I DUE BASIC DEL PC 128

Continua l'analisi del Basic del PC 128
7ª Parte

Continuiamo il nostro viaggio all'interno dell' intricato mondo del Basic del PC 128, nella speranza che ciò serva a fare chiarezza sulla maggior parte dei comandi in esso implementati.

Anche a costo di essere ripetitivi consigliamo vivamente di trascrivere e provare i piccoli listati da noi posti quali esempi: solo attraverso un paziente e costante esercizio si possono ottenere dei buoni risultati.

Se vi doveste trovare in difficoltà scrivete alla nostra redazione i vostri dubbi; cercheremo di rispondervi in modo esauriente.

END

TIPO: Istruzione

SINTASSI: END

COMMENTO:

Determina la conclusione dell'esecuzione di un programma.

Il comando END si può introdurre in qualsiasi parte di un programma, generalmente viene posto dopo un controllo, così da porre fine al programma in caso di necessità.

Come risultato, END non provoca l'azzeramento delle variabili in memoria, nè modifica le operazioni di CLEAR.

EOF

TIPO: Funzione

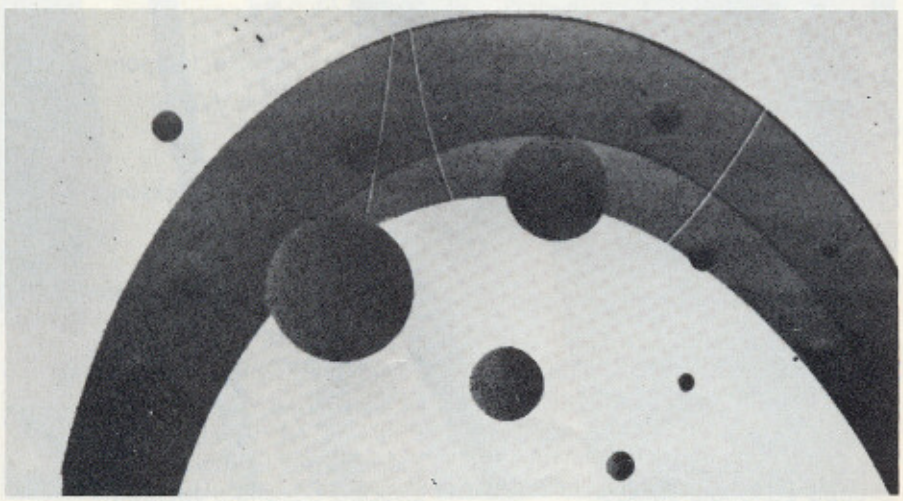
SINTASSI: EOF(n.canale)

COMMENTO:

Dove n.canale, è il numero di canale attualmente aperto.

Per mezzo di questo comando, è possibile testare la fine di un file sequenziale, infatti esso ritorna il valore -1, se la fine del file è stata raggiunta, altrimenti ritorna il valore 0 se, viceversa, tale fine non è stata raggiunta.

```
10 OPEN"I",#1,"FILE1"
20 DO
30 IF EOF(1) THEN EXIT
40 INPUT#1,A:PRINT A
60 LOOP
70 END
```



ERL

TIPO: Funzione

SINTASSI: ERL

COMMENTO:

ERL è una variabile riservata dal computer, utilizzata per contenere il numero di riga in cui si è verificato un errore. Solitamente questa variabile è associata ad un'altra variabile riservata: ERR. Quest'ultima ritorna il numero dell'errore, il quale può essere rintracciato nell'apposita tabella degli errori.

Una variabile riservata, è tale perché un qualsiasi suo uso al di fuori del campo specificato, genera un errore.

Nel caso in cui in un nostro programma si avvera una condizione d'errore, se questa possibilità non è stata contemplata nel programma stesso, l'esecuzione avrà termine e comparirà sullo schermo il messaggio d'errore corrispondente.

Nel caso in cui il programma preveda la possibilità di un errore, e ciò

è possibile per mezzo dell'apposito comando ON ERROR, al sopraggiungere dell'errore questo verrà trattato da un apposita sezione di programma, la quale sarà informata del tipo d'errore e dal "luogo" in cui si è manifestato, rispettivamente dalle variabili ERR e ERL.

Per dare un'idea di come possa essere strutturato un programma, contenente una sezione dedicata al trattamento degli errori, di seguito riportiamo una bozza di programma che di per sè non è funzionante, ma in compenso ben mostra il tipo di struttura adottata:

```
10 ON ERROR GOTO 10000
.
.
.
10000 PRINT"Errore ";ERR;" alla
      riga ";ERL
10010 IF ERR=1 THEN PRINT "Next
      senza For"
10020 IF ERR=2 THEN PRINT
      "Errore di Sintassi"
10030 IF ERR=3 THEN PRINT
      "RETURN senza GOSUB"
10040 IF ERR=4 THEN PRINT
      "DATA esauriti"
.
.
.
11000 INPUT"Numero di linea da
      cui ricominciare.";NL
11010 RESUME NL
```

ERR

TIPO: Istruzione

SINTASSI: ERR

COMMENTO:

ERR è una variabile riservata dal computer, utilizzata per contenere il numero del tipo d'errore occorso nella linea di un programma individuata dalla variabile ERL, la cui corrispondenza è riportata nell'apposita tabella in appendice.

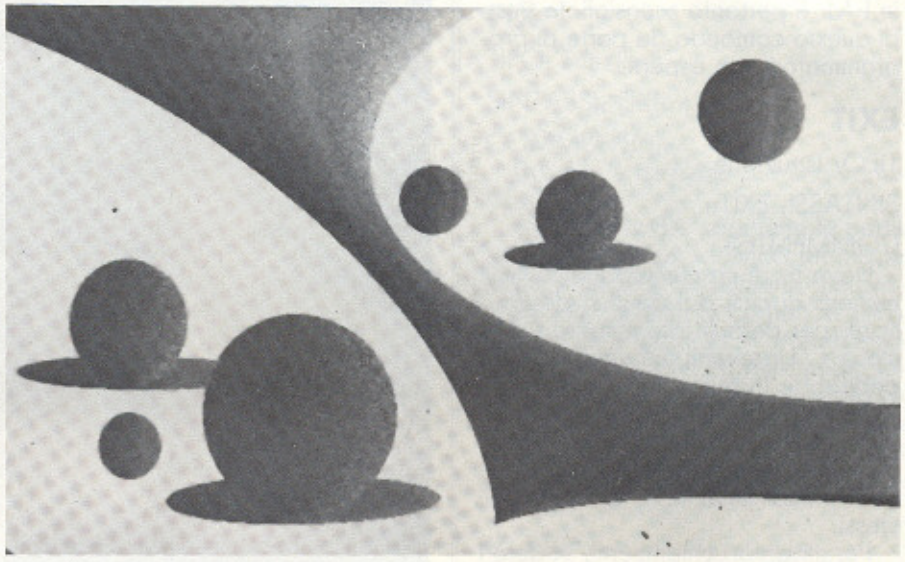
ERROR

TIPO: Istruzione

SINTASSI: ERRORn.

COMMENTO:

Dove numero è un numero, o una variabile numerica, corrispondente



ad un errore codificato nell'apposita tabella.

Per mezzo del comando ERROR, quando il programma lo incontra nel suo svolgimento, è possibile simulare un errore del tipo riportato in argomento. Provocando in tale modo un messaggio d'errore da parte del computer.

Se tale comando viene invece usato assieme a ON ERROR, per esempio per la messa a punto di una sezione di programma atta alla gestione degli errori, il computer si comporterà come se si fosse realmente verificato un errore. In tal modo sarà possibile verificare l'efficacia della sezione di controllo errori da noi studiata.

```
10 A=7
20 T=11
30 ERROR A+B
```

Dopo aver impartito, in modo diretto, il comando RUN, apparirà la scritta relativa all'"errore n.18: "Undefined User Function".

EVAL

TIPO: Funzione

SINTASSI: EVAL(stringa di caratteri)

COMMENTO:

Dove stringa di caratteri, è un'espressione codificata dalla funzione CRUNCH$.

Se l'espressione considerata non è stata codificata precedentemente dal comando CRUNCH$, il computer ritorna un messaggio d'errore "Syntax Error".

Per ulteriori informazioni, vedi il comando CRUNCH$.

EXEC

TIPO: Istruzione

SINTASSI: EXEC indirizzo, elenco parametri

COMMENTO:

Dove indirizzo è un indirizzo di memoria individuante un modulo di programma in L.M., ed elenco parametri può essere un elenco di parametri da passare al modulo in L.M., e separati da virgole.

Permette di lanciare un modulo in L.M. che si trova in memoria all'indirizzo in argomento. Nel caso in cui l'indirizzo venisse omesso, l'indirizzo considerato sarebbe quello dell'ultimo comando EXEC, oppure quello definito dall'ultimo comando LOADM. Se invece l'indirizzo è compreso tra &H6000 e &H9FFF, il banco di memoria indirizzato sarà quello definito dal comando BANK, se utilizzato, oppure, per default, il banco di numero più elevato.

Il modulo in L.M. preso in considerazione, dovrà terminare con un'istruzione del tipo RTS, al fine di provocare poi il ritorno al Basic.

È comunque da tenere presente che il comando EXEC, per essere usato correttamente, richiede da parte dell'utilizzatore una notevole conoscenza della programmazione

in L.M. è pertanto sconsigliato l'uso di questo comando da parte di programmatori non esperti.

EXIT

TIPO: Istruzione

SINTASSI: EXITn.

COMMENTO:

Dove n., è un numero indicante il numero di cicli dai quali si deve uscire, per default tale numero è posto a 1. Un eventuale valore 0, impedirebbe l'uscita.

Il comando EXIT, usato all'interno dei cicli DO...LOOP e FOR...NEXT, è utilizzato per interromperli e saltare alla prima istruzione esterna agli stessi.

Per maggiori chiarimenti, vedere i comandi DO...LOOP e FOR...NEXT.

```
10 DO
20 CLS
30 INPUT"Immetti una parola.";A$
40 IF A$="PC 128" THEN EXIT
50 CLS
60 PRINT"Non è questa!!"
70 FOR A=0 TO 1000: NEXT A
80 LOOP
90 CLS
100 PRINT"BRAVO HAI VINTO!!"
110 END
```

EXP

TIPO: Funzione

SINTASSI: EXP(n.)

COMMENTO:

Dove n. è un numero o una variabile numerica.

La funzione EXP ritorna un numero dato in doppia precisione, che è il risultato dell'operazione di elevazione alla potenza e n., dove e vale 2.712828.

Si ha un errore di Owerflow, quando il risultato supera il valore di 58.02969.

```
10 CLS
20 A=2.6
30 PRINT EXP(A)
```

FIELD

TIPO: Istruzione

SINTASSI: FIELD # n.canale,lungh.1 AS var.camp.1,lungh.2 AS var.camp.2...

COMMENTO:

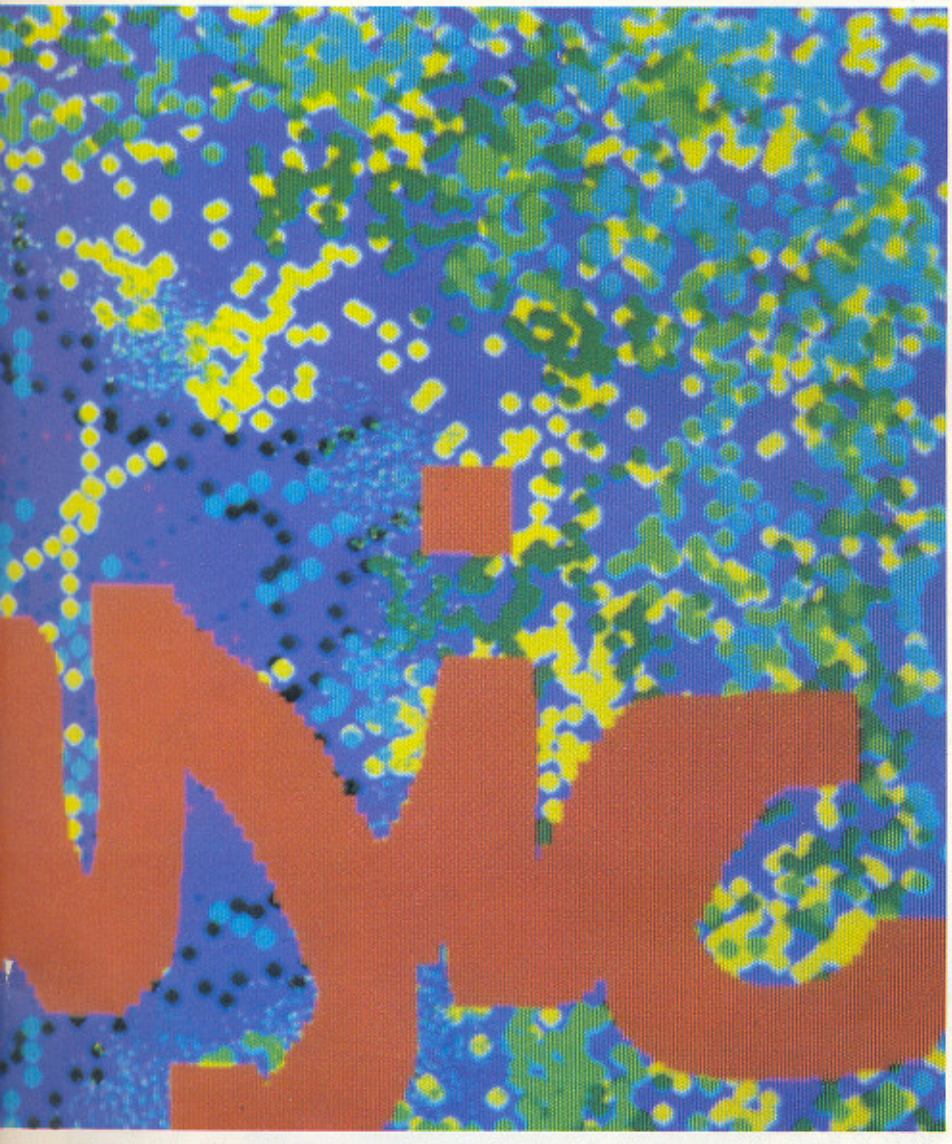
Dove # n. è il numero di canale aperto; lungh.1 è la lunghezza in caratteri dei campi specificati e var.camp.1 è la variabile stringa che designa il campo.

L'istruzione FIELD definisce l'organizzazione, in campi, di record di un file ad accesso diretto del quale è specificato il canale.

La somma delle lunghezze non deve essere superiore alla lunghezza del file: quest'ultima è stata fissata al momento dell'apertura per mezzo del comando OPEN. Se ciò non è stato fatto, i valori di default del file sono rispettivamente di 255 caratteri, per i dischetti a doppia densità, e 128 per quelli a densità singola.

Di solito l'istruzione FIELD viene usata una sola volta, dopo l'apertura del file mediante OPEN, ma ciò non è restrittivo, infatti FIELD può essere utilizzata diverse volte, al fine di permettere la lettura, o la scrittura, di file con formati diversi.

Il contenuto delle variabili di campo è sempre utilizzabile per essere visualizzato, trasformato o assegnato a un'altra variabile. Per contro, si può depositarvi un valore solo tramite i comandi LSET, RSET e MID$(). Le variabili in questione, possono ricevere dei valori dopo che questi sono stati convertiti dalle funzioni MKI$(numeri interi), MKS$(numeri



reali a precisione singola) o MKD$(numeri a precisione doppia).

FIELD #1,18 AS NAME$,5 AS NAME1$

FILES

TIPO: Istruzione

SINTASSI: FILESn.canali,lungh.,estensione area prot.

COMMENTO:

Dove n.canali è il numero di canali simultanei utilizzabili, tale numero non deve essere superiore a 15; lungh. equivale alla somma delle lunghezze dei vari record dei file ad accesso diretto, aperti simultaneamente. Tale lunghezza è fissata all'inizializzazione al valore di 256 byte, cioè due file con record di 128 byte.

L'argomento estensione area prot., riserva in memoria una zona tampone di n tracce, dove n può variare da 0 a 25, utilizzate per ottimizzare gli accessi fisici al QDD o al dischetto. I valori di default sono pari a 3 per un QDD e a 0 per un dischetto.

Secondo quanto detto, per aprire tre file ad accesso diretto le cui lunghezze dei record sono di 30, 40, 50 caratteri, occorre riservare una lunghezza corrispondente ad almeno 120 caratteri.

FIX

TIPO: Funzione

SINTASSI: FIX(n.)

COMMENTO:

Dove n. è un numero o una variabile.

Il comando FIX, sopprime la parte decimale, e così facendo ritorna la parte intera di un numero. Equivalente nei risultati al comando INT, per quanto riguarda il trattamento dei numeri positivi, si differenzia invece da quest'ultimo per quanto riguarda il trattamento dei numeri negativi.

A tal proposito osservate attentamente gli esempi:

```
10 CLS
20 A=15.12345 : B=-15.12345
30 PRINT INT(A),
40 PRINT FIX(A)
50 PRINT
60 PRINT INT(B),
70 PRINT FIX(B)
80 END
```

FKEY$

TIPO: Funzione

SINTASSI: FKEY$(n.)

COMMENTO:

Dove n. è un numero o una variabile, corrispondente al numero di un tasto funzione, e deve essere compreso tra 1 e 10.

Questo comando ritorna il numero di codice del tasto funzione corrispondente al numero riportato in argomento.

```
10 CLS
20 FOR A=1 TO 10
30 PRINT FKEY$(A)
40 PRINT
50 NEXT A
60 END
```

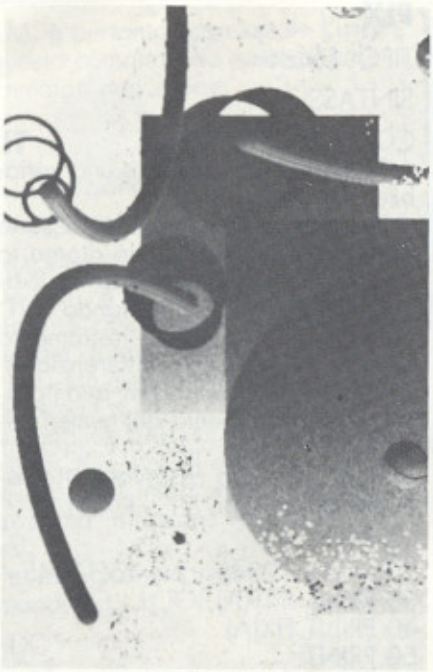
FOR...NEXT

TIPO: Controllo

SINTASSI: FOR var.1=n.1 TO n.2 STEP n.3 ... NEXT var.1

COMMENTO:

Dove var.1 è una variabile numerica decimale che viene usata come contatore e chiamata variabile di controllo; n.1 è un numero o una



variabile indicante il valore di partenza della var.1; n.2 è il valore d'arrivo della variabile di controllo e n.3 è il valore d'incremento desiderato. Se l'istruzione supplementare STEP è omessa, il valore di default dell'incremento è pari a 1.

È interessante notare che il valore d'incremento può essere anche negativo, in tal caso pero, il valore di n.1 deve essere maggiore a n.2.

Il comando NEXT determina il rinvio alla linea contenente l'istruzione FOR o la fine del ciclo di cui ha in argomento la variabile di controllo. Ciò vuol dire, che mentre il comando FOR...STEP incrementa la variabile di controllo e poi passa al programma l'esecuzione delle linee contenute fra la riga FOR e la riga NEXT, il comando NEXT valuta quante volte il ciclo è stato ripetuto e nel caso sia inferiore al valore di n2., ridà il controllo alla linea contenete il comando di incremento, nel caso contrario, termina il ciclo e ritorna il comando alla linea successiva a quella contenente il comando NEXT stesso.

Un altro modo consentito, per uscire dal ciclo è l'uso del comando EXIT, per il cui uso si rimanda alla spiegazione del comando stesso.

Per comprendere meglio questo meccanismo, abbozziamo uno schema di percorso:

- Per mezzo del comando FOR il contatore viene inizializzato, memorizzati l'incremento e il valore finale della variabile di controllo.
- Tutta la parte di programma, racchiusa tra la riga contenente l'istruzione FOR e la riga contenente l'istruzione NEXT, viene eseguita.
- Raggiunta l'istruzione NEXT, il valore della variabile di controllo viene incrementato, e il valore ottenuto confrontato con il valore finale della variabile di controllo contenuto in moria.
- Se il valore raggiunto è inferiore al valore finale, il programma ritorna alla istruzione immediatamente seguente ilcomando FOR, così da ricominciare un altro ciclo.
- Se invece il valore raggiunto supera il valore finale della variabile di controllo, il programma ricomincia dalla riga successiva a quella contenente l'istruzione NEXT.

Da quanto detto, si può ricavare che un ciclo FOR...NEXT viene eseguito almeno una volta, infatti il test di controllo viene effettuato alla riga contenente l'istruzione NEXT, appunto alla fine del ciclo. Una seconda osservazione da farsi è che il valore finale non viene sempre raggiunto, infatti il valore dell'incremento può essere tale da fare in modo che il valore di arrivo venga superato.

I cicli FOR...NEXT possono esere nidificati, ciò vuol dire che possono esserci diversi cicli uno dentro l'altro come nell'esempio:

```
10 CLS
20 FOR A=0 TO 10
30 FOR B=0 TO 5
40 PRINT A,B
50 NEXT B
60 NEXT A
70 END
```

Nel listato appena visto, si può immediatamente notare una regola fondamentele dei cicli nidificati: questi devono essere chiusi, partendo dall'ultimo ciclo aperto. Infatti nella riga 50 c'è NEXT B e nella riga 60 c'è NEXT A. Per vedere cosa accade provate pure a scrivere:

```
10 CLS
20 FOR A=0 TO 10
30 FOR B=0 TO 5
40 PRINT A,B
50 NEXT A
60 NEXT B
70 END
```

Come avete potuto constatare, il computer vi risponde con una frase d'errore del tipo "Next Without For in 60".

C'è da tener presente che non esistono limiti al numero di cicli nidificabili.

I comandi FOR e NEXT sono usati moltissimo nella stesura di un programma in Basic, e questo perché la loro utilità non è limitata ad un solo uso specifico, ma bensì a diversi campi applicativi, anche se la loro caratteristica fondamentale è di essere un contatore.

```
10 CLS
20 PRINT"Premere10 numeri, op pu-
   re 0 per fermare";
30 FOR I=1 TO 10
40 PRINT"NUMERO";I;
50 INPUT N
60 IF N=0 THEN 100
70 NEXT I
80 PRINT"Completo"
90 END
100 PRINT"Interrotto"
110 END
```

FRE

TIPO: Funzione

SINTASSI: FRE(n.)

COMMENTO:

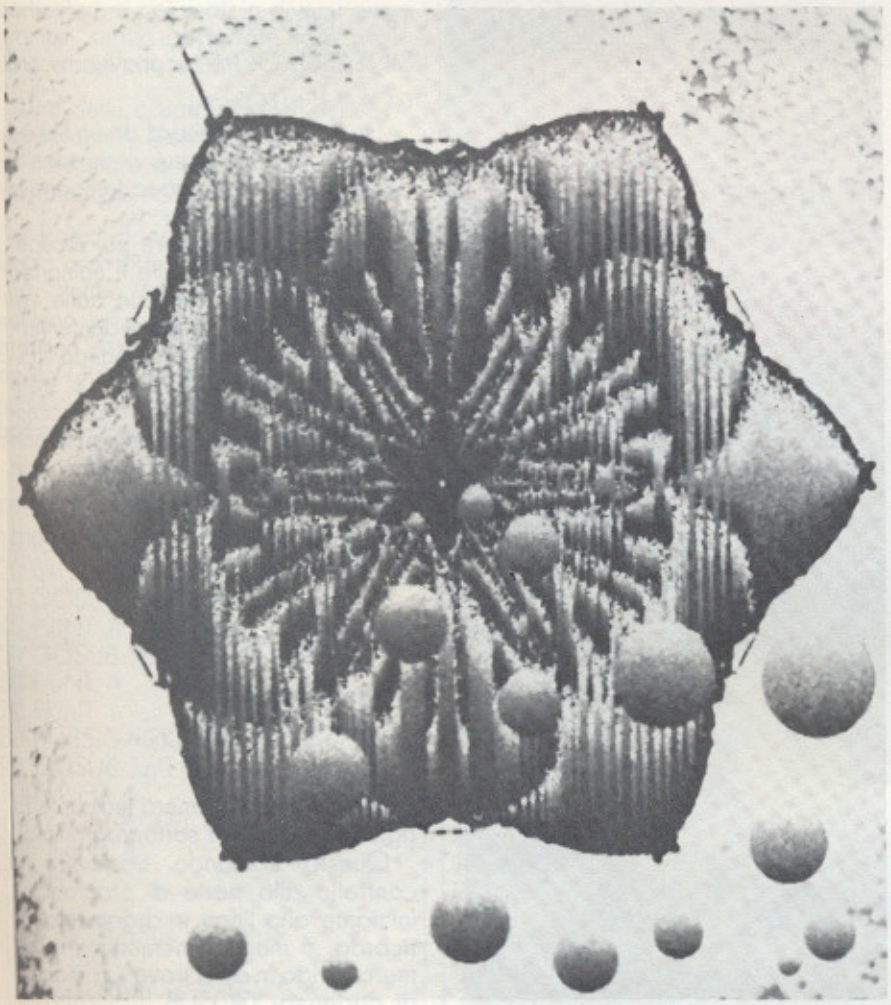
Dove n., se è un numero, è compreso fra 0 e 2, oppure è una variabile stringa di nome A$.

La funzione FRE, permette di vedere quanta memoria è ancora a disposizione dell'utente, più esattamente, al variare di n. ritorna i valori relativi alle seguenti aree di memoria:

n.=0	Spazio totale in memoria
n.=1	Ritorna il valore corrispondente all'area di lavoro dell'interprete e cioè da &H6100 a &H9FFF.
n.=2	Ritorna il valore corrispondente all'area dedicata al programma e ai dati e cioè da &HA000 a &HDFFF.
n.=A$	Ritorna il valore corrispondente all'area dedicata alle stringhe: tale spazio, per default è pari a 300 byte.

```
10 CLS
20 PRINT FRE(0); FRE(1); FRE(2);
   FRE(A$)
30 Z$="PROVA OCCUPAZIONE
   DI MEMORIA"
40 PRINT FRE(0); FRE(1);FRE(2);
   FRE(A$)50 END
```

Come si può vedere chiaramente dai risultati del programma, e come daltronde è logico, qualsiasi cosa si faccia fare al computer, questa gli sottrae memoria. Nel caso del testo, è interessante notare come l'area di memoria si riempia contando esattamente i caratteri, compresi gli spazi (ultime cifre sulla destra).



FWD

TIPO: Istruzione

SINTASSI: FWDn.

COMMENTO:

Dove n. è un numero o una variabile numerica, compresa tra -255 e 255.

Per mezzo di questo comando è possibile spostare la tartaruga di un numero di punti pari all'argomento.

Il numero negativo implica un arretramento della tartaruga, mentre un numero positivo ne comporta un avvanzamento.

Tutte le modifiche inerenti la tartaruga, quali: rotazioni, direzioni e dimensioni, vengono prese in considerazione prima che lo spostamento stesso abbia luogo.

Se in precedenza non è stato impartito il comando TURTLE, il computer ritornerà un messaggio d'errore del tipo "Illegal Function Call". FWD 45 Sposta la tartaruga di 45 punti nella direzione indicata.

GET

TIPO: Istruzione

SINTASSI: GET (cl.1,rg.1)-(cl.2,rg.2), elemento matrice

COMMENTO:

Dove (cl.1,rg.1) e (cl.2,rg.2) sono, rispettivamente, l'angolo superiore sinistro e l'angolo superiore destro di un rettangolo delimitante un'area video, in cui cl. e rg. possono essere dei numero o delle variabili numeriche intere, rappresentanti il numero di colonna e il numero di riga. È bene sottolineare, che le coordinate dei vertici sono espresse in caratteri, e devono essere comprese: per le colonne da 0 a 39 (oppure79) per le righe da 0 a 24. Se per caso il secondo vertice non viene definito, il computer lo considera per default in posizione (39,24), oppure (79,24) se ci si trova in 80 colonne. Come si può vedere queste coordinate corrispondono al vertice inferiore destro dell'area video.

La matrice numerica, di cui fa parte l'elemento di matrice, dev'essere di tipo intero e dev'essere dichiarata prima dell'uso del comando GET. In una matrice possono essere memorizzate più immagini, queste, dopo essere state compattate, vengono memorizzate nella matrice, partendo dal numero di indice più basso.

Il comando GET, viene usato per memorizzare una parte di schermo, delimitata da un rettangolo, in una matrice numerica.

L'elemento di matrice usato in questa istruzione, deve essere l'elemento di indice più alto, rispetto a questa immagine. Infatti la memorizzazione, avverrà dall'indice appena inferiore a quello designato: pertanto si può dire che l'elemento di matrice contiene il valore dell'indice del primo elemento libero della matrice. Ciò potrà essere usato per una nuova istruzione GET o LOADP che utilizzi la stessa matrice.

All'atto del dimensionamento della matrice, si deve fare attenzione che questa sia sufficientemente grande da contenere i disegni che ci servono, altrimenti il computer protesterà tramite un comando del tipo "Out of Memory"

Dopo che la porzione di schermo

è stata compattata e memorizzata, può essere ristampata in qualsiasi parte dello schermo, grazie al comando PUT, oppure può essere memorizzato in un file, mediante il comando SAVEP.

```
10 CLS
20 DIM A%(100)
30 PRINT"Prova 1"
40 GET (0,0)-(7,1),A%(100)
50 PUT(15,20),A%(100)
60 PRINT A%(100)
70 END
```

Dove la linea 30 stampa in alto a sinistra la stringa in argomento al comando PRINT; la linea 40 la memorizza nella variabile intera A% e la linea 50 la stampa in un altro punto dello schermo. La linea 60 stampa sullo schermo il valore del primo elemento libero della matrice A%(), questo valore può essere utilizzato, per esempio, per effettuare un'altra memorizzazione, del tipo:

```
GET(a,b)-(c,d),A%(61)
```

Per vedere una semplice applicazione pratica, una scritta con scrolling orizzontale, provate il programma seguente, molto simile al precedente:

```
10 CLS
20 DIM A%(130)
30 PRINT "Prova 2"
40 GET(0,0)-(7,1),A%(130)
50 GET(10,10)-(17,11),A%(91)
60 CLS
70 FOR X=0 TO 40
80 PUT(X,20),A%(130)
90 FOR Q=0 TO 10: NEXT Q
100 PUT(X-1,20),A%(91)
110 NEXT X
120 GOTO 70
```

In riga 40, come nell'esempio precedente, viene memorizzata nella matrice A%(), la stringa "Prova 2", stampata dalla riga 30, ma il bello viene alla riga 50. In riga 50, noi memorizziamo una porzione di video vuoto, e ciò per poterlo utilizzare come una mascherina per coprire, e quindi cancellare, la scritta appena stampata dalla riga 80; vedi riga 100.

GET#

TIPO: Istruzione

SINTASSI: GET # n.canale,n.record

COMMENTO:

Trasferisce un record da un file ad accesso diretto, di cui viene specificato il canale, alla zona tampone in memoria centrale.

Il record, può essere poi recuperato e utilizzato tramite il comando INPUT # , o direttamente dalle variabili di campo, se precedentemente è stato eseguito un comando FIELD. Se il numero di record è stato omesso, GET legge il record successivo del file, oppure il primo, se non è stato ancora letto o scritto alcun record.

GET # 2,10 Legge il decimo record del file n.2

GOSUB

TIPO: Istruzione

SINTASSI: GOSUBn.r

COMMENTO:

Dove n.r è un numero indicante la riga d'inizio di un sottoprogramma.

Questo comando trasferisce il controllo alla parte di programma iniziante alla linea in argomento, e ricorda, o meglio, memorizza il numero di riga in cui si trova il comando di partenza: GOSUB. Il numero di linea memorizzato, viene utilizzato dall'istruzione RETURN, la quale permette il ripristino del programma originale, a partire dalla riga immediatamente successiva la riga memorizzata.

Un sottoprogramma, è un pezzo di programma che esegue un ben determinato numero di operazioni, e termina con un comando RETURN (attenzione a non confonderlo con il tasto RETURN della tastiera).

La divisione di un programma in tanti sottoprogrammi, permette la creazione di connessioni logiche fra i vari blocchi, molto più trasparenti e intelleggibili di quelle ottenibili con un'altra istruzione di salto incondizionato: il comando GOTO, di cui parleremo in seguito.

Un altro lato positivo dei sottoprogrammi, è che questi possono essere creati al fine di risolvere dei problemi particolari, e poi usati in diversi programmi, senza apportare loro nessuna modifica.

Una delle caratteristiche dei sottoprogrammi, è che questi si possono richiamare a vicenda, o addirittura possono richiamare sè stessi.

Il comando GOSUB, può dare luogo a un messaggio d'errore del tipo "Return Without GOSUB", ciò avviene se si entra in un sottoprogramma seza essere prima passati da un GOSUB, oppure a "Undefined Line" se il numero di riga in argomento non esiste nel programma.

Gli esempi possibili sono infiniti, pertanto vediamone alcuni:

```
10 CLS
20 B=15: C=30
30 GOSUB 160
40 PRINT A,
50 GOSUB 200
60 PRINT A
70 PRINT
80 B=85:C=27
90 GOSUB 160
100 PRINT A,
110 GOSUB 200
120 PRINT A
130 END
140 '
150 ' MOLTIPLICAZIONE
160 A=B*C
170 RETURN
180 '
190 ' DIVISIONE
200 A=B/C
210 RETURN
```

Come si può vedere da questo piccolo programma, i due sottoprogrammi MOLTIPLICAZIONE e DIVISIONE, vengono utilizzati diverse volte e con valori diversi, senza che per questo sia necessario riscriverli.

Notate che in riga 130 è stata posta l'istruzione END, provate a toglierla e ne capirete immediatamente l'utilità.

Nell'esempio che segue, vi proponiamo un semplice programma di controllo dell'input, attuato proprio per mezzo del comando GOSUB:

```
10 CLS
20 X$="A"
30 PRINT "PREMI UNA ";X$
40 A$=INPUT$(1)
50 IF A$="A" THEN GOTO 80
60 GOSUB 180
70 GOTO 40
80 X$="B"
90 CLS
100 PRINT"PREMERE UNA";X$
110 A$=INPUT$(1)
120 IF A$="B" THEN GOTO 150
130 GOSUB 180
140 GOTO 110
150 CLS
160 PRINT" -BENE- "
170 END
180 PLAY"DOREMIDOMI"
190 CLS
200 PRINT"HO DETTO UNA ";X$
210 RETURN
```

Come qualcuno avrà già notato, in questo secondo programma ci sono molte ripetizioni inutili, soprattutto nella parte riguardante l'input, facilmente evitabili proprio per mezzo del comando GOSUB. Usare un unico comando di input, però, implica la conoscenza di particolari variabili chiamate FLAG che, seppure spiegate in un'altra sezione di questo testo, possono creare qualche problema a chi non ne ha sufficiente pratica. È comunque auspicabile, che molti di voi provino a rifare il programma costruendo un unico sottoprogramma di input.