studiare le seguenti definizioni e scrivere esempi delle frasi che essi potrebbero produrre.



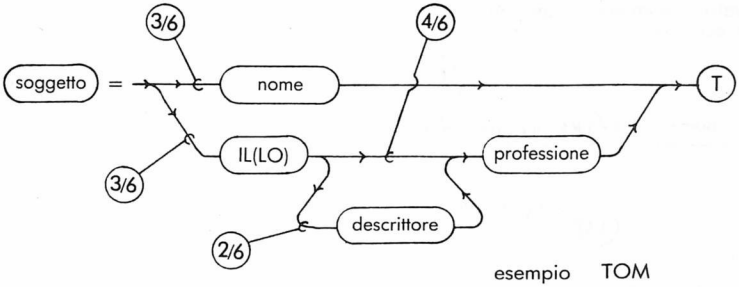
esemp.: VESCOVO

oppure .....



esemp.: PIGRO

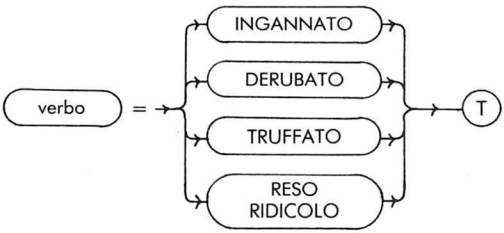
oppure .....



esempio TOM

o LO STUPIDO PILOTA

o .....

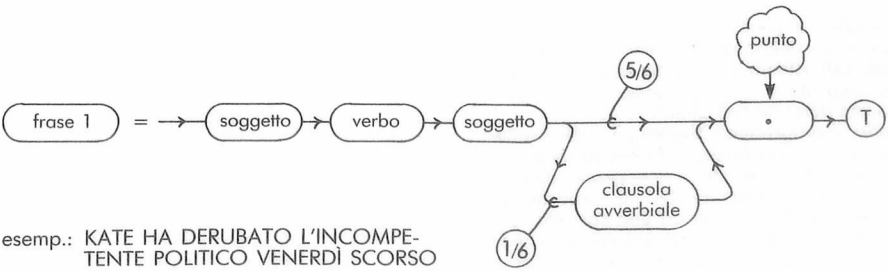


esempio DERUBATO

oppure .....

esempio IERI

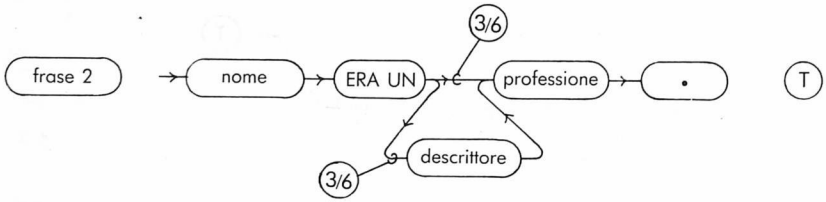
oppure .....



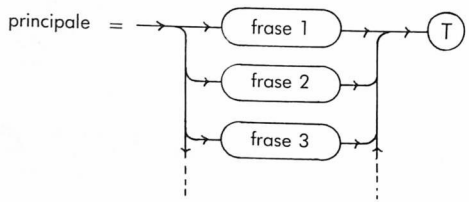
esemp.: KATE HA DERUBATO L'INCOMPETENTE POLITICO VENERDÌ SCORSO

oppure .....

È possibile creare qualsiasi numero di frasi con diverse definizioni, ad esempio:



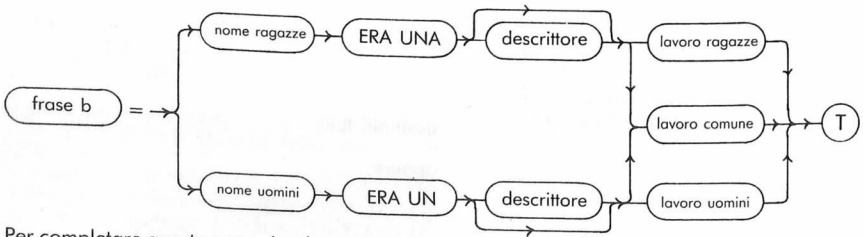
ed è possibile combinarle in un diagramma a linea tranviaria (principale) che comprende tutte le forme di frasi che si vogliono generare. Potrebbe cominciare:



A questo punto vale la pena di osservare il raggruppamento di parole. Nella serie attuale di diagrammi e linee tranviarie, una frase possibile è

TOM ERA UN ATTRICE PIGRA.

Per evitare questo tipo di assurdità, occorre separare i nomi in due gruppi e le professioni in tre: quelli limitati agli uomini (ad esempio VESCOVO), quelli limitati alle donne (ad esempio ATTRICE) e quelli aperti ad entrambi. Una definizione alternativa per la frase 2 sarebbe:



Per completare questo esempio, dovremmo consigliare alcuni dettagli pratici. Innanzitutto le frasi prodotte dal sistema dovrebbero essere correttamente disposte e ciò può essere fatto dalla subroutine descritta nell'Unità 21.
Secondo, il programma diventa più interessante per gli utenti se essi possono fornire proprie liste di parole per le varie categorie — probabilmente alterando le istruzioni DATA nel programma dopo che è stato caricato da una cassetta. Ciò implica che il programmatore non conosca nè le parole nè quante ce ne saranno in ciascun gruppo. Ci saranno chiaramente difficoltà nello scrivere espressioni indice adatte.

Questo problema può essere superato facendo in modo che il programma definisca una serie di "cartelli indicatori" ai vari gruppi di parole. Nell'attuale grammatica, abbiamo dei gruppi di parole: nomi, aggettivi, professioni, descrittori, verbi e clausole avverbiali. Diciamo all'utente di inserire la sua scelta per ciascun gruppo in una (o più) istruzioni DATA e di terminare con una "Z". L'utente potrebbe inserire:

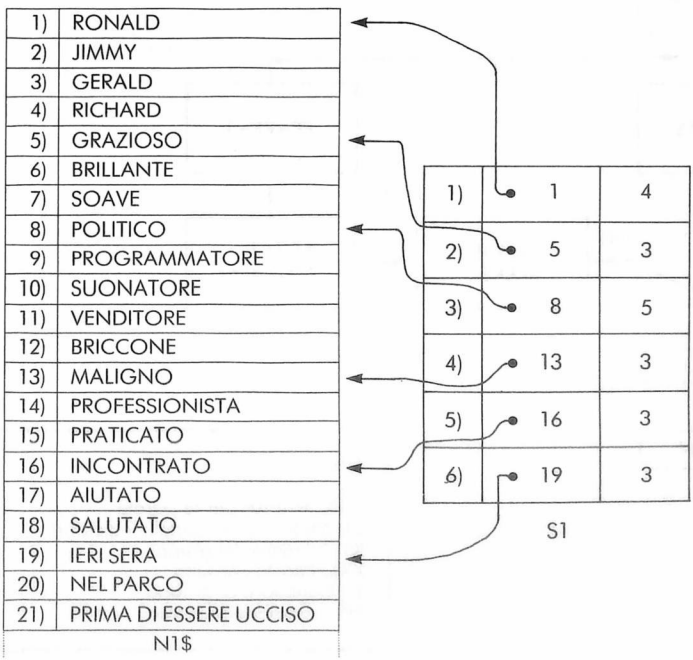
```
10 DATA RONALD,JIMMY,GERALD,
        RICHARD,Z
20 DATA GRAZIOSO,BRILLANTE,SOAVE,Z
30 DATA POLITICO,PROGRAMMATORE,
        SUONATORE,VENDITORE,
        BRICCONE,Z
40 DATA MALVAGIO,PROFESSIONALE,
        ESPERTO,Z
        50 DATA INCONTRATO,AIUTA-
        TO,
        SALUTATO,Z
60 DATA IERI SERA, NEL PARCO, PRIMA
        DI ESSERE UCCISO,Z
```

All'interno del programma disponiamo i dati in matrici: una con un elemento per ciascuna parola (salvo le Z) e una con le informazioni su ciascun gruppo. Le necessarie informazioni comprendono

a) La posizione iniziale (e cioè l'indice del primo elemento) nel gruppo

b) il numero di parole nel gruppo.

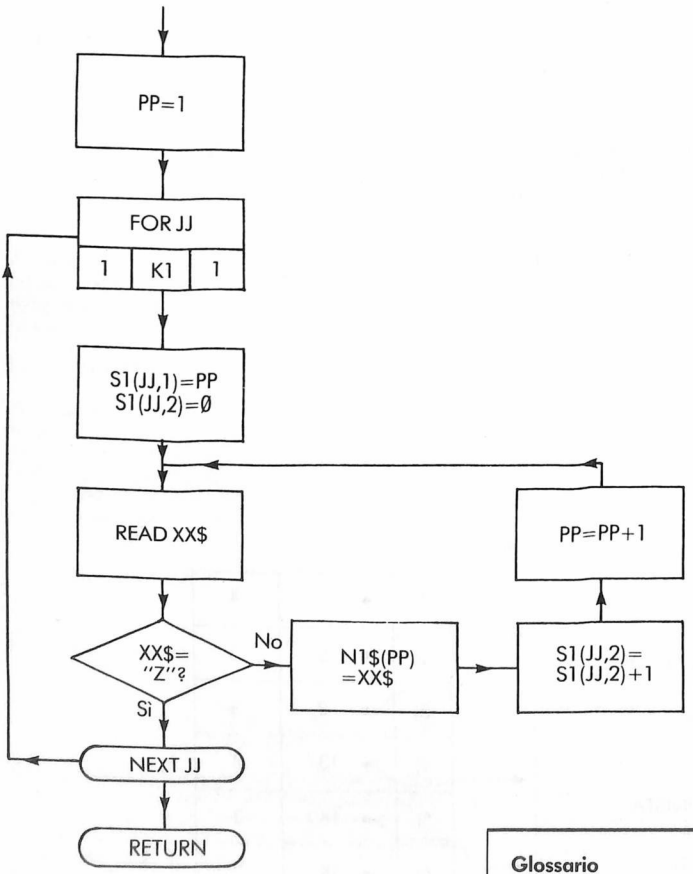
Un diagramma potrebbe aiutare a rendere tutto più chiaro:



In questa struttura di dati, la fila 3 della matrice S1 dice al programma che il gruppo 3 contiene 5 parole che iniziano in N1$(8). Un'istruzione per aggiungere una parola del gruppo 3 a X1$ verrebbe lettà così

$$X1\$=X1\$ + \text{""} + N1\$(\underbrace{S1(3,1)}_{=8} + INT(\underbrace{S1(3,2)}_{=5} \star RDN(\emptyset)))$$

Posto che i cartelli indicatori in S1 siano correttamente disposti, questa espressione funzionerà per qualsiasi raccolta di parole che l'utente fornisca.
La messa a punto dei cartelli indicatori è illustrata nel seguente schema di flusso:



Glossario

N1$: Matrice per le parole
S1: Matrice per i cartelli indicatori
K1: Numero dei gruppi
XX$: Parola corrente
JJ: Contatore di gruppo
PP: Contatore di parola

La specifica e il codice corrispondenti sono:

Specifica della subroutine

Scopo: Leggere gruppi di parole per generare frasi casuali.

Righe: 5000-5070

Parametri: *Output:* N1$ (parole), S1 (cartelli indicatori

Matrici vuote; K1: Numero di gruppi

Output: N1$, S1 (impostato come descritto nel testo)

Locali: PP,JJ,XX$

```
5000 REM LEGGI PAROLE E IMPOSTA
     SEGNAPOSTI
5010 PP=1
5020 FOR JJ = TO K1
5030 S1(JJ,1)=PP:S1(JJ,2) = 0
5040 READ XX$
5050 IF XX$<>"Z"THEN N1$(PP)=XX$:
     S1(JJ,2)=S1(JJ,2)+1:PP=PP+1:
     GOTO 5040
5060 NEXT JJ
5070 RETURN
```

Per riassumere: il programma per generare le frasi che abbiamo discusso si compone principalmente di subroutine che sono abbastanza strettamente modellate sulla grammatica delle frasi da costruire. In questo modo la struttura del problema è trasferita pressochè senza alterazione al programma stesso.

Scrivere un generatore di frasi complete che comporta parecchi tipi di frasi. Provarlo con gli amici e i parenti.

Esperimento 25.1 completato	

GIOCHI DI AVVENTURA O DI LABIRINTI

Nel successivo esempio pratico esamineremo due esempi in cui la struttura del problema è riflessa nei dati anzichè nel programma.
Esistono numerosi giochi per computer in cui l'eroe deve esplorare un castello/labirinto/universo, affrontare vari pericoli tipo dragoni o alieni e salvare una principessa/un modulatore ipertermico/e un intrepido esploratore dello spazio. La Commodore dispone di una eccellente gamma di questi giochi nella sua serie di avventure.
Il disegno di una semplice versione di uno di tali giochi può essere esposto come un diagramma del tipo illustrato in Fig. 25.2. Quando si inizia la codifica di un tale gioco, il modo più naturale consiste nel partire scrivendo mucchi di codici senza profilo tipo questo


```
10 PRINT " SHIFT e CLR HOME ";
20 PRINT "LA MISSIONE È DI RECUPERARE"
...
100 PRINT" OR B) ASPETTI E OSSERVI?"
110 INPUT X$
120 IF X$ <> "A" AND X$ <> "B" THEN
    PRINT "PROVA ANCORA": GOTO 110
130 IF X$ = "A" THEN 500
140 PRINT " SHIFT e CLR HOME ";
150 PRINT "SEI ATTACCATO DA"
...
...
270 PRINT OR B) FUGGI NEL BATTELLO DI
    SALVATAGGIO
280 INPUT X$
290 IF X$ <>"A" AND X$ <> "B" THEN
    PRINT "PROVA  ANCORA": GOTO 280
300 IF X$ = "A" THEN 500
...
e così di seguito
```

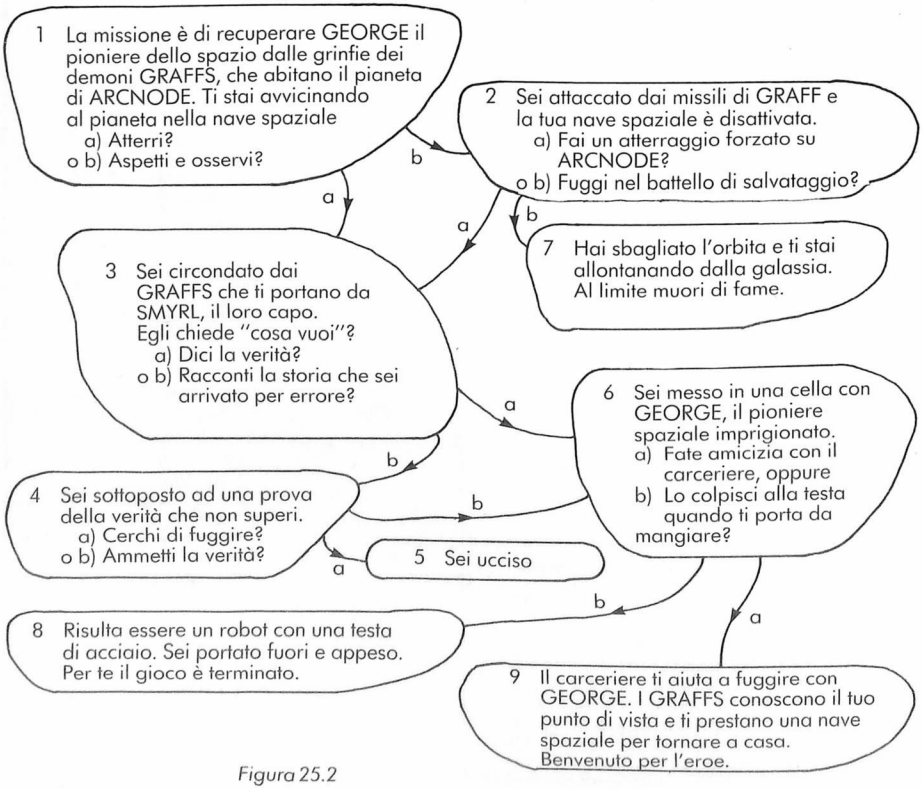


Figura 25.2

Questo programma è difficile da leggere e da modificare e si sviluppa rapidamente per occupare tutto lo spazio nella memoria.
Diamo uno sguardo a ciò che succede in ciascun punto o nodo nel diagramma. Ci sono due possibilità:

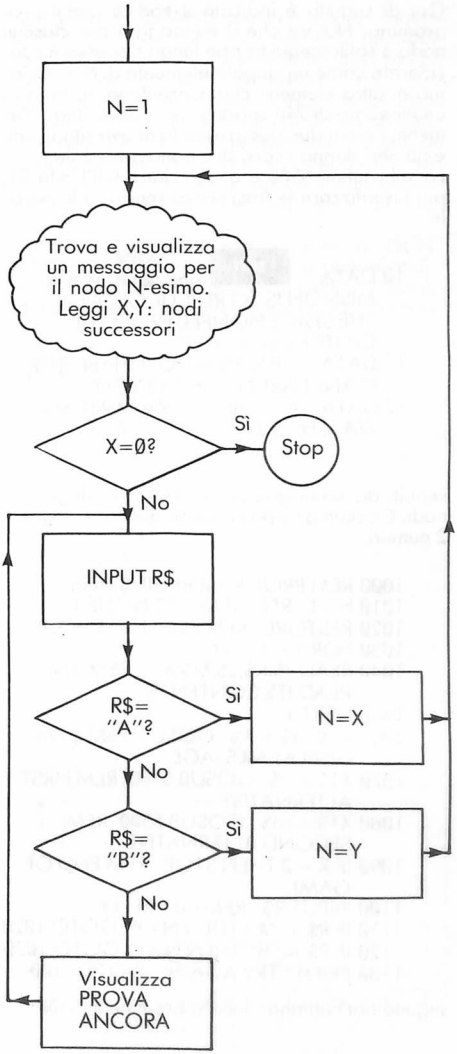
1) Il computer cancella lo schermo e visualizza un messaggio. Quindi interrompe il programma. Ciò succede quando l'eroe viene ucciso oppure quando riesce nella sua missione.

2) a) Il computer cancella lo schermo e visualizza un messaggio.

   b) Quindi invita l'utente a battere A o B e respinge tutti gli altri input.

   c) Usa la nuova risposta per scegliere un nuovo successore e ripete il processo da capo.

Ciò suggerisce che il gioco può essere gestito da un programma molto semplice, con tutta la complessità racchiusa nei dati.
Si supponga di numerare i nodi da 1 in su (come già è stato fatto nel diagramma). Quindi, per ciascun nodo, possiamo creare una "package" di dati composto dalla stringa da visualizzare e dai numeri dei due nodi successivi. Usiamo la convenzione secondo cui un numero successore di Ø significa che il gioco è terminato.
Lo schema base di flusso per il programma è ora abbastanza breve. Esso è:



Glossario:

N: Numero corrente di nodo
X,Y: Numeri dei nodi successori
R$: Risposta

Qui di seguito è indicato il codice per il programma. Notare che il messaggio per ciascun nodo è solitamente troppo lungo per essere considerato come un singolo elemento di dati. Usiamo quattro elementi che consentono un testo di un massimo di 240 caratteri per nodo. Degli elementi, i primi due descrivono la nuova situazione e gli altri danno i corsi di azione probabile.
La subroutine 5500 è quella data dall'Unità 21, per visualizzare le frasi senza separare le parole.

```
10 DATA " SHIFT and CLR HOME YOUR
   MISSION IS TO RESCUE GEORGE
   THE SPACE PIONEER FROM THE
   CLUTCHES OF THE "
11 DATA " GRAFFS, WHO INHABIT THE
   PLANET ARCNODE, DO YOU"
12 DATA " A) LAND","OR B) WAIT AND
   WATCH",3,2
...
```

seguiti da simili gruppi per ciascuno degli altri nodi. Ciascun gruppo contiene quattro stringhe e 2 numeri.

```
1000 REM PROGRAM BEGINS HERE
1010 N=1 : REM START AT NODE 1
1020 RESTORE : REM FIND N'TH NODE
1030 FOR J = 1 TO N
1040 READ J$,K$,L$,M$,X,Y : REM AND
     READ ITS CONTENTS
1050 NEXT J
1060 X1$=J$ + K$ : GOSUB 5500: REM
     DISPLAY MESSAGE
1070 X1$ = L$ : GOSUB 5500 :REM FIRST
     ALTERNATIVE
1080 X1$ = M$ : GOSUB 5500 :REM
     SECOND ALTERNATIVE
1090 IF X = 0 THEN STOP : REM END OF
     GAME
1100 INPUT R$ : REM GET REPLY
1110 IF R$ = "A" THEN N=X : GOTO 1020
1120 IF R$ = "B" THEN N=Y : GOTO 1020
1130 PRINT "TRY AGAIN": GOTO 1100
```

seguito dai comandi della subroutine in 5500.

ESPERIMENTO 25·2

Caricare l'intero programma di GRAFFS dalla cassetta di nastro e provarlo. Quindi modificarlo:

a) Per produrre interessanti effetti sonori

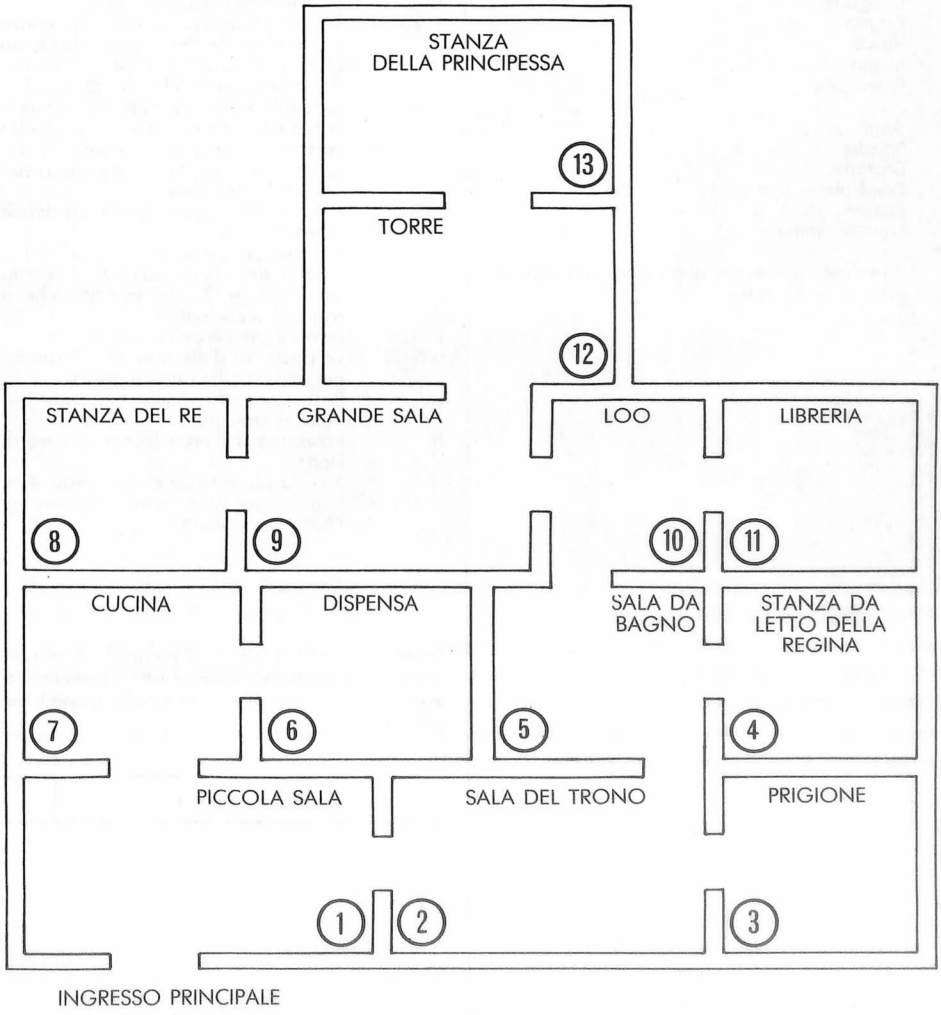
b) Per raccontare una storia diversa.

Esperimento 25.2 completato	

ESPERIMENTO 25·3

Il programma GRAFFS del precedente esperimento ha offerto un semplice esempio del modo in cui la struttura di un problema può essere rappresentata dai dati. Un esempio più complesso è dato dal programma denominato DUNGEON, anche esso caricabile da cassetta ed eseguibile. Questo programma segue un profilo popolare, ma non è altrettanto ambiguo come i suoi equivalenti commerciali in quanto deve essere "costretto" nei 3500 byte disponibili sul VIC standard.
Una volta giocata questa versione di DUNGEON alcune volte, esaminare il codice attentamente e costruire propri schemi di flusso. Potrebbero essere utili le seguenti informazioni:

1) Pianta del castello:



2) Codice interno:

Stanze

Esterno	0
Piccola sala	1
Sala del trono	2
Prigione	3
Stanza da letto della regina	4
Sala da bagno	5
Dispensa	6
Cucina	7
Stanza del re	8
Grande sala	9
Loo	10
Libreria	11
Torre	12
Stanza della principessa	13

Pericoli

Dragone	1
Ragno	2
Vespe	3
Maga	4
Pescecane	5

Armi

Spada	1
Bastone	2
Bomboletta spray	3
Pozione magica	4
Lanciafiamme	5

Il pericolo numero X può essere soltanto superato con l'arma X.

Glossario

n: Numero delle stanze

mm$(5,5): Testo per le battaglie con ciascuno dei cinque pericoli. Pertanto da mm$(1,1) fino a mm$(1,5) contengono i messaggi per i combattimenti con il dragone:
"a Dragon" (un dragone)
"You fight and" (combatti e)
"kill with your sword" (lo uccidi con la spada)
"it kills you" (lui t'uccide)
"You both run away!" (entrambi fuggite)
La subroutine della battaglia (dalla riga 3000 in su) usa questi messaggi per visualizzare i risultati delle battaglie.

w$(5): I nomi delle cinque armi.

r$(n): I nomi delle stanze nel castello.

c%(n,4): Le vie attraverso le quali le stanze sono collegate. Per esempio, la fila da c%(2,1) a c%(2,4) è 5,3,0,1. Ciò significa che la sala del trono (2) è collegata alla sala da bagno (5) andando verso nord, la prigione (3) andando verso ovest e la piccola sala (3) andando verso est. Non c'è porta a sud nella sala del trono.

di$(4): I nomi dei quattro punti cardinali: Nord, Est, Sud e Ovest.

d%(5): Le posizioni presenti (come numeri di stanze) dei cinque pericoli. Pertanto se d%(2) = 7, ciò significa che il ragno è in cucina!

I%(3): Le armi che l'eroe ha con se.

w%(5): Le posizioni delle armi che l'eroe ha perso (o non ha ancora preso).

p: Posizione della principessa (come numero di stanza).

h: Posizione dell'eroe (come numero di stanza).

q: 1 se la principessa è con l'eroe. 0 se non l'ha ancora trovata (oppure se l'ha abbandonata).

Infine, quando finalmente si è capito come funziona il programma DUNGEON, disegnare e scrivere un proprio programma sulle stesse linee generali.

Esperimento 25.3 completato	

CONCLUSIONE

CONCLUSIONE

Congratulazioni! Avete raggiunto la fine del corso e se lo avete seguito accuratamente svolgendone tutti gli esempi, avrete avuto un approccio alle problematiche generali del linguaggio BASIC. Avrete iniziato ad apprezzare l'immensa potenza del computer per fornire diletto, divertimento e utile servizio a tutti. Avrete l'esperienza e la conoscenza per applicare la vostra macchina ai giochi, agli affari, alle previsioni, per aiutare l'insegnante in classe e in molti altri campi. Con tutta probabilità preferireste interrompere lo studio e usare la vostra esperienza in vari affascinanti ed eccitanti progetti di cui avete sognato durante il corso.

Il BASIC è sotto molti aspetti un eccellente linguaggio con il quale imparare la programmazione, ma c'è un serio inconveniente che deve essere citato: non è standardizzato. Ciò significa che la versione del BASIC che troverete sulle diverse macchine, è generalmente leggermente diversa e solitamente inferiore al BASIC del VIC. Alcuni BASIC vi danno una scelta notevolmente ristretta di nomi variabili e molti non consentono addirittura l'uso delle stringhe. Le regole sull'inserimento di parecchi comandi su una riga non sono per nulla universali e gli arrangiamenti all'interno dei comandi PEEK, possono anche variare tra le varie macchine. Ci sono inoltre altre piccole differenze troppo numerose da citare: per esempio PEEK e POKE hanno risultati totalmente diversi. Se avete in progetto di trasferire i vostri programmi su qualsiasi altra marca di computer, dovete conoscerne le limitazioni prima di disegnare il programma, altrimenti vi troverete in serie difficoltà.

Potrebbe farvi piacere sentire che questa avvertenza non si applica al PET della Commodore, che ha una forma del BASIC pressochè identica a quella del VIC. Le principali differenze sono che la RAM dello schermo inizia in un posto diverso (32768) e che lo schermo stesso è di dimensioni diverse. Non ci sono funzioni per disegnare colori o per il suono; salvo questo, i programmi possono essere trasferiti tra le due macchine. Dovreste inoltre sapere che espandendo la memoria di più di 3K nel VIC, la RAM dello schermo inizia a 4096 e il colore a 37888.

Terminiamo con 10 "precetti" di buona programmazione. Oscar Wilde una volta ha detto: "Date sempre dei buoni consigli ad altri; è la sola cosa che potete fare".
È in questo spirito che viene offerto questo consiglio: prendetelo o lasciatelo, come preferite:

1) Puntate alla perfezione. Ogni parte del vostro programma e della sua documentazione devono essere il meglio che sapete fare.

2) Disegnate prima di costruire. Decidete cosa deve fare il programma prima di pensare a come farlo.

3) Siete preparati a buttare il tutto e ricominciare da capo. Non fate l'errore di innamorarvi di ciò che avete già fatto. Ricordate che ci sono molti modi per risolvere i problemi e il primo metodo che vi viene in mente, difficilmente è il migliore.

4) Pianificate e registrate il vostro schema per allocare le variabili. Il glossario è un documento di lavoro che deve essere usato costantemente quando scrivete i vostri programmi.

5) Dove occorre, scegliete buoni algoritmi. Per esempio, usate una ricerca dicotomica invece di una ricerca "diretta" o Quicksort invece di Bubble sort. Non ha importanza se la lista da esaminare o da riordinare è molto corta — usate sempre il metodo più semplice che potete trovare.

6) Fate attenzione all'interfaccia con l'utente. Dove è appropriato, assicuratevi che i vostri programmi possano essere usati da chiunque senza speciali istruzioni.

7) Usate il lavoro di altri. Non scrivete mai una subroutine se potete trovarne una valida in una libreria di programmi.

8) Non provate nulla di difficile o complicato ma suddividete il lavoro in fasi molto semplici. Usate le subroutine e osservate le convenzioni.

9) Evitate i cosiddetti "trucchi di programmazione intelligente" specialmente se non conoscete come funzionano.

10) Trovate sempre un'altra persona alla quale far compiere la prova finale del vostro programma.

Il vostro viaggio insieme attraverso il BASIC è terminato. Buona fortuna e buona programmazione!

Andrew Colin Glasgow, 1982

APPENDICI

APPENDICE A

Il computer VIC può essere usato come strumento musicale insolito e versatile. Per ottenere le migliori prestazioni della macchina, occorrono arte e scienza. Quando si esegue la prova finale di un programma musicale non si dice "è corretto" ma "suona bene?" e questo è in definitiva una questione di gusto personale.
Nell'appendice considereremo:

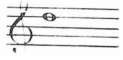
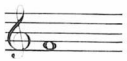
a) L'esecuzione di un motivo preprogrammato

b) La creazione di suoni con diverse qualità timbriche
c) La produzione di un'armonia a due voci

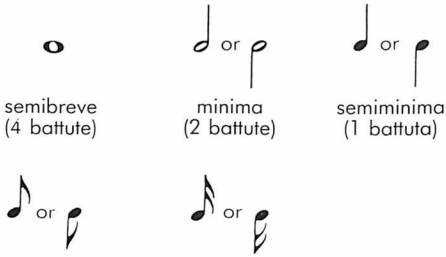
d) L'esecuzione diretta dalla tastiera

Per avere un'idea del VIC come strumento, caricare ed eseguire il programma intitolato "PRELUDE 1". Provare diverse combinazioni di velocità e di timbro e trovare quella che è più gradita. Per fare qualcosa di interessante con la musica, occorre conoscere un poco la notazione musicale standard. Molti la imparano da giovani ma se è necessario, è facile trovare qualcuno che la possa spiegare. Chiunque suona uno strumento musicale può probabilmente essere di aiuto.
Un motivo è costituito da note, generalmente suonate l'una dopo l'altra. Ogni nota ha quattro qualità:

a) *L'altezza:* che indica se la nota è alta o bassa. Nella musica scritta, l'altezza di una nota è indicata dalla sua altezza sul pentagramma cosicchè

 è più alto di 

b) *La durata:* che dice per quanto si suppone che la nota continui a suonare. La musica scritta usa diversi simboli per indicare note di varie lunghezze:



semibreve (4 battute) — minima (2 battute) — semiminima (1 battuta)

croma (1/2 battuta) — semicroma (1/4 di battuta)

c) *L'intensità* di una nota fa anche differenza nel modo in cui essa suona. Nella notazione musicale, l'intensità o volume è indicata da una serie di codici:

ppp pp p mp mf f ff fff

pianissimo ←——→ fortissimo

d) Il *timbro* di una nota dipende dallo strumento sul quale è suonata. Ogni diverso strumento ha una propria qualità che è immediatamente riconoscibile. Gli strumenti a fiato o a corda (ad esempio organi, clarini, corni o violoncelli), producono un suono *tenuto,* cosicchè la nota è ugualmente intensa per tutto il periodo in cui viene suonata. Per contro i suoni degli strumenti che vengono percossi o battuti (arpe, piano o tamburi), creano dei suoni *decadenti* che iniziano forte e si attenuano.

Sul VIC i suoni sono creati inserendo (POKE) numeri nei registri delle voci nelle locazioni da 36874 a 36878. La configurazione (come si ricorderà dal libro 1) è la seguente:

Registro	*Scopo*
36874	Voce di basso
36875	Voce di tenore
36876	Voce acuta
36877	Generatore di rumore
36878	Regolazione di volume

In questa sezione mostreremo come i numeri che s'inseriscono (POKE) e i suoni che essi producono, sono legati l'uno all'altro.
Ogni suono musicale è creato da un oggetto *vibrante* che potrebbe essere — ad esempio la corda di una chitarra o la membrana di un'altoparlante o le corde vocali della propria gola. Le vibrazioni si distribuiscono nell'area sotto forma di onde, più o meno come le onde sulla superficie di uno stagno (ma molto più velocemente). Al limite le onde raggiungono le orecchie, dove vengono percepite sotto forma di suono.
L'altezza del suono dipende soltanto dalla *frequenza* ossia del numero delle vibrazioni al

secondo. Maggiore è il numero delle vibrazioni e più alta è la nota. L'altezza è determinata con una regola molto semplice: ad ogni raddoppio della frequenza si sale di un'ottava e ad ogni dimezzamento, si scende di un'ottava. La frequenza del Do centrale sul pianoforte è di circa 264 vibrazioni al secondo. Il Do sopra quello centrale ha una frequenza esattamente doppia di 264 ossia 528 vibrazioni al secondo. Analogamente il Do due ottave al disopra del Do centrale, ha una frequenza di quattro volte 264 ossia 1056.

La regola funziona anche alla rovescia; per esempio, la frequenza del Do al disotto del Do centrale, è di un mezzo di 264 ossia 132. L'ottava è divisa in intervalli più piccoli di altezza. La maggior parte della musica occidentale usa una scala di 12 intervalli uguali detti *semitoni*. Le frequenze delle note non procedono a intervalli uguali ma secondo un rapporto costante. Il valore esatto di questo rapporto è la radice 12esima di 2 che è circa 1,059. Il significato di questo valore è che quando lo si moltiplica per se stesso 12 volte, si ottiene di nuovo 2.

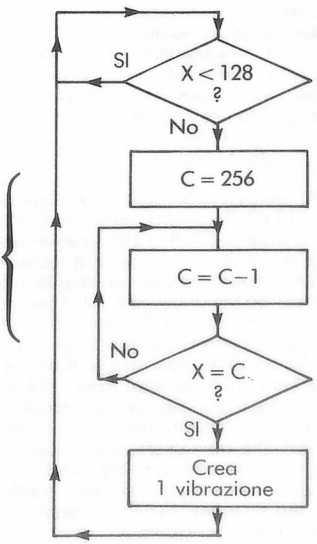
È possibile usare il valore per calcolare le frequenze delle note della scala. Per salire di un semitono, basta moltiplicare la precedente frequenza per il rapporto. Se il Do centrale è 264, la nota un semitono più alto (Do diesis o Re bemolle) avrà una frequenza di 1,059★264 ossia (arrotondato al valore intero più vicino) 280.

La successiva nota, Re ha una frequenza di 280★1.059, ossia 297 (sempre arrotondato al numero intero più vicino). Se si applica questa regola a tutte le note in una scala, si ottiene la seguente tabella:

DO	DO	RE	RE	MI	FA	FA	SOL	SOL	LA	LA	SI	DO
C	C♯ D♭	D	D♯ E♭	E	F	F♯ G♭	G	G♯ A♭	A	A♯ B♭	B	C
264	280	297	315	334	354	375	397	420	445	471	499	528

La frequenza del Do alto arriva a 528, esattamente come ci si sarebbe aspettato. Altre note, al disopra del Do alto o al disotto del Do centrale, possono essere ricavate esattamente allo stesso modo.

Sfortunatamente non possiamo dire alle voci del VIC direttamente quale frequenza suonare. Ogni voce funziona in un modo che è meglio descritto da uno schema di flusso:



Questa iterazione viene eseguita 32768 volte al secondo sulla voce acuta, 16384 volte al secondo sulla voce di tenore e 8192 volte sulla voce di basso.

Glossario

X: Numero inserito (POKE) nel registro della voce

C: Variabile interna

Lo schema di flusso mostra che:

a) Se X, il numero inserito (POKE) nella voce, è minore di 128, la voce è una pausa di silenzio.

b) Altrimenti, produce vibrazioni ad un ritmo che dipende dal valore del numero inserito (POKE) e alla velocità alla quale la iterazione viene eseguita. Per esempio, si supponga di aver inserito (POKE) 193 nella voce superiore. L'iterazione verrà eseguita (255-193) ossia 62 volte prima che venga prodotta una singola vibrazione, quindi altre 62 volte prima della successiva e così via. Dato che la "frequenza dell'iterazione" è di 32768, la frequenza delle vibrazioni deve essere 32768/62 ossia 528 vibrazioni al secondo.

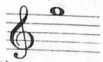
È importante notare che questo schema di flusso non costituisce un programma BASIC che si deve inserire nella macchina; esso rappresenta per contro un *meccanismo*, che fa parte di ciascuna voce. Tutti e tre i meccanismi delle voci funzionano contemporaneamente, ciascuno al limite producendo una frequenza propria.
Un poco di algebra darà una formula per trovare quale numero inserire (POKE) per qualsiasi frequenza necessaria:

Acuta: $F = \frac{32768}{255-X}$, per cui $X = 255 - \frac{32768}{F}$

Tenore: $F = \frac{16384}{255-X}$, per cui $X = 255 - \frac{16384}{F}$

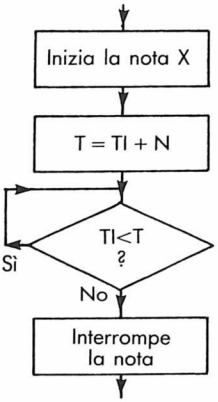
Basso: $F = \frac{8192}{255-X}$, per cui $X = 255 - \frac{8192}{F}$

In ciascun caso F è la frequenza e X è il numero che si deve inserire (POKE).
Con queste formule sembra abbastanza facile calcolare il valore esatto di X per ciascuna nota nella scala. Sfortunatamente troviamo subito un ostacolo in quanto la maggior parte dei valori risultano non essere interi. Per esempio, la frequenza giusta per la nota



è di 792 vibrazioni al secondo, e il valore corretto di X è 213.62. Il numero più vicino che possiamo avere in pratica è 214, e ciò dà un errore di circa 1/6 di un semitono — una discrepanza che un musicista addestrato può facilmente individuare. La Fig. A1 dà i migliori valori di X per le note che è possibile suonare sulla voce di acuto, unitamente con gli errori espressi in parti di un semitono. All'estremità inferiore le note sono abbastanza bene intonate man mano che si procede verso le note più elevate, gli errori diventano sempre più avvertibili. Non c'è però alcun rimedio salvo che mantenersi sulle note basse se si trova che quelle alte sono discutibili. Le note prodotte dalle altre due voci sono esattamente le stesse ma spostate rispettivamente di una e di due ottave verso il basso.

Una volta che una voce è stata istruita a suonare una nota, continua a suonarla fino a che non riceve un diverso comando. È possibile controllare la durata di una nota mediante una semplice iterazione che usa il temporizzatore interno TI. Per mantenere una nota per N jiffies (ricordarsi, 60 jiffies costituiscono un secondo), è possibile usare un'iterazione come questa:



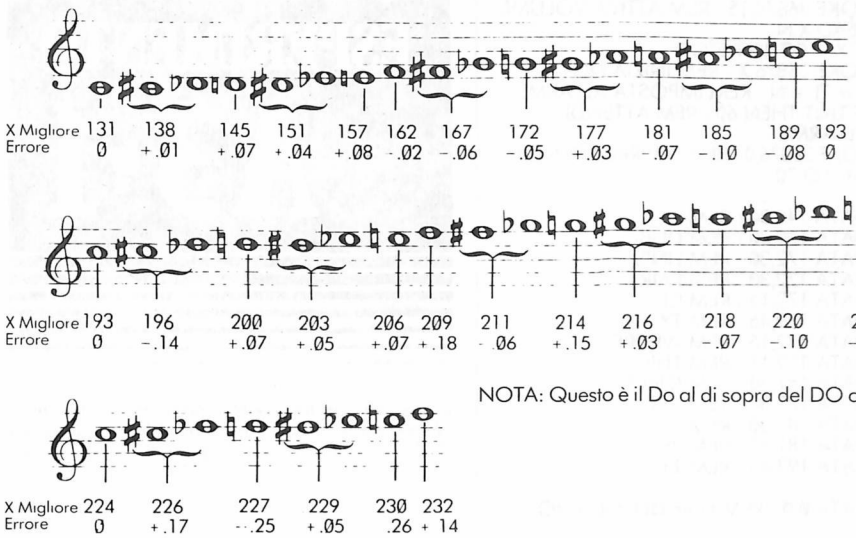
Glossario

X: Numero inserito (POKE) nella voce
N: Durata della nota
T: Allarme

oppure in codice:

```
100 POKE 36876,X
110 T=TI+N:REM PREDISPONI ALLARME
120 IF TI<T THEN 120:REM ASPETTA
130 POKE 36876,0:REM INTERROMPI NOTA
```

Se si trascurano per il momento l'intensità , il timbro, è possibile fare in modo che la macchina suoni un semplice motivo controllando l'altezza e la durata di ciascuna nota. Ecco un programma per suonare la prima frase del motivo nella Figura A2. La "battuta" è di 30 jiffies:



NOTA: Questo è il Do al di sopra del DO centrale.

Figura A1

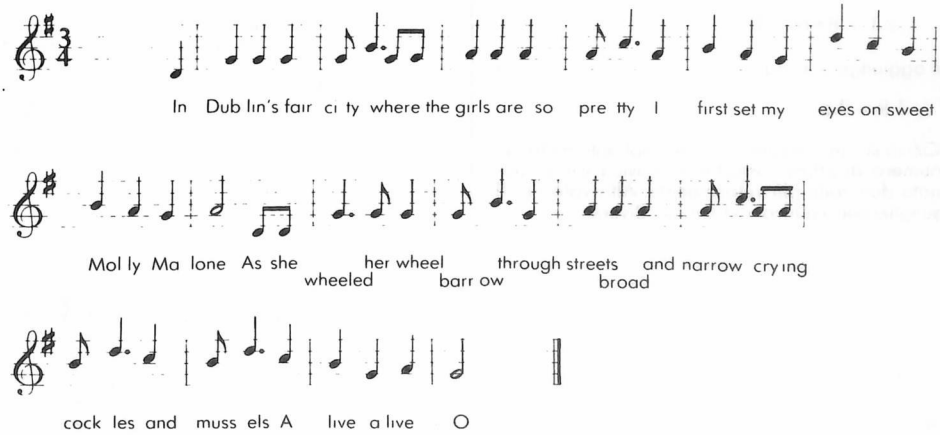


Figura A2

```
10 POKE 36878,15 : REM ATTIVA VOLUME
20 READ X,N
30 IF X = 0 THEN STOP
40 POKE 36876,X : REM INIZIA NOTA
50 T = TI + N : REM IMPOSTA ALLARME
60 IF TI<T THEN 60 : REM ATTENDI
   ALLARME
70 POKE 36876,0 : REM INTERROMPI NOTA
80 GOTO 20

100 DATA 145,30 : REM IN
110 DATA 172,30 : REM DUB-
120 DATA 172,30 : REM LIN'S
130 DATA 172,30 : REM FAIR
140 DATA 172,15 : REM CI-
150 DATA 189,45 : REM TY
160 DATA 172,15 : REM WHERE
170 DATA 172,15 : REM THE
180 DATA 181,30 : REM GIRLS
190 DATA 181,30 : REM ARE
200 DATA 181,30 : REM SO
210 DATA 181,15 : REM PRET-
220 DATA 193,45 : REM TY

1000 DATA 0,0 : REM FINE DEL MOTIVO
```


La prima istruzione DATA alla riga 100 dice al programma d'inserire 145 nel registro della voce acuta per 30 jiffies. Ciò fa suonare la prima del motivo. Le seguenti note vengono suonate allo stesso modo.
Per variare la velocità della musica è possibile cambiare il numero di jiffies usati per ciascuna nota. Per evitare di cambiare tutte le istruzioni DATA, è possibile usare un semplice "fattore di velocità". Modificare la riga 50 come segue

```
50 T = B*N + TI
```

e aggiungere la riga

```
1 B = 0.5
```

Ciò fa sì che ciascuna nota duri soltanto metà del numero di jiffies cosicchè la musica viene suonata due volte più velocemente. Altri valori di B sceglieranno a loro volta valori diversi.

ESPERIMENTO A3·1

Estendere questo programma in modo che suoni l'intero motivo. Occorreranno altre 36 istruzioni DATA (o meno se ci sono più di due numeri per riga).

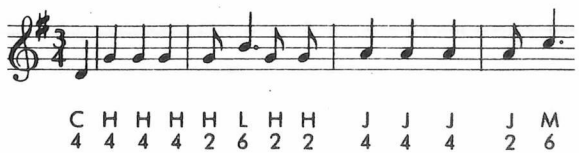
Esperimento A3.1 completato	

L'esperimento A3.1 avrà convinto che immettere musica nel VIC è un lavoro molto difficile e predisposto agli errori. Un altro inconveniente è che ciascuna istruzione DATA usa 16 byte e ciò non lascia molto spazio per motivi molto lunghi.
Per rendere le cose un poco più facili, useremo una notazione speciale che rappresenta ciascuna nota come due caratteri — uno per la sua altezza e uno per la sua durata. Il codice è indicato in figura A.3. È stato inventato specialmente per questo manuale ed è diverso da qualsiasi altro codice, in particolare non è lo stesso codice usato nel VIC SUPER-EXPANDER. Ciononostante ci si abituerà presto a codificare i motivi.
Come si vedrà, le note intervallate da un semitono sono rappresentate da carattere con codici ASCII adiacenti. Per esempio i codici dei simboli ? @ A B C D sono i numeri 63,64,65,66 e 67. Questa configurazione rende facile cercare i valori esatti da inserire (POKE) da una tabella. Una pausa (l'assenza di alcun suono) è indicata dal carattere "(" (parentesi sinistra).
Le lunghezze delle note sono misurate in quarti di nota o semicrome. Dove una nota dura più di 9 semicrome, la seconda semplicemente lungo il codice ASCII cosicchè (ad esempio) una "minima puntata" 𝅗𝅥. che è lunga 12 semicrome, è indicata come "<". Analogamente 𝅝 (16 semicrome) è scritta come "@".
Per scrivere un motivo occorre inserire due stringhe. Una stringa contiene tutti i caratteri di altezza, e la seconda tutti i caratteri di durata. Si usano tante coppie di stringhe quante sono necessarie e si fa seguire l'ultima con una "Z".
La versione codificata di "Dublin's Fair City" è la seguente:

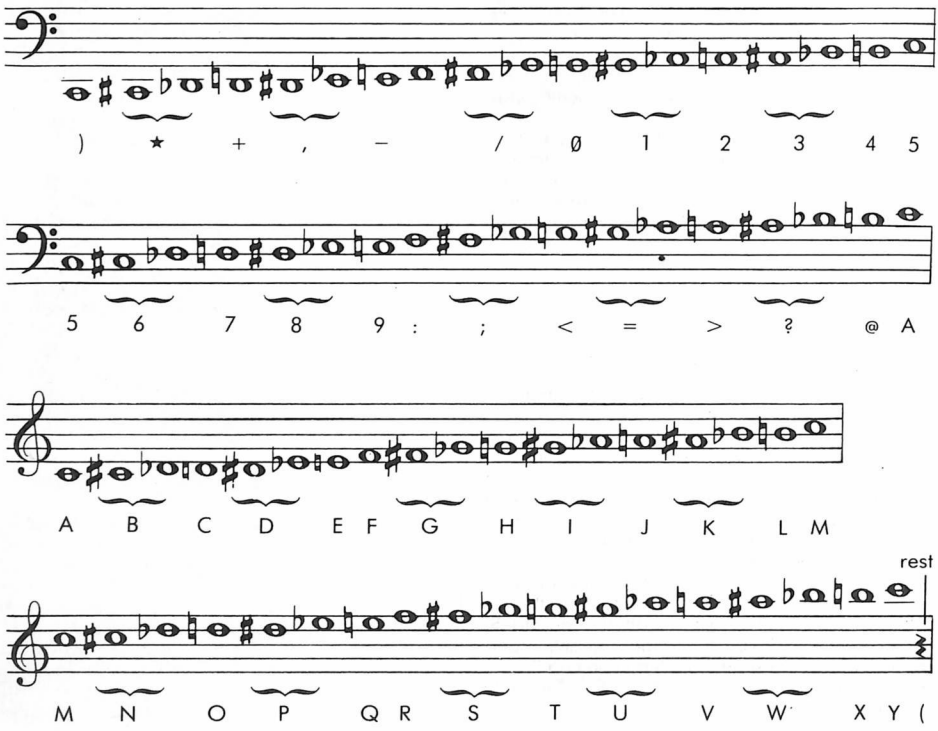
"CHHHHLHHJJJJMJLJHOMLLJHJ"
"4444262244426444444444448"

"CCHHHHHLHJJJJMJJLOMLOMLHJH"
"224442644442622642644448"

"Z"



Altezza:



Durata:

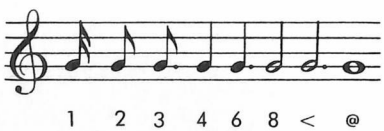
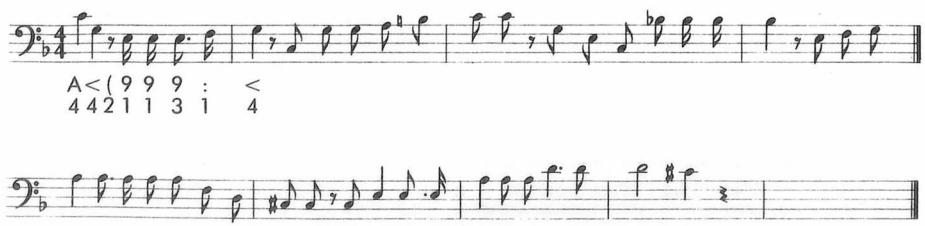


Figura A3

Un altro esempio, che è scritto in chiave di basso e usa qualche pausa è indicato qui di seguito.

(da "Elijah" Mendelssohn)



```
DATA"A<(999:<(5<<> @ AA(<95????(9:<"
DATA"44211314222222222222221146222"

DATA ">>>>>:766(6999>>>CCCB"
DATA "43122222224314226284"

DATA "Z"
```

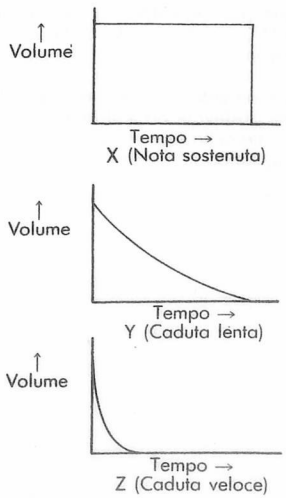
Qui di seguito è riportato un programma che suona motivi scritti in questo codice. Il primo comando della riga 1 definisce la velocità. Le righe DATA che seguono danno i valori POKE per tutte le note nella scala e lo zero iniziale è usato per produrre una pausa di silenzio per le pause della musica.
La notazione consente una gamma di quattro ottave. Il programma non si limita ad una voce, ma sceglie quella più appropriata per l'altezza della nota che viene eseguita. Ciò avviene nelle righe 150-180. I numeri 64,52 e 40 sono relativi ai codici ASCII dei caratteri A, 5 e). Quando la musica viene eseguita esattamente con gli stessi valori di tempo, spesso risulta monotona e noiosa. È possibile talvolta migliorare l'esecuzione sottolineando la nota all'inizio di ciascuna barra, facendola suonare leggermente più a lungo del suo valore "vero". Per esempio, si potrebbe allungare un semiminima da 4 semicrome a 5. Senza esagerare però, altrimenti rende la musica troppo pesante!
Un'altra variazione consiste nel rallentare gradualmente la musica man mano che si raggiungono le barre finali.

```
  1 R=3
 10 DATA 0,131,138,145,151,157,162,167,
    172,177,181,185,189
 20 DATA 193,196,200,203,206,209,211,
    214,216,218,220,222,224
 30 DIM N(25)
 40 FOR J = 0 TO 25
 50 READ N(J)
 60 NEXT J
 70 FOR J = 36874 TO 36877 : POKE J,0
    NEXT J :REM CESSA TUTTE LE VOCI
 80 POKE 36878,15: REM ATTIVA
    REGOLAZIONE VOLUME
 90 READ X$
100 IF X$ = "Z" THEN GOTO 205
110 READ Y$
120 FOR J = 1 TO LEN(X$) : REM SUONA
    OGNI NOTA NELLA COPPIA STRINGHE
130 A=ASC(MID$(X$,J,1)) : B = ASC(MID$
    (Y$,J,1))
140 T=TI+R*(B-ASC("0")) : REM DISPONI
    TEMPORIZZATORE
150 IF A > = ASC("A") THEN POKE
    36876,N(A-54) : GOTO 180
160 IF A > = ASC("5") THEN POKE
    36875,N(A-52) : GOTO 180
170 POKE 36874,N(A-40)
180 IF TI < T THEN 180 : REM ATTENDI
190 NEXT J
200 GOTO 90
205 POKE 36878,0 : STOP : REM
    INTERROMPI NOTA
210 REM IN DUBLIN'S FAIR CITY
220 DATA "CHHHHLHHJJJJMJLJHOMLL
    JHJ","444426224442644444444448"
230 DATA "CCHHHHLHJJJJMJJLOMLO
    MLHJH","22444264444262226426444
    448"
240 DATA "Z"
```

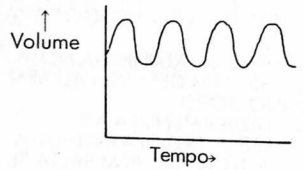
È possibile ottenere un interessante gruppo di effetti se si altera il *timbro* o la qualità sonora della musica. Finora abbiamo semplicemente attivato e disattivato i suoni, creando note sostenute. Per dare l'effetto di uno strumento a corda, occorre iniziare la nota molto intensamente e attenuarla gradualmente. Impostare il seguente programma e provare:

```
 10 GET A$
 20 IF A$ = "X" THEN P = 1 : GOTO 60
 30 IF A$ = "Y" THEN P = 1.1 : GOTO 60
 40 IF A$ = "Z" THEN P = 1.5 : GOTO 60
 50 GOTO 10
 60 POKE 36876,220 : REM INIZIA NOTA
 70 T = TI + 60 : REM DEFINISCI ALLARME UN
    SECONDO DOPO
 80 J = 15 : INIZIA INTENSITÀ A 15
 90 POKE 36878, J : REGOLA INTENSITÀ
100 IF TI > = T THEN 130 : REM SALTA SE
    TEMPO SCADUTO
110 J = J/P
120 GOTO 90
130 POKE 36878,0 : REM INTERROMPI NOTA
140 GOTO 10
```

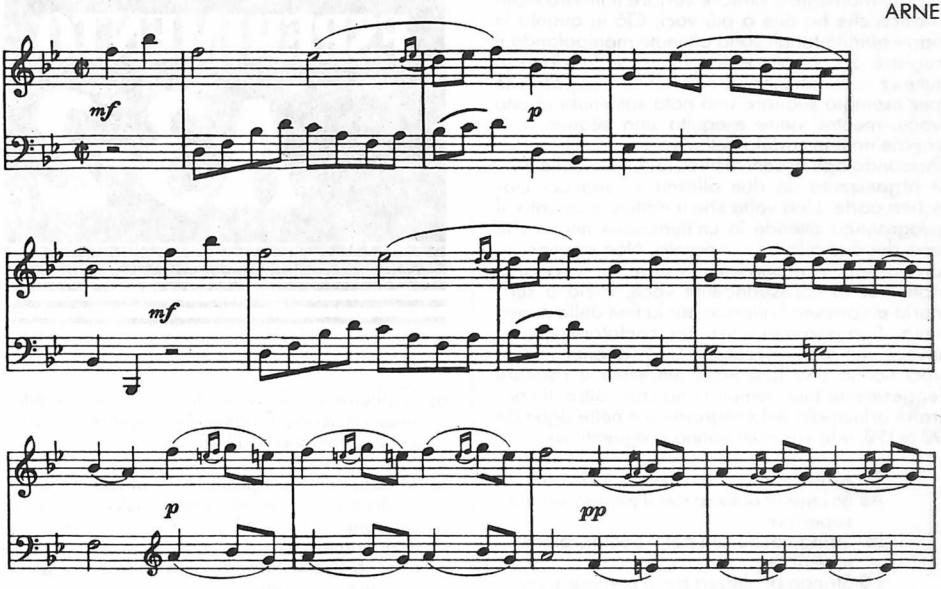
Quando si inizia questo programma, non succede nulla fino a che non si premono i tasti X, Y o Z. X dà una nota sostenuta che dura 1 secondo. Y suona come un'arpa, e Z una nota smorzata come un mandolino o un violino pizzicati. In tutti i casi, il modo in cui l'intensità della nota varia nel tempo è detto *inviluppo*. I grafici degli inviluppi usati in questo programma sono indicati qui di seguito:



Nel programma, la variazione di volume è organizzata impostando un "fattore di caduta" nella variabile P. La variabile J, che controlla l'intensità effettiva è regolata ogni volta durante l'iterazione di temporizzazione.
Quando P=1, il valore di J non varia mai producendo così una nota sostenuta. Quando P=.1, il valore inserito (POKE) nel registro di volume, declina lentamente secondo la sequenza di numeri 15,13,12,11,10,9,8,7,6,6,5,5,4,4,3,3,2,2...
Quando P=1.5, la caduta è più veloce, essendo i numeri 15,10,6,4,2,1,1,0. È possibile sperimentare facilmente con altri inviluppi sia calcolando i valori d'intensità quando la nota viene suonata oppure ricavandoli da una tabella memorizzata in anticipo. Se si aumenta il volume di ciascuna nota mentre viene suonata, si ottiene un effetto che è impossibile produrre su qualsiasi strumento convenzionale. Un altro tipo di variazione tonale, è detto "vibrato". Il suo inviluppo è come questo:



Per vedere come funziona, caricare da cassetta ed eseguire il programma VIBRATO. Si può decidere che l'uso di questo timbro non va preso in considerazione.
Successivamente considereremo la possibilità di suonare musica in un'armonica a due voci. Caricare da cassetta ed eseguire il programma denominato "GAVOTTE". Il pezzo è di Thomas Arne (1710-1778) ed è leggermente adattato dalla versione originale per tastiera.
Le armonie sono eseguite sul VIC usando contemporaneamente due o più voci. La parte superiore usa la voce acuta e la parte inferiore la voce di tenore o di basso a seconda dei casi.
La notazione per immettere la musica nel programma GAVOTTE è la stessa dell'esempio precedente. In ogni caso ciascuna delle due voci usa proprie stringhe di altezza e di durata, cosicchè in qualsiasi momento la musica è il risultato delle informazioni prelevate da 4 stringhe di caratteri. le prime battute nella Gavotte di Arne e la rispettiva codifica, sono:




```
1000 DATA "RWRPOPRKOHPMOKMJKRWRPOPRKOHPOOMMKKJ"
1010 DATA "22441222211111114224411222211111122"
```
} Pentagramma superiore

```
1020 DATA "(7:?CA>:>?AC7385: .33(7:?CA>:>?AC7389:"
1030 DATA "41111111111222222222411111111111222444"
```
} Pentagramma inferiore

Il gruppo finale di stringhe è seguito da quattro Z:

```
2000 DATA Z,Z,Z,Z
```

È possibile cancellare le istruzioni DATA nel programma (da 1000 in avanti) e sostituirle con altre proprie ma ci sono tre speciali limitazione da osservare:

1) La voce superiore è limitata al campo

da Do a Do

2) La voce inferiore è limitata al campo

da Do a Mi

3) Le divisioni tra le stringhe devono cadere nello stesso posto per le voci superiore e inferiore.

Sfortunatamente è difficile variare il timbro della musica che ha due o più voci. Ciò in quanto le varie qualità tonali sono ottenute manipolando il registro di controllo e di volume che funziona su tutte le voci contemporaneamente. È impossibile per esempio suonare una nota sostenuta in una voce, mentre viene eseguita una sequenza di rapide note in caduta nell'altra.
Passando agli aspetti tecnici, la durata delle note è organizzata da due allarmi — uno per ciascuna parte. Una volta che il motivo è avviato, il programma attende in un'iterazione fino a che uno dei due allarmi sia pronto. Non appena ne arriva uno, il programma estrae la successiva nota per la corrispondente voce, inizia a suonarla e ripristina l'allarme per la fine della nuova nota. Il programma usa un contatore interno invece del temporizzatore TI, in quanto le due voci hanno una fastidiosa tendenza a risultare leggermente fuori tempo l'una con l'altra. Il controllo principale del programma è nelle righe da 90 a 190, e le variabili hanno i seguenti usi:

- Z Matrice di valori X per le varie note
- A$ Stringa di altezza per il pentagramma superiore
- B$ Stringa di durata per il pentagramma superiore
- C$ Stringa di altezza per il pentagramma inferiore
- D$ Stringa di durata per il pentagramma inferiore
- P Puntatore a A$ e B$: mostra quale carattere viene successivamente
- Q Puntatore a C$ e D$: mostra quale carattere viene successivamente
- R Allarme per il pentagramma superiore
- S Allarme per il pentagramma inferiore
- L Temporizzatore interno
- V Costante per controllare la velocità di esecuzione

ESPERIMENTO A3·2

a) Sostituire la Gavotte di Arne con un altro pezzo a scelta. È possibile scegliere le invenzioni a due voci di J.S. Bach.

b) Modificare il programma in modo da poter controllare l'intensità della musica che viene suonata.
Suggerimento:
Probabilmente occorre una quinta stringa di "controllo di volume".

Esperimento A3.2 completato	

Infine, osserveremo un programma che consente di usare il VIC come strumento a tastiera. Caricare e provare il programma "KEYBOARD". Si scoprirà presto che la tastiera del VIC è stata riprodotta il più possibile per assomigliare ad un pianoforte. L'intera seconda fila di tasti è usata per le note "bianche" e alcuni dei tasti nella fila superiore, rappresentano le note "nere".
Lo strumento VIC è munito di due altri comandi:

a) Il volume che può essere aumentato premendo ripetutamente A e abbassato premendo ripetutamente S. Il volume massimo è 15 e il minimo è zero.

b) Il fattore di caduta per le note che può essere diminuito battendo K, è aumentato battendo L. Ciò consente di variare la qualità tonale da un tono ampio sostenuto ad un "plink smorzato".

Se si lista e si esamina il programma, si vedrà che le tabelle sono alquanto irregolari in quanto le note sono assegnate ai caratteri pressochè a caso dal punto di vista del computer. Altrimenti il programma è lineare e facilmente modificabile per dare, ad esempio — un vibrato opzionale. Un'altra modifica consiste nel ricordare le note del motivo che si sta suonando memorizzando i valori X e i tempi in una matrice. La macchina può quindi ripetere successivamente l'esecuzione.

Questa appendice ha fornito soltanto la più breve delle introduzioni al potenziale musicale del VIC. La macchina può essere estremamente gratificante nella composizione nonchè nelle prestazioni e i soli limiti importanti sono dati dalla rispettiva esperienza e senso artistico congiuntamente alla qualità di programmatori e musicisti.

APPENDICE B

LIBRERIA DI PROGRAMMI

Questa sezione contiene una piccola libreria di utili subroutine. Tutte le subroutine sono state accuratamente controllate. Di regola, esse non variano i valori dei rispettivi parametri di input a meno che le stesse variabili siano anche specificate come parametri di output. (Un'eccezione è la subroutine per risolvere le equazioni simultanee).
Quando si disegna un nuovo programma, occorre scegliere e incorporare una qualsiasi di queste subroutine necessarie; come risultato, il programma sarà più affidabile e funzionerà più rapidamente.
Ci sono due modi per usare le subroutine:

a) Se si dispone di una configurazione VIC minima (con 3,5K di memoria) copiare le subroutine dal testo che segue. Si consiglia vivamente di controllare l'accuratezza del lavoro impostando il programma di gestione indicato con ciascuna subroutine e controllando che si ottengano gli stessi risultati o risultati analoghi. Il programma di gestione può essere cancellato una volta soddisfatti.

b) Se si dispone di un modulo di memoria extra da 3K (o più), è possibile usare il programma LIBRARY. Caricarlo dalla cassetta di nastro ed eseguirlo attraverso il menù rispondendo "YES" o "NO" a ciascuna subroutine. Dopo l'ultima voce, il programma LIBRARY si cancella da solo e lascia soltanto le subroutine effettivamente occorrenti. A questo punto occorre salvarle (SAVE) su un nastro separato.

Notare che il programma LIBRARY non può essere fatto ripartire salvo che ricaricandolo dal nastro. Se viene introdotto a metà, può lasciare il VIC in una condizione non standard nella quale è impossibile caricare i programmi correttamente. Ciò può sempre essere rimediato spegnendo il computer e riaccendendolo.

Le subroutine sono state disposte in quattro sezioni:

Sezione A: Input da tastiera.

1. Input tollerante: accetta 1 e Ø e fornisce chiari messaggi di errore.
2. Input robusto: non risponde a qualsiasi carattere salvo quelli legittimi.

Sezione B: Output su schermo.

3. Grandi lettere: visualizza il testo in formato quattro volte il normale.
4. Output formattato: visualizza numeri con numero di cifre decimali specificato dall'utente.
5. Visualizzazione di stringhe: visualizza una lunga stringa senza interrompere le parole sulle righe.
6. Convertitore binario: visualizza un numero in binario.

Sezione C: Manipolazione interna

7. Estrazione del nome: estrae il nome da una stringa che contiene il nome completo di una persona.
8. Ricerca di liste: analizza una lista alla ricerca di un'entrata specifica.
9. Bubble sort: riordina un piccolo numero di stringhe in ordine alfabetico.
10. Quick sort: riordina una lista di numeri (o stringhe).

Sezione D: Matematica

11. Semplificatore di frazioni: riduce la frazione ai suoi minimi termini.
12. Equazioni simultanee: risolve le equazioni simultanee.

1. INPUT TOLLERANTE

Scopo: Consentire ad un utente inesperto di immettere numeri. Tutti gli spazi sono ignorati e le lettere l e O sono intese rispettivamente come cifre 1 e Ø. Tutti gli altri errori sono chiaramente spiegati.

righe: 4500-4660

Parametri: Output: il risultato è fornito in X1.

Variabili locali: XX$, YY$, JJ, CC$

```
4500 REM INPUT TOLLERANTE DI NUMERI
4510 INPUT XX$
4520 YY$=" "
4530 FOR JJ = 1 TO LEN (XX$)
4540 CC$ = MID$ (XX$,JJ,1)
4550 IF CC$ = "O" THEN YY$ = YY$ + "Ø":
     GOTO 4600 : REM SOSTITUISCI
     LETTERA O CON CIFRA Ø
4560 IF CC$ = "l" THEN YY$ = YY$ + "1"
     GOTO 4600
4570 IF CC$ = " " THEN 4600
4580 IF NOT (CC$<= "9" AND CC$ > =
     "Ø" OR CC$ = "+" OR CC$ = "–"
     OR CC$= ".") THEN 4620
4590 YY$ = YY$ + CC$
4600 NEXT JJ
4610 X1 = VAL(YY$) : RETURN
4620 PRINT "NUMERI CONSISTONO IN"
```

```
4630 PRINT "CIFRE DECIMALI 0-9,"
4640 PRINT "+, – E . SOLTANTO"
4650 PRINT "PROVA ANCORA"
4660 GOTO 4510
```

Programma di gestione:

```
10 GOSUB 4500
20 PRINT "VALORE ="; X1
30 GOTO 10
```

Risultati della prova:

```
RUN
? 778
VALORE = 778
? 101
VALORE = 101
? –34.56
VALORE = –34.56
? 45.K
NUMERI CONSISTONO IN
CIFRE DECIMALI 0-9,
+, – E . SOLTANTO
PROVA ANCORA
? 7.7
VALORE = 7.7
```

2. INPUT ROBUSTO

Scopo: Leggere un numero dalla tastiera, ignorando tutti i caratteri privi di significato. Può essere usato DEL e un numero è sempre terminato da RETURN;

Righe: 7000-7090

Parametri: Output: Risultato fornito in X1.

Variabili locali: PP, AA$, XX$

```
7000 REM INPUT NUMERI ROBUSTO
7010 XX$ = " ":P = 0
7020 GEST AA$ : IF AA$ = " " THEN 7020
7030 IF AA$ > = "0" AND AA$ < = "9"
     THEN PRINT AA$;: XX$ = XX$ + AA$:
     PP = PP + 1 : GOTO 7020
7040 IF ASC(AA$) <> 20 THEN 7070:
     REM CERCA DEL
7050 IF PP=0 THEN 7020: REM NON
     POSSO CANCELLARE NULLA
7060 PRINT "[SHIFT] e [←→] spazio
     [SHIFT] e [←→]";: PP=PP–1:
     XX$=LEFT$(XX$,PP) : GOTO 7020
7070 IF ASC (AA$)<> 13 THEN 7020 : REM
     CERCA RETURN
7080 IF PP=0 THEN 7020 : REM DEVE ESSERE
     UNA QUALCHE CIFRA
7090 X1=VAL (XX$) : RETURN
```

Programma di gestione:

```
10 GOSUB 7000
20 PRINT X1
30 GOTO 10
```

3. BIG LETTERS (GRANDI LETTERE)

Scopo: Visualizzare i caratteri VIC quattro volte la loro dimensione abituale.

Righe: da 8000 a 8200

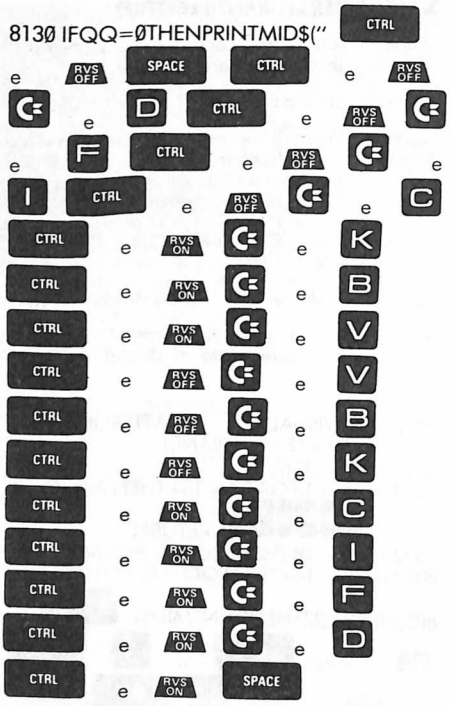
Parametri Input: Il successivo carattere da visualizzare è fornito in A1$. Può essere qualsiasi carattere stampabile o spazio, RETURN, CLR/HOME, un codice di colore, CTRL e RVS ON o CTRL e RVS OFF.

Variabili locali: AA,BB,JJ,KK,LL,MM,NN,QQ.

Nota: QQ tiene nota dello stato RVS corrente e non deve essere usato al difuori della subroutine.

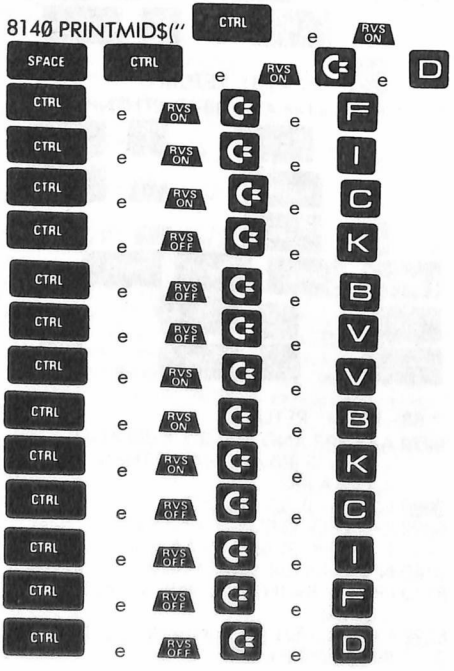

```
8000 REM VISUALIZZA I CARATTERI IN A1$
     4 VOLTE PIÙ GRANDI
8010 BB = ASC (A1$)
8020 IF BB = 13 OR BB = 141 THEN PRINT
     " [CRSR] [CRSR] [CRSR] ": RETURN
8030 IF BB = 18 THEN QQ = 1: RETURN
8040 IF BB = 146 THEN QQ = 0: RETURN

8050 IFBB<32THENPRINTMID$(" [CTRL] e
[RVS OFF] 5 volte [CTRL] e [CTRL 2] [CTRL]
e [RVS OFF] 13 volte [CLR HOME] [CTRL] e
[RVS OFF] 8 volte [CTRL] e [CTRL 2] [CTRL]
e [RVS OFF] [CTRL] e [CTRL 6] [CTRL]
e [7] ",BB+1,1);:RETURN
8060 IFBB>=144 AND BB<160THENPRINT
     MID$(" [CTRL] e [CTRL 1] [CTRL]
e [RVS OFF] [CTRL] e [RVS OFF] [SHIFT]
e [CLR HOME] [CTRL] e [RVS OFF] 3 volte
[SPACE] [SPACE] [SPACE] [SPACE]
[CTRL] e [CTRL 5] [CTRL] e [RVS OFF]
[CTRL] e [CTRL 8] [CTRL] e [CTRL 4]

",BB–143,1);: RETURN
8070 AA= (BB AND 31)+0·5 ★ (BB AND
     128): IF (BB AND 64)=0 THEN
     AA=AA+32
8080 FOR JJ=0 TO 6 STEP 2
8090 KK=PEEK (32768+8★AA+JJ):
     LL=PEEK (32769+8★AA+JJ)
8100 NN=64: FOR MM=0 TO 3
8110 PP = 1+8★ INT (KK/NN)+2★INT
     (LL/NN)
8120 KK=KK–INT (KK/NN)★NN:LL=LL–
     INT (LL/NN)★NN
```

8130 IFQQ=ØTHENPRINTMID$("



",PP,2);:GOTO8150

8140 PRINTMID$("



CTRL e RVS OFF SPACE ",PP,2);

8150 NN=INT(NN/4):NEXT MM

8160 PRINT" CRSR ↓ SHIFT e CRSR → 4 volte";
8170 NEXTJJ

8180 PRINT" SHIFT e CRSR ↓ 4 volte CRSR ←
4 volte";

8190 IFPEEK(211)>18THENPRINT" CRSR ↑
3 volte"
8200 RETURN

Programma di gestione

```
10 GET A1$ : IF A1$= " " THEN 10
20 GOSUB 8000
30 GOTO 10
```

Passata campione: Non riproducibile. (Provare per credere).

4. NUMERO FORMATTATO

Scopo: Visualizzare un numero in un formato controllato.

Righe: da 5000 a 5130

Parametri: Input: Numero da visualizzare in X1
Numero di cifre decimali richieste in Y1.

Variabili locali: NN$, PP, JJ, XX

Nota: Se il numero da visualizzare è maggiore di 999999999, viene visualizzato in virgola mobile senza arrotondamento, altrimenti è arrotondato e visualizzato con Y1 cifre decimali dopo il punto decimale. Se Y1 = 0, il numero è arrotondato all'intero più vicino.

```
5000 REM VISUALIZZA DA X1 a Y1 CIFRE
     DECIMALI
5010 XX=X1: IF Y1 >0 AND ABS(XX) <=
     999999999 THEN 5050
5020 XX=XX+0.5
5030 PRINT INT (XX);
5040 RETURN
5050 IF XX <0 THEN XX=XX-0.5*10↑-
     Y1:GOTO 5070
5060 XX=XX+0.5*10↑-Y1
5070 NN$= STR$ (XX)
5080 FOR PP = 1 TO LEN (NN$)
5090 IF MID$(NN$, PP, 1) = "." THEN
     PRINT LEFT$(NN$, PP+Y1);: RETURN
5100 NEXT PP
5110 PRINT NN$; ".";
5120 FOR JJ=1 TO Y1: PRINT "0";: NEXT JJ
5130 RETURN
```

Programma di gestione

```
10 FOR J= 667 TO 670
20 FOR Y1 =0 TO 3
30 X1 = SQR(J)
40 GOSUB 5000
50 NEXT Y1
60 PRINT
70 NEXT J
80 STOP
```

Passata campione:

26	25·8	25·83	25·826
26	25·8	25·85	25·846
26	25·9	25·87	25·865
26	25·9	25·88	25·884

5. VISUALIZZAZIONE DI STRINGHE

Scopo: Visualizzare una stringa senza dividere le parole sulle righe.

Righe: 5700-5800

Parametri: Input: stringa da visualizzare in X1$.

Variabili locali: XX$, PP, QQ, RR.

```
5700 REM VISUALIZZA X1$ SENZA
     DIVIDERE LE PAROLE
5710 XX$=X1$
5720 PP=LEN(XX$)
5730 IF PP <=22 THEN RR=PP:GOSUB
     5780: RETURN
5740 FOR QQ=23 TO 1 STEP -1
5750 IF MID$(XX$,QQ,1) = " " THEN
     RR=QQ-1:GOSUB 5780: XX$=
     RIGHT$(XX$, PP-QQ):GOTO
     5720
5760 NEXT QQ
5770 RR=22:GOSUB 5780: XX$=RIGHT$
     (XX$, PP-22): GOTO 5720
5780 REM SUBROUTINE INTERNA
5790 PRINT LEFT$(XX$,RR);: IF RR<22
     THEN PRINT
5800 RETURN
```

Programma di gestione

```
10 X1$="BATTI QUALSIASI STRINGA LUNGA
   FINO A TRE RIGHE PER PROVARE LA
   SUBROUTINE
20 GOSUB 5700
30 INPUT X1$: GOSUB 5700
40 GOTO 30
```

Output campione: provare e vedere!

6. CONVERTITORE BINARIO

Scopo: Visualizzare il profilo binario di un numero nel campo da 0 a 255.

Righe: da 1000 a 1060

Parametri: Input: Numero da visualizzare in X1

Variabili locali: YY, KK, XX.

```
1000 REM CONVERTI X1 IN BINARIO E
     VISUALIZZA
1010 YY=256: XX=X1: FOR KK = 1 TO 8
1020 YY=YY/2
1030 IF XX>= YY THEN XX=XX-YY:
     PRINT"★";:GOTO 1050
1040 PRINT" ";
1050 NEXT KK
1060 PRINT:RETURN
```

Programma di gestione:

```
20 FOR J=0 TO 9
20 X1=J: GOSUB 1000
30 NEXT J
40 STOP
```

Output campione:

7. ESTRAZIONE DEL NOME

Scopo: Estrarre un nome dal nome intero di una persona.

Righe: da 4100 a 4200

Parametri: Input: Il nome di una persona, in NI$, in una qualsiasi delle seguenti forme:
J.X. SMITH
HEORGE ELLIOT
ALVA T EDISON
WELLINGTON-COO
K.O'SHAUGNESSY

Output: Il cognome della persona in YI$. Il cognome è definito con una sequenza interrotta di lettere, trattini e apostrofi vicino alle estremità della stringa in NI$.

Esempi sono: SMITH
ELLIOT
EDISON
WELLINGTON-COO
O'-SHAUGHESSY

Se il cognome non può essere trovato, YI$ è fornito vuoto.

Variabili locali: JJ, KK, CC$

Note: Questa subroutine funziona correttamente per i nomi europei ma richiede modifiche per i nomi di altre parti del mondo — ad esempio Cina.

```
41100 REM ESTRAI COGNOME DA NI$ E
      PRESENTALO IN YI$
4110 JJ=LEN (NI$)
4120 IF JJ=0 THEN YI$= " ": RETURN
4130 IF MID$(NI$, JJ, 1) <"A" OR MID$
      (NI$, JJ, 1) > "Z" THEN JJ=JJ-1:
      GOTO 4120
4140 FOR KK=JJ TO 1 STEP -1
4150 CC$=MID$(NI$, KK, 1)
4160 IF NOT (CC$ >= "A" AND CC$ <=
      "Z" OR CC$= "-" OR CC$= "'")
      THEN 4190
4170 NEXT KK
4180 KK=0
4190 YI$=MID$(NI$, KK+1, JJ -KK)
4200 RETURN
```

Programma di gestione:

```
10 INPUT "NOME PREGO"; NI$
20 GOSUB 4100
30 PRINT "IL COGNOME È"; YI$
40 GOTO 10
```

Passata campione:

```
NOME PREGO? J. X. SMITH
IL COGNOME È SMITH
NOME PREGO? GEORGE ELLIOT
IL COGNOME È ELLIOT
NOME PREGO? WELLINGTON-COO
IL COGNOME È WELLINGTON-COO
NOME PREGO? K. O'SHAUGHNESSY
IL COGNOME È O'SHAUGHNESSY
```

8. RICERCA IN LISTE

Scopo: Cercare una lista ordinata, un'entrata specifica, usando il metodo della ricerca dicotomica

Righe: 6000-6050

Parametri: Input: La Lista da esaminare (deve essere in ordine alfabetico o crescente) in AI$.
Indice della prima entrata in L1; indice dell'ultima entrata in H1.
Stringa da osservare in XI$.

Output: Se l'entrata viene trovata, M1 contiene il suo indice. Se non è trovato M1=−1.

Variabili locali: HH, LL.

```
6000 REM CERCA IN LISTA ORDINATA AI$
6005 HH=H1 : LL=L1
6010 IF HH<LL THEN M1 =-1 : RETURN
6020 M1 = INT (0.5*(HH+LL))
6030 IF XI$= AI$(M1) THEN RETURN
6040 IF XI$ < AI$(M1) THEN HH=M1 -1 :
      GOTO 6010
6050 LL=M1+1 : GOTO 6010
```

Programma di gestione:

```
10 DATA ABLE, BAKER, CHARLIE, DOG,
   ERNIE, FRED, GORDON
20 DATA HARRY, IONA, JILL, KATE, LYDIA,
   MURIEL, NICK
30 DIM AI$(14)
40 FOR J=1 TO 14 : READ AI$(J) : NEXT J
50 INPUT"BATTI UN NOME";XI$
60 H1=14:L1=1:GOSUB 6000
70 IF M1<0 THEN PRINT "NON TROVATO":
   GOTO 50
80 PRINT"TROVATO ALL'ENTRATA";
   M1 : GOTO 50
```

Passata campione:

```
BATTI UN NOME? CHARLIE
TROVATO ALL'ENTRATA 3
BATTI UN NOME? DAVID
NON TROVATO
BATTI UN NOME? NICK
TROVATO ALL'ENTRATA 14
...
```

9. BUBBLE SORT

Scopo: Riordinare poche voci di una stringa in ordine alfabetico.

Righe: 6500-6560

Parametri: Input: Elenco di voci da riordinare in AI$(1) fino a AI$(N1).
Numero degli elementi in N1.

Output: Lista ordinata da AI$(1) fino a AI$(N1).

Variabili locali: KK, DD$, SS$

```
6500 REM BUBBLE SORT DI STRINGHE IN AI$
6510 SS$="NO"
6520 FOR KK=1 TO N1-1
```

```
6530 IF AI$(KK)>AI$(KK+1) THEN DD$=
     AI$(KK): AI$(KK)=AI$(KK+1):
     AI$(KK+1)=DD$: SS$= "YES"
6540 NEXT KK
6550 IF SS$= "YES" THEN 6510
6560 RETURN
```

Programma di gestione:

```
10 INPUT "N1"; N1
20 PRINT"BATTI"; N1; "PAROLE"
30 DIM AI$(N1)
40 FOR J=1 TO N1 : INPUT AI$(J) : NEXT J
50 GOSUB 6500
60 FOR J=1 TO N1 : PRINT AI$(J) : NEXT J
70 STOP
```

Passata campione:

```
RUN
N1 ?7
BATTI 7 PAROLE
? PEARS
? CHERRIES
? BANANAS
? ORANGES
? DATES
? PLUMS
? APPLES
APPLES
BANANAS
CHERRIES
DATES
ORANGES
PEARS
PLUMS
```

10. QUICK SORT

Scopo: Riordinare elementi in ordine numerico usando l'algoritmo Quick Sort di Hoare.

Numeri di riga: da 6200 a 6380

Parametri: Input: Lista dei numeri da riordinare da A1(1) fino a A1(N1).

Output: La lista riordinata compare da A1(1) fino a A1(N1).

Variabili locali: SS, SS%, AA, BB, XX, YY, ZZ, DD, PP.

Note: (i) SS non deve essere usato altrove nel programma se la subroutine di riordino è usata più di una volta.

(ii) La subroutine riordinerà stringhe invece di numeri se vengono effettuate le seguenti sostituzioni:
AI$ per A1; ZZ$ per ZZ;
DD$ per DD

(iii) Se la routine viene usata per riordinare una serie di record con i campi che abbracciano parecchi matrici, le seguenti istruzioni devono essere modificate per assicurare che tutti i campi di ogni record siano spostati:
6280 6290
L'operazione di confronto nelle istruzioni 6260 e 6270, può essere invertita o sostituita se si richiede un diverso ordinamento.

```
6200 REM QUICKSORT
6210 IF SS=1 THEN 6230
6220 DIM SS%(100):SS=1: REM DECLARE
     STACK
6230 AA=1: BB=N1: SS%(0)=1: PP=1
6240 XX=AA: YY=BB: ZZ=A1(BB)
6250 IF XX >= YY THEN 6290
6260 IF A1(XX) <= ZZ THEN XX=XX+1:
     GOTO 6250
6270 IF A1(YY) >= ZZ THEN YY=YY-1:
     GOTO 6250
6280 DD=A1(YY): A1(YY)=A1(XX): A1(XX)=
     DD: GOTO 6250
6290 A1(BB)=A1(XX): A1(XX)=ZZ
6300 IF XX-AA <=1 THEN 6340
6310 SS%(PP)=XX: SS%(PP+1)=BB:
     SS%(PP+2)=2: PP=PP+3
6320 BB=XX-1: GOTO 6240
6330 PP=PP-3: XX=SS%(PP): BB=SS%
     (PP+1)
6340 IF BB-XX <=1 THEN 6370
6350 SS%(PP)=3: PP=PP+1: GOTO 6240
6360 PP=PP-1
6370 ON SS%(PP-1) GOTO 6380, 6330,
     6360
6380 RETURN
```

Programma di gestione:

```
10 INPUT "QUANTI";N1
20 DIM A1 (N1)
30 FOR J=1 TO N1:A1(J)=INT(1000*RND(0))
   : NEXT J
40 GOSUB 6200
50 FOR J=1 TO N1:PRINT A1(J);: NEXT J
60 STOP
```

Passata campione:

```
QUANTI? 6
331  342  369  540  870  912
(cioè 6 numeri in ordine
ascendente)
```

11. SEMPLIFICATORE DI FRAZIONI

Scopo: Ridurre le frazioni ai rispettivi minimi termini.

Numeri di riga: da 5500 a 5630

Parametri: Input: *A1 (Numeratore della frazione)*
B1 (Denominatore di frazione)
Output: C1 (Numeratore di frazione semplificata)
D1 (Denominatore di frazione semplificata)

Variabili locali: JJ, KK

```
5500 REM RIDUCI FRAZIONE A1/B1 AI SUOI
     MINIMI TERMINI
```

```
5510 REM RISULTATO IN C1/D1. LOCALI
     SONO JJ, KK
5520 REM ERRORE SE AI O B1 NON SONO
     NUMERI INTERI O SE B1 < 1
5530 IF AI=INT(AI) AND B1=INT(B1)
     AND B1 > 1 THEN 5550
5540 PRINT"PARAMETRI ERRATI PER
     SEMPLIFICATORE FRAZIONE"; AI;B1:STOP
5550 IF AI=0 THEN C1=0:D1=1:RETURN
5560 JJ= AI:KK=B1
5570 IF AI < 0 THEN JJ=-AI
5580 IF KK=0 THEN 5620
5590 IF JJ=0 THEN JJ=KK:GOTO 5620
5600 JJ > KK THEN JJ=JJ-INT(JJ/KK)*
     KK:GOTO 5580
5610 KK=KK-INT  (KK/JJ)*JJ:GOTO 5580
5620 C1= AI/JJ:D1=B1/JJ
5630 RETURN
```


Programma di gestione:

```
10 INPUT "INDICA FRAZIONE"; A1,B1
20 GOSUB 5500
30 PRINT C1;"/"; D1
40 GOTO 10
```

Passata campione:

```
RUN
INDICA FRAZIONE? 2, 4
1/2
INDICA FRAZIONE? 123, 456
41/152
INDICA FRAZIONE? 375, 1000
3/8
INDICA FRAZIONE? 0, 1234
0/1
INDICA FRAZIONE? 0.5, 0.75
PARAMETRI ERRATI PER SEMPLIFICATORE
FRAZIONE .5
.75
RUN
INDICA FRAZIONE? -50, 100
-1/2
INDICA FRAZIONE? 77,0
PARAMETRI ERRATI PER SEMPLIFICATORE
FRAZIONE 77
0
```

12. EQUAZIONI SIMULTANEE

Scopo: Risolvere equazioni simultanee, ad esempio N1 equazioni in N1 incognite

Righe: da 9000 a 9270

Parametri: *Input:* N1: il numero di equazioni da A1(1,1) a A1(N1,N1); una matrice bidimensionale che contiene la matrice dei coefficienti da B1(1) a B1(N1): il vettore dei membri di destra.
Output: da X1(1) a X1(N1) contiene il vettore delle soluzioni.

Variabili locali: DD, JJ, KK, LL

Nota: I valori iniziali delle matrici A1 e B1 sono distrutti.

Esempio: Si considerino tre equazioni in tre incognite:

$$3x + 2y + 1z = 19$$
$$2x + 7y + 2z = 55$$
$$4x + 1y + 4z = 19$$

Un programma, per risolvere questa equazione, potrebbe essere scritto come segue:

```
10 DATA 3, 2, 1, 19
20 DATA 2, 7, 2, 55
30 DATA 4, 1, 4, 19
40 DIM A1(3,3), B1(3), X1(3)
50 N1=3
60 FOR j=1 TO N1: FOR K=1 TO N1:
   READ A1 (K,J): NEXT K: READ B1(J):
   NEXT J
70 GOSUB 9000: REM RICHIAMA
   SUBROUTINE
80 FOR J=1 TO N1: PRINT X1(J):
   NEXT J
90 STOP
```

Tempo: Il tempo richiesto per risolvere una serie di N1 equazioni è all'incirca proporzionale al cubo di N1. Valori tipici sono:

N1	Tempo (secondi)
5	2
10	10
15	30
20	65
25	121

```
9000 REM RISOLVI N1 EQUAZIONI
     SIMULTANEE A1.X1=B1
9010 IF N1=1 THEN X1(1)=B1(1)/A1(1,1):
     RETURN
9020 FOR JJ=1 TO N1-1:REM TROVA PIVOT
9030 DD=ABS(A1(JJ,JJ)):LL=JJ
9040 FOR KK=JJ TO N1
9050 IF ABS(A1(KK,JJ)) > DD THEN DD=
     ABS(A1(KK,JJ)): LL=KK
9060 NEXT KK
9070 IF LL=JJ THEN 9120
9080 FOR KK=JJ TO N1: REM INTERSCAMBIA
     EQUAZIONI
9090 DD=A1(JJ,KK): A1(JJ,KK)=A1(LL,KK):
     A1(LL,KK)=DD
9100 NEXT KK
9110 DD=B1 (JJ): B1 (JJ)=B1 (LL): B1 (LL)=DD
9120 FOR KK=JJ+1 TO N1: DD=A1(KK,JJ)/A1
     (JJ,JJ)
9130 FOR LL=JJ TO N1 : REM ELIMINA
9140 A1 (KK,LL)=A1 (KK,LL)-DD*A1 (JJ,LL)
9150 NEXT LL
9160 B1 (KK)=B1 (KK)-DD*B1 (JJ)
9170 NEXT KK
9180 NEXT JJ
9190 FOR JJ=N1 TO 1 STEP-1 : REM
     RISOSTITUISCI
9200 DD=B1(JJ)
9210 IF JJ=N1 THEN 9250
9220 FOR KK=JJ+1 TO N1
9230 DD=DD-X1(KK)*A1 (JJ,KK)
```

```
9240 NEXT KK
9250 X1 (JJ)=DD/A1(JJ,JJ)
9260 NEXT JJ
9270 RETURN
```

Programma di gestione:

```
 10 INPUT "N1";N1
 20 DIM A1(N1,N1), B1(N1), X1(N1), Y1(N1)
 30 FOR J=1 TO N1
 40 FOR K=1 TO N1
 50 A1(J,K)= 100 *(RND(0)-0.5)
 60 NEXT K
 70 PRINT "Y1(";J; ")";: INPUT Y1(J)
 80 NEXT J
 90 FOR J=1 TO N1
100 B1(J)=0
110 FOR K=1 TO N1
120 B1(J)= B1(J)+Y1(K)*A1(J,K)
130 NEXT K,J
140 X=TI
150 GOSUB 9000
160 X=TI-X
170 FOR J=1 TO N1
180 PRINT Y1(J);X1(J)
190 NEXT J
200 PRINT "TEMPO="; INT (X/600.5)
210 STOP
```

NOTA SUL PROGRAMMA DI GESTIONE

Questo programma è studiato per trovare la subroutine per la risoluzione di equazioni simultanee e per presentare i suoi risultati in una forma che possa essere facilmente controllata.
Il programma inizia chiedendo all'utente il numero di equazioni da risolvere. Richiede quindi un numero appropriato di valori per le "incognite".
Successivamente, il programma costruisce una serie di equazioni per le incognite usando coefficienti casuali. Esso le risolve e presenta una serie di risultati unitamente ai valori originali. I risultati dovrebbero essere gli stessi, salvo piccoli errori di arrotondamento.

Passata campione:

```
RUN
N1?  1
Y1(1)?  6
  6  6                    (Uguale!)
TEMPO = 0
RUN
N1?  4
Y1(1)?  4
Y1(2)?  1
Y1(3)?  7
Y1(4)?  8
  4  4
  1  1                       Uguale
  7  7
  8  8.000000001   Circa uguale
TEMPO = 0
```

APPENDICE C

UNITA': 16

Esperimento 16.1

```
 10 INPUT"STIPENDI IN CENTS";W
 20 FOR J=1 TO 8
 30 READ V,N$
 40 T=INT(W/V)
 50 PRINT T;N$
 60 W=W-V*T
 70 NEXTJ
 80 STOP
1000 DATA 5000, BANCONOTE 50 DOLLARI
1010 DATA 1000, BANCONOTE 10 DOLLARI
1020 DATA 500, BANCONOTE DA 5 DOLLARI
1030 DATA 100, DOLLARI
1040 DATA 25, QUARTI DI DOLLARO
1050 DATA 10, DECIMI DI DOLLARO
1060 DATA 5,5 CENTS
1070 DATA 1,CENTS
```

Esperimento 16.2A

```
 10 FOR K=1 TO 12
 20 READ M$
 30 PRINT M$
 40 NEXT K
 50 STOP
1000 DATA GENNAIO, FEBBRAIO, MARZO,
     APRILE
1010 DATA MAGGIO, GIUGNO, LUGLIO,
     AGOSTO
1020 DATA SETTEMBRE, OTTOBRE, NOVEMBRE,
     DICEMBRE
```

Esperimento 16.2B

```
 10INPUT G,M,A
 20 FOR J=1 TO M
 30 READ M$
 40 NEXT J
 50 PRINT G;M$;A
 60 RESTORE
 70 GOTO 10
1000 DATA GENNAIO, FEBBRAIO, MARZO,
     APRILE
1010 DATA MAGGIO, GIUGNO, LUGLIO,
     AGOSTO
1020 DATA SETTEMBRE, OTTOBRE, NOVEMBRE,
     DICEMBRE
```

Esperimento 16.3

```
 10 T=0
 20 S=0
 30 PRINT" SHIFT e CLR HOME "
 40 READA$
 50 IF A$="END" THEN 240
 60 READ B$
 70 T=T+1
 80 J=1
 90 PRINTA$
100 PRINT
110 INPUT X$
120 IF X$=B$ THEN 200
130 IF J=3 THEN 170
140 J=J+1
150 PRINT"ERRATO. RIPROVA"
160 GOTO 90
170 PRINT"RISPOSTA="
180 PRINT B$
190 GOTO 40
200 PRINT"ESATTO!"
210 IF J > 1 THEN 40
220 S=S+1
230 GOTO 40
240 PRINT"HAI RISPOSTO";S;"ESATTO"
250 PRINT"AL PRIMO COLPO"
260 PRINT"SU";T;"DOMANDE"
270 STOP
280 DATA CHI COMPOSE IL MESSIA, HANDEL
290 DATA QUANTE SONO LE SINFONIE
     SCRITTE DA BEETHOVEN, NOVE
300 DATA CHI HA SCRITTO LA CARMEN, BIZET
310 DATA COSA SUONAVA PAGANINI,
     VIOLINO
320 DATA END
```

UNITA': 17

Esperimento 17.1A

```
10 INPUT"QUANTI MINUTI";M:R=TI+M
   *3600
20 IF TI < R THEN 20
30 PRINT"TEMPO SCADUTO":STOP
```

Esperimento 17.1B

```
10 PRINT"USA 1000000 PER":PRINT"
   TERMINARE INPUT":S=0:N=0
20 INPUT"NUMERO SUCCESSIVO";X:IF X =
   1000000 THEN 40
30 S=S+X:N=N+1:GOTO 20
40 PRINT"MEDIA=";S/N:STOP
```

Esperimento 17.1C

```
 10 REM DEBUG THIS PROGRAM!
 20 INPUT"NAME";N$
 30 IFN$="JIM"THEN A$="JAMES":GOTO
    100
 40 IFN$="BOB"THEN A$="ROBERT":GOTO
    100
 50 IFN$="KATE"THEN A$="KATHERINE":
    GOTO100
 60 IFN$="PENNY"THEN A$="PENELOPE":
    GOTO100
 70 PRINT N$;" IS NOT"
 80 PRINT"SHORT FOR ANYTHING."
 90 GOTO10
100 PRINT N$;" IS SHORT"
110 PRINT"FOR ";A$
120 GOTO10
```

Esperimento 17.2A

Più generoso: SSF o SWWWW

```
NOT ((S < = 1 AND F < = 2 AND W = 0) OR
(S < = 1 AND F = 0 AND W < = 4)).
```

Più restrittivo: SFF o WWWW

```
NOT ((S < = 1 AND F < = 2 AND W = 0) OR
(S = 0 AND F = 0 AND W < =4)).
```

Esperimento 17.2B

```
N$<>"JONES" AND N$<>"SMITH"
AND N$<>"BROWN"
X>15 OR X<4
```

Esperimento 17.2C

```
 10 REM ESERCIZIO 17.2C
 20 IF T < 0.1 THEN
    PRINT"FANTASTICO!!": GOTO 100
 30 IF T < 0.15 THEN
    PRINT"ECCEZIONALE!": GOTO 100
 40 IF T > = 0.15 AND T < 0.2 THEN
    PRINT"MOLTO BUONO":GOTO 100
 50 IF T > = 0.2 AND T < 0.25 THEN
    PRINT"BUONO":GOTO 100
 60 IF T > = 0.25 AND T < 0.28 THEN
    PRINT"DISCRETO":GOTO 100
 70 IF T > = 0.28 AND T < 0.33 THEN
    PRINT"MOLTO LENTO": GOTO 100
 80 IF T > = 0.33 AND T < 0.4 THEN
    PRINT"SVEGLIA!":GOTO 100
 90 IF T > 0.4 THEN PRINT"PROVA
    ANCORA QUANDO SEI SOBRIO!!"
100 STOP
```

Esperimento 17.2D

```
10 INPUT"PAROLA";W$
20 IF W$ > ="ABRAHAM" AND W$ < =
   "FRANCE" THEN PRINT"USA VOL 1":STOP
30 IF W$ > ="FRANCHISE" AND W$ < =
   "LEVANTE" THEN PRINT"USA VOL 2":STOP
40 IF W$ > ="LEVITAZIONE" AND W$ < =
   "QUOTA" THEN PRINT"USA VOL 3":STOP
50 IF W$ > ="QUOZIENTE" AND W$ < =
   "XILOFONO" THEN PRINT"USA VOL 4":
   STOP
60 PRINT"QUESTA PAROLA NON È"
70 PRINT"NELL'ENCICLOPEDIA"
80 STOP
```

UNITA': 18

Esperimento 18.1A

```
 10 PRINT"[SHIFT] e [CLR HOME] PROVA
    ARITMETICA"
 20 PRINT"CALCOLA SOMME SEGUENTI"
 30 S=0
 40 FOR A=1 TO 10
 50 X = INT(10*RND (0)+1)
 60 Y = INT(10*RND (0)+1)
 70 PRINT X;"+";Y;"=";
 80 INPUT Z
 90 GOSUB 1000:REM CREA SUONO
100 GOSUB 2000:REM CAMBIA COLORE
    MARGINE
110 NEXT A
120 PRINT"ESATTO";S;"SU 10"
130 FOR R=1 TO S
140 GOSUB 1000:REM CREA SUONO
150 NEXT R
160 STOP
```

```
1000 REM SUBROUTINE PER CREARE SUONO
     PIP
1010 POKE 36878,15:POKE 36876,245
1020 FOR MM=1 TO 100:NEXT MM
1030 POKE 36878,0
1040 FOR MM=1 TO 800:NEXT MM
1050 RETURN
2000 REM SUBROUTINE PER CAMBIARE
     COLORE MARGINE
2010 IF Z=X+Y THEN POKE 36879,28:S=S+1:
     GOTO 2030:REM RISPOSTA ESATTA—
     MARGINE PORPORA
2020 POKE 36879,24:REM RISPOSTA ERRATA
     —MARGINE NERO
2030 FOR KK=1 TO 800:NEXT KK
2040 POKE 36879,27:REM RIPRISTINA
     COLORE INIZIALE
2050 RETURN
```


Esperimento 18.1B

```
 10 PRINT" SHIFT e CLR HOME PROVA
    CONTEGGIO"
 20 PRINT"CONTA I PIP"
 30 S=0
 40 FOR A=1 TO 10
 50 FOR T=1 TO 5000:NEXT T:REM ASPETTA
 60 X=INT(9*RND(0)+1)
 70 FOR J=1 TO X
 80 GOSUB 1000
 90 NEXT J
100 INPUT Z
110 IF Z=X THEN GOSUB 5000:GOSUB 2000:
    GOSUB 4000:GOTO 130
120 GOSUB 3000:GOSUB 4000
130 NEXT A
140 PRINT"QUESTO È";S;"GIUSTO"
150 STOP
1000 REM FA UN PIP
1010 POKE 36878,15:POKE 36876,245
1020 FOR MM=1 TO 100:NEXT MM
1030 POKE 36878,0
1040 FOR MM=1 TO 800:NEXT MM
1050 RETURN
2000 REM SUBROUTINE PER CAMBIARE COLORE
     MARGINE PORPORA-RISPOSTA ESATTA
2010 IFZ=XTHENPOKE36879,28:S=S+1:
     RETURN
3000 REM SUBROUTINE PER CAMBIARE COLORE
     MARGINE NERO-RISPOSTA ERRATA
3010 POKE36879,24:RETURN
4000 REM SUBROUTINE PER RIPRISTINARE
     COLORE INIZIALE
4010 FORKK=1TO800:NEXT KK
4020 POKE 36879,27:RETURN
5000 REM MISTERO
5010 POKE 36878,15
5020 FOR MM=150 TO 200 STEP 1
5030 POKE 36876,MM
5040 FOR TT=1 TO 10:NEXT TT
5050 NEXT MM
5060 POKE 36878,0
5070 RETURN
```

Esperimento 18.2

```
10 PRINT" SHIFT e CLR HOME ";
20 FOR X1=0TO16

30 C1$=" CTRL e RED "
40 GOSUB 500
50 FOR T=1TO150 :NEXT T

60 C1$=" CTRL e WHT "
70 GOSUB500
80 NEXT X1
90 GOTO 90
500 REM  MOSTRO

510 PRINT" CLR HOME CRSR ↓ CRSR ↓ ";C1$;
520 IF X1=0THEN540

530 FORJJ=1 TO X1:PRINT" CRSR → ";:NEXT JJ

540 PRINT" SPACE SPACE CTRL e
RVS ON SHIFT e £ e *
CRSR ↓ SHIFT e CRSR → 2 volte SPACE
CRSR ↓ SHIFT e CRSR → SPACE
CRSR ↓ SHIFT e CRSR → 2 volte
SHIFT e £ SPACE e *
CRSR ↓ SHIFT e CRSR → 3 volte
SPACE 3 volte CRSR ↓ SHIFT e
CRSR → 3 volte CTRL e RVS OFF e
* CTRL e RVS ON SPACE CTRL
e RVS OFF SHIFT e £ CRSR ↓
SHIFT e CRSR → 3 volte SHIFT e
N SPACE SHIFT e M CRSR ↓
SHIFT e CRSR → 4 volte SHIFT e
V SPACE 3 volte SHIFT e V ";
550 RETURN
```

Esperimento 18.3

```
10 PRINT" [SHIFT] e [CLR HOME] "
20 X1=1:Y1=22:N1=17:GOSUB2000
30 X1=1 :Y1=5:N1=3:GOSUB3000
40 X1=4:Y1=3:N1=3:GOSUB4000
50 X1=7:Y1=6:N1=5:GOSUB2000
60 X1=7:Y1=10:N1=13:GOSUB1000
70 X1=20:Y1=10:N1=11:GOSUB2000
80 X1=1:Y1=21:N1=20:GOSUB1000
90 X1=1:Y1=21:N1=20:GOSUB1000
100 X1=3:Y1=5:GOSUB5000
110 Y1=15
120 FORX1=2 TO 19 STEP 4
130 GOSUB6000
140 NEXT X1
150 X1=4:Y1=0:GOSUB7000:REM DRAW
    CROSS
160 GOTO160
500 REM POSITION CURSOR TO X1,Y1
510 PRINT" [CLR HOME] ";
520 IF X1=0 THEN 540
530 FORKK=1 TO X1:PRINT" [CRSR →] ";:NEXT KK
540 IF Y1=0 THEN RETURN
550 FORKK=1 TO Y1:PRINT" [CRSR ↑] ";:NEXT KK
560 RETURN
1000 REM TO DRAW HORIZONTAL LINE FOR
     N1 UNITS FROM X1,Y1
1010 GOSUB 500:REM POSITION CURSOR
1020 FOR JJ=1 TO N1
1030 PRINT" [CTRL] e [RVS ON] [SPACE] ";
1035 NEXT JJ
1037 PRINT" [CTRL] e [RVS OFF] ";
1040 RETURN
2000 REM DRAW VERTICAL LINE N1 UNITS
     DOWN FROM X1,Y1
2010 GOSUB 500:REM POSITION CURSOR
2020 FOR JJ=1 TO N1
2030 PRINT" [CTRL] e [RVS ON] [SPACE]
     [CRSR ↓] [SHIFT] e [CRSR →] ";
2040 NEXT JJ
2050 RETURN
3000 REM DRAW LINE DIAGONALLY UPWARDS
     AND RIGHT FROM X1,Y1
3010 GOSUB 500:REM POSITION CURSOR
3020 FOR KK=1 TO N1
3030 PRINT" [CTRL] e [RVS ON] [SHIFT]
     e [£] [CTRL] e [RVS OFF] [SHIFT]
     e [£] [SHIFT] e [CRSR ↓] [SHIFT]
     e [CRSR →] ";
3040 NEXT KK
3050 RETURN
4000 REM DRAW LINE DIAGONALLY DOWN-
     WARDS AND RIGHT FROM X1,Y1
4010 GOSUB 500:REM POSITION CURSOR
4020 FOR KK=1 TO N1
4030 PRINT" [C=] e [*] [CTRL] e
     [RVS ON] [C=] e [*] [CTRL] e
     [RVS OFF] [CRSR ↓] [SHIFT] e [CRSR →] ";
4040 NEXT KK
4050 RETURN
5000 REM DRAW WINDOW
5010 GOSUB 500:REM POSITION CURSOR
5020 PRINT" [C=] e [A] [SHIFT] e
     [*] [C=] e [S] [CRSR ↓] [SHIFT] e
     [CRSR →] 3 volte [SHIFT] e [-] [SPACE]
     [SHIFT] e [-] [CRSR ↓] [SHIFT] e
     [CRSR →] 3 volte [C=] e [Z] [SHIFT] e
     [*] [C=] e [X] ";
5030 RETURN
6000 REM PAINT ARCHED WINDOW
6010 GOSUB 500
6020 PRINT" [SHIFT] e [N] [SHIFT]
     e [M] [CRSR ↓] [SHIFT] e [CRSR →] 2 volte
     [C=] e [+] 2 volte [CRSR ↓] [SHIFT] e
     [CRSR →] 2 volte [C=] e [+] 2 volte [CRSR ↓]
     [SHIFT] e [CRSR →] 2 volte [C=] e
     [+] 2 volte ";
6030 RETURN
7000 REM DRAW CROSS
7010 GOSUB 500
7020 PRINT" [CTRL] e [RVS ON] [SPACE]
     [CRSR ↓] [SHIFT] e [CRSR →] 2 volte
     [SPACE] 3 volte [CRSR ↓] [SHIFT] e
     [CRSR →] 2 volte [SPACE] [CRSR ↓] [SHIFT] e
     [CRSR →] [SPACE] ";
7030 RETURN
```

UNITA': 19

Esperimento 19.1

```
  10 INPUT"PRIMA FRAZIONE";P,Q
  20 INPUT"SECONDA FRAZIONE"; S,T
  30 A1=P*T+Q*S
  40 B1=Q*T
  50 GOSUB 5500
  60 PRINT"RISULTATO=";C1;"/"; D1
  70 STOP
5500 REM RIDUCI FRAZIONE R1/B1 AI
     MINIMI TERMINI
5510 REM RISULTATO IN C1/D1 LOCALI
     SONO JJ,KK
5520 JJ=A1:KK=B1
5530 IF JJ=KK THEN 5560
5540 IF JJ<KK THEN KK=KK-JJ:GOTO 5530
5550 JJ=JJ-KK:GOTO 5530
5560 C1=A1/JJ:D1=B1/JJ
5570 RETURN
```

Esperimento 19.2B

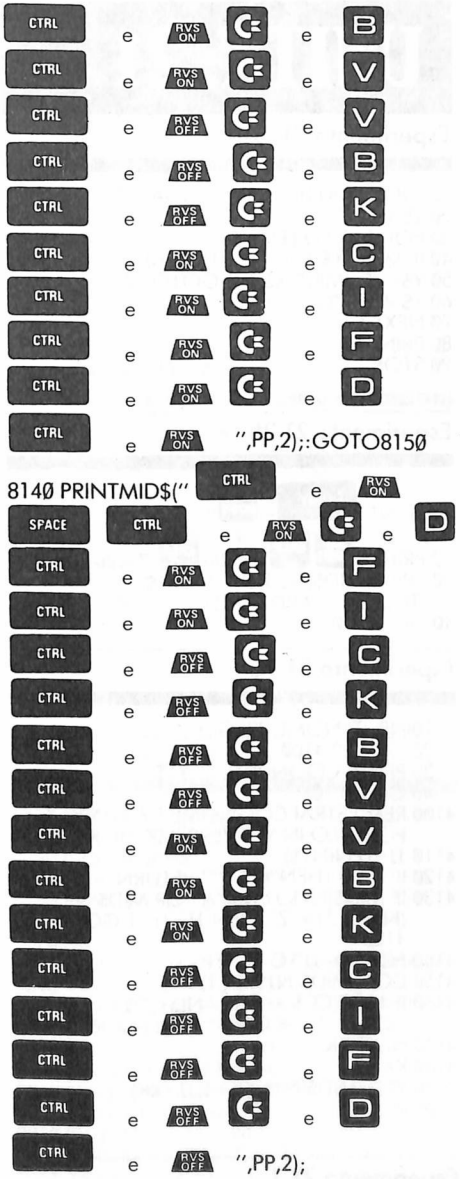
```
  10 INPUT"PRIMA FRAZIONE";P,Q
  20 INPUT"SECONDA FRAZIONE";S,T
  30 A1=P*T+Q*S
  40 B1=Q*T
  50 GOSUB 5500
  60 PRINT"RISULTATO=";C1;"/";D1
  70 STOP
5500 REM RIDUCI FRAZIONE A1/B1 AI
     MINIMI TERMINI USANDO DIVISIONE
     NON SOTTRAZIONE
5510 REM RISULTATO IN C1/D1 LOCALI
     SONO JJ,KK,LL
5520 REM ERRORE SE A1 O B1 NON SONO
     NUMERI INTERI O SE B1<1
5530 IF A1=INT(A1) AND B1=INT(B1)AND
     B1>=1 THEN 5550
5540 PRINT"PARAMETRI SBAGLIATI-":PRINT
     A1;B1:STOP
5550 IF A1=0THEN C1=0:D1=1:RETURN
5560 LL=1:IFA1<0THEN LL=-1:A1=-A1
5570 JJ=A1:KK=B1
5580 IFKK=0THEN 5620
5590 IFJJ=0THZN JJ=KK:GOTO5620
5600IFJJ>KK THEN JJ=JJ-INT(JJ/KK)*KK:
     GOTO5580
5610 KK=KK-INT(KK/JJ)*JJ:GOTO 5580
5620 C1=LL*A1/JJ:D1=B1/JJ
5630 RETURN
```

Esperimento 19.3

```
  10 INPUT"TRE NUMERI";A1,B1,C1
  20 GOSUB 1000
  30 PRINT"IL MAGGIORE=";X1
  40 GOTO 10
1000 REM TROVA MAGGIORE DI A1,B1,C1
1010 REM E DAI RISULTATO IN X1
1020 X1=A1
1030 IF X1<B1 THEN X1=B1
1040 IF X1<C1 THEN X1=C1
1050 RETURN
```

Esperimento 19.4

```
  10 GET A1$:IFA1$=""THEN 10
  20 GOSUB 8000
  30 GOTO 10
8000 REM VISUALIZZA CARATTERE IN A1$
     4 VOLTE PIÙ GRANDE
8010 BB=ASC(A1$)
8020 IFBB=13ORBB=141THENPRINT"
     [CRSR] 3 volte":RETURN
8030 IFBB=18THENQQ=1:RETURN
8040 IFBB=146THENQQ=0:RETURN
8050 IFBB<32THENPRINTMID$("[CTRL] e
     [RVS OFF] 5 volte [CTRL] e [" 2 "] [CTRL]
     e [RVS OFF] 13 volte [CLR HOME] [CTRL] e
     [RVS OFF] 9 volte [CTRL] e [" 2 "] [CTRL]
     e [RVS OFF] [CTRL] e [& 6] [CTRL] e
     [' 7] ",BB+1,1);:RETURN
8060 IFBB>=144 AND BB<160THENPRINT
     MID$("[CTRL] e [! 1] [CTRL]
     e [RVS OFF] 2 volte [SHIFT] e [CLR HOME]
     [CTRL] e [RVS OFF] 9 volte [CTRL] e
     [% 5] [CTRL] e [RVS OFF] [CTRL] e
     [( 8] [CTRL] e [$ 4] ",BB-143,1);:
     RETURN
8070 AA=(BBAND31)+0.5*(BBAND128):IF
     (BBAND64)=0THENAA=AA+32
8080 FORJJ=0 TO 6 STEP 2
8090 KK=PEEK(32768+8*AA+JJ):LL=PEEK
     (32769+8*AA+JJ)
8100 NN=64:FORMM=0TO3
8110 PP=1+8*INT(KK/NN)+2*INT(LL/NN)
8120 KK=KK-INT(KK/NN)*NN:LL=LL-INT
     (LL/NN)*NN
8130 IFQQ=0THENPRINTMID$("[CTRL]
     e [RVS OFF] [SPACE] [CTRL] e [RVS OFF]
     [C=] e [D] [CTRL] e [RVS OFF] [C=]
     e [F] [CTRL] e [RVS OFF] [C=] e
     [I] [CTRL] e [RVS OFF] [C=] e [C]
     [CTRL] e [RVS ON] [C=] e [K]
```



UNITA': 20

Esperimento 20.1

```
10 DIM W$(100)
20 N=0
30 INPUT"NOME";X$
40 IF X$="ZZZZ" THEN 60
50 N=N+1:W$(N)=X$:GOTO30
60 FOR J=N TO 1 STEP -1
70 PRINT W$(J)
80 NEXT J
90 STOP
```


Esperimento 20.2A

```
 10 DIM T$(40)
 20 FOR J=0 TO 40
 30 READ T$(J)
 40 NEXT J
 50 DATA NIL,I,II,III,IV,V,VI,VII,VIII,IX,X
 60 DATAXI,XII,XIII,XIV,XV,XVI,XVII,XVIII,XIX,XX
 70 DATAXXI,XXII,XXIII,XXIV,XXV,XXVI,XXVII,
    XXVIII,XXIX,XXX
 80 DATAXXXI,XXXII,XXXIII,XXXIV,XXXV,XXXVI,
    XXXVII,XXXVIII,XXXIX,XXXX
100 PRINT"DATI DUE NUMERI"
110 INPUT X$,Y$
120 A1$=X$:GOSUB 1000:X=B1
130 A1$=Y$:GOSUB 1000:Y=B1
140 Z=X+Y
150 IF Z>40 THEN PRINT"I RISULTATI
    SUPERANO CAPACITÀ":STOP
160 PRINT"SUM =";T$(Z)
170 STOP
1000 REM CONVERTI A1$ NEL NUMERO
     ROMANO B1
1010 FOR JJ=0 TO 40
1020 IF A1$=T$(JJ) THEN 1050
1030 NEXT JJ
1040 PRINT"NON TROVATA ENTRATA":STOP
1050 B1=JJ
1060 RETURN
```

Esperimento 20.2B1

```
  7 REM SOLUZIONE CON MATRICI
 10 DIM N$(20),T$(20)
 20 FORJ=1 TO 20
 30 READ N$(J),T$(J)
 40 NEXTJ
 50 INPUT"NOME";X$
 60 FOR J=1 TO 20
 70 IF X$=N$(J) THEN PRINT X$;"HA NUMERO
DI TELEFONO";T$(J): PRINT:GOTO 50
 80 NEXT J
 90 PRINTX$;"NON È NELLA LISTA"
100 PRINT "TELEFONICA"
110 GOTO 30
```

```
1000 DATA MAXWELL,3398123
1010 DATABOHR,558
1020 DATAEINSTEIN,4073189
1030 DATAVON NEUMANN,777000
1040 DATANEWTON,3074
1050 DATA ZUSE,222
1060 DATAPLANCK,1237543
1070 DATABOYLE,146543
1080 DATABABBAGE,03474
1090 DATAPALACE,5674
1100 DATAPTOLEMY,54863
1110 DATAARISTOTELE,66543
1120 DATAMCCARTHY,47
1130 DATADIJKSTRA,645
1140 DATABERZELIUS,777
1150 DATACHARLES,5543
1160 DATAMENDELEEV,645634
1170 DATATSIOLKOVSKY,645332
1180 DATAARCHIMEDES,2
1190 DATAHOYLE,21352
```


Esperimento 20.2B2

```
  7 REM SOLUZIONE SENZA MATRICI
 10 RESTORE
 20 INPUT"NOME";X$
 30 FORJ=1 TO 20
 40 READ N$,T$
 50 IF N$=X$THEN PRINTX$;"HA NUMERO
    DI TELEFONO";T$:PRINT:GOTO 10
 60 NEXT J
 70 PRINTX$;"NON È NELLA LISTA"
 80 PRINT"TELEFONICA"
 90 GOTO 10
1000 DATA MAXWELL,3398123
1010 DATABOHR,558
1020 DATAEINSTEIN,4073189
1030 DATAVON NEUMANN,777000
1040 DATANEWTON,3074
1050 DATA ZUSE,222
1060 DATAPLANCK,1237543
1070 DATABOYLE,146543
1080 DATABABBAGE,03474
1090 DATAPALACE,5674
1100 DATAPTOLEMY,54863
1110 DATAARISTOTELE,66543
1120 DATAMCCARTHY,47
1130 DATADIJKSTRA,645
1140 DATABERZELIUS,777
1150 DATACHARLES,5543
1160 DATAMENDELEEV,645634
1170 DATATSIOLKOVSKY,645332
1180 DATAARCHIMEDES,2
1190 DATAHOYLE,21352
```

UNITA': 21

Esperimento 21.1A

```
10 INPUT"BATTI UNA STRINGA";X$
20 Y$=""
30 FOR J=1 TO LEN(X$)
40 IF MID$(X$,J,1)="E" THEN 60
50 Y$=Y$+MID$(X$,J,1):GOTO 70
60 Y$=Y$+"O"
70 NEXT J
80 PRINT Y$
90 STOP
```

Esperimento 21.2B

```
10 PRINT" SHIFT e CLR HOME "
20 PRINT" CLR HOME ↑ CRSR ↓ 6 volte → CRSR 7 volte";
30 PRINT MID$(TI$,1,2);"/";MID$
   (TI$,3,2);"/";MID$(TI$,5,2)
40 GOTO 20
```

Esperimento 21.1C

```
   10INPUT"NOME PREGO";N1$
   20 GOSUB 4100
   30 PRINT"COGNOME È";Y1$
   40 GOTO 10
4100 REM ESTRAI COGNOME DA N1$ E
       PORTALO IN Y1$(VERS. MIGLIORATA)
4110 JJ=LEN(N1$)
4120 IF JJ=0 THEN Y1$="":RETURN
4130 IF MID$(N1$,JJ,1)<"A" OR MID$
       (N1$,JJ,1)>"Z"THEN JJ=JJ-1:GOTO
       4120
4140 FORKK=JJ TO 1 STEP-1
4150 CC$=MID$(N1$,KK,1)
4160 IF NOT(CC$>="A" AND CC$<="Z" OR
       CC$="-" OR CC$="")THEN 4190
4170 NEXT KK
4180 KK=0
4190 Y1$=MID$(N1$,KK+1,JJ-KK)
4200 RETURN
```

Esperimento 21.2

```
   10FOR J=667TO 677
   20 FOR Y1=0 TO 3
   30 X1=SQR(J)
   40 GOSUB5000
   50 NEXT Y1
   60 PRINT
   70 NEXT J
   80 STOP
5000 REM VISUALIZZA DA X1 A Y1 DECIMALI
5010 IF Y1>0 AND ABS(X1)<=999999999
       THEN GOTO 5050
```

```
5020 X1=X1+0.5
5030 PRINT INT(X1);
5040 RETURN
5050 IF X1<0THEN X1=X1-0.5*10↑-Y1:
     GOTO 5070
5060 X1=X1+0.5*10↑-Y1
5070 NN$=STR$(X1)
5080 FOR PP = 1 TO LEN(NN$)
5090 IF MID$(NN$,PP,1)="." THEN PRINT
     LEFT$(NN$,PP+Y1);:RETURN
5100 NEXT PP
5110 PRINT NN$; ".";
5120 FORJJ=1TO Y1:PRINT"0";:NEXTJJ
5130 RETURN
```

Esperimento 21.3A

```
10 INPUT"STRINGA";N$
20 FORJ=1 TO LEN(N$)
30 IF MID$(N$,J,1)>="A" AND MID$(N$,J,1)
   <="Z" THEN 60
40 NEXT J
50 PRINT"NESSUNA PAROLA":STOP
60 N=VAL(LEFT$(N$,J-1))
70 N$=RIGHT$(N$,LEN(N$)-J+1)
80 PRINT N$,2*N
90 GOTO 10
```

Esperimento 21.3B

```
  10 DIM N1$(10),Q1(10)
  20 INPUT"LISTA";X1$
  30 GOSUB 8000
  40 FORJ=1TOX
  50 PRINT N1$(J),Q1(J)
  60 NEXT J
  70 STOP
8000 REM ANALIZZA LISTA ACQUISTI IN X1$
8010 X=0:JJ=1:LL=LEN(X1$)
8020 GOSUB 8200:REM PRIMA CERCA PRIMA
     CIFRA DI UN NUMERO
8030 IF JJ>LL THEN RETURN:REM SE
     STRINGA TERMINATA
8040 GOSUB 8300:REM ESTRAI NUMERO
     (FORNITO IN NN)
8050 IF JJ>LL THEN 8100:REM SEGNALA
     ERRORE SE STRINGA TERMINATA
8060 X=X+1:Q1(X)=NN:REM SCARTA NUMERO
8070 GOSUB 8400:REM ESTRAI PAROLA IN
     S1$.Z1=0 SE PAROLA NON SI TROVA
8080 IF Z1=0THEN 8100
8090 N1$(X)=S1$:GOTO 8020
8100  PRINT"NON  COMPRENDO":X=0:RE-
     TURN
8200 REM CERCA INIZIO NUMERO IN X1$
8210 IF JJ>LL THEN RETURN
8220 CC$=MID$(X1$,JJ,1)
8230 IFCC$<"0" OR CC$>"9"THEN JJ=JJ+1:
     GOTO8210
8240 RETURN
8300 REM ESTRAI NUMERO DA X1$
     INIZIANDO IN JJ, E PORTALI IN NN.
     AVANZA
8310 KK=JJ:JJ=JJ+1
8320 IFJJ > LLTHEN RETURN
8330 CC$=MID$(X1$,JJ,1)
8340 IF CC$>="0" AND CC$<="9" THEN
     JJ=JJ+1:GOTO8320
8350 NN=VAL(MID$(X1$,KK,JJ-KK)): RETURN
8400 REM ESTRAI PAROLA DA X1$
     INIZIANDO DA JJ.PORTALA IN S1$, E
     AUMENTA JJ
8405 REM Z1=0 SE NON TROVA PAROLA
8410 IFJJ>LL THEN Z1=0:RETURN
8420 CC$=MID$(X1$,JJ,1)
8430 IF CC$<"A" OR CC$>"Z" THEN JJ=JJ+1
     :GOTO 8410
8440 KK=JJ:JJ=JJ+1
8450 IF JJ>LLTHEN8480
8460 CC$=MID$(X1$,JJ,1)
8470 IFCC$>="A" AND CC$<="Z"THEN
     JJ=JJ+1:GOTO8450
8480 S1$=MID$(X1$,KK,JJ-KK):Z1=1:RETURN
```


UNITA': 22

Esperimento 22.1

```
  10 DATABAIN,BEAVIS,BOWEY,BURNS
     CLARK,FLEMING
  20 DATAGORDON,GREEN,HOOD,KIDD,
     MCCABE,MAVER
  30 DATAMARSHALL,MILLER,NORTH,PACK,
     PERKINS,REED,ROSE
  40 DATAROSS,SIMPSON,SMITH,SYKES,
     TEDFORD,WEBSTER,WOOD
  50 DIMA$(26)
  60 FORJ=1 TO 26:READA$(J):NEXT J
  70 INPUT"BATTI UN NOME";X$
  80 H1=26:L1=1: GOSUB6000
  90 IFM1=-1THEN PRINTX$;"NON
     TROVATO":GOTO 70
 100 PRINT X$;"TROVATO IN ENTRATA";M1
 110 GOTO 70
6000 REM CERCA LISTA ORDINATA:LISTA A$
6010 IF H1<L1 THEN M1=-1:RETURN
6020 M1=INT(0.5*(H1+L1))
6030 IF X$=A$(M1)THEN RETURN
6040 IF X$<A$(M1)THEN
     H1=M1-1:GOTO6010
6050 L1=M1+1:GOTO6010
```

Esperimento 22.2

```
 10DATAROSS,SIMPSON,SMITH,SYKES,
    TEDFORD,WEBSTER,WOOD
 20 DATAMARSHALL,MILLER,NORTH,PACK,
    PERKINS,REED,ROSE
 30 DATAGORDON,GREEN,HOOD,KIDD,
    MCCABE,MAVER
```

```
  40 DATABAIN,BEAVIS,BOWEY,BURNS,
     CLARK,FLEMING
  50 DIMA1$(26)
  60 FORJ=1 TO 26:READA1$(J):NEXT J
  70 N1=26:GOSUB6500
  80 FOR J=1 TO 26
  90 PRINT A1$(J)
 100 NEXT J
 110 STOP
6500 REM BUBBLE SORT DI N1 CAMPI DA
     A1$(1) FINO A A1$(N1)
6510 SS$="NO"
6520 FOR KK=1 TO N1 -1
6530 IFA1$(KK)>A1$(KK+1)THENDD$=A1$
     (KK):A1$(KK)=A1$(KK+1):A1$(KK+1)=
     DD$:SS$="SI"
6540 NEXT KK
6550 IF SS$="SI"THEN 6510
6560 RETURN
```


Esperimento 22.3A

```
  10 DATAADAMS,27
  20 DATABRIGGS,66
  30 DATACHILVERS,29
  40 DATADALE,38
  50 DATACOLIN,12
  60 DATAMACSNOOT,67
  70 DATAWILSON,96
  80 DATA THOMSON,53
  90 DATAWILMOTT,31
 100 DATABAIN,42
 110 DATAMUNDY,64
 120 DATAKRESTIN,85
 130 DATAMCILDOWIE,10
 140 DATAWRAITH,99
 150 DATAGREEN,72
 160 DATAGREENE,52
 170 DATASHEPHERD,53
 180 DATAHUTCHISON,64
 190 DATABLACK,45
 200 DATABAXTER,1
 210 DATASMYRL,99
 220 DATAFLASHMAN,2
 230 DATAMORRIS,75
 240 DATAELLIS,42
 250 DATAMATTHEWS,66
 260 DATAFOLEY,91
 270 DATACOLLINS,36
 280 DATADANIELS,93
 290 DATAJACKSON,77
 300 DATAPEIRCE,78
 310 DATADEWEY,34
 320 DATAMACGREGOR,15
 330 DATAEASON,69
 340 DATAPARSONS,6
 350 DATATHATCHER,45
 360 DATACLAYMORE,85
 370 DATAO'FLAHERTY,66
 380 DATABUNN,5
 390 DATASULLIVAN,85
 400 DATAGILBERT,41
 500 N1=40:REM NUMBER OF PUPILS
 510 DIM S$(N1),M(N1),A1(N1)
 520 FOR J=1TON1
 530 READ S$(J),M(J)
 540 A1(J)=M(J)
 550 NEXT J
 560 REM ORA RIORDINA VOTI IN A1
 570 GOSUB 6000
 580 REM TROVA LIMITE(TRE QUARTI LISTA)
 590 P=A1 (INT(0.75★N1+1))
 600 PRINT"LIMITE =";P
 610 REM ORA VISUALIZZA STUDENTI CHE
     HANNO PASSATO L'ESAME
 620 FOR J=1 TO N1
 630 IF M(J)>=P THEN PRINT S$(J),M(J)
 640 NEXT J
 650 STOP
6000 REM QUICKSORT:RIORDINA N1
     ELEMENTI DI A1
6010 IFSS=1 THEN6030
6020 DIMSS%(100):SS=1: REM DICHIARA CODA
6030 AA=1:BB=N1:SS%(0)=1:PP=1
6040 XX=AA:YY=BB:ZZ=A1(BB)
6050 IFXX>=YYTHEN6090
6060 IFA1(XX)<=ZZTHENXX=XX+1:GOTO
     6050
6070 IFA1(YY)>=ZZTHENYY=YY-1:GOTO
     6050
6080 DD=A1(YY):A1(YY)=A1(XX):A1(XX)=
     DD:GOTO6050
6090 A1(BB)=A1(XX):A1(XX)=ZZ
6100 IFXX-AA<=1THEN6140
6110 SS%(PP)=XX:SS%(PP+1)=BB:SS%(PP+2)
     =2:PP=PP+3
6120 BB=XX-1:GOTO6040
6130 PP=PP-3:XX=SS%(PP):BB=SS%(PP+1)
6140 IFBB-XX<=1THEN6170
6150 SS%(PP)=3:PP=PP+1:AA=XX+1:GOTO
     6040
6160 PP=PP-1
6170 ONSS%(PP-1)GOTO6180,6130,6160
6180 RETURN
```

Esperimento 22.3B

```
 10 REM PRIMA PARTE PROGRAMMA VITA
 20 DIMX$(9,9),Y$(9,9)
 30 PRINT" SHIFT e CLR HOME BATTI
    PROFILO INIZIALE"
 40 PRINT"CON ★ E SPAZI,"
 50 PRINT"USANDO COMANDI CURSORE"
 60 PRINT"PER ENTRARE NELLA"
 70 PRINT"SCATOLA. USA RETURN"
 73 PRINT"PER TERMINARE CIASCUNA FILA"
 77 PRINT"(ANCHE QUELLE VUOTE)."
 80 PRINT" 3 spazi C= e A
    SHIFT e * 9 volte C= e S"
 90 FORJ=1 TO 9
100 PRINT" 3 spazi SHIFT e -
    9 spazi SHIFT e -"
```

```
110 NEXTJ
120 PRINT" 3 spazi [C=] e [Z]
    [SHIFT] e [*] 9 volte [C=] e [X]"
130 PRINT" [CLR HOME] [CRSR ↓] 8 volte"
140 FOR J=1 TO 9
150 T$="":INPUT T$
160 FORK=2TO 10
170 X$(J,K-1)=MID$(T$,K,1)
180 NEXTK,J
190 REM VISUALIZZA POSIZIONE CORRENTE
200 PRINT" [SHIFT] e [CLR HOME] [CRSR ↓]
    3 volte 3 spazi [C=] e [A] [SHIFT]
    e [*] 9 volte [C=] e [S]"
210 FORJ=1 TO 9
220 PRINT" 3 spazi [SHIFT] e [-]";
230 FORK=1TO9:PRINTX$(J,K);:NEXTK
240 PRINT" [SHIFT] e [-]"
250 NEXTJ
260 PRINT" 3 spazi [C=] e [Z]
[SHIFT] e [*] 9 volte [C=] e [X]"
270 FOR J=2 TO 8
280 FOR K=2 TO 8
290 T=0
300 IFX$(J-1,K-1)="★"THEN T=T+1
310 IFX$(J-1,K)="★"THEN T=T+1
320 IFX$(J-1,K+1)="★"THEN T=T+1
330 IFX$(J,K-1)="★"THEN T=T+1
340 IFX$(J,K+1)="★"THEN T=T+1
350 IFX$(J+1,K-1)="★"THEN T=T+1
360 IFX$(J+1,K)="★"THEN T=T+1
370 IFX$(J+1,K+1)="★"THEN T=T+1
380 Y$(J,K)=X$(J,K)
390 IF T<2 OR T>3THEN Y$(J,K)="  "
400 IF T=3THENY$(J,K)="★"
410 NEXT K,J
420 FORJ=2 TO 8
430 FORK=2 TO 8
440 X$(J,K)=Y$(J,K)
450 NEXT K,J
460 GOTO190
```

UNITA': 23

Esperimento 23.1B

```
 10 INPUT"PEEK (TROVA) INDIRIZZO";J
 20 FOR K=0 TO 7
 30 X1=PEEK(J+K)
 40 GOSUB 1000
 50 NEXT K
 60 PRINT:PRINT
 70 GOTO 10
1000 REM DATA UNA LOCAZIONE IN X1
     CALCOLA CORRISPONDENTE PROFILO
     BINARIO
1010 YY=256:FOR KK=1 TO 8
1020 YY=YY/2
1030 IFX1>=YY THEN X1=X1-YY:PRINT"★";:
     GOTO 1050
1040 PRINT" ";
1050 NEXTKK
1060 PRINT:RETURN
```

UNITA': 24

Esperimento 24.1

```
 10 PRINT" [SHIFT] e [CLR HOME] ";
 20 X=11:Y=11
 30 POKE 7680+22★Y+X,160
 40 POKE 38400+22★Y+X,0
 50 GET A$:IF A$=" "THEN50
 60 IF ASC(A$)<> 133 THEN 80
 70 IF Y>0THEN Y=Y-1:GOTO30
 80 IFASC(A$)<>134 THEN 100
 90 IF X<21 THEN X=X+1:GOTO30
100 IF ASC(A$)<>135THEN120
110 IF Y<22THEN Y=Y+1:GOTO30
120 IF ASC(A$)<>136THEN GOTO50
130 IF X>0THEN X=X-1:GOTO 30
140 GOTO50
```

Esperimento 24.2A

```
 10 DEF FNA(X)=X↑3+(X+7)↑2-100
 50 FOR J=2TO3STEP0.1
 60 PRINT J;FNA(J)
 70 NEXT J
 80 STOP
100 REM MIGLIOR STIMA PER SOLUZIONE
    = CIRCA 2.33
```

Esperimento 24.2B

```
10 DEF FNA(X)=X↑3+(X+7)↑2-100
20 DEF FNB(X)=3★X↑2+2★(X+7)
30 X=2
40 Y=FNA(X)
50 PRINT "X=";X
60 PRINT"Y=";Y
70 IF ABS(Y)>0.0000001 THEN X=X-Y/FNB
   (X):GOTO 40
80 PRINT"SOLUZIONE=";X
90 STOP
```

Esperimento 24.3A

```
 10 INPUT"NOME FILE";F$
 20 OPEN1,1,2,F$
 30 PRINT
 40 PRINT"BATTI FRASE E TERMINALA CON
    PUNTO"
 50 PRINT"NON SUPERARE 75 CARATTERI"
 60 PRINT"ULTIMA FASE DEVE ESSERE ZZ."
 70 S$=""
 80 GETA$:IFA$=""THEN 80
 90 IF ASC(A$)=20THENPRINT" [SHIFT] e
    [CRSR] ";:GOTO80
100 PRINTA$;:S$=S$+A$
110 IF A$<>"." AND LEN(S$)<=75 THEN 80
120 PRINT#1,S$
130 IFLEFT$(S$,2) <>"ZZ"THEN 30
140 CLOSE1
150 STOP
```

Esperimento 24.3B

```
 10 DIM J(26,5)
 20 FORQ=1TO26
 30 FORR=1TO5
 40 READJ(Q,R)
 50 NEXTR,Q
100 DATA1,3,0,0,0
110 DATA3,1,1,1,0
120 DATA3,1,3,1,0
130 DATA3,1,1,0,0
140 DATA1,0,0,0,0
150 DATA1,1,3,1,0
160 DATA3,3,1,0,0
170 DATA1,1,1,1,0
180 DATA1,1,0,0,0
190 DATA1,3,3,3,0
200 DATA3,1,3,0,0
210 DATA1,3,1,1,0
220 DATA3,3,0,0,0
230 DATA3,1,0,0,0
240 DATA3,3,3,0,0
250 DATA1,3,3,1,0
260 DATA3,3,1,3,0
270 DATA1,3,1,0,0
280 DATA1,1,1,0,0
290 DATA3,0,0,0,0
300 DATA1,1,3,0,0
310 DATA1,1,1,3,0
320 DATA1,3,3,0,0
330 DATA3,1,1,3,0
340 DATA3,1,3,3,0
350 DATA3,3,1,1,0
500 INPUT"NOME FILE";Q$
510 OPEN 1,1,0,Q$
520 INPUT"VELOCITÀ";R
530 INPUT # 1,Z$
535 IF LEN(Z$)=0THEN530
540 IF LEFT$(Z$,2)="ZZ" THEN 650
550 IF ST<>0 THEN 800
560 POKE36876,240
570 FOR W=1 TOLEN(Z$)
580 A1$=MID$(Z$,W,1)
590 GOSUB2000
600 NEXTW
610 GOTO 530
650 PRINT"FINE DEL TESTO"
660 STOP
800 PRINT"ERRORE DEL NASTRO"
810 STOP
1000 REM CREA PIP DURATA T1 VOL V1
1010 POKE 36878,V1
1020 FOR TT=1 TO T1:NEXTTT
1030 RETURN
2000 REM TRASMETTI CARATTERE IN A1$
2010 AA=ASC(A1$)
2020 IF AA=32 THEN T1=4★R:GOSUB1000:
     RETURN
2030 IFAA<65ORAA>90THEN RETURN
2040 AA=AA-64:PP=1
2050 IF J(AA,PP)=0 THEN T1=2★R:GOSUB
     1000:RETURN
2060 T1=J(AA,PP)★R:V1=15:GOSUB 1000
2070 T1=R:V1=0:GOSUB1000:PP=PP+1:
     GOTO2050
```

Esperimento 24.4

```
 10 DATAEDINBURGH,GLASGOW,DUNDEE,
    ABERDEEN
 20 DATAFOOTBALL,TENNIS,HILLWALKING,
    OPERA,JAZZ,ROCK,THEATRE,READING,
    POLITICS
 30 DATACHESS,GAMBLING,HORSERACING,
    CARS,MOTORBIKES,CYCLING,MEETING
    PEOPLE
 40 DATACONSERVATIVE,LABOUR,LIBERAL,
    SDP,OTHER,NONE
 60 DIMP$(26)
 70 FORJ=1TO26:READP$(J):NEXT
 80 PRINT" [SHIFT] e [CLR HOME] COMPUTER
    DATING"
 90 PRINT"WHAT TOWN?"
100 A1=1:B1=4:GOSUB1000:XT$=P1$
110 PRINT" [SHIFT] e [CLR HOME] WHAT IS
    YOUR MAIN"
120 PRINT"INTEREST?"
130 A1=5:B1=20:GOSUB1000:XH$=P1$
```

```
140 PRINT" [SHIFT] e [CLR HOME] WHAT IS
    YOUR SECOND"
150 PRINT"INTEREST?"
160 A1=5:B1=20:GOSUB1000:X1$=P1$
170 PRINT" [SHIFT] e [CLR HOME] WHAT ARE
    YOUR"
180 PRINT"POLITICS?"
190 A1=21:B1=26:GOSUB1000:XP$=P1$
200 PRINT"ARE YOU MALE OR FE-"
210 PRINT"MALE (SAY M OR F)
220 INPUTXS$
230 IF XS$<>"M" AND XS$<>"F"THEN 200
240 INPUT"AGE";XA
250 INPUT"HEIGHT IN INCHES";XH
260 OPEN1,1,0,"COMPUTER DATES"
265 M=-100:PRINT" [SHIFT] e [CLR HOME] ";
270 INPUT#1,C$,A$,T$,S$,A,H,H1$,H2$,PO$
280 IFST<>0 THEN 700
290 IFT$<>XT$ORS$=XS$THEN 270
320 S=0
330 D=XA-A:IFXS$="F"THEND=-D:REM
    D=MALE'S AGE — FEMALE'S AGE
340 IFD>=0 AND D<=4 THEN S=S+5
350 D=XH-H:IFXS$="F"THEND=-D
360 IFD>=1 AND D<=3 THEN S=S+3
370 IFXH$=H1$ORXH$=H2$THENS=S+6
380 IFXI$=H1$ORXI$=H2$THENS=S+6
390 IFXP$=P0$THENS=S+4
400 IFXP$="CONSERVATIVE"ANDPO$=
    "LABOUR" THEN S=S-2
410 IFPO$="CONSERVATIVE"ANDXP$=
    "LABOUR" THEN S=S-2
430 IFS>=MTHENM=S:PRINT" [SHIFT] e
    [CLR HOME] BEST SOLUTION SO FAR":GOSUB
    1100:PRINT"SCORE=";S
440 GOTO270
700 STOP
999 STOP
1000 REM DISPLAY MENU A1 TO B1 OF P$.
     SELECT A WORD AND RETURN IN P1$
1010 JJ=1
1020 FORKK=A1TOB1
1030 PRINTJJ;">";P$(KK):JJ=JJ+1
1040 NEXTKK
1045 PRINT
1050 INPUT"CHOOSE A NUMBER";LL:LL=INT
     (LL)
1060 IF LL<1 OR LL>B1-A1+1THEN 1010
1070 P1$=P$(LL+A1-1):RETURN
1100 REM DISPLAY PERSON
1110 PRINT"NAME:";C$
1120 PRINT"ADDRESS:";A$
1130 PRINT"TOWN:";T$
1140 PRINT"AGE:";A
1150 PRINT"HEIGHT:";H
1160 PRINT"HOBBIES:";H1$
1170 PRINT"          ";H2$
1180 PRINT"POLITICS:";PO$
1190 RETURN
2000 OPEN1,1,0
2010 GET#1,A$
2020 PRINTA$
2030 GOTO2010
```

INDICE ALFABETICO

Commodore Home Computer Division

675 Ajax Avenue, Slough Trading Estate,
Slough, Berks. SL1 4BG England

commodore COMPUTER

2502 - 1/84 - 5M