# I DUE BASIC DEL PC 128

**Proseguiamo con la nostra analisi del Basic relativo al PC 128**

**5° Parte**

Eccoci qui per la quinta volta a parlare del vostro linguaggio «preferito». Allora, come proseguono i vostri viaggi nell'intricato mondo del Basic? Speriamo bene, anzi, ne siamo convinti!

Chissà quali programmi avrete redatti, grazie alle numerose scoperte fatte. Comunque è meglio non indagare troppo a fondo, rimedieremmo sicuramente una figuraccia: sarete ormai diventati talmente abili da farci impallidire!

Ma c'è ancora qualcosa che forse potremmo insegnarvi.

Eccovi quindi l'elenco dei comandi analizzati per voi in questo numero.

## COPY

TIPO: Istruzione

SINTASSI: COPY*nomefile1* TO *nomefile2*

COMMENTO:

Dove *nomefile1* è il nome del file sorgente e *nomefile2* è il nome del file destinazione. Questo formato è necessario se la copia del file viene fatta sullo stesso dischetto.

Per mezzo di questo comando è possibile copiare integralmente un file da un dischetto all'altro, o sul medesimo dischetto. Ciò è possibile, sia con un drive che con due.

COPY"0:FILE1"TO"1:FILE2"

COPY"FILE1"TO"FILE2"

COPY"FILE1"

## COS

TIPO: Funzione matematica

SINTASSI: COS(n.)

COMMENTO:

Dove n. può essere un numero, una funzione o una variabile che rappresenta un angolo espresso in radianti.

Il risultato ottenibile per mezzo del comando COS è il coseno di un angolo.

```
10 CLS
20 SCREEN1,7,0,1
30 LINE(0,100)-(319,100),2
40 LINE(0,50)-(319,50),4
50 LINE(0,150)-(319,150),4
60 FOR A=1 TO 319
70 ANGOLO=A/20
80 PSET(A,100-50*COS
   (ANGOLO)),0
90 NEXT A
100 END
```

## CRUNCH$

TIPO: Comando

SINTASSI: CRUNCH$(stringa)

COMMENTO:

Dove stringa deve essere o una espressione stringa o una espressione numerica.

Il comando CRUNCH permette di codificare dei dati, per esempio ottenuti da un comando di input, e di renderli valutabili dal comando EVAL. Ciò permetterà di ottenere, direttamente, il risultato di un'operazione, sia che essa riguardi delle stringhe che dei numeri.

È opportuno sottolineare che il formato con cui vengono immessi i dati, deve rispettare la corretta sintassi Basic delle funzioni di cui si vuole il risultato.

Per meglio comprendere, vediamo un breve programma in cui utilizzeremo anche il comando EVAL e ciò al fine di mostrare l'interdipendenza dei due comandi.

```
10 CLS
20 SCREEN0,2,8
30 LINE INPUT"Immissione dati:
   ";A$
40 PRINT EVAL(CRUNCH$(A$))
50 ONKEY="C" GOTO10
60 LOCATE9,10
70 COLOR5,,1
80 PRINT"Per continuare pre-
   mere C"
90 PRINT CHR$(20)
100 GOTO100
```

Nel programma sopra riportato, provate ad immettere quanto segue:

RIGHT$("CIAO PIPPO",5)

"CIAO"+" PIPPO"

8*16+(32-3)

La riga 100 crea un loop, bloccando così il programma in attesa della pressione del tasto C, che consente di poter continuare ad usare l'input. Ciò determina un minimo controllo sull'introduzione dei dati, che può essere da voi e-

steso, per esempio, sul controllo dell'uscita dal programma.

La riga 90, serve ad eliminare la fastidiosa presenza del cursore lampeggiante, mentre leggiamo il messaggio del computer.

## CSNG

TIPO: Funzione di conversione

SINTASSI: CSNG(n.)

COMMENTO:

Dove n. può essere sia un numero, sia una espressione numerica, sia una variabile.

Converte un numero qualsiasi in un numero a precisione singola, effettuando un arrotondamento automatico.

```
10 CLS
20 A#=(32.34565120*
45.619199)
30 PRINT CSNG(A#)
40 PRINT A#
50 END
```

## CSRLIN

TIPO: Funzione

SINTASSI: CSRLIN

COMMENTO:

Ritorna il numero di linea in cui si trova il cursore mentre viene eseguita l'istruzione CRSLIN.

```
10 CLS
20 LOCATE32,13
30 PRINT CSRLIN
```

Il programma stamperà sul video il numero 13, che corrisponde al numero di riga, in posizione colonna 32 e riga 13.

## CVD

TIPO: Funzione di conversione

SINTASSI: CVD(variabile stringa)

COMMENTO:

La variabile stringa deve contenere dati ottenuti tramite una conversione, per mezzo dell'istruzione MKD$(). Questa istruzione fa solitamente riferimento ad una variabile di campo precedentemente definita per mezzo dell'istruzione FIELD.

L'istruzione CVD permette la

conversione del contenuto di una variabile alfanumerica in un numero a precisione doppia.

A#=CVD(STRINGA$)

## CVI

TIPO: Funzione di conversione

SINTASSI: CVI(variabile stringa)

COMMENTO:

Analoga al comando precedente, questa istruzione permette la conversione in numero intero, del contenuto della variabile alfanumerica.

A%=CVI(STRINGA$)

## CVS

TIPO: Funzione di conversione

SINTASSI: CVI(variabile stringa)

COMMENTO:

Questa è la terza funzione di questo tipo, e infatti la sua sintassi e i modi sono identici a quelli dei comandi CVD e CVI: per suo mezzo si può convertire una variabile stringa in un numero a precisione singola.

A=CVS(STRINGA$)

## DATA

TIPO: Istruzione di trattamento dati

SINTASSI: DATA val.1,val.2,...

COMMENTO:

Dove val.1, val.2 eccetera, sono un elenco di costanti.

Tramite l'istruzione DATA, possiamo creare un elenco di costanti, sia numeriche che alfanumeriche, di lunghezza indefinita e tramite l'istruzione READ, possiamo poi immetterle in variabili. È opportuno precisare che le variabili utilizzate dall'istruzione READ devono essere dello stesso tipo delle costanti da loro immagazzinate.

Le costanti elencate possono essere di tutti i tipi: intere, reali, a precisione singola, a precisione doppia e alfanumeriche. In tale elenco non possono comparire delle espressioni, e le costanti alfanumeriche contenenti spazi, o caratteri separatori, devono essere racchiuse fra virgolette.

Un altro comando molto utile, alla corretta gestione dei DATA, è l'istruzione RESTORE, che permette la rilettura dei DATA a partire dal numero di riga in argomento. Per verificare l'utilità del comando RESTORE provare i due programmi riportati di seguito:

```
10 CLS
20 DATA32,67,73,65,79,32,80,
   73,80,80,79
30 FOR A=1 TO 11
40 READ B
50 PRINT CHR$(B)
60 NEXT
70 GOTO 30
```

```
10 CLS
20 DATA32,67,73,65,79,32,80,
   73,80,80,79
30 FOR A=1 TO 11
40 READ B
50 PRINT CHR$(B)
60 NEXT
70 RESTORE
80 IF CSRLIN=20 THE END
90 GOTO 30
```

L'istruzione DATA è molto usata per caricare direttamente in memoria dei dati tramite l'istruzione POKE, così da ottenere delle parti di programma che lavorino direttamente in linguaggio macchina anche da Basic. Oltre a questo, la tecnica appena descritta, permette di memorizzare direttamente, in opportune zone di memoria, dei dati che in seguito potranno essere riletti con l'istruzione PEEK.

```
10 CLS
20 20 DATA32,67,73,65,79,32,
   80,73,80,80,79
30 FOR A=0 TO 10
40 READ B
50 POKE15000+A,B
60 NEXT
70 FOR A=0 TO 10
80 C(A)=PEEK(15000+A)
90 NEXT
100 FOR A=0 TO 10
110 PRINT CHR$(C(A));
120 NEXT
130 END
```

Come si può vedere, il programma sopra riportato, non è ottimizzato al meglio, e questo perché sopra tutto esiste una necessità didattica alla quale difficilmente si può far fronte diversamente.

Alla riga 50 il programma carica in memoria, a partire dall'indirizzo 15000, i valori letti dall'istruzione DATA di riga 20, tramite la riga 40. Mentre l'istruzione PEEK(), in riga 80 fa l'operazione inversa, andando a leggere i valori contenuti in memoria dalla locazione specificata in argomento.

In questo programma è altresì interessante notare l'importanza dei cicli FOR NEXT, di cui parleremo in modo più esteso in una prossima rivista, in cui chiariremo in modo inequivocabile tutti i segreti e tutte le applicazioni di tale, oserei definirle, fondamentale istruzione.

## DEF FN

TIPO: Funzione

SINTASSI: DEF FNnome(var1, var2,...)= espressione

COMMENTO:

Dove nome è il nome di una variabile, o numerica o stringa, Basic e pertanto ne deve seguire le regole di formazione; var1, var2 eccetera, sono un elenco di variabili fittizie che si riferiscono all'espressione. Alla destra dell'uguale c'è l'espressione che definisce la funzione.

La funzione stessa è richiamabile tramite l'istruzione FNnome(varA, varB...), dove nome deve essere il nome di una variabile precedentemente definita con DEF, e varA, varB... eccetera, devono essere delle variabili utilizzate per passare i valori alla funzione, divenendo così il vero argomento della funzione.

Solitamente questo comando, erroneamente considerato troppo macchinoso e pertanto poco usato, permette di definire delle funzioni anche molto complesse, che possono essere poi facilmente richiamate attraverso il loro nome e l'immissione dei giusti parametri. La possibilità di far eseguire la

stessa funzione con parametri di volta in volta diversi, per esempio, in un programma di grafica che si basi su delle curve calcolabili matematicamente tramite funzioni, può far risparmiare molti passaggi e risparmiarci pertanto una notevole mole di lavoro.

Nel breve programma che segue abbiamo cercato di mostrare l'uso dell'istruzione DEF FN, sia in campo numerico che in quello alfanumerico. È importante notare l'uso delle variabili.

```
10 CLS
20 I$=" CIAO PIPPO ":
   B$="1234567890"
30 A=15: B=65: C=3: D=6:
   E=4: H=1
40 DEF FN NUM(X,Y)= SQR(X+Y)
50 DEF FN CH$(R)=MID$
   (STR$(R),2)
60 DEF FN R$(F,G)=MID$
   (I$,F,G)
70 A=FN NUM(A,B)
80 A$=FN CH$(C)
90 B$=FN R$(D,E)
100 PRINT A
110 PRINT A$
120 PRINT B$
130 FOR Q=1 TO 10
140 PRINT FN R$(Q,1)
150 NEXT Q
160 END
```

## DEFDBL

TIPO: Funzione

SINTASSI: DEFDBL lettera1-lettera2,lettera3-lettera4,...

COMMENTO:
Questo comando permette di definire, in doppia precisione, un gruppo di variabili aventi la stessa lettera iniziale nel nome.

Tale istruzione, è utile porla in testa ai programmi.

## DFINT

TIPO: Funzione

SINTASSI: DEFINT lettera1-lettera2,lettera3-lettera4,...

COMMENTO:
Questo comando permette di definire un gruppo di variabili, di tipo intero, aventi la stessa lettera iniziale nel nome.

L'interesse di questo comando, sta nel fatto che le variabili di tipo intero occupano molto meno spazio in memoria di quanto non ne occupino gli altri tipi di variabili, inoltre, le operazioni svolte, usando questo tipo di variabili, vengono eseguite con una rapidità decisamente superiore; e ciò non guasta mai. Pertanto, se siamo sicuri che le variabili di cui necessitiamo in un programma saranno sempre di tipo intero, ci converrà dichiararle come tali.

## DEFSNG

TIPO: Funzione

SINTASSI: DEFSNG lettera1-lettera2,lettera3-lettera4,...

COMMENTO:
Questo comando permette di definire, in singola precisione, un gruppo di variabili aventi la stessa lettera iniziale nel nome.

## DEFSTR

TIPO: Funzione

SINTASSI: DEFSTR lettera1-lettera2,lettera3-lettera4,...

COMMENTO:
Questo comando permette di definire un gruppo di variabili, di tipo alfanumerico, aventi la stessa lettera iniziale nel nome.

È opportuno ricordare nuovamente, che tutti i comandi di tipo DEF... vanno collocati in testa ai programmi.

## DEFGR$

TIPO: Istruzione

SINTASSI: DEFGR$(n.)= n1,n2,n3...,n8

COMMENTO:
Dove n. è un numero compreso tra 0 e 127, e n1,...,n8 sono 8 numeri rappresentanti il valore di ogni singolo byte costituente il carattere ridefinito. Questi otto numeri possono essere dati sotto forma di costanti, variabili, array, eccetera, e possono essere sia decimali che binari. Quest'ultimi devono seguire le regole di rappresentazione dei numeri binari del Basic del PC 128, devono cioè essere preceduti dal prefisso &B.

Questa istruzione, deve essere obbligatoriamente preceduta dal comando CLEAR, necessario all'organizzazione della memoria utente.

Tramite DEFGR$() è possibile definire dei caratteri personalizzati o delle immagini grafiche ottenute dall'unione di più caratteri ridefiniti.

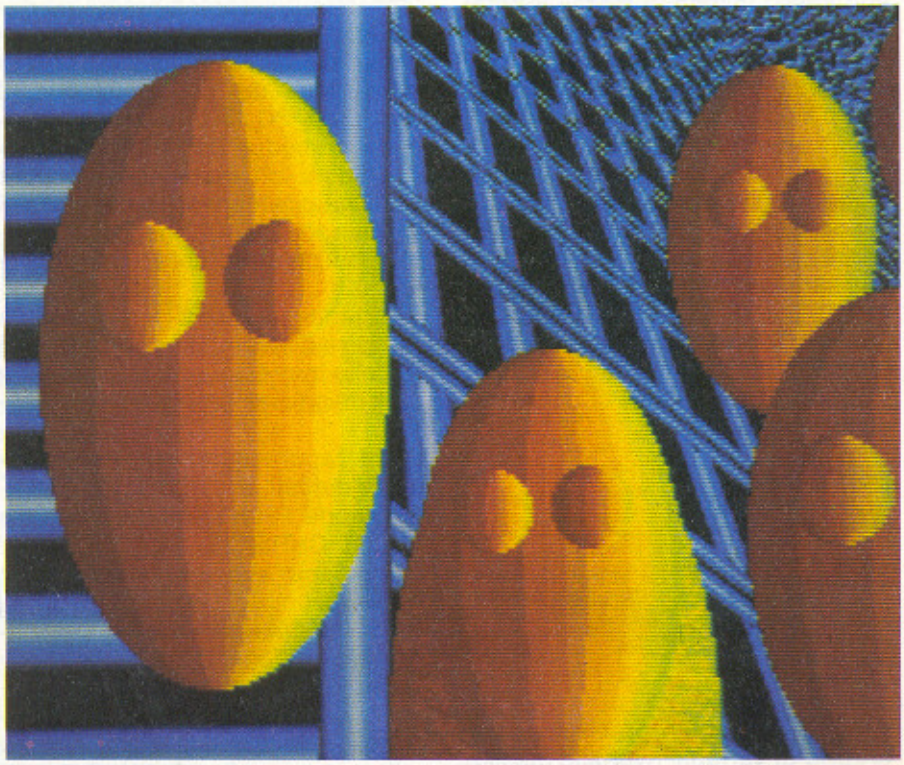
La rappresentazione dei caratteri ridefiniti, si avrà poi tramite l'istruzione PRINT GRS(n.), oppure PRINT CHR$(128+n.).

Vediamo ora, per mezzo di un breve esempio, come sia possibile ottenere delle semplici rappresentazioni grafiche, le quali però possono essere solo delle anticipazioni di creazioni molto più complesse, da voi realizzabili grazie alla potenza e flessibilità di questa tecnica.

Per prima cosa si deve disegnare un riquadro composto da 8 per 8 caselle, questa sarà la matrice da utilizzare per ridisegnare un carattere. Il perché siano necessarie 8 caselle per 8, lo dovreste oramai sapere, ma sarà opportuno chiarire nuovamente che ogni carattere visualizzato dal PC 128 è costituito da 8 byte messi in colonna uno sopra l'altro, da ciò quindi la tabella sopra descritta.

All'interno del nostro riquadro, segneremo con una crocetta ogni punto che desideriamo sia visualizzato, così da definire la nuova forma del carattere. Una volta terminata questa prima operazione, puramente creativa, alla destra di ogni riga scriveremo un numero binario in cui ripoteremo, in ordine da sinistra verso destra, con la cifra 1 i riquadri da noi segnati con una crocetta, e con lo 0 quelli vuoti. Alla fine di tale operazione ci dovremmo trovare con otto numeri binari incolonnati, rappresentanti la situazione di ogni bit dell'immagine definita.

Visto che il comando DEFGR$() accetta in argomento anche nu-



meri decimali, potremmo ora trasformare i numeri binari, appena ottenuti, nei corrispondenti numeri decimali. A tal fine, sopra ogni colonna del riquadro, dovremo segnalare, a partire da sinistra, i seguenti valori: 128, 64, 32, 16, 8, 4, 2, 1, i quali sono i valori di ogni bit, componente un byte. Questi sono determinati dalla loro posizione all'interno della serie.

Ora che siamo in possesso dei valori desiderati, possiamo finalmente scrivere un programma atto alla visualizzazione dei caratteri appena ridefiniti.

Tale programma è riportato di seguito e mostra alcune possibilità di manipolazione, dei dati ottenuti in precedenza.

```
10 CLS
20 SCREEN0,5,3
30 CLEAR,,4
40 DEFGR$(0)=24,60,126,255,
   102,102,102,255
50 DEFGR$(1)=92,42,95,124,28,
   12,12,255
60 DEFGR$(2)=&B00011000,
   &B00111100,&B01111110,
   &B11111111,&B01100110,
   &B01100110,&B01100110,
   &B11111111
70 DEFGR$(3)=&B01011100,
   &B00101010,&B01011111,
   &B01111100,&B00011100,
   &B000011,&B00001100,
   &B11111111
80 LOCATE18,10
90 PRINTGR$(0);
100 PRINTGR$(1);
110 PRINTGR$(3);
120 PRINTGR$(2);
130 A$=CHR$(128+0)+CHR$
   (128+1)+CHR$(128+2)+
   CHR$(128+3)
140 PRINT A$
150 LOCATE,5
160 FOR A=0 TO 9
170 PRINT A$
180 NEXT A
```

| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | BIN. | DEC. |
|---|---|---|---|---|---|---|---|---|---|
| | | | X | X | | | | 00011000 | = 24 |
| | | X | X | X | X | | | 00111100 | = 60 |
| | X | X | X | X | X | X | | 01111110 | = 126 |
| X | X | X | X | X | X | X | X | 11111111 | = 255 |
| | X | X | | | X | X | | 01100110 | = 102 |
| | X | X | | | X | X | | 01100110 | = 102 |
| | X | X | | | X | X | | 01100110 | = 102 |
| X | X | X | X | X | X | X | X | 11111111 | = 255 |

| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | BIN. | DEC. |
|---|---|---|---|---|---|---|---|---|---|
| | X | | X | X | X | | | 01011100 | = 92 |
| | | X | | X | | X | | 00101010 | = 42 |
| | X | | X | X | X | X | X | 01011111 | = 95 |
| | X | X | X | X | X | | | 01111100 | = 124 |
| | | | X | X | X | | | 00011100 | = 28 |
| | | | | X | X | | | 00001100 | = 12 |
| | | | | X | X | | | 00001100 | = 12 |
| X | X | X | X | X | X | X | X | 11111111 | = 255 |

Le righe 60 e 70 non sono altro che una ripetizione delle righe 40 e 50, esse sono state introdotte solo a pura dimostrazione, e per questo i dati riportati sono scritti in forma binaria. È interessante notare, sia l'uso del comando CLEAR, riga 30, che il trattamento dei dati, definiti dal comando DEFGR$(), identico al trattamento delle variabili stringa e ampiamente descritto dai blocchi di programma compresi tra le righe 80 e 120, 130 e 140, e per finire, 150 e 180.

Come è dato vedere, da questo semplice esempio, il comando DEFGR$() è molto potente e allo stesso tempo semplice da usare, pertanto grazie ad esso vi sarà molto facile creare animazioni, fondi, ed altro ancora.