IBM

Personal Computer

BASIC

Manuale di riferimento

6834177

Attenzione

In Italia valgono solo le seguenti disposizioni in lingua italiana. Eventuali altre condizioni o contratti di licenza di programmi IBM stampati all'interno del prodotto non sono applicabili e devono pertanto essere ignorate.

Il cliente è responsabile della scelta di questo prodotto software al fine del raggiungimento dei risultati voluti, nonché dell'installazione, dell'uso dello stesso e dei relativi risultati.

Condizioni di uso

Questo prodotto software contiene materiale oggetto di diritti esclusivi della IBM Italia-Distribuzione Prodotti s.r.l. (qui di seguito denominata IBM) o dei suoi danti causa. In ogni caso devono essere tassativamente osservate le seguenti condizioni.

1. L'uso del programma è consentito su una sola macchina per volta.

2. Il programma può essere copiato in forma leggibile dal sistema di elaborazione o stampato esclusivamente come supporto per tale uso. Inoltre i programmi, contrassegnati "Copy Protected" oppure "Copia Protetta" possono contenere dispositivi per limitarne o impedirne la copiatura.

3. Il programma può essere modificato o collegato ad altri programmi esclusivamente per uso su detta macchina; ogni copia integrale, parziale o modificata sarà soggetta alle presenti condizioni.

4. L'indicazione di copyright deve essere riprodotta ed inclusa in ogni copia integrale, parziale o modificata del programma.

5. Il cliente che ceda il prodotto software ad altri deve immediatamente cessarne l'uso e non può trattenerne alcuna copia, sia essa integrale, parziale o modificata. Il nuovo acquirente è tenuto ad osservare le presenti condizioni.

E' altrimenti vietata la riproduzione, l'uso ed il trasferimento del prodotto software.

In caso di esportazione, il cliente dovrà ottenere le eventuali licenze di esportazione nazionali e degli Stati Uniti d'America, che siano necessarie per l'esportazione o riesportazione di questo prodotto software.

Caratteristiche del programma e avvertimenti per l'uso

Il programma non è garantito dalla IBM né dai suoi concessionari. La IBM non garantisce che le funzioni contenute nel programma soddisfino le esigenze dell'utente o funzionino in tutte le combinazioni che possono essere scelte per l'uso da parte dell'utente. L'utente inoltre dovrà controllare il programma ed ovviare a proprie spese ad eventuali errori o malfunzionamenti.

IBM *Personal Computer*

BASIC

Manuale di riferimento

Terza Edizione (Maggio 1984)

La IBM Italia fornisce questo manuale senza offrire le garanzie usuali come per i prodotti IBM. La IBM si riserva il diritto di apportare miglioramenti o modifiche al prodotto od al programma descritto nel manuale in qualsiasi momento e senza preavviso.

Questa pubblicazione potrebbe contenere informazioni tecniche inconsistenti od errori tipografici. Le correzioni relative saranno incluse nelle nuove edizioni della pubblicazione.

E' possibile che questo materiale contenga riferimenti od informazioni su prodotti (macchine o programmi), programmi o servizi IBM, non annunciati. Tali riferimenti od informazioni non possono significare in alcun modo che la IBM Italia intende annunciare tali prodotti, programmi o servizi.

Richieste di ulteriori copie di questo prodotto od informazioni tecniche sullo stesso vanno indirizzate al punto di vendita autorizzato.

© Copyright International Business Machines Corporation 1982 - 1983-1984

Prefazione

Il Personal Computer IBM comprende tre versioni progressive compatibili dell'interprete BASIC: su Cassetta, su Disco, e Avanzato. Questo volume e il suo complementare manuale BASIC, costituiscono un riferimento per le tre versioni del BASIC a partire dal BASIC 1.Ø.

E' necessario tener presente che questi volumi non insegnano a programmare ma rappresentano solo un riferimento al linguaggio di programmazione BASIC. Volendo imparare a programmare in BASIC è consigliabile consultare specifici volumi.

Questo volume contiene, secondo la logica dell'enciclopedia, in ordine alfabetico, la sintassi e la semantica di ogni comando, specifica e funzione in BASIC.

Il manuale BASIC contiene informazioni generali sull'uso del BASIC. Vi sono delle sezioni il cui fine è quello di avvicinare l'utente al BASIC ed altre che contengono informazioni su argomenti avanzati per programmatori esperti.

Questi due manuali BASIC hanno riferimenti incrociati e sono provvisti di indici, appendici ed utili informazioni. Inoltre una sezione di rapida consultazione elenca tutti i comandi, specifiche e funzioni in BASIC, catalogate per tipologie.

Nota: Se si possiede un PC*jr* IBM, usare il volume BASIC specifico per questo tipo di PC.

Il *Compilatore BASIC* del Personal Computer IBM è un insieme di programmi opzionali disponibili presso il venditore PC. Se si possiede il *Compilatore BASIC* utilizzare il relativo manuale in congiunzione a questo volume e al manuale *BASIC*.

Riferimenti

Utili riferimenti possono essere trovati nelle seguenti pubblicazioni:

- *Guida Operativa* del Personal Computer IBM.
- *Dos* del Personal Computer IBM.
- *Technical Reference* al DOS del PC IBM.
- *Technical Reference* del PC IBM.

Sommario delle modifiche

Questo sommario elenca, prima le modifiche apportate al BASIC 2.Ø, e successivamente quelle al BASIC 3.Ø.

Modifiche al BASIC 2.0 e 2.1

Le modifiche apportate al BASIC 2.Ø sono qui di seguito brevemente descritte.

- Tre miglioramenti sono stati apportati alla specifica di comando BASIC:

  - E' stato aggiunto il parametro facoltativo *mass ampbloc* all'opzione /M: che permette di caricare programmi al di sopra dell'area di lavoro del BASIC.

  - Le funzioni ATN, COS, EXP, LOG, SIN, SQR e TAN permettono ora il calcolo a doppia precisione specificando /D sulla linea di comando BASIC.

  - Usando i parametri < *stdin* o > *stdout* nella specifica di comando BASIC è possibile variare il dispositivo standard di immissione e emissione.

- La pressione dei tasti Ctrl-Stampa fa sì che il testo inviato al proprio schermo sia inviato anche alla propria stampatrice.

- Nei comandi in cui è presente il parametro *specfile,* è stata ampliata la sintassi che ora permette di specificare un *collegamento* ad un dispositivo o ad un file. Tutti i comandi e le specifiche che accettavano il parametro *specfile* accettano ora anche il parametro specifica di *collegamento.* Tali comandi e specifiche sono: BLOAD, BSAVE, KILL, LOAD, MERGE, NAME, RUN, SAVE, CHAIN e OPEN.

- La sintassi del comando DELETE è stata ampliata per permettere la cancellazione di righe dalla linea specificata sino alla fine del programma.

- Alla specifica OPEN"COM... è stato aggiunto il parametro PE per permettere il controllo di parità.

- La specifica PLAY ha due nuovi parametri (disponibili solo col BASIC avanzato).

  - > *n* aumenta l'ottava e suona la nota *n*.

  - < *n* diminuisce l'ottava e suona la nota *n*.

- La specifica DRAW ha due nuovi parametri (disponibili solo col BASIC avanzato).

  - TA *(n)* ruota l'angolo *n* da -360 a 360 gradi.

  - P *paint, boundary* imposta il colore della figura e del bordo.

- La funzione POINT ammette ora la forma v=POINT *(n)* che fornisce il valore corrente delle coordinate grafiche x o y. Disponibile solo col BASIC avanzato.

- La specifica RANDOMIZE permette ora espressioni in precisione doppia.

- La specifica LINE ha la nuova opzione *style*, che usa valori esadecimali per tracciare una sequenza di punti sullo schermo. Disponibile solo col BASIC avanzato.

- La specifica PAINT ha la nuova funzione *sfondo*, che permette di colorare una zona con un modello, piuttosto che con un colore uniforme. Disponibile solo col BASIC avanzato.

- Nelle specifiche ON KEY (n), KEY (n), e KEY è possibile ora definire sei tasti in aggiunta ai precedenti (15-20). Disponibile solo col BASIC avanzato.

- Le specifiche GET e PUT sono state migliorate per permettere numeri di record compresi tra 1 e 16.777.215 per l'utilizzo di grandi file con ridotta lunghezza di record.

- La funzione EOF (Ø) fornisce la condizione di fine file per i dispositivi standard di input usata con reindirizzamento di I/E.

- La funzione LOF fornisce l'effettivo numero di bytes riservati a un file.

- Le specifiche grafiche CIRCLE, DRAW, LINE, PAINT, POINT, PSET, PRESET, VIEW e WINDOW ora usano il taglio delle linee.

Sono state aggiunte tre nuove funzioni:

- La funzione PLAY (n) fornisce il numero di note presenti nella memoria di transito della musica di sottofondo (MB). Disponibile solo col BASIC avanzato.

- La funzione PMAP permette di tradurre coordinate spaziali in coordinate fisiche e viceversa. Disponibile solo col BASIC avanzato.

- La funzione TIMER fornisce il numero di secondi trascorsi dalla mezzanotte o dall'ultimo ripristino del sistema.

Sono state aggiunte quattro nuove specifiche:

- La specifica ON PLAY permette di mantenere il suono di una musica durante l'esecuzione di un programma. Disponibile solo col BASIC avanzato.

- La specifica ON TIMER trasferisce il controllo ad una data riga di specifica nel programma BASIC quando è trascorso un determinato periodo di tempo. Disponibile solo col BASIC avanzato.

- La specifica VIEW permette di definire una porzione di schermo (o area) entro i limiti fisici del video. Disponibile solo col BASIC avanzato.

- La specifica WINDOW permette di ridefinire le coordinate dello schermo o parte di esso. Disponibile solo col BASIC avanzato.

Sono stati aggiunti tre nuovi comandi:

- Il comando CHDIR permette di cambiare l'indirizzario corrente.

- Il comando MKDIR crea un indirizzario sul disco specificato.

- Il comando RMDIR rimuove un indirizzario dal disco specificato.

Modifiche in BASIC 3.0

Nel BASIC 3.Ø sono state apportate le seguenti modifiche:

- Un supporto dell'unità permette al BASIC di comunicare con altre unità installate dall'utente mediante la specifica IOCTL o la funzione IOCTL$.

- IOCTL e IOCTL$ vengono usate per inviare o ricevere informazioni e/o dati da diversi canali dell'unità.

- La specifica ENVIRON e la funzione ENVIRON$ permettono di modificare i parametri nella tabella ambiente del BASIC.

- ERDEV e ERDEV$ sono variabili di errore di unità che permettono di leggere i codici di errore INT 24.

- SHELL permette di eseguire comandi DOS ed elaborazioni secondarie dal BASIC.

Nota: I termini disco, minidisco, e disco fisso vengono di frequente usati in questo manuale. Quando si parla di minidisco ci si riferisce unicamente a unità minidisco o a minidischi. Il termine disco fisso è applicabile solo all'unità disco fisso non asportabile IBM. Il termine disco invece, viene indifferentemente usato per indicare sia i dischi fissi che i minidischi.

Note:

Indice

Comandi, specifiche e funzioni BASIC 1
Come utilizzare questo manuale 1
Funzione ABS 4
Funzione ASC 5
Funzione ATN 6
Comando AUTO 8
Specifica BEEP 10
Comando BLOAD 11
Comando BSAVE 15
Specifica CALL 17
Funzione CDBL 19
Specifica CHAIN 20
Comando CHDIR 23
Funzione CHR$ 25
Funzione CINT 27
Specifica CIRCLE 28
Comando CLEAR 32
Specifica CLOSE 34
Specifica CLS 36
Specifica COLOR 38
Specifica COLOR in modo testo 39
Specifica COLOR in modo grafici 45
Specifica COM(n) 46
Specifica COMMON 47
Comando CONT 48
Funzione COS 50
Funzione CSNG 51
Variabile CSRLIN 52
Funzioni CVI, CVS, CVD 53
Specifica DATA 55
Variabile e specifica DATE$ 57
Specifica DEF FN 59
Specifica DEF SEG 62
Specifica DEF tipo 64
Specifica DEF USR 66
Comando DELETE 68
Specifica DIM 70
Specifica DRAW 72
Comando EDIT 79
Specifica END 80
Specifica ENVIRON 81

Funzione ENVIRON$ 84
Funzione EOF 87
Specifica ERASE 89
Variabili ERDEV e ERDEV$ 91
Variabili ERR e ERL 94
Specifica ERROR 96
Funzione EXP 98
Specifica FIELD 99
Comando FILES 102
Funzione FIX 105
Specifiche FOR e NEXT 106
Funzione FRE 111
Specifica GET (file) 113
Specifica GET (grafici) 115
Specifiche GOSUB e RETURN 118
Specifica GOTO 120
Funzione HEX$ 122
Specifica IF 123
Variabile INKEY$ 127
Funzione INP 129
Specifica INPUT 130
Specifica INPUT# 133
Funzione INPUT$ 135
Funzione INSTR 137
Funzione INT 138
Specifica IOCTL 139
Funzione IOCTL$ 141
Specifica KEY 142
Specifica KEY(n) 148
Comando KILL 150
Funzione LEFT$ 152
Funzione LEN 153
Specifica LET 154
Specifica LINE 156
Specifica LINE INPUT 160
Specifica LINE INPUT# 161
Comando LIST 163
Comando LLIST 166
Comando LOAD 167
Funzione LOC 170
Specifica LOCATE 172
Funzione LOF 175
Funzione LOG 177
Funzione LPOS 178
Specifiche LPRINT e LPRINT USING 179
Specifiche LSET e RSET 182

Comando MERGE 184
Funzione e specifica MID$ 186
Comando MKDIR 189
Funzioni MKI$, MKS$, MKD$ 191
Specifica MOTOR 193
Comando NAME 194
Comando NEW 196
Funzione OCT$ 197
Specifica ON COM(n) 198
Specifica ON ERROR 201
Specifiche ON...GOSUB e
ON...GOTO 203
Specifica ON KEY(n) 205
Specifica ON PEN 209
Specifica ON PLAY(n) 211
Specifica ON STRIG(n) 214
Specifica ON TIMER 217
Specifica OPEN 220
Specifica OPEN "COM... 226
Specifica OPTION BASE 233
Specifica OUT 234
Specifica PAINT 236
Funzione PEEK 246
Specifica e funzione PEN 247
Specifica PLAY 250
Funzione PLAY(n) 255
Funzione PMAP 256
Funzione POINT 258
Specifica POKE 261
Funzione POS 262
Specifica PRINT 263
Specifica PRINT USING 267
Specifiche PRINT # e
PRINT # USING 273
Specifiche PSET e PRESET 277
Specifica PUT (file) 279
Specifica PUT (grafici) 281
Specifica RANDOMIZE 286
Specifica READ 289
Specifica REM 291
Comando RENUM 293
Comando RESET 295
Specifica RESTORE 296
Specifica RESUME 297
Specifica RETURN 299
Funzione RIGHT$ 300
Comando RMDIR 301

Funzione RND 303
Comando RUN 306
Comando SAVE 308
Funzione SCREEN 310
Specifica SCREEN 312
Funzione SGN 316
Specifica SHELL 317
Funzione SIN 322
Specifica SOUND 323
Funzione SPACE$ 326
Funzione SPC 327
Funzione SQR 328
Funzione STICK 329
Specifica STOP 331
Funzione STR$ 333
Specifica e funzione STRIG 334
Specifica STRIG(n) 336
Funzione STRING$ 337
Specifica SWAP 338
Comando SYSTEM 339
Funzione TAB 340
Funzione TAN 341
Variabile e specifica TIME$ 342
Funzione TIMER 344
Comandi TRON e TROFF 345
Funzione USR 346
Funzione VAL 352
Funzione VARPTR 353
Funzione VARPTR$ 355
Specifica VIEW 357
Specifica WAIT 362
Specifiche WHILE e WEND 364
Specifica WIDTH 366
Specifica WINDOW 370
Specifica WRITE 375
Specifica WRITE # 377

Appendice A. Messaggi errore A-3

Appendice B. Sottoprogrammi in linguaggio macchina B-1
Dove allocare in memoria i sottoprogrammi B-2
Come parte residente del DOS B-2
In un segmento di dati BASIC B-3
Oltre il segmento dati del BASIC B-5

Come chiamare e caricare i sottoprogrammi . . . B-6
Immettere o assegnare un sottoprogramma in memoria B-6
Caricamento di un sottoprogramma da un file . . B-10
Esempio di sottoprogramma B-12
Spiegazione dell'esempio del sottoprogramma B-14
Caricamento del sottoprogramma come estensione residente del DOS B-19
Come il BASIC supporti i sottoprogrammi in linguaggio macchina B-24
La specifica CALL B-26
Mappa della memoria B-29

Appendice C. Comunicazioni C-1

Come aprire un file per le comunicazioni C-1
I/E della comunicazione C-1
Esempio di programmi C-4
Operazione dei segnali di controllo C-6
Controllo dei segnali di emissione con OPEN . C-6
Uso dei segnali di controllo di immissione . . C-6
Prova dei segnali di controllo del modem C-7
Controllo diretto dei segnali di controllo di emissione C-8
Errori di comunicazione C-9

Appendice D. Codici carattere ASCII D-1
Codici estesi D-6

Appendice E. Codici di scansione E-1

Glossario . . . . . Glos 1

Indice . . . . . Ind 1

Note:

Comandi, specifiche e funzioni BASIC

Come utilizzare questo manuale

In questo volume sono incluse le descrizioni di tutti i comandi, specifiche e funzioni BASIC. Ogni voce è disposta in ordine alfabetico. Per ulteriori e generali informazioni sul BASIC far riferimento al manuale BASIC che accompagna questa pubblicazione.

Le funzioni incorporate e le variabili del BASIC possono essere usate in qualsiasi programma senza ulteriori informazioni.

La distinzione tra un comando ed una specifica è dovuta in gran parte alla consuetudine. I comandi, poiché operano generalmente sui programmi, vengono immessi direttamente. Le specifiche generalmente dirigono il flusso dall'interno del programma e perciò sono abitualmente immesse indirettamente, come parte di una riga di programma. Attualmente, la maggior parte dei comandi e delle specifiche BASIC può essere immessa direttamente o indirettamente.

In questo capitolo, la descrizione di ogni comando, specifica, funzione o variabile ha il seguente formato:

Scopo: Indicare come opera il comando, la specifica, la funzione o la variabile.

Versioni: Indica quale versione del BASIC consente il suo impiego. Ad esempio, se si cerca "Specifica CHAIN" si noterà che dopo **Versioni** appare:

Cassetta	Disco	Avanzato	Compilatore
	***	***	(**)

Gli asterischi indicano quali versioni di BASIC supportano la funzione. Questo esempio mostra che si può usare la specifica CHAIN per i programmi scritti su disco in versione BASIC e BASIC avanzato.

Si noterà anche che gli asterischi sotto la parola "Compilatore" sono tra parentesi. Ciò significa che vi sono delle differenze tra il modo in cui la specifica lavora con l'interprete BASIC e col compilatore BASIC. Il compilatore BASIC del Personal Computer IBM è un pacchetto di programmi facoltativo disponibile presso l'IBM. Il manuale relativo spiega le differenze.

Formato: Mostra il formato corretto dei comandi, specifiche, funzioni o variabili. Una spiegazione dettagliata del formato della sintassi è fornita nella prefazione. Tenere presente che:

- Le parole in lettere maiuscole sono parole chiave e devono essere immesse così come sono eccetto quelle che possono essere immesse in una qualsiasi combinazione di lettere maiuscole e minuscole: il BASIC le convertirà sempre in maiuscole (a meno che non facciano parte di una stringa tra apici, di una nota o della specifica DATA).

- Devono essere forniti tutti gli elementi in lettere minuscole corsive.

- Gli elementi tra parentesi quadre ([ ]) sono facoltativi.

- Un'ellissi (...) indica un elemento che può essere ripetuto più di una volta.

- Tutta la punteggiatura, eccetto le parentesi quadre, (virgole, parentesi, punto e virgola, trattini o segni di uguale) devono essere inclusi come si trovano.

Note: Descrivono in dettaglio come si usano i comandi, le specifiche, le funzioni e le variabili.

Esempio: Mostra direttamente tramite programmi o segmenti di programma l'uso del comando, specifica, funzione o variabile.

Se si fornisce un valore a precisione singola o doppia invece di un intero, il BASIC arrotonda la porzione di frazione ed usa l'intero che ne deriva.

ABS (Funzione)

Scopo: Fornire il valore assoluto dell'espressione *x*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v*=ABS(*x*)

Note: *x* può essere qualsiasi espressione numerica.

Il valore assoluto di un numero è sempre positivo o nullo.

Esempio: Questo esempio mostra che il valore assoluto di –35 è il positivo 35.

```
PRINT ABS(7*(-5))
 35
```

ASC (Funzione)

Scopo: Fornire il codice ASCII per il primo carattere della stringa *(x$)*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v*=ASC(*x$*)

Note: *x$* può essere una espressione a stringa qualsiasi.

Il risultato della funzione ASC è un valore numerico, cioè il codice ASCII del primo carattere della stringa *x$*. (Vedere nell'Appendice D i codici ASCII). Se *x$* è nulla viene emesso l'errore **Illegal function call.**

La funzione CHR$ è l'inverso della funzione ASC e converte il codice ASCII in un carattere.

Esempio: Questo esempio mostra che il codice ASCII per la lettera maiuscola **T** è 84. La specifica **PRINT ASC ("TEST")** è corrispondente.

```
10 XS = "TEST"
20 PRINT ASC(X$)
RUN
 84
```

ATN
(Funzione)

Scopo: Fornire l'arcotangente di x.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: $v=\text{ATN}(x)$

Note: x può essere un'espressione numerica di qualsiasi tipo.

La funzione ATN fornisce l'angolo la cui tangente è x. Il risultato è un valore in radianti nella gamma compresa tra –PI/2 e PI/2, dove PI=3,141593.

Se si vogliono convertire i radianti in gradi, moltiplicare per 180/PI.

Nel BASIC 2.Ø e successive versioni, questo calcolo può essere eseguito in precisione doppia specificando **/D** nella riga di comando BASIC quando questo viene inizialmente caricato. Vedere la sezione "Opzioni nel comando BASIC" del manuale BASIC.

Esempio: Il primo esempio mostra l'uso della funzione ATN per calcolare l'arcotangente di 3.

```
PRINT ATN(3)
 1.249046
```

ATN
(Funzione)

Il secondo esempio trova l'angolo la cui tangente è 1. Cioè, 7853983 radianti, oppure 45 gradi.

```
1Ø PI=3.141593
2Ø RADIANS=ATN(1)
3Ø DEGREES=RADIANS*18Ø/PI
4Ø PRINT RADIANS,DEGREES
RUN
 .7853983      45
```

AUTO
(Comando)

Scopo: Generare automaticamente un numero di riga ogni volta che il tasto di immissione viene premuto.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	

Formato: AUTO [*numero*] [,[*incremento*]]

Note: *numero* è il numero usato per avviare la numerazione delle righe. In sostituzione del numero della riga, è possibile usare un punto (.) per indicare la riga corrente.

incremento è il valore che sarà aggiunto ad ogni numero di riga per ottenere quello successivo.

La *numerazione* inizia dal numero ed aumenta dell'*incremento* ogni numero di riga successivo. Se si omettono entrambi i valori, il valore assunto per difetto è 10,10. Se il *numero* è seguito da un virgola ma non si specifica l'*incremento,* si assume l'ultimo incremento specificato in un comando AUTO. Se si omette il *numero,* ma si include l'incremento, la numerazione della riga inizia con Ø.

Generalmente si usa AUTO per immettere i programmi, evitando così di dover battere ogni volta il numero della riga.

Se AUTO genera un numero di riga già esistente nel programma, dopo il numero viene stampato un asterisco (*) per avvertire l'utente che qualsiasi immissione sostituisce la riga esistente. Tuttavia se si preme il tasto di immissione subito dopo l'asterisco, la riga esistente non viene sostituita ed AUTO genera il successivo numero di riga.

AUTO (Comando)

AUTO termina quando si premono i tasti Ctrl-Interr. La riga in cui si premono Ctrl-Interr non viene salvata. Dopo Ctrl-Interr, il BASIC ritorna a livello di comando.

> **Nota:** In modo AUTO è possibile eseguire delle modifiche solo nella riga corrente. Se si vuole modificare un'altra riga sullo schermo, assicurarsi di uscire dal modo AUTO premendo Ctrl-Interr.

Esempio:

```
AUTO
```

Questo comando genera numeri di riga 10, 20, 30, 40...

```
AUTO 100,50
```

Questo genera i numeri di riga 100,150, 200,...

```
AUTO 500,
```

Questo genera i numeri di riga 500, 550, 600, 650,... L'incremento è 50, poiché 50 era l'incremento indicato nel precedente comando AUTO.

```
AUTO ,20
```

Questo genera i numeri di riga 0, 20, 40, 60,...

BEEP
(Specifica)

Scopo: Far emettere suoni all'altoparlante.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: BEEP

Note: La specifica BEEP fa suonare l'altoparlante a 800 Hz per 1/4 di secondo. BEEP ha lo stesso effetto di:

```
PRINT CHR$(7);
```

Esempio: In questo esempio, il programma controlla se la X è fuori limite. In caso affermativo, l'elaboratore lo segnala con un suono.

```
100 IF X < 20 THEN BEEP
```

BLOAD (Comando)

Scopo: Caricare in memoria un file ad immagine della memoria

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: BLOAD *specfile* [,*pos. relativa*]

Note:

specfile è un'espressione a stringa per la specifica del file. Nel BASIC 2.0 e successivi livelli può contenere un percorso. Deve essere conforme alle regole indicate nella sezione "Denominazione dei File" del Capitolo 3, del manuale BASIC altrimenti si verifica un errore.

pos. relativa è un'espressione numerica compresa nella gamma da 0 a 65535. Questo è l'indirizzo specificato come posizione relativa dal quale comincia il caricamento, nel segmento corrente dichiarato dall'ultima specifica DEF SEG.

Se si omette *pos. relativa,* si assume la posizione relativa specificata in BSAVE. Cioè, il file viene caricato nella stessa posizione in cui è stato BSAVED (salvato).

Quando si esegue un comando BLOAD, il file indicato viene caricato in memoria a partire dalla posizione specificata. Se il file deve essere caricato dall'unità CAS1:, il registratore a cassette viene acceso automaticamente.

Se si usa il BASIC su cassetta e si omette l'unità indicata, si assume CAS1:. CAS1: è l'unica unità disponibile per BLOAD nel BASIC su cassetta. Se si usa il BASIC su disco o avanzato e si omette il nome dell'unità si usa per difetto il modulo a minidischi del DOS.

BLOAD (Comando)

BLOAD viene usato con un file precedentemente salvato con BSAVE. BLOAD e BSAVE sono utili per caricare e salvare i programmi in linguaggio macchina. Tuttavia non si limitano a questi. Qualsiasi segmento può essere specificato come punto di arrivo o di origine per queste specifiche tramite DEF SEG. L'utente dispone di un modo utile per salvare e visualizzare le immagini dello schermo: salvandole e caricandole nella memoria di transito. Vedere anche l'appendice B, "Sottoprogrammi in linguaggio macchina".

Attenzione:
Il BASIC non esegue alcun controllo sull'indirizzo. Cioè, è possibile usare BLOAD ovunque in memoria. Non si deve usare BLOAD con stack del BASIC, aree di variabili BASIC o programmi BASIC. Vedere la mappa della memoria nell'Appendice B.

Note sull'utilizzo di CAS1:

1. Se si immette il comando BLOAD in modo diretto, i nomi dei file sul nastro saranno visualizzati seguiti da un punto (.) e da una lettera che indica il tipo di file. Questa è seguita dal messaggio **"Skipped"** (Saltato) per i file che non corrispondono a quello indicato e **"Found"** (Trovato) quando questo viene trovato. I tipi di file e le lettere associate sono:

   .B per programmi BASIC in formato interno (creati con comando SAVE).

   .P per programmi BASIC protetti in formato interno (creati con comando SAVE,P).

.A per programmi BASIC in formato ASCII (creati con comando SAVE ,A).

.M per file ad immagine della memoria (creati con comando BSAVE).

.D per file di dati (creati da OPEN seguito da specifiche di emissione).

Se si esegue il comando BLOAD in un programma BASIC, i nomi dei file saltati e trovati non vengono visualizzati.

2. Si possono premere i tasti Ctrl e Interr in qualsiasi momento durante BLOAD. Questo consente al BASIC di terminare la ricerca e ritornare al modo diretto tra i file o dopo un periodo di supero tempo. Non cambia il precedente contenuto della memoria.

3. Se si specifica CAS1: come unità e si omette il nome del file, viene caricato il successivo file ad immagine della memoria (.M) sul nastro.

BLOAD
(Comando)

Esempio: Questo esempio carica la memoria di transito dello schermo per l'Adattatore del video grafico a colori che si trova all'indirizzo assoluto esadecimale HB8000. Se si carica la memoria di transito dello schermo per l'Adattatore video IBM monocromatico e stampatrice parallela, si deve modificare la riga 30 in &HB000. La riga 50 carica PICTURE alla posizione relativa 0, indirizzo &HB800.

```
10 'carica la memoria di transito dello schermo
20 'SEG punta alla memoria di transito dello schermo
30 DEF SEG= &HB800
40 'carica PICTURE nella memoria di transito dello schermo
50 BLOAD "PICTURE", 0
```

L'esempio di BSAVE alla sezione successiva illustra come è stato salvato PICTURE.

BSAVE (Comando)

Scopo: Salvare porzioni della memoria dell'elaboratore sull'unità specificata.

Versioni:	Cassetta	Disco	Avanzato	Compilatore
	***	***	***	***

Formato: BSAVE *specfile, pos. relativa, lunghezza*

Nota:

specfile è un'espressione a stringa per la specifica del file. Nel BASIC 2.Ø e successivi livelli può contenere un percorso. Deve essere conforme alle regole indicate nella sezione "Denominazione dei File" del Capitolo 3, del manuale BASIC, altrimenti si verifica un errore.

pos. relativa è un'espressione numerica intera compresa tra Ø e 65535. Questa è la posizione relativa nel segmento dichiarato dall'ultima specifica DEF SEG. Il salvataggio inizierà da questa posizione. Vedere la specifica "DEF SEG".

lunghezza è un'espressione numerica intera compresa tra 1 e 65535. Questa è la lunghezza dell'immagine di memoria da salvare.

Se si omette *pos. relativa,* o *lunghezza,* si verifica un errore **Syntax error** ed il salvataggio viene annullato.

Se si omette il nome dell'unità nel BASIC su cassetta, si assume CAS1:. CAS1: è l'unica unità consentita per BSAVE nel BASIC su cassetta. Se si omette il nome dell'unità nel BASIC su disco o avanzato, si assume per difetto l'unità a minidischi del DOS.

BSAVE (Comando)

Se si sta salvando su CAS1:, il registratore a cassette verrà acceso ed il file dell'immagine di memoria sarà scritto immediatamente sul nastro.

Con la specifica DEF SEG, è possibile specificare qualsiasi segmento come punto di origine per queste specifiche. Ad esempio, si può salvare un'immagine dello schermo eseguendo BSAVE della memoria di transito dello schermo.

Esempio: Come indicato nel comando BLOAD, l'indirizzo della memoria di transito dello schermo di 16K per l'adattatore per il video grafico a colori è HB8000. L'indirizzo del segmento della memoria di transito dello schermo di 4K per l'Adattatore del video IBM monocromatico e stampatrice parallela è HB0000.

La specifica DEF SEG deve essere usata per impostare l'indirizzo del segmento all'avvio della memoria di transito dello schermo. La posizione relativa 0 e la lunghezza &H4000 specificano che deve essere salvata l'intera memoria di transito dello schermo di 16K.

```
10 'Salva la memoria di transito dello schermo
15 'SEG punta alla memoria di transito dello schermo
20 DEF SEG= &HB800
25 'Salva la memoria di transito nel file PICTURE
30 BSAVE "PICTURE",0,&H4000
```

CALL (Specifica)

Scopo: Richiamare un sottoprogramma in linguaggio macchina.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: CALL *varnum* [(*variabile* [,*variabile*]...)]

Note:

varnum è il nome di una variabile numerica. Il valore della variabile indica la posizione relativa del sottoprogramma da richiamare nell'attuale segmento di memoria (come definito dall'ultima specifica DEF SEG).

variabile è il nome della variabile che deve essere passata come argomento nel sottoprogramma in linguaggio macchina.

La specifica CALL è una delle interfacce tra i programmi in linguaggio macchina ed il BASIC. Fare riferimento all'Appendice B "Sottoprogrammi in linguaggio macchina" per specifiche considerazioni al riguardo.

CALL
(Specifica)

Esempio: La riga 10 imposta un segmento nel segmento del BASIC. La 30 dichiara tutti i valori scalari e schiere usate nel programma. La riga 40 calcola la posizione relativa di ARRAY nel segmento dati del BASIC. La riga 50 carica il file in una schiera intera e la riga 60 richiama il programma. Le variabili Q,B$ e C vengono passate come argomenti al programma.

```
10 DEF SEG: OPTION BASE 1
20 DEFINT A-Z
30 DIM ARRAY (512): P=0: Q=5: B$=TRUE":B$="TRUE": C=0
40 P=VARPTR(ARRAY(1))
50 BLOAD "ASM.FIL",P
60 CALL P(Q,B$,C)
```

CDBL (Funzione)

Scopo: Convertire *x* in un numero a precisione doppia.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v*=CDBL(*x*)

Note: *x* può essere qualsiasi espressione numerica.

Per la conversione da una precisione numerica all'altra valgono le regole descritte al Capitolo 3 del manuale BASIC. Per la conversione dei numeri in interi e a precisione singola, fare riferimento anche alle funzioni CINT e CSNG.

Esempio: Il valore di CDBL(A) è preciso solo al secondo decimale dopo l'arrotondamento. Le cifre in eccesso non hanno significato. Questo perché con A sono fornite solo due posizioni decimali.

```
10 A = 454.67
20 PRINT A;CDBL(A)
RUN
 454.67  454.669829101563
```

CHAIN (Specifica)

Scopo: Trasferire il controllo ad un altro programma e passargli le variabili del programma attuale.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	(**)

Formato: CHAIN [MERGE] *specfile* [,[*riga*] [,[ALL] [,DELETE *gamma*]]]

Note: MERGE porta una sezione di codice nel programma BASIC come sovrapposizione. Cioè, si esegue un'operazione MERGE con il programma concatenante e quello concatenato. Se deve essere fuso, il programma concatenato deve essere un file ASCII. Esempio:

```
CHAIN MERGE "A:OVRLAY",1000
```

specfile è una espressione a stringa per la specifica del file. In BASIC 2.0 e successive versioni può contenere un percorso. Segue le regole per la specifica dei file descritte al Capitolo 3 nella sezione "Denominazione dei file". Il nome del file è quello del programma al quale viene trasferito il controllo. Esempio:

```
CHAIN "A:PROG1"
```

riga è un numero di riga o un'espressione che calcola un numero di riga nel programma concatenato. Specifica la riga dalla quale inizia il programma concatenato. Se si omette, l'esecuzione inizia dalla prima riga del programma concatenato.

Riga non è interessata dal comando RENUM. Se si rinumera PROG1, la specifica CHAIN di questo esempio deve essere modificata in modo da puntare al nuovo numero di riga.

```
CHAIN "A:PROG1,1000
```

ALL specifica che ogni variabile del programma deve essere passata al programma concatenato. Se si omette l'opzione ALL, si deve includere una specifica COMMON nel programma concatenante per passare le variabili al programma concatenato. Vedere la "specifica COMMON". Esempio:

```
CHAIN "A:PROG1,1000,ALL
```

Per eseguire questa operazione, si usa l'opzione DELETE che si comporta come il comando DELETE. Come in questo comando, i numeri di riga specificati come prima e ultima riga nella gamma, devono esistere altrimenti si verifica un errore di **Illegal function call.** Dopo aver usato una sovrapposizione, normalmente la si vuole cancellare in modo da poterne effettuare un'altra.
Esempio:

```
CHAIN MERGE "A:OVRLAY2",1000,DELETE 1000-5000
```

CHAIN (Specifica)

Questo esempio cancella le righe da 1000 a 5000 del programma concatenante prima di effettuare la sovrapposizione (programma concatenato). I numeri di riga nella *gamma* sono interessati dal comando RENUM.

Note:

1. La specifica CHAIN lascia aperti i file.

2. La specifica CHAIN con l'opzione MERGE conserva l'attuale impostazione OPTION BASE.

3. Se si omette l'opzione MERGE, CHAIN non conserva i tipi di variabili o le funzioni definite dall'utente utilizzate dal programma concatenato. Cioè, qualsiasi specifica DEFINT, DEFSNG, DEFDBL, DEFSTR o DEF FN contenente variabili condivise deve essere impostata nel programma concatenato.

4. La specifica CHAIN effettua un RESTORE prima di eseguire il programma concatenato.

CHDIR (Comando)

Scopo: Permettere di cambiare l'indirizzario corrente. (Per il Basic 2.Ø e successive versioni).

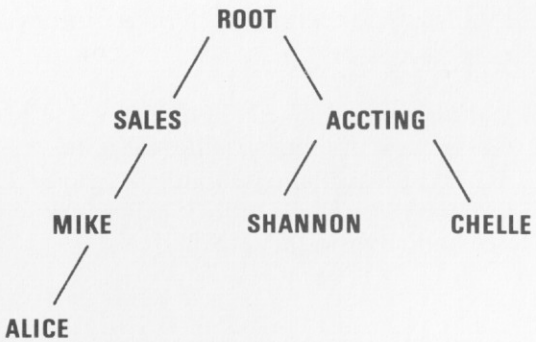
Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	

Formato: CHDIR *collegamento*

Note: *collegamento* è una espressione a stringa non superiore a 63 caratteri che identifica il nuovo indirizzario corrente. Per ulteriori informazioni consultare la sezione "Denominazione dei file" e "Indirizzari strutturati ad albero" nel capitolo 3 del manuale BASIC.

Esempio:



CHDIR (Comando)

Gli esempi seguenti fanno riferimento alla struttura ad albero della pagina precedente.

Per passare da un sotto-indirizzario qualsiasi all'indirizzario-radice immettere:

```
CHDIR "\"
```

Per passare dall'indirizzario-radice all'indirizzario ALICE immettere:

```
CHDIR "SALES\MIKE\ALICE"
```

Per passare dall'indirizzario ACCTING all'indirizzario CHELLE immettere

```
CHDIR "CHELLE"
```

Per passare dall'indirizzario MIKE a SALES immettere:

```
CHDIR ".."
```

Per far diventare SALES l'indirizzario corrente sull'unità corrente (unità A) e INVENTORY l'indirizzario corrente sull'unità C: immettere:

```
CHDIR "SALES"
CHDIR "C:INVENTORY"
```

L'indirizzario INVENTORY deve esistere nell'unità C. Ora se si usa una *specfile* sull'unità A, si fa riferimento ai file che si trovano nell'indirizzario SALES. Se si usa una *specfile* sull'unità C si fa riferimento ai file dell'indirizzario INVENTORY.

CHR$ (Funzione)

Scopo: Convertire un codice ASCII nel carattere equivalente.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v$*=CHR$(*n*)

Note: *n* deve essere compreso tra 0 e 255.

La funzione CHR$ fornisce la stringa di un carattere relativa al codice ASCII *n* (I codici ASCII sono elencati nella appendice D). Normalmente si usa CHR$ per inviare un carattere speciale allo schermo o stampatrice. Ad esempio, il carattere BEL, che fa emettere i suoni alla stampatrice, può essere incluso come CHR$ (7) come prefazione ad un messaggio di errore (invece di usare BEEP). Vedere sotto la funzione ASC come riconvertire un carattere nel codice ASCII relativo.

Esempio: Questo esempio stampa il carattere equivalente al codice ASCII 66.

```
PRINT CHR$(66)
B
```

Il successivo esempio imposta il tasto funzionale F1 alla stringa formata da "AUTO" più il tasto di immissione. Questo è un ottimo metodo per impostare i tasti funzionali in modo che l'immissione venga eseguita automaticamente quando si preme il tasto funzionale.

```
KEY 1,"AUTO"+CHR$(13)
```

CHR$
(Funzione)

Il seguente esempio è un programma che mostra tutti i caratteri visualizzabili, con i loro codici ASCII, sullo schermo di 80 colonne. Può essere usato sia con l'Adattatore per il video IBM monocromatico e stampatrice parallela che con l'Adattatore per il video grafico a colori:

```
10 CLS
20 FOR I=1 TO 255
30 ' ignora i caratteri non visualizzati
40 IF (I>6 AND I<14) OR (I>27 AND I<32) THEN 100
50 COLOR 0,7 ' nero su bianco
60 PRINT USING"###" ; I ; ' codici ASCII su tre posizioni di stampa
70 COLOR 7,0 ' bianco su nero
80 PRINT " "; CHR$(I); " ";
90 IF POS(0)>75 THEN PRINT ' va alla riga successiva
100 NEXT I
```

CINT (Funzione)

Scopo: Convertire *x* in un intero.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v*=CINT(*x*)

Note: *x* può essere una qualsiasi espressione numerica. Se *x* non è compreso tra –32768 e 32767, si verifica un errore di **Overflow.**

x è convertito in un intero arrotondando per eccesso la parte decimale.

Vedere le funzioni FIX e INT che forniscono entrambe degli interi. Vedere anche le funzioni CDBL e CSNG per la conversione dei numeri in precisione singola o doppia.

Esempio: Osservare in entrambi gli esempi come si verifica l'arrotondamento.

```
PRINT CINT(45.499)
 45

PRINT CINT(-2.89)
-3
```

CIRCLE (Specifica)

Scopo: Disegnare sullo schermo un'ellisse con centro (x,y) e raggio r.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
		***	***

Solo modo grafici.

Formato: CIRCLE $(x,y),r$[,*colore*[, *avvio, fine* [,*aspetto*]]]

Note:

(x,y) sono le coordinate del centro dell'ellisse. Le coordinate possono essere indicate in forma assoluta o relativa. Vedere "Come specificare le coordinate" nel Capitolo 3 del manuale BASIC.

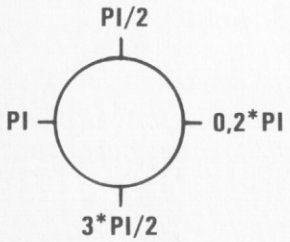
r è il raggio (asse maggiore) dell'ellisse in punti.

Colore è un numero o un'espressione che specifica l'attributo (colore) nella gamma (dei colori possibili per il tipo di schermo corrente). A media risoluzione *colore* sceglie il colore della tavolozza, come definito dalla specifica COLOR.
Per la media risoluzione sono disponibili quattro attributi (Ø-3); per l'alta risoluzione due (Ø-1). Zero (Ø) è il colore di fondo. L'attributo assunto per difetto per il colore di primo piano è sempre il massimo attributo per quel tipo di schermo a media risoluzione; 1 ad alta risoluzione.

avvio, fine sono gli angoli in radianti e possono variare da -2*PI a 2*PI, dove PI è 3.141593.

aspetto è un'espressione numerica.

avvio e fine specificano dove deve iniziare e finire il disegno dell'ellisse. Gli angoli sono posizionati nel modo matematico standard, con Ø alla destra e in movimento antiorario.

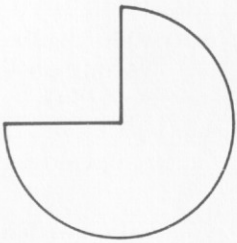


Se l'angolo di avvio o di fine è negativo, (–Ø non è consentito) l'ellisse viene collegata al centro con una linea e gli angoli vengono considerati positivi (notare che non corrisponde ad aggiungere 2*PI). L'angolo di avvio può essere maggiore o minore rispetto a quello di fine. Ad esempio:

```
1Ø PI=3.141593
2Ø SCREEN 1
3Ø CIRCLE (16Ø,1ØØ),6Ø,,-PI,-PI/2
```

Disegna una parte di un cerchio simile a questa:

CIRCLE (Specifica)

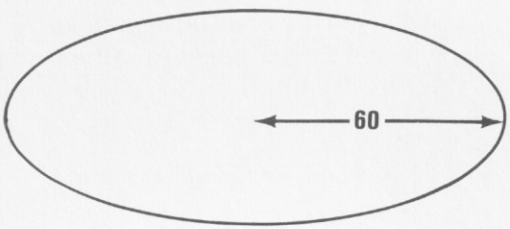


aspetto riguarda il rapporto tra il raggio x ed il raggio y. Il valore assunto per *aspetto* è 5/6 a media risoluzione e 5/12 ad alta. Questi valori forniscono un cerchio che assume l'aspetto dello schermo standard in rapporto di 4/3.

Se *aspetto* è minore di uno, allora *r* è il raggio x. Cioè il raggio è misurato in punti nella direzione orizzontale. Se *aspetto* è maggiore di uno, allora *r* è il raggio. Ad esempio:

```
10 SCREEN 1
20 CIRCLE (160,100),60,,,,5/18
```

disegna un'ellisse come questa:



In molti casi, un *aspetto* 1 darà dei cerchi migliori a definizione media. Il cerchio sarà anche disegnato più velocemente.

L'ultimo punto a cui si è fatto riferimento dopo il disegno di un cerchio è il suo centro.

I punti che si trovano fuori dallo schermo non vengono disegnati da CIRCLE.

Esempio: il seguente esempio disegna un superficie.

```
10 PI=3.141593
20 SCREEN 1 ' grafici a media risoluzione
30 COLOR 0,1 'sfondo nero, tavolozza 1
40 'due circonferenze con colore 1 (azzurro)
50 CIRCLE (120,50),10,1
60 CIRCLE (200,50),10,1
70 'due ellissi orizzontali
80 CIRCLE (120,50),30,,,,5/18
90 CIRCLE (200,50),30,,,,5/18
100 'arco con colore 2 (magenta)
110 CIRCLE (160,0),150,2, 1.3*PI, 1.7*PI
120 'arco connesso al centro da una parte
130 CIRCLE (160,52),50,, 1.4*PI, -1.6*PI
```

CLEAR (Comando)

Scopo: Impostare tutte le variabili numeriche a zero e vuotare tutte le variabili a stringa. Le opzioni impostano il massimo numero di byte che il BASIC indirizzerà e la quantità di spazio di stack.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: CLEAR [,[*n*] [,*m*]]

Note:

n è un totale di byte che, se specificato, imposta il numero massimo di byte per lo spazio di lavoro del BASIC (su cui sono memorizzati i dati e il programma unitamente all'area di lavoro dell'interpretatore).
Il valore assunto di *n* è 65535. Si può specificare *n* come più piccolo valore da sottrarre allo spazio indirizzabile del BASIC. Questo incrementa l'ammontare della memoria disponibile dietro il segmento dati del BASIC nella parte alta della memoria. Si include *n* se si deve riservare spazio in memoria per i programmi in linguaggio macchina dietro il segmento dati del BASIC.

m riserva dello spazio a parte per il BASIC. Il valore assunto è 512 byte oppure un ottavo della memoria disponibile (anche se limitata). Si può includere *m* se nel programma si usano molte specifiche GOSUB nidificate o iterazioni FOR... NEXT, oppure se si usa PAINT per immagini complesse.

CLEAR (Comando)

CLEAR libera tutta la memoria usata per i dati senza cancellare il programma che si trova in memoria. Dopo CLEAR, le schiere sono indefinite; le variabili numeriche hanno un valore zero; le variabili a stringa hanno un valore nullo e qualsiasi informazione impostata da una specifica DEF viene persa (questo comprende le specifiche DEF FN, DEF SEG, DEF USR, così come, DEFINT, DEFDBL, DEFSNG e DEFSTR).

Eseguendo un comando CLEAR si disattiva qualsiasi suono e si ripristina il primo piano della musica. Anche PEN e STRIG vengono ripristinati ad OFF.

La specifica ERASE può essere utile per liberare la memoria senza cancellare tutti i dati del programma. Cancella dall'area di lavoro solo le schiere specificate. Per dettagli, vedere la specifica ERASE.

Esempio: questo esempio cancella tutti i dati dalla memoria (senza cancellare il programma).

```
CLEAR
```

Il successivo esempio cancella i dati ed imposta l'ampiezza massima dell'area di lavoro a 32K-byte:

```
CLEAR ,32768
```

L'esempio seguente cancella i dati ed imposta l'ampiezza della memoria a 2000 byte:

```
CLEAR ,,2000
```

L'ultimo esempio cancella i dati, imposta il massimo per l'area di lavoro del BASIC a 32K-byte ed imposta l'ampiezza della memoria a 2000 byte:

```
CLEAR ,32768,2000
```

CLOSE
(Specifica)

Scopo: Porre termine all'I/E su un'unità o file.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: CLOSE [[#] *numfile* [,[#] *numfile*]...]

Note: *numfile* è il numero usato nella specifica OPEN.

L'associazione tra un determinato file o unità ed il relativo numero, si arresta quando viene eseguita CLOSE. Le successive operazioni di I/E che specificano quel numero di file non sono valide. Il file o l'unità può essere riaperta usando lo stesso o un diverso numero di file; oppure il numero del file può essere riutilizzato per aprire qualsiasi unità o file.

Una specifica CLOSE su un file o unità aperta per emissioni sequenziali causa la scrittura della memoria di transito finale sul file o sulla unità.

Una CLOSE senza numeri di file specificati causa la chiusura di tutti i file e unità aperte.

L'esecuzione di END, NEW, RESET, SYSTEM o RUN senza l'opzione **R** produce la chiusura automatica di tutti i file e unità aperte. STOP non chiude alcun file o unità.

Per informazioni sull'apertura dei file vedere la specifica OPEN.

CLOSE (Specifica)

Esempio: Questo primo esempio causa la chiusura dei file e delle unità associate ai numeri di file 1, 2, 3:

```
100 CLOSE#1,#2,#3
```

Questo secondo esempio causa la chiusura di tutti i file ed unità aperte:

```
100 CLOSE
```

CLS (Specifica)

Scopo: Cancellare lo schermo.

Versioni:	Cassetta	Disco	Avanzato	Compilatore
	***	***	***	***

Formato: CLS

Note: Se lo schermo è in modo testo, il contenuto della pagina attiva viene eliminato e lo schermo viene riportato al colore di fondo. Vedere le specifiche "COLOR" e "SCREEN".

Se lo schermo è in modo grafici (media o alta risoluzione), l'intera memoria di transito dello schermo viene svuotata fino al colore di fondo.

La specifica CLS riporta anche il cursore alla posizione iniziale. In modo testo, ciò significa che il cursore si trova nell'angolo superiore sinistro dello schermo. In modo grafici, ciò significa che l'ultimo punto a cui si è fatto riferimento, per le specifiche grafiche successive, è il centro dello schermo (160, 100) a media risoluzione, (320, 100) ad alta risoluzione.

La modifica del modo o dell'ampiezza dello schermo con le specifiche SCREEN o WIDTH cancella anche lo schermo. Lo schermo può essere cancellato anche premendo i tasti Ctrl e Rit Curs.

Quando si usa la specifica VIEW, la specifica CLS cancella solo la parte visualizzata. Per pulire l'intero schermo si deve usare prima VIEW per disabilitare la parte di schermo visualizzata, quindi la specifica CLS. (Le parti visualizzate sono usate nel BASIC 2.0 e nei livelli successivi)

CLS
(Specifica)

Esempio: Con l'Adattatore per il video grafico a colori, questo esempio svuota lo schermo fino al blu:

```
10 SCREEN 0,0,0
20 COLOR 10,1
30 CLS
```

COLOR
(Specifica)

Scopo: Impostare i colori per il primo piano, lo sfondo ed il bordo dello schermo. Per una spiegazione di questi termini, fare riferimento a "Modo Testo" al Capitolo 3 del manuale BASIC.

La sintassi della specifica COLOR dipende dal modo in cui ci si trova, testo o grafici, come definito dalla specifica SCREEN.

Prima di eseguire qualsiasi operazione BASIC il colore è inizialmente impostato sul bianco (per il primo piano) e il nero (sfondo e bordo).

In modo testo si può definire quanto segue:

Primo piano:	1 dei 16 colori lampeggiamento del carattere, se desiderato
Sfondo:	1 degli 8 colori
Bordo:	1 dei 16 colori

In modo grafici, a media risoluzione, si può definire quanto segue:

Sfondo:	1 dei 16 colori
Tavolozza:	1 di 2 tavolozze con 3 colori ciascuna

Il bordo è uguale al colore dello sfondo.

COLOR
(Specifica)

La Specifica COLOR in modo testo

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: COLOR [*primopiano*] [,[*sfondo*] [,*bordo*]]

Note:

primopiano è un'espressione numerica compresa tra Ø e 31 che rappresenta il colore del carattere

sfondo è un'espressione numerica compresa tra Ø e 7 per il colore dello sfondo.

bordo è un'espressione numerica compresa tra Ø e 15. E' il colore per il bordo dello schermo.

Se si ha un Adattatore per il video grafico a colori
per il *primo piano* sono consentiti i seguenti colori:

Ø	Nero	8	Grigio
1	Blu	9	Blu chiaro
2	Verde	10	Verde chiaro
3	Azzurro	11	Azzurro chiaro
4	Rosso	12	Rosso chiaro
5	Magenta	13	Magenta chiaro
6	Marrone	14	Giallo
7	Bianco	15	Bianco ad alta intensità

I colori e l'intensità possono variare a seconda dell'unità video.

I colori 8-15 si possono considerare come chiari o ad alta intensità (rispetto ai colori Ø-7).

COLOR (Specifica)

E' possibile far lampeggiare i caratteri definendo il *primopiano* uguale a 16 più il numero del colore desiderato. Cioè, un valore da 16 a 31 causa il lampeggiamento dei caratteri.

Per lo sfondo in modo testo si possono scegliere solo i colori Ø-7.

Se si possiede l'Adattatore per l'unità video IBM monocromatico e stampatrice parallela, per il *primo piano* possono essere utilizzati i seguenti colori:

Ø	Nero
1	Caratteri sottolineati con colore di primo piano standard
2-7	Colore di primo piano standard

Con il video grafico a colori, aggiungendo 8 al numero del colore desiderato, questo viene visualizzato ad alta intensità. Ad esempio, il valore 15 fornisce il colore standard ad alta intensità.

Con l'Adattatore per il video grafico a colori, è possibile far lampeggiare il carattere aggiungendo 16 al numero del colore desiderato. Quindi con 31 si ottengono i caratteri di colore standard ad alta intensità.

Con l'Adattatore per l'unità video IBM monocromatico e stampatrice parallela, per lo *sfondo* è possibile scegliere i seguenti valori:

Ø-6	Nero
7	Colore di primo piano standard

Nota: Il colore 7, come colore di fondo, viene visualizzato come colore standard sull'unità video IBM monocromatica solo se utilizzato con un colore di primo piano Ø, 8, 16 o 24 (nero).

Il nero (colore Ø,8,16 o 24), come colore di primo piano, viene visualizzato in nero solo se utilizzato con un colore di sfondo che crea i caratteri in inversione di fondo. Il nero usato con un attributo di sfondo Ø rende i caratteri invisibili.

Altre combinazioni di colori di primo piano e sfondo producono risultati standard sull'unità video IBM monocromatica.

Note per entrambi gli adattatori

1. Il colore in primo piano può essere uguale a quello di fondo. Questo produce l'effetto di rendere i caratteri invisibili. Modificando successivamente questi due colori, i caratteri ritornano visibili.

2. Può essere omesso qualsiasi parametro. I parametri omessi assumono il vecchio valore.

3. Se la specifica COLOR termina con una virgola (,), si verifica un errore **Missing Operand,** ma il colore cambia. Ad esempio,

```
COLOR 1,7,
```

non è valida.

4. Qualsiasi valore immesso, non compreso tra Ø e 255 produce l'errore **Illegal function call** I precedenti valori vengono mantenuti:

Esempio: Questo esempio definisce un primo piano giallo, uno sfondo blu ed un bordo nero sullo schermo.

```
10 COLOR 14,1,0
```

COLOR
(Specifica)

Il seguente esempio può essere utilizzato sia con l'adattatore per il video grafico a colori che con l'adattatore per l'unità video IBM monocromatica e stampatrice parallela:

```
10 PRINT "Enter your ";
20 COLOR 15,0 'highlight next word
30 PRINT "password";
40 COLOR 7    'ritorna al bianco su nero
50 PRINT " here: ";
60 COLOR 0    'non visibile (nero su nero)
70 INPUT PASSWORD$
80 IF PASSWORD$="secret" THEN 120
90 ' messaggio di errore lampeggiante ad alta intensità
100 COLOR 31: PRINT "Wrong Password": COLOR 7
110 GOTO 10
120 COLOR 0,7  'immagine inversa (nero su bianco)
130 PRINT "Program continues...";
140 COLOR 7,0  'ritorna al bianco su nero
```

COLOR (Specifica)

La Specifica COLOR in modo grafici

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Modo grafici, solo media risoluzione.

Formato: COLOR [*sfondo*] [,[*tavolozza*]]

Note:

sfondo è un'espressione numerica compresa tra Ø-15 che specifica il colore di fondo.

tavolozza è un'espressione numerica che sceglie una delle due tavolozze di colore.

In modo grafici la specifica COLOR imposta un colore di sfondo e sceglie una delle due tavolozze di quattro colori ciascuna (Ø-3). Il colore Ø è sempre lo sfondo in uso. Si può scegliere uno dei tre attribuiti di colore per il colore di primo piano da usare con le specifiche PSET, PRESET, LINE, CIRCLE, PAINT, VIEW e DRAW. La specifica COLOR in modo grafici ha significato solo se usata a media risoluzione. Se COLOR viene usato ad alta risoluzione si verificherà un errore di **Illegal function call.**
I colori selezionati quando si sceglie ogni tavolozza sono:

Colore	**Tavolozza 0**	**Tavolozza 1**
1	Verde	Azzurro
2	Rosso	Magenta
3	Marrone	Bianco

Se *tavolozza* è un numero pari, si sceglie la tavolozza Ø. Questo associa i colori Verde, Rosso e Marrone ai numeri 1, 2 e 3. Si sceglie la tavolozza 1 (Azzurro/Magenta/Bianco) quando *tavolozza* è un numero dispari.

COLOR
(Specifica)

Il modo grafici può visualizzare un testo in qualsiasi colore dei tre disponibili della tavolozza in uso. Tuttavia, se non si sta usando una tastiera U.S. fare riferimento al comando "GRAFTABL" nel manuale di riferimento del DOS per informazioni che riguardino il supporto di caratteri addizionali per l'adattatore del video grafico a colori ed altre tastiere.

Il colore di primo piano dei caratteri può essere modificato da 3 a 2 a 1 immettendo:

```
DEF SEG: POKE &HFE, COLOR
```

dove COLOR è l'attributo 1, 2 o 3; non è permesso l'attributo Ø. Successivamente PRINT userà l'attributo del colore specificato.
Il colore selezionato per il fondo può essere lo stesso o uno qualsiasi dei colori della tavolozza.

Dalla specifica COLOR può essere omesso qualsiasi parametro. Questo non cambia la tavolozza o il colore del fondo in uso. Ogni valore immesso al di fuori della gamma Ø-255 genera un errore di **Illegal function call.** I valori precedenti vengono mantenuti.

COLOR
(Specifica grafici)

Esempio:

```
10 SCREEN 1
20 COLOR 9,0
```

Imposta lo sfondo al blu chiaro e sceglie la tavolozza 0.

```
10 COLOR ,1
```

Lo sfondo resta in blu chiaro e si sceglie la tavolozza 1.

COM(n) (Specifica)

Scopo: Abilitare o disabilitare l'attività di intercettazione delle comunicazioni per l'adattatore per comunicazioni.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
		***	(**)

Formato:
COM(*n*) ON
COM(*n*) OFF
COM(*n*) STOP

Note: *n* è il numero dell'adattatore per le comunicazioni (1 o 2).

Deve essere eseguita una specifica COM(*n*) ON per consentire alla specifica ON COM(*n*) di eseguire l'intercettazione. Se nella specifica ON COM(*n*) è stato specificato un numero di riga diverso da zero, il BASIC controlla ogni volta che viene eseguita una nuova specifica se qualche carattere è entrato nell'adattatore per le comunicazioni.

Se COM(*n*) è OFF, non viene eseguita alcuna intercettazione e l'attività di comunicazione non viene ricordata anche se eseguita.

Se è stata eseguita una specifica COM(*n*) STOP, non può aver luogo nessuna intercettazione. Tuttavia qualsiasi attività di comunicazione eseguita viene ricordata in modo che si verifichi un'immediata intercettazione quando COM(*n*) ON viene eseguita.

COMMON (Specifica)

Scopo: Passare le variabili ad un programma concatenato.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	(**)

Formato: COMMON *variabile*[,*variabile*]...

Note:

variabile è il nome di una variabile che deve essere passata al programma concatenato. Le variabili a schiera vengono specificate aggiungendo "( )" al nome della variabile.

Si usa la specifica COMMON in unione con CHAIN. Le specifiche COMMON possono apparire in un qualsiasi punto del programma, sebbene sia raccomandabile usarle all'inizio. Nel programma può apparire un numero qualsiasi di specifiche COMMON, ma la stessa variabile non può apparire in più di una specifica COMMON. Se devono essere passate tutte le variabili, usare CHAIN con l'opzione ALL ed omettere la specifica COMMON.

Non è necessario che le schiere passate siano dimensionate nel programma concatenato.

Esempio: Questo esempio concatena il programma PROG3 nel minidisco dell'unità A: e passa la schiera D con le variabili A, BEE1, C e G$.

```
100 COMMON A,BEE1,C,D(),G$
110 CHAIN "A:PROG3"
```

CONT (Comando)

Scopo: Riprendere l'esecuzione del programma dopo un'interruzione.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	

Formato: CONT

Note: Si può utilizzare il comando CONT per riprendere l'esecuzione del programma dopo aver premuto i tasti Ctrl-Interr, dopo l'esecuzione di una specifica STOP o END o dopo il verificarsi di un errore. L'esecuzione continua dal punto in cui si è verificata l'interruzione. Se l'interruzione si è verificata dopo una richiesta da una specifica INPUT, l'esecuzione continua con la ristampa della richiesta.

Normalmente CONT viene usata con STOP per la ricerca degli errori. Quando si arresta l'esecuzione, è possibile esaminare o modificare i valori delle variabili usando le specifiche in modo diretto. Poi si può utilizzare CONT per riprendere l'esecuzione, oppure GOTO in modo diretto, che riprende l'esecuzione da un determinato numero di riga.

CONT non è valido se il programma è stato editato durante l'interruzione.

CONT (Comando)

Esempio: Nel seguente esempio si crea una lunga iterazione.

```
10 FOR A=1 TO 50
20 PRINT A;
30 NEXT A
RUN
 1  2  3  4  5  6  7  8  9  10  11  12
 13  14  15  16  17  18  19  20  21  22
 23  24  25  26  27  28  29
```

(A questo punto si interrompe l'iterazione premendo i tasti Ctrl-Interr).

```
   .
   .
   .
Break in 20

CONT
 30  31  32  33  34  35  36  37  38  39
 40  41  42  43  44  45  46  47  48  49
 50
```

COS (Funzione)

Scopo: Fornire la funzione trigonometrica del coseno.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: $v=\text{COS}(x)$

Note:

x è l'angolo di cui si deve calcolare il coseno. Il valore di x deve essere in radianti. Per eseguire la conversazione da gradi in radianti, moltiplicare i gradi per PI/180, dove PI=3.141593.

In BASIC 2.Ø e successivi livelli, questo calcolo può essere eseguito a precisione doppia specificando **/D** nella riga di comando BASIC quando il BASIC è caricato inizialmente. Far riferimento a "Opzioni nel comando BASIC" del manuale BASIC.

Esempio: Questo esempio mostra che il coseno di PI radianti è uguale a -1. Poi calcola il coseno di 180 gradi convertendo prima i gradi in radiante (180 gradi corrispondono a PI radianti).

```
1Ø PI=3.141593
2Ø PRINT COS(PI)
3Ø DEGREES=18Ø
4Ø RADIANS=DEGREES*PI/18Ø
5Ø PRINT COS(RADIANS)
RUN
-1
-1
```

CSNG (Funzione)

Scopo: Convertire *x* in un numero a precisione singola.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v*=CSNG (*x*)

Note:

x è un'espressione numerica che verrà convertita in precisione singola.

Per la conversione si usano le regole descritte al Capitolo 3 nella sezione "Come il BASIC converte i numeri da una precisione ad un'altra".

Vedere inoltre le funzioni CINT e CDBL per informazioni sulla conversione dei numeri in interi o a precisione doppia.

Esempio: Il valore del numero a precisione doppia A# è arrotondato alla settima cifra e ritornato come CSNG (A#).

```
10 A#= 975.3421222#
20 PRINT A#; CSNG(A#)
RUN
 975.3421222  975.3421
```

CSRLIN (Variabile)

Scopo: Fornire la coordinata verticale del cursore.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v*=CSRLIN

Note: la variabile CRSLIN fornisce il numero di riga del cursore sulla pagina attiva (per una spiegazione della pagina attiva, vedere la specifica SCREEN). Il valore fornito sarà compreso tra 1 e 25.

La funzione POS fornisce la colonna di posizione del cursore. Fare riferimento alla funzione POS.

Fare riferimento alla specifica LOCATE per informazioni su come definire la riga del cursore.

Esempio: Questo esempio salva le coordinate del cursore nelle variabili X e Y, poi sposta il cursore alla riga 24 per inserirvi le parole "HI MOM". Il cursore viene quindi riportato alla posizione originaria.

```
10 Y = CSRLIN 'memorizza la riga corrente
20 X = POS(0) 'memorizza la colonna corrente
29 'stampa HI MOM sulla riga 40
30 LOCATE 24,1: PRINT "HI MOM"
40 LOCATE Y,X 'ripristina la posizione
```

CVI, CVS, CVD (Funzioni)

Scopo: Convertire in numeriche le variabili a stringa.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	***

Formato: *v*=CVI(*stringa di 2 byte*)

v=CVS(*stringa di 4 byte*)

v=CVD(*stringa di 8 byte*)

Note: I valori numerici letti da un file ad accesso casuale devono essere convertiti da stringhe in numeri. CVI converte una stringa di due byte in un intero. CVS converte una stringa di quattro byte in un numero a precisione singola. CVD converte una stringa di otto byte in un numero a precisione doppia.

Le funzioni CVI, CVS e CVD *non* modificano i byte del dato attuale. Viene modificato solo il modo della loro interpretazione da parte del BASIC.

Vedere anche le funzioni MKI$, MKS$, MKD$ e l'appendice A del manuale BASIC "Immissione ed emissione su disco nel BASIC".

CVI, CVS, CVD (Funzioni)

Esempio: Questo esempio usa un file ad accesso casuale (#1) che ha dei campi definiti come alla riga 100. La riga 110 legge un record dal file. La riga 120 usa la funzione CVS per interpretare i primi quattro byte (N$) del record come un numero a precisione singola. N$, probabilmente, era in origine un numero scritto nel file usando la funzione MKS$.

```
100 FIELD #1,4 AS N$, 12 AS B$
110 GET #1
120 Y=CVS(N$)
```

DATA (Specifica)

Scopo: Memorizzare le costanti numeriche e a stringa lette dalla specifica READ del programma.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: DATA *costante*[,*costante*]...

Note:

costante può essere una costante numerica o a stringa. Nella lista non sono consentite espressioni. Le costanti numeriche possono essere in qualsiasi formato, intero, punto decimale fisso, punto decimale mobile, esadecimale o ottale. Non è necessario che, nella specifica DATA, le costanti a stringa vengano racchiuse tra doppi apici a meno che non contengano virgole, due punti, o spazi significativi iniziali o finali.

Le specifiche DATA non sono eseguibili e possono essere poste in un qualsiasi punto del programma. Una specifica DATA può contenere tante costanti quante ne contiene una riga ed in un programma può essere usato un numero qualsiasi di specifiche DATA. Le informazioni contenute nelle specifiche DATA possono essere considerate come una lista continua di elementi, senza tener conto di quanti elementi si trovino in una riga o di dove le righe siano poste nel programma. Le specifiche READ accedono a quelle DATA in base al numero di riga.

DATA
(Specifica)

Il tipo di variabile (numerica o stringa) fornito nella specifica READ deve corrispondere alla relativa costante nella specifica DATA, altrimenti si verfica un **Syntax error.**

L'apice (') non può essere usato per aggiungere un commento alla fine della specifica DATA. Se ciò viene fatto il BASIC lo considera parte della stringa. Si può tuttavia aggiungere una nota usando :REM.

Si può utilizzare la specifica RESTORE per rileggere le informazioni da qualsiasi riga nella lista delle specifiche DATA (vedere la specifica RESTORE).

Esempio: Vedere l'esempio della specifica READ.

DATE$
(Variabile e Specifica)

Scopo: Impostare o ricercare la data.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	***

Formato: Come variabile:

*v$* =DATE$

Come specifica:

DATE$=*x$*

Note: **Per la variabile (v$=DATE$):**

Viene fornita una stringa di 10 caratteri nella forma *mm-gg-aaaa,* dove *mm* è il mese, *gg* è il giorno e *aaaa* l'anno. La data può essere impostata dal DOS prima di entrare nel BASIC.

Per la specifica (DATE$=x$):

x$* è un'espressione a stringa utilizzata per impostare la data attuale. Si può immettere *x$ in uno dei seguenti modi:

mm-gg-aa
mm/gg/aa
mm-gg-aaaa
mm/gg/aaaa

DATE$
(Variabile e Specifica)

L'anno deve essere compreso tra 1980 e 2099. Se si usa solo una cifra per il mese o il giorno, si assume uno Ø (zero) davanti ad essa. Se si indica solo una cifra per l'anno, si aggiunge uno zero per renderlo di due cifre. Se si indicano solo due cifre per l'anno, si assume che esso sia 19*aa*.

Esempio: Nell'esempio si imposta la data al 29 Agosto 1984. Si noti che, leggendo la data al contrario con la funzione DATE$, è stato inserito uno zero davanti al mese per renderlo di due cifre e l'anno è diventato 1984. Inoltre il mese, il giorno e l'anno sono separati da trattini anche se sono stati immessi con barre.

```
1Ø DATE$= "8/29/84
2Ø PRINT DATE$
RUN
Ø8-29-1984
```

DEF FN (Specifica)

Scopo: Definire e denominare una funzione che si scrive.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: DEF FN*nome*[(*arg*[,*arg*]...)] =*espressione*

Note:

nome è un nome di variabile valido. Questo nome, preceduto da FN, diventa il nome della funzione.

arg è un argomento. E' un nome di variabile nella definizione della funzione che sarà sostituito da un valore quando la funzione è richiamata. Gli argomenti della lista rappresentano, sulla base di uno-a-uno, i valori forniti quando la funzione è richiamata.

espressione definisce il valore di ritorno della funzione. Il tipo dell'*espressione* (numerica o stringa) deve corrispondere a quello dichiarato dal *nome*.

La definizione della funzione è limitata ad una specifica. Gli argomenti (*arg*) che appaiono nella definizione della funzione servono solo per definirla; non interessano le variabili di programma aventi lo stesso nome. Un nome di variabile usato *nell'espressione* non deve apparire nella lista degli argomenti. In caso contrario, il valore dell'argomento viene fornito quando si richiama la funzione. Altrimenti si usa il valore attuale della variabile.

DEF FN (Specifica)

Il tipo di funzione determina se la funzione fornisce un valore numerico o a stringa. Il tipo della funzione è dichiarato dal *nome,* nello stesso modo in cui vengono dichiarate le variabili. Vedere "Come dichiarare i tipi di variabile" nel Capitolo 3 del manuale BASIC. Se il tipo di *espressione* (stringa o numerica) non corrisponde a quello della funzione, si verifica l'errore **Type mismatch.** Se la funzione è numerica il valore dell'*espressione* viene convertito nella precisione specificata dal *nome* prima che venga restituito alla specifica chiamante.

Prima di poter richiamare una funzione questa deve essere definita mediante una specifica DEF FN. Se si richiama una funzione prima di averla definita, si verifica un errore di **Undefined user function.** D'altra parte, una funzione può essere definita più di una volta. Si usa la definizione eseguita più recentemente. Funzioni ricorrenti non sono supportate.

DEF FN non è valida in modo diretto.

Esempio: In questo esempio la riga 20 definisce la funzione FNAREA, che calcola l'area di un cerchio con raggio R. La funzione viene richiamata a riga 40.

```
10 PI=3.141593
20 DEF FNAREA(R)=PI*R^2
30 INPUT "Radius? ",RADIUS
40 PRINT "Area is" FNAREA(RADIUS)
RUN
Radius?
```

(Supponiamo di rispondere 2)

```
Radius? 2
Area is 12.56637
```

DEF FN (Specifica)

Scopo: Definire e denominare una funzione che si scrive.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: DEF FN*nome*[(*arg*[,*arg*]...)] =*espressione*

Note:

nome è un nome di variabile valido. Questo nome, preceduto da FN, diventa il nome della funzione.

arg è un argomento. E' un nome di variabile nella definizione della funzione che sarà sostituito da un valore quando la funzione è richiamata. Gli argomenti della lista rappresentano, sulla base di uno-a-uno, i valori forniti quando la funzione è richiamata.

espressione definisce il valore di ritorno della funzione. Il tipo dell'*espressione* (numerica o stringa) deve corrispondere a quello dichiarato dal *nome*.

La definizione della funzione è limitata ad una specifica. Gli argomenti (*arg*) che appaiono nella definizione della funzione servono solo per definirla; non interessano le variabili di programma aventi lo stesso nome. Un nome di variabile usato *nell'espressione* non deve apparire nella lista degli argomenti. In caso contrario, il valore dell'argomento viene fornito quando si richiama la funzione. Altrimenti si usa il valore attuale della variabile.

DEF FN (Specifica)

Questo è un esempio con due argomenti:

```
10 DEF FNMUD(X,Y)=X-(INT(X/Y)*Y)
20 A = FNMUD(7.4,4)
30 PRINT A
RUN
 3.4
```

DEF SEG (Specifica)

Scopo: Definire l'attuale "segmento" di memoria. Una successiva definizione BLOAD, BSAVE, CALL, PEEK, POKE o DEF USR definisce il reale indirizzo fisico dell'operazione come posizione relativa in questo segmento.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: DEF SEG [=*indirizzo*]

Note: *indirizzo* è un'espressione numerica compresa tra Ø e 65535.

L'impostazione iniziale del segmento, quando viene avviato il BASIC, è segmento di dati del BASIC (DS). Questo segmento è l'inizio dello spazio di lavoro dell'utente in memoria. Se si esegue una specifica DEF SEG che modifica il segmento, il valore *non* è ripristinato al DS del BASIC quando si emette un comando RUN.

Se nella specifica DEF SEG si omette indirizzo, il segmento viene impostato al segmento di dati del BASIC.

Il valore di quest'ultimo può essere trovato nell'indirizzo Ø, posizione relativa &HØ51Ø e &HØ511.

Se viene dato l'indirizzo, esso deve essere un valore basato su un limite di 16 byte. Il valore è spostato a sinistra di 4 bit (moltiplicati per 16) per formare l'indirizzo di segmento per la successiva operazione. Cioè se, l'indirizzo è in esadecimale, si aggiunge uno Ø (zero) per ottenere l'attuale indirizzo di segmento. Il BASIC non esegue alcun controllo per assicurarsi che il valore del segmento sia valido.

DEF SEG (Specifica)

DEF e SEG devono essere separate da uno spazio, altrimenti il BASIC interpreta la specifica DEFSEG=100; il che significa "assegnare il valore 100 alla variabile DEFSEG."

Qualsiasi valore immesso, non compreso nella gamma indicata, causa l'errore **Illegal funtion call.** Viene conservato il precedente valore.

Per maggiori informazioni sull'uso di DEF SEG, vedere l'appendice B "Sottoprogramma in linguaggio macchina".

Esempio: Il primo esempio memorizza un segmento nel segmento dati del BASIC.

```
DEF SEG ' ripristina il segmento al segmento dati del BASIC
```

Nel secondo esempio, la memoria di transito dello schermo per l'Adattatore per il video grafico a colori è al segmento esadecimale B800, pos. relativa 0. Poiché i segmenti sono specificati sui limiti di 16 byte, l'ultima cifra esadecimale viene omessa nella specifica DEF SEG.

```
DEF SEG=&HB800
```

DEF tipo (Specifiche)

Scopo: Dichiarare i tipi di variabile come interi, precisione singola, precisione doppia o stringhe.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: DEF *tipo lettera* [*-lettera*] [,*lettera* [*-lettera*]]...

Note: *tipo* può essere INT, SNG, DBL o STR.

lettera può essere una lettera dell'alfabeto (A-Z).

Una specifica DEF *tipo* dichiara che i nomi di variabile che iniziano con la lettera o le lettere specificate saranno quel tipo di variabile. Tuttavia, un carattere di dichiarazione del tipo (%, !, #, $) ha sempre la precedenza su una specifica DEF *tipo* nell'immissione di una variabile. Fare riferimento a "Come dichiarare i tipi di variabile", al Capitolo 3 del manuale BASIC.

Se non esistono specifiche di dichiarazione del tipo, il BASIC assume che tutte le variabili senza caratteri di dichiarazione siano variabili a precisione singola.

Se si usano le specifiche di dichiarazione del tipo, esse devono trovarsi all'inizio del programma. La specifica DEF *tipo* deve essere eseguita prima di utilizzare una delle variabili che dichiara.

DEF tipo (Specifiche)

Esempio: La riga 10 dichiara che tutte le variabili che iniziano con la lettera L, M, N, O, P devono essere a precisione doppia.

La riga 20 fa in modo che tutte le variabili che iniziano con la lettera A siano stringhe.

La riga 30 dichiara che tutte le variabili che iniziano con la lettera X, D, E, F, G o H devono essere intere.

```
10 DEFDBL L-P
20 DEFSTR A
30 DEFINT X,D-H
40 ORDER = 1#/3: PRINT ORDER
50 ANIMAL = "CAT": PRINT ANIMAL
60 X=10/3: PRINT X
RUN
 .3333333333333333
CAT
 3
```

DEF USR (Specifica)

Scopo: Specificare l'indirizzo iniziale di un sottoprogramma in linguaggio macchina, successivamente richiamato dalla funzione USR.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: DEF USR [*n*] = *posizione relativa*

Note:

n può essere qualsiasi cifra da Ø a 9. Identifica il numero del programma USR di cui si specifica l'indirizzo. Se si omette *n*, si assume DEF USRØ.

pos. relativa è un'espressione intera compresa tra Ø e 65535. Il valore della *posizione relativa* è aggiunto al valore di segmento attuale per ottenere l'indirizzo iniziale reale del programma USR. Vedere la specifica DEF SEG.

In un programma può apparire qualsiasi numero di specifiche DEF USR consentendo così l'accesso a tutti i sottoprogrammi necessari. Per la posizione relativa si usa il valore usato più recentemente.

DEF USR (Specifica)

Esempio: Questo esempio carica un sottoprogramma in linguaggio macchina in una schiera. La *n* della riga 60 è determinata dall'ampiezza del sottoprogramma. La pos. relativa passata a DEF USR è la posizione relativa del segmento dati del BASIC.

```
10 OPTION BASE 1
20 DEFINT A-Z
30 'Definisce tutte le variabili prima di VARPTR
40 SUBRT=0: I=0: J=0
50 'Dimensiona la schiera per il sottoprogramma
60 DIM ARRAY (n)
70 'Ottiene la posizione relativa del 1° elemento
   della schiera nel segmento dati del BASIC
80 SUBRT = VARPTR(ARRAY(1))
90 'Carica il programma nella schiera
100 BLOAD "ASMFILE",SUBRT
    .
    .
    .
1000 'Invia la posizione relativa a DEF USR
1010 DEF USR0 = VARPTR(ARRAY(1))
1020 'Esegue il sottoprogramma
1030 J = USR0(I)
```

DELETE (Comando)

Scopo: Cancellare righe di programma.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	

Formato: DELETE [*riga1*] [-*riga2*]

DELETE [*riga1*-]

Note: *riga 1* è il numero della prima riga da cancellare.

riga 2 è il numero dell'ultima riga da cancellare.

Il comando DELETE cancella dal programma il gruppo di righe specificato. Il BASIC ritorna sempre a livello di comando dopo l'esecuzione di DELETE.

DELETE [*riga1*-] cancella tutte le righe a partire dal numero specificato fino alla fine del programma. Questa forma è disponibile per il BASIC livello 2.Ø e successivi livelli.

Per indicare la riga attuale, è possibile usare un punto (.) invece del numero di riga. Se si specifica un numero di riga che non esiste nel programma, si verifica il seguente errore: **"Illegal function call".**

Esempio: Questo esempio cancella la riga 4Ø:

```
DELETE 4Ø
```

DELETE (Comando)

Il successivo esempio cancella le righe comprese tra 40 e 100 incluse:

```
DELETE 40-100
```

Il seguente esempio cancella dalla riga 40 fino alla fine del programma:

```
DELETE 40-
```

L'ultimo esempio cancella tutte le righe fino alla 40 compresa:

```
DELETE -40
```

DIM
(Specifica)

Scopo: Specificare i valori massimi per gli indici di variabili a schiera e di conseguenza assegnare la memoria.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: DIM *variabile*(*indici*) [,*variabile* (*indici*)]...

Nota: *variabile* è il nome da utilizzare per la schiera.

indici è una lista di espressioni numeriche, separate da virgole, che definiscono le dimensioni della schiera.

Se eseguita, la specifica DIM imposta tutti gli elementi delle schiere numeriche specificate ad un valore iniziale zero. Gli elementi di schiera a stringa, sono tutti a lunghezza variabile, con un valore iniziale nullo (lunghezza zero).

Se si usa un nome di schiera senza una specifica DIM, si assume che il valore massimo del suo indice sia 10. Se si usa un indice superiore al massimo specificato, si verifica l'errore **Subscript out of range.**

Il valore minimo per un indice è sempre 0, a meno che non sia specificato altro con la specifica OPTION BASE (vedere la specifica "OPTION BASE"). Il numero massimo di dimensioni per una schiera è 255.

Se si tenta di dimensionare una schiera più di una volta si verifica un errore di **Duplicate definition.** Tuttavia si può usare la specifica ERASE per cancellare una schiera in modo da poterla ancora dimensionare. Per maggiori informazioni sulle schiere, vedere il Capitolo 3 del manuale BASIC.

DIM (Specifica)

Esempio: Questo esempio crea due schiere: una schiera numerica monodimensionale denominata SIS con 13 elementi, da SIS (Ø) a SIS(12) ed una schiera a stringa bi-dimensionale denominata WRR$, con tre righe e tre colonne.

```
1Ø WRRMAX=2
2Ø DIM SIS(12) , WRR$(WRRMAX,2)
3Ø DATA 26.5, 37, 8, 29, 8Ø, 9.9, &H8ØØ
4Ø DATA 7, 18, 55, 12, 5, 43
5Ø FOR I=Ø TO 12
6Ø READ SIS(I)
7Ø NEXT I
8Ø DATA SHERRY, ROBERT, "A:"
9Ø DATA "HI, SCOTT", HELLO, GOOD-BYE
1ØØ DATA BOCA RATON, DELRAY, MIAMI
11Ø FOR I=Ø TO 2:FOR J=Ø TO 2
12Ø READ WRR$(I,J)
13Ø NEXT J,I
14Ø PRINT SIS(3); WRR$(2,Ø)
RUN
 29 BOCA RATON
```

DRAW
(Specifica)

Scopo: Disegnare un oggetto come specificato dalla *stringa*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
		***	(**)

Solo modo grafici.

Formato: DRAW *stringa*

Note: Si usa la specifica DRAW per eseguire un disegno usando un *linguaggio di definizione dei grafici.* I comandi del linguaggio sono contenuti nella *stringa* dell'espressione. La stringa definisce un oggetto che sarà disegnato quando il BASIC esegue la specifica DRAW. Durante l'esecuzione, il BASIC esamina il valore di *sringa* ed interpreta i comandi di una lettera dai contenuti della stringa. Quando viene dato un comando di spostamento una linea viene tracciata dall'ultimo punto referenziato.

n nei comandi seguenti quantifica lo spostamento. Il numero di punti spostati è *n* volte il fattore di spostamento (impostato dal comando S). I comandi di spostamento sono esposti dettagliatamente nella pagina successiva.

DRAW (Specifica)

U n Spostamento verso l'alto

D n Spostamento verso il basso.

L n Spostamento a sinistra.

R n Spostamento a destra.

E n Spostamento diagonale verso l'alto e a destra.

F n Spostamento diagonale verso il basso e a destra.

G n Spostamento diagonale verso il basso e a sinistra.

H n Spostamento diagonale verso l'alto e a sinistra.

M x,y Spostamento assoluto o relativo. Se prima di *x* si trova un segno più (+) o meno (–), è relativo. Altrimenti è assoluto.

I seguenti due prefissi di comandi possono precedere uno dei comandi di spostamento elencati in precedenza.

B Spostamento, ma non disegna punti.

N Spostamento, ma ritorna, al termine, alla posizione originale.

Sono disponibili anche i seguenti comandi:

A n Imposta l'angolo *n*. *n* può essere compreso tra 0 e 3, dove 0 è 0 gradi, 1 è 90, 2 è 180 e 3 è 270. Le figure ruotate di 90 o 270 gradi sono disposte in modo da apparire delle stesse dimensioni di 0 o 180 gradi su uno schermo con proporzione standard di 4/3.

DRAW
(Specifica)

TA n Ruota l'angolo *n*. *n* può variare da −360 a +360. Se è positivo (+), la rotazione è in senso antiorario, se negativo (−) in senso orario. Se si immette un valore al di fuori dell'intervallo −360 +360 si causa l'errore **Illegal function call**. Il comando è disponibile in BASIC 2.Ø e successive versioni.

C n Imposta il colore *n*. *n* può essere compreso tra Ø-3 a media risoluzione e Ø-1 ad alta risoluzione. A media risoluzione *n* sceglie il colore della attuale tavolozza, definita dalla specifica COLOR. Ø è il colore di fondo. Il valore assunto in mancanza di indicazioni per il primo piano è sempre il massimo compatibile col tipo di schermo corrente: 3 a media risoluzione, 1 ad alta.

S n Imposta il fattore di scala. *n* può essere compreso tra 1 e 255. *n* diviso 4 è il fattore di scala. Per esempio, se *n*=1 allora il fattore di scala è 1/4. Il fattore, moltiplicato per le distanze indicate da U, D, L, R, E, F, G, H ed i relativi comandi M, fornisce lo spostamento reale. Il valore assunto è 4, in modo che il fattore di scala sia 1.

X variabile; Esegue la sottostringa: questo consente di eseguire una seconda stringa all'interno di un'altra.

P figura, bordo

Il parametro *figura* imposta il colore della figura, il parametro *bordo* quello del bordo. Il parametro *figura* è un numero intero. Esso sceglie un colore dalla gamma del modo schermo in uso. A media risoluzione, questo colore corrisponde ad uno dei colori della tavolozza corrente definita dalla specifica COLOR. A media risoluzione sono disponibili 4 colori (Ø-3). Ad alta risoluzione i colori disponibili sono 2 (Ø-1): Ø indica il nero ed 1 il bianco. Il parametro *bordo* indica il colore del bordo della figura che deve essere disegnata al suo interno e viene scelto dalla gamma di attributi del modo schermo in uso. Sia *figura* che *bordo* devono essere specificati altrimenti si verifica un errore. Il comando è disponibile col BASIC 2.Ø e successive versioni.

In tutti questi comandi, l'argomento *n, x o y* può essere una costante come 123, oppure può essere =**variabile;** dove *variabile* è il nome di una variabile numerica. Il punto e virgola (;) è indispensabile se si usa una variabile in questo modo o nel comando X. Altrimenti il punto e virgola è facoltativo nei comandi. Nella *stringa* si ignorano gli spazi. Ad esempio, è possibile utilizzare le variabili in un comando di spostamento in questo modo:
M+=X1;-=X2;
Le variabili possono essere specificate anche nella forma VARPTR$*variabile* invece di =**variabile;**. Questa è l'unica forma che può essere usata nei programmi compilati. Ad esempio:

DRAW (Specifica)

Metodo	Metodo alternativo
DRAW "XA$;"	DRAW "X"+VARPTR$(A$)
DRAW "S=SC";	DRAW "S="+VARPTR$(SC)

Il comando X può rappresentare una parte molto utile di DRAW. Permette infatti di definire dei segmenti di un disegno in diverse variabili X e cambiare successivamente queste variabili in una singola specifica DRAW. In questo modo si possono creare delle stringhe lunghe più di 255 caratteri. Per esempio, se si sta disegnando una casa con un camino ed un albero, ognuno di questi aspetti può essere definito in una variabile X cosicché la specifica DRAW può apparire in questa forma:

```
DRAW "XHOUSE$;XCHIM$;XTREE$;"
```

La proporzione visiva dello schermo determina poi gli spazi dei punti orizzontali, verticali e diagonali. La specifica DRAW non tiene conto delle proporzioni visive del modo schermo in uso; cioè DRAW "R50 V50" traccia 50 punti a destra e 50 in alto ma le due linee non sembrano avere la stessa lunghezza.

La proporzione visiva è usata per correggere la forma di un oggetto disegnato su una superficie non lineare. L'idea sarebbe quella di disegnare un quadrato che sembri un quadrato.

DRAW (Specifica)

Se ci sono 640 x 640 punti sullo schermo uniformemente spaziati lungo gli assi x e y, la proporzione visiva è 1 a 1 oppure 1/1. Questa è una specifica ideale. Se si esegue la specifica:

```
DRAW "R100 D100 L100 U100"
```

il disegno si presenterà quadrato.

Comunque non è questo il caso del BASIC, il quale supportà 2 tipi di risoluzioni di schermo, ciascuna delle quali con una propria proporzione visiva

Risoluzione	**Punti**	**Proporzione visiva**
Media	320x200	5/6
Alta	640x200	5/12

Al fine di poter ottenere un disegno che appaia quadrato in ciascuna risoluzione è necessario rapportare l'asse y con il corrispondente valore di proporzione visiva, ovvero l'asse x con il suo inverso. Per esempio per disegnare un quadrato alto 100 occorre rapportare l'asse x come segue:

```
10 '100*6/5 is 120
20 DRAW "U100 R120 D100 L120
```

DRAW
(Specifica)

Esempio: Per disegnare un quadrato:

```
10 SCREEN 1
20 A=2030 DRAW "U=A;R=A;D=A;L=A;"
```

Per disegnare un quadrato e colorarlo internamente:

```
10 DRAW "U50R50D50L50" 'DRAW a box
20 DRAW "BE10" 'Muove in alto e a destra nel quadrato
30 DRAW "P1,3" 'Colora l'interno
```

Per disegnare un triangolo:

```
10 SCREEN 1
20 DRAW "E15 F15 L30"
```

Per creare una "stella cadente":

```
10 SCREEN 1,0: COLOR 0,0: CLS
20 DRAW "BM300,25" ' initial point
30 STAR$= "M87,17 M-17,-12 M+20,0 M-17,12 M+7,-17"
40 FOR SCALE=1 TO 40 STEP 2
50 DRAW "C1;S=SCALE; BM-2,0;XSTAR$;"
60 NEXT
```

Per disegnare raggi di cerchio:

```
10 SCREEN 1,0:CLS
20 FOR D=0 TO 360 STEP 10
30 DRAW "TA=D; NU50"
40 NEXT D
```

EDIT (Comando)

Scopo: Visualizzare una riga per modifiche.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	

Formato: EDIT *riga*

Note:

riga è il numero di una riga esistente nel programma. Se la riga non esiste, si verifica un errore.

La specifica EDIT visualizza semplicemente la riga specificata e posiziona il cursore sotto la prima cifra del numero di riga. La riga può essere modificata come descritto nella sezione "L'editore programmi BASIC" del manuale BASIC.

Invece del numero di riga è possibile utilizzare un punto (.) per far riferimento alla riga attuale. Ad esempio, se si è appena immessa una riga e si desidera tornare indietro per modificarla, il comando EDIT rivisualizza la riga per l'editazione.

Si può utilizzare anche LIST per visualizzare le righe di programma per la modifica. Vedere il comando LIST.

END (Specifica)

Scopo: Terminare l'esecuzione del programma, chiudere tutti i file e ritornare al livello di comando.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: END

Note: Le specifiche END possono essere poste in qualsiasi punto del programma per terminare l'esecuzione. END si differenzia da STOP in due modi:

- END non produce la stampa di un messaggio di interruzione.
- END chiude tutti i file.

Una specifica END alla fine di un programma è facoltativa. Il BASIC ritorna sempre a livello di comando dopo l'esecuzione di una specifica END.

Esempio: Questo esempio termina il programma se K è maggiore di 1000: altrimenti, il programma salta alla riga 20.

```
100 IF K>1000 THEN END ELSE GOTO 20
```

ENVIRON (Specifica)

Scopo: Modificare i parametri nella tabella di ambiente del BASIC. La specifica ENVIRON è usata per modificare il parametro "PATH" per elaborazioni secondarie o per passare dei parametri a tali elaborazioni creando un nuovo parametro ambiente. Vedere ENVIRON$, SHELL e il comando PATH del DOS.

Non disponibile per versioni antecedenti al BASIC 3.0.

Versioni:	Cassetta	Disco	Avanzato	Compilatore
		***	***	

Formato: ENVIRON *parametro=stringa*

Note: *parametro* è il nome del parametro, come ad esempio "PATH".

stringa è il testo che definisce il nuovo parametro

parametro deve essere separato dalla *stringa* mediante un segno di uguale o da uno spazio. ENVIRON prende come *parametro* qualsiasi carattere a sinistra del primo spazio bianco o del segno di uguale. Qualsiasi carattere (eccetto lo spazio in bianco ed il segno di uguale) dopo *parametro* è considerato *stringa*.

Se stringa è un carattere nullo o consiste solo in un punto e virgola ";" come ad esempio:

```
"PATH=;"
```

il parametro viene rimosso dalla tabella ambiente e la tabella stessa viene compressa.

ENVIRON (Specifica)

Se *parametro* non esiste, il nuovo parametro viene aggiunto alla fine della tabella.

Se *parametro* esiste, viene cancellato, ed aggiunto alla fine della tabella dopo che la stessa è stata compressa.

> **Nota:** Quando il BASIC viene chiamato, l'ampiezza della sua tabella di ambiente è uguale all'ampiezza della tabella di ambiente del DOS (arrotondata sino ai 16 successivi byte del limite del paragrafo). Il BASIC non può ampliare la propria tabella. Se si desiderano aggiungere degli elementi alla tabella BASIC, è necessario espandere la tabella dal DOS sino all'ampiezza necessaria all'applicazione prima di chiamare il BASIC.

Esempio: Con la seguente specifica si può creare un PATH nell'indirizzario radice sull'unità a disco A:

```
ENVIRON "PAT=A:\"
```

Ora si può richiamare il DOS dal proprio programma BASIC usando la specifica SHELL ed immettere qualsiasi comando DOS. Se è necessario un file su disco per eseguire il comando, il DOS ne ricerca automaticamente uno (.COM, .EXE, o .BAT) nell'indirizzario radice sull'unità A.

ENVIRON (Specifica)

```
SHELL            'richiama una copia di COMMAND.COM.
A> REM modifica l'indirizzo di "WORK" sul drive B
A> CD B:\WORK
B> REM Carica PROJ1 sotto DEBUG sebbene non sia
   specificata nessuna unità e disco. DEBUG e PROJ1 sono
   caricati su differenti unità.
B< DEBUG PROJ1
     .
     .
     .
B> REM ritorno al programma BASIC
B> EXIT (esce dal DOS e ritorna al programma BASIC)
```

Si può aggiungere un nuovo parametro alla tabella di ambiente:

```
ENVIRON "HELP = C:\HELP" 'definisce
                  il parametro del file chiamato "HELP"
CHDIR ENVIRON$ ("HELP")'modifica l'indirizzario di "HELP"
```

Si può togliere questo parametro dalla tabella con:

```
ENVIRON "HELP=;"'cancella il parametro "HELP"
                        dalla tabella.
```

L'ambiente creato dall'applicazione BASIC viene passato a COMMAND.COM quando viene richiamato dalla specifica SHELL. Questo rende possibile trasmettere dei parametri da un programma collaterale (BASIC) ad un programma dipendente attraverso la tabella ambiente.

> **Nota:** Per ulteriori informazioni è consigliabile vedere la "Funzione ENVIRON$" e la "Specifica SHELL" in questo manuale. Vedere anche il "Comando SET" del DOS e la "EXEC Function Call" nel manuale di riferimento tecnico DOS.

ENVIRON$ (Funzione)

Scopo: Richiamare e visualizzare la stringa specificata dalla tabella di ambiente del BASIC.

Non è valida per versioni antecedenti al BASIC 3.Ø.

Versioni:	Cassetta	Disco	Avanzato	Compilatore
		***	***	

Formato:

*v$* =ENVIRON$ (*parametro*)
oppure
*v$* =ENVIRON$ (*n*)

Note:

parametro è un'espressione a stringa contenente il parametro da richiamare.

n è un numero intero che restituisce un valore compreso tra 1 e 255.

Se viene usata, una stringa ENVIRON$ restituisce dalla tabella ambiente, una stringa contenente il testo che segue *parametro*. Se *parametro* non viene trovato, oppure nessun testo segue il segno di uguale (=), in questo caso viene restituita una stringa nulla.

Se viene usata una espressione numerica, ENVIRON$ fornisce una stringa contenente l'ultimo *parametro* della tabella ambiente insieme a *parametro* =testo. Se il *parametro* non esiste, viene restituita una stringa nulla.

ENVIRON$ distingue fra lettere maiuscole e minuscole. Se in tabella si immette questa espressione:

```
ENVIRON$ "load = high"
```

e si vuole verificare il successo dell'operazione usare la funzione ENVIRON$ in questo modo

```
PRINT ENVIRON$ ("load")
```

Se si immette invece

```
PRINT ENVIRON$ ("LOAD")
```

ENVIRON$ fornisce una stringa nulla in quanto "LOAD" a differenza di "load" non esiste nella tabella.

Esempio: Quando il DOS inizialmente viene caricato, imposta un parametro chiamato "COMSPEC" che informa il DOS su dove caricare il file COMMAND. COM, senza impostare un percorso.

Per vedere il contenuto iniziale della tabella ambiente immettere dal BASIC quanto segue:

```
PRINT ENVIRON$ (1)
```

si vedrà quindi sullo schermo:

```
PATH=
```

Se si immette:

```
PRINT ENVIRON$ (2)
```

si vedrà visualizzato:

```
COMSPEC = A:\ COMMAND.COM
```

Nota: Usando un disco fisso usare nel precedente esempio, *C:* invece di *A:*, per la specifica dell'unità a disco.

Se si immette :

ENVIRON$ (Funzione)

```
PRINT ENVIRON$ ("COMSPEC")
```

la risposta dell'elaboratore sarà

```
A:\COMMAND.COM
```

Il seguente programma salva la tabella ambiente del BASIC in una schiera in modo da poter essere modificata per una elaborazione dipendente. Dopo che l'elaborazione è completa, l'ambiente viene ripristinato.

```
10 DIM TABLE$(10) 'assume no more than 10 parms
20 PARMS = 1     'initial number of parameters
30 WHILE LEN(ENVIRON$(PARMS)) > 0
40 TABLE$(PARMS) = ENVIRON$(PARMS)
50 PARMS = PARMS+1
60 WEND
70 PARMS = PARMS - 1  'adjust to correct number
80 'now store new environment
90 ENVIRON "DATAIN = C:\DATAIN\INP.FIL"
100 ENVIRON "SORT.DAT = SORT.DAT<" +
   ENVIRON$ ("DATAIN") +">LPT1:"
    .
    .
    .
1000 SHELL ENVIRON$("SORT.DAT") 'data is sorted
1010 FOR I=1 TO PARMS
1020 ENVIRON TABLE$(I)  'restore parameters
1030 NEXT I
    .
    .
    .
```

Nota: Vedere anche la specifica "ENVIRON" e il comando "SHELL". Come pure il comando "SET" e la funzione EXEC nei manuali di riferimento DOS.

EOF (Funzione)

Scopo: Indicare una condizione di fine file.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: *v* = EOF(*numfile*)

Note: *numfile* è il numero indicato nella specifica OPEN.

La funzione EOF è utile per evitare un errore relativo all'immissione dopo la fine. EOF fornisce –1 (vero) se è stata raggiunta la fine del file specificato. Fornisce Ø in caso contrario.

EOF ha significato solo per un file aperto per un'immissione sequenziale da disco o cassetta, o per un file di comunicazioni. Un-1 per un file di comunicazioni significa che la memoria di transito è vuota.

EOF
(Funzione)

Esempio: Questo esempio legge le informazioni dal file sequenziale denominato "DATA". I valori sono letti nella schiera M fino a quando si raggiunge la fine del file.

```
10 OPEN "DATA" FOR INPUT AS #1
20 C=0
30 IF EOF(1) THEN END
40 INPUT #1,M(C)
50 C=C+1: GOTO 30
```

In BASIC 2.0 e successive versioni, EOF (0) fornisce la condizione di fine file sui dispositivi di immissione standard usati con il reindirizzamento dell'I/E.

ERASE (Specifica)

Scopo: Eliminare le schiere da un programma.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	

Formato: ERASE *nome schiera* [,*nome schiera*]...

Note: *nome schiera* è il nome di una schiera da cancellare.

Si può usare la specifica ERASE se si sta utilizzando poco spazio di memoria nell'esecuzione del programma. Dopo la cancellazione delle schiere, lo spazio di memoria, che era stato allocato per le schiere, può essere utilizzato per altri scopi.

Si può utilizzare ERASE anche per ridimensionare le schiere nel programma. Se si tenta di ridimensionare una schiera senza prima cancellarla, si verifica un errore di **Duplicate definition.**

Si usa il comando CLEAR per cancellare *tutte* le variabili dall'area di lavoro.

ERASE
(Specifica)

Esempio: Questo esempio usa la funzione FRE per illustrare l'utilizzo di ERASE per liberare la memoria. La schiera BIG utilizzava fino a circa 40K-byte di memoria (62808-21980) quando era stata dimensionata come BIG(100,100). Dopo la cancellazione, può essere ridimensionata in BIG(10,10) ed occupare poco più di 500 byte (62808-62289).

I valori reali forniti dalla funzione FRE possono essere diversi sull'elaboratore dell'utente.

```
10 START=FRE("")
20 DIM BIG(100,100)
30 MIDDLE=FRE("")
40 ERASE BIG
50 DIM BIG(10,10)
60 FINAL=FRE("")
70 PRINT START, MIDDLE,FINAL
RUN
 62808          21980         62289
```

ERDEV e ERDERV$ (Variabili)

Scopo: Variabili di sola lettura. Mantengono il codice di errore di interruzione 24 dell'errore di unità, ed il nome dell'unità che lo ha generato.

Non è disponibile per versioni BASIC antecedenti al BASIC 3.Ø.

Versioni:	Cassetta	Disco	Avanzato	Compilatore
		***	***	

Formato: *v* = ERDEV

*v$* = ERDEV$

Note: ERDEV è una variabile di sola lettura. Quando un errore nel DOS viene scoperto, ERDEV trattiene il codice errore INT 24 negli ultimi 8 bit mentre nei primi 8 contiene i bit 13, 14, 15 della parola attributo del blocco di testa dell'unità.

ERDEV$ è una variabile di sola lettura. Se l'errore è stato trovato su una unità, ERDEV$ contiene gli 8 byte carattere del nome dell'unità. Se l'errore non si trovava sull'unità, ERDEV$ contiene il nome di due caratteri dell'unità (A:,B:, C:, ecc).

ERDEV e ERDERV$
Variabili

Esempio: Aprire l'unità a disco B ed immettere:

```
FILE "B:"
```

il BASIC risponde

```
Disco non pronto
```

Immettere quindi:

```
PRINT ERDEV, ERDEV$
```

il BASIC risponde

```
2           B:
```

Nota: Facendo riferimento al manuale tecnico DOS sotto la lista del codice di errore INT24 si può notare che l'errore 2 indica **Drive not ready.** Gli 8 bit di ordine superiore (i bit della parola) sono tutti a zero. Come spiegato nel manuale di riferimento tecnico del DOS in particolare nella sezione "Attribute Field" sotto "Installable Device Drivers" i bit 13, 14, e 15 impostati a zero significano che B: è un'unità, che IOCTL non è supportato e che l'unità è in formato IBM.

Vedere anche la specifica IOCTL e la funzione IOCTL$.

ERDEV e ERDERV$
Variabili

Questo esempio simula un errore della stampatrice:

```
10 CLS
20 ON ERROR GOTO 50
30 LPRINT"The printer is ready"
40 PRINT"The printer is ready"
50 END
60 V$=HEX$(ERDEV)
70 PRINT "ERDEV = ";V$
80 D$=ERDEV$
90 PRINT "ERDEV$ = ";D$
100 RESUME NEXT
```

Se questo esempio viene eseguito con la stampatrice spenta il computer visualizzerà:

```
ERDEV = 8009
ERDEV$ = LPT1
```

Gli 8 bit inferiori (Ø-7) dell'equivalente binario sono uguali a 9 che significa come codice di errore INT24 **Printer out of paper.** Il significato dei bit 13, 14 e 15 del valore ottenuto da ERDEV è spiegato nella sezione "Attribute Field" del manuale di riferimento tecnico DOS sotto il capitolo "Installable Device Drivers".

ERR e ERL (Variabili)

Scopo: Fornire il codice di errore ed il numero di riga associato all'errore.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v* = ERR

v = ERL

Nota: La variabile ERR contiene il codice dell'ultimo errore e la variabile ERL contiene il numero della riga in cui si è verificato l'errore. Normalmente si usano le variabili ERR e ERL nelle specifiche IF...THEN per dirigere il flusso di programma nel programma di gestione dell'errore (vedere la specifica ON ERROR).

Se si prova ERL in una specifica IF...THEN, assicurarsi di porre il numero di riga sul lato destro dell'operatore di relazione, in questo modo:

```
IF ERL = numero riga THEN...
```

Il numero deve essere sul lato destro dell'operatore in modo che sia rinumerato da RENUM.

Se la specifica che ha causato l'errore era in modo diretto, ERL contiene 65535. Poiché non si vuole modificare questo numero durante RENUM, se si desidera provare se l'errore si è verificato in una specifica in modo diretto si deve usare la forma:

```
IF 65535 = ERL THEN...
```

ERR e ERL possono essere impostate usando la specifica ERROR.

ERR e ERL (Variabili)

I codici di errore BASIC sono elencati nell'appendice A "Messaggi di errore".

Esempio: Questo esempio serve a verificare se la levetta dell'unità a disco è aperta quando il programma necessita di accedere ad un file.

```
10 ON ERROR GOTO 100
20 OPEN "DATA" FOR INPUT AS #1
30 END
   .
   .
   .
100 IF ERR=71 THEN LOCATE 23,1:
    PRINT "DISK IS NOT READY": RESUME
```

ERROR (Specifica)

Scopo:

- Simulare il ripetersi di un errore BASIC; oppure
- Consentire di definire i propri codici di errore.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: ERROR *n*

Note: *n* deve essere un'espressione intera compresa tra 0 e 255.

Se il valore di *n* è uguale ad un codice di errore usato dal BASIC (vedere l'Appendice A "Messaggi di errore"), la specifica ERROR simula il ripetersi dell'errore. Se è stato definito un programma di gestione errori (con la specifica ON ERROR) il controllo viene passato al suddetto programma, altrimenti viene visualizzato il messaggio di errore corrispondente al codice e l'esecuzione si arresta. Vedere il primo esempio sotto riportato.

Per definire il proprio codice di errore, usare un valore diverso da quelli usati dal BASIC (è consigliabile usare i valori più alti a disposizione; ad esempio, valori superiori a 200). Questo nuovo codice di errore può essere provato in un programma di gestione dell'errore, come qualsiasi altro errore (vedere il secondo esempio).

Se si definisce in questo modo il proprio codice e non viene gestito in un programma di gestione dell'errore, il BASIC visualizza il messaggio di **Unprintable error** e l'esecuzione si arresta.

Esempio: il primo esempio simula un errore di stringa troppo lunga.

```
10 T = 15
20 ERROR T
RUN
String too long in line 20
```

L'esempio successivo fa parte di un programma di gioco che consente di fare scommesse. Usando un codice di errore 210, che non è utilizzato dal BASIC, il programma intercetta l'errore se si supera il limite.

```
100 ON ERROR GOTO 1000
110 INPUT  "WHAT IS YOUR BET";B
120 IF B > 5000 THEN ERROR 210
   .
   .
   .
1000 IF ERR = 210 THEN PRINT
     "HOUSE LIMIT IS $5000"
1010 IF ERL = 120 THEN RESUME 110
```

EXP (Funzione)

Scopo: Calcolare la funzione esponenziale.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: $v = \text{EXP}(x)$

Note: *x* può essere qualsiasi espressione numerica.

Questa funzione fornisce il numero matematico *e* elevato alla potenza di *x*. *e* è la base per i logaritmi naturali. Si verifica un'eccedenza se *x* è maggiore di 88.Ø2969.

In BASIC 2.Ø e successivi livelli questo calcolo può essere eseguito a precisione doppia specificando **/D** nella linea di comando BASIC quando il BASIC viene inizialmente caricato. Far riferimento a "Opzioni nel comando BASIC" nel manuale BASIC.

Esempio: Questo esempio calcola *e* elevato alla potenza (2-1), che è semplicemente *e*.

```
1Ø X = 2
2Ø PRINT EXP(X-1)
RUN
 2.718282
```

FIELD (Specifica)

Scopo: Allocare spazio per le variabili nella memoria di transito di un file ad accesso casuale.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	***

Formato: FIELD [#] *numfile, ampiezza* AS *varstringa* [,*ampiezza* AS *varstringa*]...

Note:

numfile è il numero con il quale è stato aperto il file.

ampiezza è un'espressione numerica che specifica il numero di posizioni carattere da allocare alla *variabile a stringa.*

varstringa è una variabile a stringa che sarà utilizzata per accedere ai file ad accesso casuale.

Una specifica FIELD definisce le variabili usate per estrarre dati da una memoria di transito ad accesso casuale dopo una funzione GET o per immettere dati nella memoria di transito per una funzione PUT.

La specifica:

```
FIELD 1, 20 AS N$, 10 AS ID$, 40 AS ADD$
```

alloca le prime 20 posizioni (byte) della memoria di transito del file ad accesso casuale alla variabile a stringa N$, le successive 10 posizioni a ID$ e le altre 40 a ADD$. FILED *non* pone realmente i dati nella memoria di transito del file ad accesso casuale. Questo viene fatto dalle specifiche LSET e RSET.

FIELD (Specifica)

FIELD non elimina nemmeno i dati dal file. Le informazioni vengono lette dal file nella memoria di transito del file ad accesso casuale con la specifica GET (file). Le informazioni vengono lette dalla memoria di transito facendo semplicemente riferimento alle variabili definite nella specifica FIELD.

Il numero totale di byte allocati in una specifica FIELD non deve superare la lunghezza del record, specificata quando è stato aperto il file. Altrimenti si verifica un errore di **Field overflow.**

Per lo stesso numero di file può essere eseguito qualsiasi numero di specifiche FIELD, e tutte le specifiche FIELD che sono state eseguite sono valide contemporaneamente. Ogni nuova specifica FIELD ridefinisce la memoria di transito dalla prima posizione carattere, in modo da avere più definizioni di campo per gli stessi dati.

> **Nota:** Attenzione nell'uso di un nome di variabile sulla quale sia stata eseguita una specifica FIELD, in una specifica di immissione o assegnazione. Quando il nome di una variabile è stato definito in una specifica FIELD, esso punta alla posizione esatta nella memoria di transito del file ad accesso casuale. Se si esegue una successiva specifica di immissione o LET con quel nome di variabile sul lato sinistro del segno di uguale, la variabile viene spostata nello spazio a stringa e non si trova più nella memoria di transito del file.

Vedere l'Appendice A del manuale BASIC per una completa spiegazione sull'uso dei file ad accesso casuale.

FIELD (Specifica)

Esempio: Questo esempio apre un file denominato "CUST" come file ad accesso casuale. La variabile CUSTNO$ viene assegnata alle prime 2 posizioni di ogni record, CUSTNAME$ alle successive 30 posizioni e ADDR$ alle successive 35. Le righe 30-50 immettono le informazioni nella memoria di transito, e la specifica PUT a riga 60 scrive la memoria di transito nel file. La riga 70 rilegge lo stesso record e la riga 90 visualizza i tre campi. Notare a riga 80 che è giusto usare un nome di variabile che è stato definito in una specifica FIELD sul lato *destro* di una specifica di assegnazione.

```
10 OPEN "A:CUST" AS#1
20 FIELD 1, 2 AS CUSTNO$, 30 AS CUSTNAME$,
     35 AS ADDR$
30 LSET CUSTNAME$="O'NEIL INC"
40 LSET ADDR$="50 SE 12TH ST, NY, NY"
50 LSET CUSTNO$=MKI$(7850)
60 PUT 1,1
70 GET 1,1
80 CNUM%= CVI(CUSTNO$): N$ = CUSTNAME$
90 PRINT CNUM%, N$, ADDR$
```

Il seguente programma mostra come creare una memoria di transito di un file ad accesso casuale con più specifiche FIELD in una riga.

```
10 OPEN  "FOO" AS #1
20 FIELD 1, 100 AS A$, 200 AS B$
30 FIELD 1, 300 AS DUMMY$,40 AS C$
```

Notare che nella riga 30 DUMMY$ muove il puntatore nella memoria di transito del file in modo da non perdere l'informazione della riga 20.

FILES (Comando)

Scopo: Visualizzare i nomi dei file che si trovano sull'indirizzario corrente di un minidisco. Il comando FILES del BASIC è simile al comando DIR del DOS.

Versioni:	Cassetta	Disco	Avanzato	Compilatore
		***	***	(**)

Formato: FILES [*specfile*]

Note: *specfile* è un'espressione a stringa per la specifica del file, deve essere conforme alle regole indicate nel paragrafo "Denominazione dei file" nel capitolo 3 del manuale BASIC altrimenti si verifica l'errore **Bad file name.** Se si omette *specfile,* vengono listati tutti i file che si trovano sull'indirizzario corrente del modulo a minidischi assunto per difetto dal DOS.

Vengono visualizzati tutti i file che corrispondono al nome del file. Quest'ultimo può contenere punti interrogativi (?). Un punto interrogativo corrisponde ad un carattere del nome o dell'estensione. Un asterisco (*) come primo carattere del nome o dell'estensione corrisponde ad un nome oppure ad un'estensione.

Se si specifica un'unità minidisco come parte di *specfile,* i file che sull'indirizzario in uso corrispondono al nome del file specificato verranno listati. Altrimenti si usa l'unità assunta per difetto dal DOS.

FILES (Comando)

Esempio:

```
FILES
```

Visualizza tutti i file dell'indirizzario corrente dell'unità minidisco assunta per difetto dal DOS.

```
FILES "*.BAS"
```

Visualizza tutti i file con l'estensione .BAS dell'indirizzario corrente dell'unità minidisco assunta per difetto dal DOS.

```
FILES "B:*.*"
```

Visualizza tutti i file del modulo B:

```
FILES "TEST??.BAS"
```

Lista tutti i file dell'indirizzario corrente dell'unità minidisco assunta per difetto dal DOS con un nome di file che inizia con TEST seguito da un massimo di due caratteri ed un'estensione .BAS.

Nel BASIC livello 2.Ø e successivi livelli si possono listare tutti i nomi dei file presenti sull'indirizzario corrente nell'unità B immettendo:

```
FILES "B:"
```

Oltre a listare tutti i nomi di file dell'indirizzario corrente dell'unità minidisco il BASIC visualizza anche il nome dell'indirizzario corrente e il numero di byte disponibili.

Usando indirizzari strutturati ad albero è consigliabile ricordare che ogni sottoindirizzario contiene due speciali registrazioni, che si possono visualizzare usando il comando FILES per listare il sottoindirizzario stesso. La prima contiene un punto al posto del nome del file che identifica questo "file" come un sottoindirizzario. La seconda contiene due punti al posto del nome del file.

FILES (Comando)

Ciò identifica l'indirizzario di livello superiore, che definisce questo sottoindirizzario. (BASIC 2.0 e successive versioni).

Questo esempio lista tutti i file del sottoindirizzario corrente chiamato LEVEL1 sull'unità A. Notare che l'indirizzario è vuoto.

```
FILES "A:\LEVEL1"

.       <DIR>          ..      <DIR>
32824 Bytes free
```

Il comando FILES può essere usato anche per listare file in altri indirizzari. L'esempio lista tutti i file nel sottoindirizzario LVL1. La barra inversa " \" deve seguire il nome dell'indirizzario.

```
FILES "LVL1\"
```

Questo esempio lista tutti i file nell'indirizzario LVL2 con una estensione .BAS.

```
FILES "LVL2\*.BAS"
```

FIX (Funzione)

Scopo: Troncare *x* ad un intero.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v* = FIX(*x*)

Note: *x* può essere qualsiasi espressione numerica.

FIX toglie tutte le cifre alla destra del punto decimale e fornisce il valore delle cifre alla sinistra dello stesso. La differenza tra FIX e INT è che FIX non fornisce il successivo numero inferiore quando *x* è negativo.

Vedere le funzioni INT e CINT che forniscono anche valori interi.

Esempio: Notare negli esempi come FIX *non* arrotondi la parte decimale quando esegue la conversione ad intero.

```
PRINT FIX(45.67)
 45

PRINT FIX(-2.89)
-2
```

FOR e NEXT (Specifiche)

Scopo: Eseguire un certo numero di volte una serie di istruzioni in un'iterazione.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: FOR *variabile* = *x* TO *y* [STEP *z*]

.
.
.

NEXT [*variabile*] [,*variabile*]...

Note:

variabile è una variabile intera o a precisione singola da utilizzare come contatore.

x è un'espressione numerica che rappresenta il valore iniziale del contatore.

y è un'espressione numerica che rappresenta il valore finale del contatore.

z è un'espressione numerica da utilizzare come incremento.

Le righe di programma che seguono la specifica FOR vengono eseguite finché non si incontra la specifica NEXT. Poi il contatore è incrementato della quantità specificata dal valore STEP *(z)*. Se non si specifica un valore per *z* si assume che l'incremento sia 1. Viene eseguito un controllo per vedere se il valore del contatore è ora maggiore del valore finale *y*. In caso negativo, il BASIC ritorna alla specifica dopo la specifica FOR e si ripete l'elaborazione.

In caso affermativo, l'esecuzione continua con la specifica successiva a NEXT. Questa è un'iterazione FOR...NEXT.

Se il valore di *z* è negativo, la prova viene eseguita al contrario. Il contatore è decrementato ogni volta nell'iterazione e questa viene eseguita finché il contatore è inferiore al valore finale.

La parte centrale dell'iterazione viene saltata se *x* è già maggiore di *y* quando il valore STEP è positivo, oppure se *x* è inferiore a *y* quando il valore STEP è negativo. Se la *z* è zero, si crea un'iterazione infinita a meno che non si trovi un modo per impostare il contatore ad un valore superiore a quello finale.

Se si usano, dove possibile, contatori interi, le prestazioni del programma verranno migliorate.

Iterazioni nidificate

Le iterazioni FOR...NEXT possono essere nidificate; cioè, un'iterazione FOR...NEXT può essere posta in un'altra iterazione FOR...NEXT. Quando le iterazioni sono nidificate, ognuna di esse deve avere un nome di variabile esclusivo come contatore. La specifica NEXT dell'iterazione interna deve apparire prima della specifica NEXT dell'iterazione esterna. Se le iterazioni nidificate hanno lo stesso punto finale, può essere usata una sola specifica NEXT per tutte.

FOR e NEXT (Specifiche)

Una specifica NEXT nella forma:

```
NEXT var1, var2, var3...
```

equivale alla sequenza di specifiche:

```
NEXT var1
NEXT var2
NEXT var3
.
.
.
```

La variabile nella specifica NEXT può essere omessa, nel qual caso NEXT corrisponde alla più recente specifica FOR. E' consigliabile includere le variabili in ordine, in modo da evitare confusione, ma può essere addirittura necessario se si eseguono dei salti al di fuori delle iterazione nidificate (tuttavia, utilizzando i nomi di variabili nelle specifiche NEXT si rallenta il programma).

Le iterazioni potrebbero aver termine o mediante un'impostazione del contatore fuori dai limiti oppure legando le stesse ad una specifica di condizione in modo che ogni FOR abbia la propria corrispondente NEXT.

Se si incontra una specifica NEXT prima della corrispondente FOR, si verifica un errore **NEXT without FOR.**

FOR e NEXT (Specifiche)

Esempio: Il primo esempio mostra un'iterazione FOR...NEXT con un valore STEP 2.

```
10 J=10: K=30
20 FOR I=1 TO J STEP 2
30 PRINT I;
40 K=K+10
50 PRINT K
60 NEXT
RUN
 1  40
 3  50
 5  60
 7  70
 9  80
```

Nel successivo esempio, l'iterazione non viene eseguita perché il valore iniziale dell'iterazione è superiore a quello finale.

```
10 J=0
20 FOR I=1 TO J
30 PRINT
40 NEXT I
RUN
```

Il successivo esempio terminerà con un errore NEXT without FOR. Per ogni FOR può esserci solo un NEXT. (Ciò è diverso da altre versioni del BASIC che permettono differenti NEXT quando si esce dalla iterazione).

```
10 FOR I=1 TO 5
20 IF I=2 GOTO 50
30 NEXT
40 GOTO 60
50 NEXT
60 END
```

FOR e NEXT (Specifiche)

Nell'ultimo esempio, l'iterazione viene eseguita 10 volte. Il valore finale della variabile nell'iterazione è sempre impostato prima di quello iniziale (si differenzia da altre versioni BASIC, che impostano il valore iniziale del contatore prima di quello finale. In altri BASIC l'iterazione di questo esempio viene eseguita sei volte).

```
10 I=5
20 FOR I=1 TO I+5
30 PRINT I;
40 NEXT
RUN
 1  2  3  4  5  6  7  8  9  10
```

FRE (Funzione)

Scopo: Fornire il numero di byte di memoria che non sono utilizzati dal BASIC. Questo numero non include l'ampiezza della porzione riservata all'area di lavoro dell'interpretatore.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: *v* = FRE(*x*)

v = FRE(*x$*)

x e *x$* sono argomenti fittizi.

Poiché nel BASIC le stringhe possono avere lunghezza variabile (ogni volta che si effettua una assegnazione ad una stringa la sua lunghezza può cambiare), esse sono trattate dinamicamente. Per questa ragione, lo spazio di una stringa può risultare frammentato, diminuendo così le prestazioni del programma.

L'esecuzione di programmi che contengono più funzioni a stringa, può essere migliorata usando la specifica MID$ per accedere a sottostringhe che formano una stringa più ampia. Questo impedisce la frammentazione dello spazio della stringa. Per un esempio vedere la "specifica MID$".

FRE con qualsiasi valore di stringa causa un riordino prima di dichiarare il numero di byte liberi. Per *riordino* si intende che il BASIC raccoglie tutti i dati utili e libera le aree di memoria ora inutilizzate, ma precedentemente utilizzate per le stringhe. I dati vengono compressi in modo da poter continuare fino alla completa mancanza di spazio.

FRE (Funzione)

Il BASIC esegue automaticamente un riordino quando vi è carenza di spazio di lavoro. Questa operazione può essere lunga.

> **Nota:** I tasti Ctrl. Interr non possono essere usati durante il riordino della memoria.

Esempio: Il valore fornito da FRE può essere diverso da quello dell'esempio.

```
PRINT FRE(Ø)
 14542
```

GET
(Specifica File)

Scopo: Leggere un record da un file ad accesso casuale in una memoria di transito ad accesso casuale.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	***

Formato: GET [#] *numfile* [,*numero*]

Note:

numfile è il numero con il quale è stato aperto il file.

numero è il numero del record da leggere, compreso tra 1 e 16 Mb. Se si omette *numero* nella memoria di transito viene portato il record successivo (dopo l'ultima GET).

Dopo una specifica GET, per leggere i caratteri della memoria di transito di un file ad accesso casuale si possono usare INPUT#, LINE INPUT# o dei riferimenti a variabili definite nella specifica FIELD. Per informazioni complete sull'uso di GET, fare riferimento all'Appendice A del manuale BASIC.

Poiché il BASIC e il DOS bloccano più record possibili in settori di 512 byte, la specifica GET non necessariamente esegue una lettura fisica dal disco.

GET può essere utilizzata anche per file di comunicazione. In questo caso *numero* è il numero di byte che devono essere letti dalla memoria di transito di comunicazione. Questo numero non può superare il valore definito dall'opzione LEN nella specifica OPEN "COM...

GET
(Specifica file)

Esempio: Questo esempio apre il file "CUST" per l'accesso casuale, con i campi definiti a riga 2Ø. La specifica GET a riga 3Ø porta un record nella memoria di transito del file. La riga 4Ø visualizza le informazioni del record letto.

```
1Ø OPEN "A:CUST" AS #1
2Ø FIELD 1, 3Ø AS CUSTNAME$, 3Ø AS ADDR$,
       35 AS CITY$
3Ø GET 1
4Ø PRINT CUSTNAME$, ADDR$, CITY$
```

GET
(Specifica grafici)

Scopo: Leggere i punti da un'area dello schermo.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
		***	***

Solo modo grafici.

Formato: GET *(x1,y1)-(x2,y2),schiera*

Note: *(x1,y1),(x2,y2)*
sono coordinate in forma relativa e assoluta. Per maggiori informazioni sulle coordinate vedere il Capitolo 3 del manuale BASIC.

schiera è il nome della schiera che deve contenere le informazioni.

GET legge gli attributi dei punti che si trovano nel rettangolo specificato nella schiera. Il rettangolo specificato ha i punti *(x1,y1)* e *(x2,y2)* agli angoli opposti (questo è uguale al rettangolo disegnato dalla specifica LINE usando l'opzione **B**).

GET e PUT possono essere utilizzate per movimenti veloci di oggetti in modo grafici. Si possono considerare GET e PUT come operazioni che muovono i bit nello (PUT) e dallo (GET) schermo. Ricordare che PUT e GET si usano anche per file ad accesso casuale, ma la sintassi di queste specifiche è diversa.

GET
(Specifica grafici)

La schiera si usa semplicemente per contenere l'immagine e deve essere numerica; può avere qualsiasi precisione. L'ampiezza in byte richiesta per la schiera è:

4+INT((*x***bitperpunto*+7)/8)**y*

dove *x* e *y* sono rispettivamente le lunghezze dei lati verticali e orizzontali del rettangolo. Il valore di *bitperpunto* è 2 a media risoluzione e 1 ad alta risoluzione.

Ad esempio, si supponga di usare GET per ottenere un'immagine di 10 per 12 a media risoluzione. Il numero di byte necessari è 4+INT((10*2+7)/8)*12, oppure 40 byte. I byte per gli elementi di una schiera sono:

- 2 per stringhe di interi
- 4 per stringhe a precisione singola
- 8 per stringhe a precisione doppia

Perciò si può usare una schiera di interi con almeno 20 elementi.

Le informazioni dello schermo sono memorizzate nella schiera nel modo seguente:

1. due byte che forniscono la dimensione *x* in bit
2. due byte che forniscono la dimensione *y* in bit
3. i dati stessi

E' possibile esaminare le dimensioni x e y e persino gli stessi dati se si usa una schiera di interi. La dimensione x è nell'elemento 0 della schiera, mentre la dimensione y è nell'elemento 1. Ricordare, tuttavia, che gli interi sono memorizzati a partire dal byte basso verso quello più alto; i dati vengono trasferiti in ordine inverso, prima il byte alto poi quello basso.

GET
(Specifica grafici)

I dati per ogni fila di punti del rettangolo sono allineati a sinistra su un limite di byte, in modo che se è memorizzato meno di un multiplo di otto bit, il resto del byte sarà riempito di zeri.

PUT e GET lavorano più velocemente a media risoluzione quando *x1* MOD4 è uguale a zero e ad alta risoluzione quando *x1* MOD8 è uguale a zero. Questo è un caso speciale in cui i limiti del rettangolo coincidono con quelli del byte.

Esempio: Vedere la specifica "PUT (grafici)" per un esempio.

GOSUB e RETURN (Specifiche)

Scopo: Saltare ad, e ritornare da un sottoprogramma.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: GOSUB *riga*

.
.
.

RETURN

Note: *riga* è il numero della prima riga del sottoprogramma.

Un sottoprogramma può essere richiamato più volte in un programma e può essere richiamato anche da un altro sottoprogramma. Questa nidificazione di sottoprogrammi è limitata solo dalla disponibilità di memoria.

La specifica RETURN fa ritornare il BASIC alla specifica che segue la più recente GOSUB. Un sottoprogramma può contenere più di una specifica RETURN; si può quindi tornare nel sottoprogramma da diversi punti. I sottoprogrammi possono apparire ovunque nel programma.

Per evitare che il programma immetta accidentalmente un sottoprogramma, si può inserire una specifica STOP, END o GOTO prima del sottoprogramma, in modo che il programma possa controllare l'operazione.

Usare ON-GOSUB per passare a diversi sottoprogramma sulla base del risultato di un'espressione.

GOSUB e RETURN (Specifiche)

Esempio: Questo esempio mostra come lavora un sottoprogramma. GOSUB a riga 10 richiama il sottoprogramma a riga 40. In questo modo il programma salta alla riga 40 ed inizia l'esecuzione delle specifiche finché non trova RETURN a riga 70. A questo punto il programma ritorna alla specifica dopo il richiamo del sottoprogramma, cioè a riga 20. La specifica END a riga 30 impedisce una seconda esecuzione del sottoprogramma.

```
10 GOSUB 40
20 PRINT "BACK FROM SUBROUTINE"
30 END
40 PRINT "SUBROUTINE";
50 PRINT " IN";
60 PRINT " PROGRESS"
70 RETURN
RUN
SUBROUTINE IN PROGRESS
BACK FROM SUBROUTINE
```

GOTO (Specifica)

Scopo: Saltare incondizionatamente da una sequenza di programma normale ad un numero di riga specificato.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: GOTO *riga*

Note: *riga* è il numero della riga del programma

Se *riga* è il numero di riga di una specifica eseguibile, viene eseguita questa specifica e quelle successive. Se *riga* si riferisce ad una specifica non eseguibile (come REM o DATA), il programma continua dalla prima specifica dopo *riga*.

La specifica GOTO può essere utilizzata in modo diretto per reimmettere un programma dal punto desiderato. Ciò è utile nella ricerca e correzione di errori.

Usare ON-GOTO per saltare a diverse righe in base al risultato di un'espressione.

GOTO (Specifica)

Esempio: La specifica GOTO a riga 60 pone il programma in un'iterazione infinita, che si arresta quando il programma esaurisce i dati della specifica DATA (si noti che saltando alla specifica DATA non si sono aggiunti altri valori alla tabella di dati interni).

```
10 DATA 5,7,12
20 READ R
30 PRINT  "R = ";R,
40 A = 3.14*R^2
50 PRINT  "AREA = ";A
60 GOTO 10
RUN
R = 5           AREA = 78.5
R = 7           AREA = 153.86
R = 12          AREA = 452.16
OUT OF DATA IN 20
```

HEX$ (Funzione)

Scopo: Fornire una stringa che rappresenta il valore esadecimale dell'argomento decimale.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v$* = HEX$*(n)*

Note: *n* è un'espressione numerica compresa tra -32768 e 65535.

Se *n* è negativa, si usa la forma del completamento a due. Cioè, HEX$*(-n)* corrisponde a HEX$ (65536-*n*).

Vedere la funzione OCT$ per la conversione ottale.

Esempio: Il seguente esempio usa la funzione HEX$ per calcolare la rappresentazione esadecimale dei due valori decimali immessi.

```
10 INPUT X
20 A$ = HEX$(X)
30 PRINT X "DECIMAL IS "; A$ "HEXADECIMAL"
RUN
? 32
 32 DECIMAL IS 20 HEXADECIMAL
RUN
? 1023
 1023 DECIMAL IS 3FF HEXADECIMAL
```

IF (Specifica)

Scopo: Prendere una decisione riguardo un flusso di programma basato sul risultato di un'espressione.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: IF *espressione*[,] *THEN clausola*[ELSE *clausola*]

IF *espressione*[,] GOTO *riga*[[,] ELSE *clausola*]

Note:

espressione può essere qualsiasi espressione numerica.

clausola può essere una specifica BASIC o una sequenza di specifiche (separate da due punti); oppure può essere semplicemente il numero di una riga a cui saltare.

riga è il numero di una riga esistente nel programma.

Se *l'espressione* è vera (non zero), viene eseguita la clausola THEN o GOTO. THEN può essere seguita da un numero di riga per il salto o da una o più specifiche da eseguire. GOTO è sempre seguita da un numero di riga.

Se il risultato dell'*espressione* è falso (zero), si ignora la clausola THEN o GOTO e si esegue la clausola ELSE, se presente. L'esecuzione continua con la successiva specifica eseguibile.

Se si immette una specifica IF...THEN in modo diretto, ed essa dirige il controllo ad un numero di

riga, si verifica un errore **Undefined line number,** a a meno che non sia stata precedentemente immessa una riga con il numero specificato.

> **Nota:** Se si usa IF per provare l'uguaglianza di un valore che è il risultato di un calcolo a precisione singola o doppia, ricordarsi che la rappresentazione interna del valore può non essere esatta (questo perché i valori a precisione singola o doppia sono memorizzati internamente in un formato binario con punto decimale mobile). Perciò la prova deve essere eseguita sulla *gamma* per la quale l'accuratezza del valore può variare. Ad esempio per provare una variabile A calcolata paragonandola al valore 1.Ø usare:
>
> ```
> IF ABS (A-1.Ø)<1.ØE-6 THEN ...
> ```
>
> Questa prova fornisce un risultato vero se il valore di A è 1.Ø con un errore relativo inferiore a 1.Ø E-6.

Notare anche che IF...THEN...ELSE sono una specifica sola. Se IF è presente tutto ciò che segue IF nella riga è parte della specifica. Poiché IF...THEN... ELSE è un'unica specifica, la clausola ELSE non può essere una riga separata di programma. Ad esempio:

```
1Ø IF A=B THEN X=4
2Ø ELSE P=Q
```

non è valido. Deve essere

```
1Ø IF A=B THEN X=4 ELSE P=Q
```

Nidificazione di specifiche IF: le specifiche IF... THEN...ELSE... possono essere nidificate. La nidificazione è limitata solo dalla lunghezza della riga. Ad esempio:

```
IF X>Y THEN PRINT "GREATER" ELSE IF Y>X
THEN PRINT "LESS THAN" ELSE PRINT "EQUAL"
```

è una specifica valida. Se la specifica non contiene lo stesso numero di clausole ELSE e THEN, ogni ELSE viene confrontata con la più vicina THEN non corrispondente. Ad esempio:

```
IF A=B THEN IF B=C THEN PRINT "A=C"
  ELSE PRINT "A<>C"
```

non stamperà "A <>C" quando A <>B.

Esempio: Questa specifica estrae il record I se I non è zero:

```
200 IF I THEN GET #1,I
```

Nell'esempio seguente, se I è compreso tra 10 e 20, viene calcolato DB e l'esecuzione salta a riga 300. Se I non è compreso in questa gamma, viene stampato il messaggio **Out of range.** Notare l'uso delle due specifiche nella clausola THEN.

```
100 IF (I>10) AND (I<20) THEN
        DB=1982-I: GOTO 300
    ELSE PRINT "OUT OF RANGE"
```

IF
(Specifica)

A riga 20 del seguente esempio tutto ciò che sta dopo THEN è parte della clausola. Questo significa che NEXT non è eseguito a meno che N non sia uguale ad I. Quando viene eseguita la riga 20, N non è uguale ad I così la valutazione di IF risulta falsa. Perciò NEXT non è eseguito ed il programma dà errore a riga 30. NEXT deve essere codificato su riga a parte se si vuole che il programma continui fino a che N = I.

```
10 N=15
20 FOR I=1 TO 20: IF N=I THEN 40:NEXT
30 PRINT "N <> I": END
40 PRINT "N = I"
RUN

N <> I
```

INKEY$ (Variabile)

Scopo: Leggere un carattere dalla tastiera.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v$* = INKEY$

Note: INKEY$ legge solo un carattere, anche se ve ne sono altri nella memoria di transito della tastiera. Il valore risultante è zero-, uno- o una stringa di due caratteri.

- Una stringa nulla (lunghezza zero) indica che non ci sono caratteri in sospeso sulla tastiera.
- Una stringa di un carattere contiene il carattere corrente letto dalla tastiera.
- Una stringa di due caratteri indica un codice esteso speciale. Il primo carattere sarà l'esadecimale ØØ. Per una lista completa di questi codici vedere l'appendice D "Codici caratteri ASCII".

Si deve assegnare il risultato di INKEY$ ad una variabile a stringa prima di usare il carattere con una specifica o funzione BASIC.

Durante l'utilizzo di INKEY$, i caratteri non possono essere visualizzati e vengono passati al programma, ad eccezione di:

- Ctrl-Interr, che arresta il programma.
- Ctrl-Bloc Num, che manda il sistema in uno stato di pausa.
- Altrn-Ctrl-Canc che esegue un ripristino del sistema.
- Stampa, che stampa lo schermo.

INKEY$ (Variabile)

Se si preme il tasto di immissione in risposta a INKEY$, il carattere di ritorno a margine passa al programma.

Esempio: La seguente sezione di un programma arresta il programma finché non si preme un tasto:

```
100 PRINT "Premere un tasto qualsiasi per continuare"
110 A$=INKEY$: IF A$="" THEN 110
```

L'esempio successivo mostra le righe di programma che possono essere utilizzate per provare un codice di due caratteri che viene ritornato:

```
100 KB$=INKEY$
110 IF LEN(KB$)=2 THEN KB$=RIGHT$(KB$,1)
```

INP (Funzione)

Scopo: Fornire il byte letto dalla porta *n*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v* = INP(*n*)

Note: *n* deve essere compreso tra Ø e 65535.

INP è la funzione complementare della specifica OUT. Vedere la specifica "OUT".

INP esegue la stessa funzione dell'istruzione IN nel linguaggio assemblatore. Per una descrizione dei numeri delle uscite valide (indirizzi I/E) vedere il manuale di riferimento del Personal Computer.

Esempio: Questa istruzione legge un byte dalla porta 255 o lo assegna alla variabile A.

```
1ØØ A=INP(255)
```

INPUT (Specifica)

Scopo: Ricevere l'immissione dalla tastiera durante l'esecuzione del programma.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: INPUT[;][*"richiesta";*] *variabile*[,*variabile*]...

Note: *"richiesta"* è una costante a stringa utilizzata per richiedere l'immissione desiderata.

variabile è il nome di una variabile numerica a stringa oppure di un elemento di schiera che riceve l'immissione.

Quando il programma trova una specifica INPUT, si ferma e visualizza un punto interrogativo sullo schermo per indicare che è in attesa di dati. Se si include una *"richiesta"*, la stringa viene visualizzata. Se *"richiesta"* è seguita da un punto e virgola (;) un punto interrogativo seguirà la stringa visualizzata; se è seguita da una virgola il punto interrogativo non verrà visualizzato. Si può usare una virgola al posto del punto e virgola dopo la stringa "richiesta" per eliminare il punto interrogativo.

Ad esempio, la specifica INPUT "ENTER BIRTHDATE", B$ stampa la richiesta senza il punto interrogativo.

Dopo che la richiesta e il punto interrogativo sono stati visualizzati i dati richiesti dalla tastiera possono essere immessi.

I dati che si immettono sono assegnati alla *variabile* indicata nella lista di variabili. Le voci relative ai dati che si indicano devono essere separate da virgole ed il loro numero deve essere uguale a quello delle variabili nella lista.

INPUT (Specifica)

Il tipo di ogni voce, relativa ai dati, che si immette deve corrispondere a quello specificato dal nome di variabile (le stringhe immesse in risposta ad una specifica INPUT non devono essere delimitate da apici).

Se si risponde a INPUT con troppe o con poche voci o con un tipo di valore errato (lettere invece di numeri, ecc.), il BASIC visualizza il messaggio **? Redo from start**. Se si richiede un'unica variabile, si può premere semplicemente il tasto di immissione per indicare il valore assunto in mancanza di indicazioni: Ø, per un'immissione numerica, o nullo, per una stringa. Tuttavia, se è necessaria più di una variabile, premendo il tasto di immissione si causa la stampa del messaggio **"? Redo from start"** poiché sono state immesse poche voci. Il BASIC non assegna alcun valore di immissione alle variabili finché non si fornisce una risposta accettabile.

Nel BASIC su disco e nel BASIC avanzato, se INPUT è seguita immediatamente da un punto e virgola, premendo il tasto di immissione, per immettere i dati, non si produce sullo schermo una sequenza ritorno a margine/ avanzamento riga. Ciò significa che il cursore rimane sulla stessa riga della risposta.

Esempio: In questo esempio, il punto interrogativo visualizzato dall'elaboratore è una richiesta di immissione. Si supponga di immettere 5.

```
1Ø INPUT X
2Ø PRINT X "IL QUADRATO E'" X^2
3Ø END
RUN
?
```

INPUT (Specifica)

Il programma continua:

```
    .
    .
    .
?   5
 5  IL QUADRATO E' 25
```

Per questo secondo esempio è stata inclusa una richiesta nella riga 2Ø, in modo che l'elaboratore chieda "QUALE E' IL RAGGIO?".

```
1Ø PI=3.14
2Ø INPUT "QUALE E' IL RAGGIO";R
3Ø A=PI*R^2
4Ø PRINT "L'AREA DEL CERCHIO E' ";A
5Ø END
RUN
QUALE E' IL RAGGIO?
```

Supponendo di rispondere 7.4 il programma continua:

```
   .
   .
   .
QUALE E' IL RAGGIO?  7.4
L'AREA DEL CERCHIO E' 171.9464
```

INPUT # (Specifica)

Scopo: Leggere le voci relative ai dati da un file o unità sequenziale e assegnarle alle variabili di programma.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: INPUT # numfile, variabile [,variabile]...

Note:

numfile è il numero usato all'apertura del file per l'immissione.

variabile è il nome di una variabile con una voce nel file ad essa assegnato. Può essere una variabile a stringa o numerica, oppure un elemento di schiera.

Il file sequenziale può trovarsi su minidisco o cassetta; può essere un flusso di dati sequenziale da un adattatore per le comunicazioni; oppure può essere la tastiera (KYBD:).

Il tipo di dati del file deve corrispondere a quello specificato dal nome di variabile. A differenza di INPUT, con INPUT# non viene visualizzato alcun punto interrogativo.

Le voci relative ai dati nel file devono apparire come se i dati fossero immessi in risposta ad una specifica INPUT. Con i valori numerici, gli spazi iniziali, il ritorno a margine e l'avanzamento di riga vengono ignorati. Si assume che il primo carattere incontrato, che non sia uno spazio, un ritorno a margine o un avanzamento di riga, sia l'inizio del numero. Il numero termina con uno spazio, un ritorno a margine, un avanzamento di riga o una virgola.

INPUT
(Specifica)

Se il BASIC esegue una scansione dei dati alla ricerca di una voce relativa ad una stringa, si ignorano gli spazi iniziali, i ritorni a capo e gli avanzamenti di riga. Si assume che il primo carattere che si incontra, che non sia uno dei precedenti, sia l'inizio della voce relativa alla stringa. Se come primo carattere vi sono gli apici ("), la voce relativa alla stringa comprende tutti i caratteri letti tra i primi apici e gli ultimi. Perciò, una stringa fra apici non può contenere gli apici come carattere. Se il primo carattere della stringa non sono apici, la stringa ne è priva e termina quando si legge una virgola, un ritorno a margine, un avanzamento di riga, oppure dopo 255 caratteri. Se si raggiunge la fine del file quando si sta immettendo una voce relativa ad un numero o ad una stringa, la voce viene annullata.

Esempio: Vedere l'appendice A "Immissione ed emissione su disco nel BASIC" nel manuale BASIC.

INPUT$ (Funzione)

Scopo: Fornire una stringa di *n* caratteri, letta dalla tastiera o dal numero del file *numfile*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v$* = INPUT$ (*n*[,[#]*numfile*])

Note:

n è il numero di caratteri da leggere dal file.

numfile è il numero del file usato nella specifica OPEN. Se si omette *numfile,* si legge la tastiera.

Se per l'immissione si usa la tastiera, i caratteri non saranno visualizzati. Vengono passati tutti i caratteri (compresi quelli di controllo) ad eccezione di Ctrl-Interr, che è utilizzato per interrompere l'esecuzione della funzione INPUT$. Rispondendo a INPUT$ dalla tastiera, non è necessario premere il tasto di immissione.

La funzione INPUT$ consente di leggere i caratteri dalla tastiera, che sono significativi per l'editore di programmi BASIC, come il ritorno unitario (codice ASCII 8). Se si vogliono leggere questi caratteri speciali, si deve usare INPUT$ o INKEY$ (non INPUT o LINE INPUT).

Per i file di comunicazione, si preferisce la funzione INPUT$ piuttosto che le specifiche INPUT# e LINE INPUT#, poiché tutti i caratteri ASCII possono essere significativi nelle comunicazioni. Vedere l'Appendice C "Comunicazioni".

INPUT$ (Funzione)

Esempio: il seguente programma lista i contenuti di un file sequenziale in esadecimale.

```
10 OPEN "DATA" FOR INPUT AS #1
20 IF EOF(1)THEN 50
30 PRINT HEX$(ASC(INPUT$(1,#1)));
40 GOTO 20
50 PRINT
60 END
```

L'esempio successivo legge un carattere dalla tastiera in risposta ad una domanda.

```
100 PRINT "BATTERE P PER PROCEDERE O S PER STOP"
110 X$=INPUT$(1)
120 IF X$="P" THEN 500
130 IF X$="S" THEN 700 ELSE 100
```

INSTR (Funzione)

Scopo: Ricercare la prima corrispondenza della stringa *y$* in *x$* e fornire la posizione in cui si trova la corrispondenza stessa. La posizione relativa *n*, facoltativa, definisce la posizione di inizio per la ricerca in *x$*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v* = INSTR([*n*,]*x$*,*y$*)

Note:

n è un'espressione numerica compresa tra 1 e 255.

x$*,*y$ possono essere variabili, espressioni o costanti di stringa.

Se n > LEN (*x$*), o se *x$* è nulla, o se non è possibile trovare *y$*, INSTR fornisce Ø. Se *y$* è nulla, INSTR fornisce *n* (o 1 se *n* non è specificata).

Se *n* non è compresa nei limiti, viene dato un errore **Illegal function call**.

Esempio: Questo esempio ricerca la stringa "B" nella stringa "ABCDEB". Se la ricerca viene effettuata dall'inizio, "B" si trova a posizione 2; se la ricerca inizia a posizione 4, "B" si trova a posizione 6.

```
1Ø A$ = "ABCDEB"
2Ø B$ = "B"
3Ø PRINT INSTR(A$,B$);INSTR(4,A$,B$)
RUN
 2  6
```

INT (Funzione)

Scopo: Fornire l'intero più grande che è minore o uguale a *x*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v* = INT(*x*)

Note: *x* è una qualsiasi espressione numerica.

E' quella chiamata funzione "floor" in alcuni altri linguaggi programma.

Vedere le funzioni FIX e CINT che forniscono anch'esse valori interi.

Esempio: Questo esempio mostra come INT tronchi gli interi positivi, ma arrotondi per eccesso i numeri negativi (in una direzione negativa).

```
PRINT INT(45.67)
 45

PRINT INT(-2.89)
-3
```

IOCTL (Specifica)

Scopo: Permettere al BASIC di inviare una stringa di dati di controllo ad una unità ogni volta che l'unità viene aperta tramite la specifica OPEN.

Non è valido per versioni del BASIC antecedenti alla versione 3.0.

Versioni:	Cassetta	Disco	Avanzato	Compilatore
		***	***	

Formato: IOCTL [#] *numfile, stringa*

Note: *numfile* è il numero del file per un'unità

stringa è una espressione che contiene i dati di controllo

Il sistema di file di Immissione/Emissione del BASIC permette di creare ed installare le proprie unità a disco. La specifica IOCTL e la funzione IOCTL$ inviano e leggono i dati di controllo e di lettura dalla propria unità.

Una stringa di comandi IOCTL può essere lunga sino a 255 byte. Comandi multipli nell'ambito di una stringa possono essere separati da punti e virgola.

```
"LF;PL66;LW132"
```

In questo modo vengono definiti il formato ed il contenuto della stringa. I comandi possibili vengono determinati dalle caratteristiche dell'unità installata.

IOCTL (Specifica)

Esempio: Inizialmente i dispositivi LPT1:, LPT2:, e LPT3: sono installati, ma possono essere sostituiti. Se si installa una unità chiamata LPT1 in sostituzione di LPT1: e supponendo che la stessa sia in grado di impostare o modificare la lunghezza della pagina il comando IOCTL da usare potrebbe essere:

```
"PLn" (dove n è la lunghezza della nuova pagina)
```

Si può quindi aprire la nuova unità LPT1 e stabilire la lunghezza della pagina con:

```
OPEN "LPT1" FOR OUTPUT AS #1
IOCTL #1, "PL60"
```

Si può, ad esempio, scrivere una unità minidisco che controlli il monitor e che sia in grado di impostare il modo dello schermo sul colore come pure di determinare l'ampiezza dello schermo. Per esempio:

```
OPEN "OPT" FOR OUTPUT AS #2
IOCTL #2, "CL:W40"
```

Assumendo che la nuova unità sia in grado di accettare un comando chiamato "CL" al fine di modificare lo schermo sul colore ed un comando chiamato "W*n*" per impostare l'ampiezza dello schermo, il precedente esempio passa quei comandi alla propria unità generando la successiva risposta dello schermo.

Nota: Per informazioni al riguardo fare riferimento alla funzione IOCTL$ in questo manuale, alla sezione "Unità a disco" del manuale BASIC e alla relativa sezione del manuale di riferimento tecnico DOS.

IOCTL$ (Funzione)

Scopo: Leggere una stringa di dati di controllo da una unità a dischi aperta.

Non è valida per versioni BASIC antecedenti alla versione 3.Ø.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	

Formato: *v$* = IOCTL$([#]*numfile*)

Note: *numfile* è il numero del file aperto nell'unità a dischi.

La funzione IOCTL$ può essere usata per venire a conoscenza se il comando IOCTL è andato a buon fine o meno. Può essere anche usata per aver informazioni sulla configurazione dell'unità e sulla sua ampiezza.

Esempio: Questo esempio verifica che i dati per il controllo siano stati ricevuti regolarmente.

```
1Ø OPEN "COM" AS #1
2Ø IOCTL #1, "SW132;GW"
3Ø IF IOCTL$(1) = "132"
     THEN PRINT "AMPIEZZA IMPOSTATA CON SUCCESSO"
```

Se l'unità a dischi "COM" fornisce un valore non uguale a 132 dopo la richiesta IOCTL$, ciò significa che il comando non è stato processato con successo e si deve quindi ricercare l'errore. Se l'errore si è verificato sull'unità controllare le variabili di sistema di ERDEV e di ERDEV$.

KEY
(Specifica)

Scopo: Impostare o visualizzare i tasti temporanei.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: KEY ON

KEY OFF

KEY LIST

KEY *n, x$*

KEY*n,* CHR$(KB*flag*) e CHR$(*cod. tasto*)
(per BASIC 2.Ø avanzato e successivi livelli).

Note: KEY ON produce la visualizzazione dei valori del tasto temporaneo alla riga 25. Se l'ampiezza è 4Ø, vengono visualizzati 5 dei 1Ø tasti. Se l'ampiezza è 8Ø vengono visualizzati i dieci tasti. Con entrambe le ampiezze, vengono visualizzati solo i primi sei caratteri di ogni valore.

KEY OFF elimina la visualizzazione dei tasti funzionali dalla 25esima riga, rendendola disponibile al programma. Non disabilita i tasti funzionali.

Dopo aver eliminato con KEY OFF la visualizzazione dei tasti temporanei, si può usare LOCATE 25,1 seguita da PRINT per visualizzare ciò che si vuole sull'ultima riga dello schermo. Le informazioni sulla riga 25 non vengono fatte scorrere come per le righe da 1 a 24.

KEY (Specifica)

KEY LIST lista i valori dei dieci tasti sullo schermo. Tutti i 15 caratteri di ogni valore vengono visualizzati.

KEY *n,x$* permette di impostare ogni tasto funzionale per scrivere automaticamente una qualsiasi stringa di caratteri. ON è lo stato assunto per difetto per la visualizzazione dei tasti funzionali.

n è il numero del tasto funzionale compreso tra 1 e 10.

x$ è una espressione a stringa assegnata al tasto. (Ricordarsi di chiudere le *costanti* a stringa tra apici).

Il valore del tasto funzionale *n* è riassegnato dal valore della stringa *x$*. Se il valore immesso per *n* non è compreso tra 1 e 10 si verifica un errore **(Illegal function call)**. In questo caso viene conservata la precedente stringa assegnata a quel tasto funzionale. *x$* può essere lunga da 1 a 15 caratteri. Se dovesse essere più lunga, verranno assegnati solo i primi 15 caratteri. L'assegnazione di una stringa nulla (lunghezza zero) ad un tasto temporaneo disabilita il tasto funzionale come tasto temporaneo.

Quando si preme un tasto temporaneo la funzione INKEY$ fornisce un carattere della stringa relativa al tasto ogni volta che viene richiamata. Il primo carattere è zero binario, il secondo è il codice di scansione del tasto, come indicato nell'Appendice D, "Codici carattere ASCII".

KEY (Specifica)

Vi sono altre sei possibili condizioni definibili generate dalla pressione combinata di quei tasti sulla tastiera. Con questa possibilità si possono impostare i tasti Ctrl, Bloc Maius.

Disponibile solo col BASIC avanzato 2.Ø e successivi livelli.

Questi tasti funzionali si definiscono con la specifica:

```
KEY n,CHR$(valore)+CHR$(cod.tasto)
```

n è un'espressione numerica compresa tra 15 e 2Ø

valore è un valore numerico che corrisponde al valore esadecimale del tasto. I valori per ogni tasto sono:

Bloc Maius &HØ - Bloc Maius non è attivo

Bloc Maius &H4Ø - Bloc Maius è attivo

Bloc Num &HØ - Bloc Num non è attivo

Bloc Num &H2Ø - Bloc Num è attivo

Altrn &HØ8 - Il tasto Altrn è premuto

Ctrl &HØ4 - Il tasto di controllo è premuto

Maius a sinistra &HØ2 - Il tasto di maiuscolo a sinistra è premuto

Maius a destra &H01 - Il tasto di maiuscolo a destra è premuto

Cod. tasto è un numero che identifica uno degli 83 tasti associabili. Vedere l'appendice E. "Codici di scansione".

KEY (Specifica)

Notare che associando i tasti vengono considerate equivalenti la maiuscola di destra e quella di sinistra perciò per definire il tasto shift (maiuscole) si può usare il valore &H01, &H02 o &H03 (la somma esadecimale dei due).

Si possono anche sommare più valori insieme come ad esempio Ctrl e Altrn che verranno premuti insieme. I valori dello stato di maiuscolo devono sempre essere in esadecimale. Quando si associa un tasto o una combinazione di tasti, è necessario conoscere lo stato di Bloc Num e Bloc Maius.

Una volta associati, i tasti vengono poi elaborati nel seguente ordine:

1. Ctrl-Stampa, che attiva la stampatrice, viene elaborato per primo.
   Anche se Ctrl-Stampa è stato definito come tasto associabile può sempre essere premuto per ottenere una copia su stampa del contenuto dello schermo.

2. Poi vengono elaborati i tasti funzionali da F1 a F10, freccia in alto, in basso, a destra e a sinistra (1-14). Scegliendo come tasti da associare i codici da 59 a 68 e 72, 75, 77 o 80 non si ottiene nessun effetto poiché questi sono considerati di tipo predefinito.

3. Per ultimi vengono elaborati i tasti definiti dal 15 al 20.

Note:

- I tasti associati non interessano la memoria di transito BIOS. Cosicché il solo BASIC sarà in grado di riconoscere la pressione degli stessi.

KEY (Specifica)

Fare attenzione quando si associano Ctrl-Interr e Ctrl-Altrn-Canc perché, se non si è prevista una uscita nel punto in cui avviene l'associazione bisognerà spegnere l'elaboratore per fermare il programma.

Vedere nelle pagine seguenti la specifica KEY(n) per sapere come abilitare o disabilitare l'associazione di tasti funzionali nel BASIC avanzato.

Esempio: visualizza i tasti temporanei sulla 25esima riga.

```
50 KEY ON
```

cancella la visualizzazione dei tasti temporanei. I tasti sono ancora attivi, ma non vengono visualizzati.

```
10 KEY OFF
```

assegna la stringa "FILES" + tasto di immissione al tasto temporaneo 1. Questo è un modo per assegnare un comando usato frequentemente ad un tasto funzionale.

```
10 KEY 1, "FILES"+CHR$(13)
```

disabilita il tasto funzionale 1 come tasto temporaneo.

```
10 KEY 1,""
```

Imposta la lettera maiuscola P. Notare che tutte e tre le specifiche KEY (KEY, KEY(n) e ON KEY) sono usate per l'intercettazione dei tasti.

```
10 KEY 15, CHR$(&H40)+CHR$(25)
20 ON KEY(15) GOSUB 1000
30 KEY(15) ON
```

Imposta una associazione di tasti tra Ctrl- ↑ A. Notare come i rispettivi valori esadecimali di Ctrl (&H04) e ↑ (&H03) vengono sommati per stabilire lo stato di maiuscolo.

```
10 KEY 20, CHR$(&H04+&H03)+CHR$(30)
20 ON KEY(20) GOSUB 2000
30 KEY(20) ON
```

KEY(n)
(Specifica)

Scopo: Attivare e disattivare l'intercettazione del tasto specificato in un programma BASIC. Vedere anche ON KEY(n).

Versioni:

Cassetta	Disco	Avanzato	Compilatore
		***	(**)

Formato: KEY(n) ON

KEY(n) OFF

KEY(n) STOP

Nota: *n* è un'espressione numerica compresa tra 1 e 20 e indica il tasto da intercettare.

1-10 Tasti funzionali F1-F10
11 Spostamento del cursore verso l'alto
12 Spostamento del cursore a sinistra
13 Spostamento del cursore a destra
14 Spostamento del cursore verso il basso
15-20 Tasti definiti dalla specifica:
KEY *n*,CHR$ *(valore)*, CHR$ (*cod. tasto*).
Questi tasti sono validi solo con il BASIC 2.0 e successive versioni.

KEY *(n)* ON deve essere eseguita per attivare l'intercettazione del tasto funzionale o dell'attività del tasto di controllo del cursore. Dopo KEY*(n)* ON, se nella specifica ON KEY *(n)* è stato specificato un numero di riga diverso da zero, ogni volta che il BASIC avvia una nuova specifica controllerà se è stato premuto il tasto specificato. In caso affermativo eseguirà una GOSUB al numero di riga indicato nella specifica ON KEY *(n)*. Una specifica KEY *(n)* non può precedere una specifica ON KEY *(n)*.

KEY(n) (Specifica)

Se KEY *(n)* è OFF, non si esegue alcuna intercettazione e anche se si è premuto il tasto l'azione non viene registrata.

Dopo l'esecuzione della specifica KEY *(n)* STOP, non avviene alcuna intercettazione. Tuttavia, se si preme il tasto specificato, l'azione viene registrata in modo che avvenga un'immediata intercettazione se KEY *(n)* ON viene eseguita.

KEY *(n)* ON non interessa la visualizzazione dei valori del tasto temporaneo in fondo allo schermo.

Se si usa una specifica KEY *(n)* nel BASIC su disco o su cassetta si ottiene un **Syntax error.** Vedere anche la "Specifica KEY".

KILL (Comando)

Scopo: Cancellare un file da un minidisco. Il comando KILL del BASIC è simile al comando ERASE del DOS.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	

Formato: KILL *specfile*

Note: *specfile* è una espressione a stringa per la specifica del file, nel BASIC 2.0 e successivi livelli può contenere un percorso. *Specfile* deve essere conforme alle regole esposte nella sezione "Denominazione dei File" del Capitolo 3 del manuale BASIC. In caso contrario si verifica un errore.

Si può usare KILL per tutti i file su disco. Il nome deve contenere l'estensione, se esiste. Ad esempio, si può salvare un programma BASIC con il comando:

```
SAVE "TEST"
```

Il BASIC fornisce l'estensione **.BAS** per il comando SAVE, ma non per KILL. Se successivamente si vuole cancellare quel file di programma, si deve immettere KILL "TEST.BAS" e non KILL "TEST".

Se si immette una specifica KILL per un file ancora aperto, si verifica un errore **(File already open)**.

KILL (Comando)

Esempio: Questo esempio cancella il file "DATA1" nel modulo A:

```
200 KILL "A:DATA1"
```

Questo esempio cancella il file PROG.BAS dal sotto-indirizzario LEVEL2.

```
KILL "LEVEL1\LEVEL2\PROG. BAS"
```

Notare che KILL cancella soltanto dei file. Per cancellare degli indirizzari occorre usare il comando RMDIR.

LEFT$ (Funzione)

Scopo: Fornire gli ultimi *n* caratteri a sinistra di *x$*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v$* = LEFT$*(x$,n)*

Note:

x$ è un'espressione a stringa.

n è un'espressione numerica che deve essere compresa tra 0 e 255. Specifica il numero di caratteri che devono trovarsi nel risultato.

Se n è maggiore di LEN*(x$)*, l'intera stringa *(x$)* viene fornita. Se *n* = 0, viene fornita la stringa nulla (lunghezza zero).

Vedere anche le funzioni MID$ e RIGHT$.

Esempio: In questo esempio, si usa la funzione LEFT$ per estrarre i primi cinque caratteri della stringa "BASIC PROGRAM".

```
10 A$ = "BASIC PROGRAM"
20 B$ = LEFT$(A$,5)
30 PRINT B$
RUN
BASIC
```

LEN (Funzione)

Scopo: Fornire il numero di caratteri di *x$*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v* = LEN(*x$*)

Note: *x$* è un'espressione a stringa.

Nel totale del numero di caratteri si includono i caratteri non stampabili e gli spazi in bianco.

Esempio: Nella stringa "BOCA RATON, FL" vi sono 14 caratteri, poiché vengono conteggiati anche lo spazio in bianco e la virgola.

```
10 X$ = "BOCA RATON,FL"
20 PRINT LEN(X$)
RUN
 14
```

LET
(Specifica)

Scopo: Assegnare il valore di un'espressione ad una variabile.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: [LET] *variabile=espressione*

Note:

variabile è il nome della variabile o dell'elemento di schiera che deve ricevere un valore. Può essere una variabile a stringa o numerica, oppure un elemento di schiera.

espressione è l'espressione il cui valore viene assegnato alla *variabile*. Il tipo dell'espressione (stringa o numerica) deve corrispondere a quello della variabile, altrimenti si verifica l'errore **"Type mismatch".**

LET
(Specifica)

Il termine LET è facoltativo; il segno di uguale è sufficiente quando si assegna una espressione ad un nome di variabile.

Esempio: Questo esempio assegna il valore 12 alla variabile DORI. Poi assegna il valore 14, che è il valore dell'espressione DORI+2, alla variabile E. La stringa "HORA" è assegnata alla variabile FDANCE$.

```
10 DORI=12
20 E=DORI+2
30 FDANCE$="HORA"
```

Le stesse specifiche possono essere scritte:

```
10 LET DORI=12
20 LET E=DORI+2
30 LET FDANCE$="HORA"
```

LINE
(Specifica)

Scopo: Disegnare una riga o un quadrato sullo schermo.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Solo modo grafici.

Formato: LINE [(*x1,y1*)]—(*x2,y2*) [,[*attributo*] [,B[F]] [,*style*]]

Note: *(x1,y1), (x2,y2)*

Sono coordinate sia in forma assoluta che relativa. Fare riferimento al paragrafo "Definizione delle coordinate" nel Capitolo 3 del manuale BASIC.

attributo è un'espressione intera che sceglie un attributo dalla gamma disponibile del modo schermo in uso. A media risoluzione il colore è l'attributo definito nella specifica COLOR. Sono disponibili 4 attributi (Ø-3). Ad alta risoluzione gli attributi disponibili sono 2 (Ø-1). Zero è l'attributo per lo sfondo. L'attributo assunto per il colore del primo piano è sempre il massimo attributo per quel tipo di schermo: 3 a media risoluzione 1 ad alta.

style è una maschera intera di 16 bit usata per porre i punti sullo schermo. Il parametro *style* può essere usato per righe e quadrati normali, ma non può essere usato per riquadri pieni (BF). Usare *style* con BF genera l'errore **Syntax Error** (per il BASIC 2.Ø e successive versioni).

LINE
(Specifica)

La forma più semplice di LINE è:

```
LINE -(X2,Y2)
```

Questa disegna una riga dall'ultimo punto a cui si è fatto riferimento al punto *(x2,y2)* del colore di primo piano.

Si può includere anche un punto di partenza:

```
LINE (Ø,Ø)-(319,199) 'linea diagonale
LINE (Ø,1ØØ) 'linea orizzontale
```

Si può indicare il colore per disegnare la riga:

```
LINE (1Ø,1Ø)-(2Ø,2Ø), 2' disegna col colore 2

1Ø 'traccia linee casuali con colori casuali
2Ø SCREEN 1,Ø,Ø,Ø: CLS
3Ø LINE -(RND*319,RND*199),RND*4
4Ø GOTO 2Ø

1Ø 'modello alternativo-linea on, linea off
2Ø SCREEN 1,Ø,Ø,Ø: CLS
3Ø FOR X=Ø TO 319
4Ø LINE (X,Ø)-(X,199),X AND 1
5Ø NEXT
```

Altro argomento **LINE** è **B** o **BF.** E' possibile tralasciare *l'attributo* ed includere l'argomento finale.

```
LINE (Ø,Ø)-(1ØØ,1ØØ),,B 'quadrato in colore di primopiano
```

oppure si può includere l'attributo.

```
LINE (Ø,Ø)-(1ØØ,1ØØ),2,BF 'quadrato riempito con colore 2
```

LINE (Specifica)

La B indica al BASIC di disegnare un rettangolo con i punti *(x1,y1)* e *(x2,y2)* come angoli opposti. Questo evita l'immissione di quattro comandi LINE:

```
LINE (XI,Y1)-(X2,Y1)
LINE (X1,Y1)-(X1,Y2)
LINE (X2,Y1)-(X2,Y2)
LINE (X1,Y2)-(X2,Y2)
```

che eseguono la funzione equivalente.

BF significa disegnare lo stesso rettangolo come viene disegnato da B, ma anche riempire l'interno di punti nel colore scelto.

L'ultimo parametro è *style*. In questo parametro ogni bit rappresenta un punto. Se il bit è 1 il punto viene tracciato se è Ø no. Dopo ogni punto viene selezionata la posizione successiva del bit in *style*. Quando viene raggiunto l'ultimo bit si ricomincia dal primo.

Notare che il bit Ø (zero) indica che non è stata eseguita nessuna cancellazione o memorizzazione del punto sullo schermo. Si può tracciare una riga di fondo prima di una riga mediante *style* per forzare uno sfondo già noto.

Il parametro style si può usare per tracciare una riga punteggiata attraverso lo schermo. Poiché style è ampio 16 bit il modello per una linea tratteggiata è:

```
1 Ø 1 Ø 1 Ø 1 Ø 1 Ø 1 Ø 1 Ø
```

Questo è simile a HAAAA in termini esadecimali.

Esempio: Per disegnare una riga punteggiata:

```
1Ø SCREEN 1,Ø
2Ø LINE (Ø,Ø)-(319,199),,,&HAAAA
```

LINE (Specifica)

Per disegnare un quadrato azzurro tratteggiato:

```
10 SCREEN 1,0
20 LINE (0,0)-(100,100),1,B,&HCCCC
```

Nel BASIC 2.0 le coordinate non comprese nei limiti non sono visibili. Nel BASIC 1.1 invece vengono riportate nella riga orizzontale successiva.

L'ultimo punto a cui si è fatto riferimento dopo una specifica LINE è il punto *(x2,y2)*. Se per la seconda coordinata si usa la forma relativa, è relativa rispetto alla prima coordinata. Ad esempio:

```
LINE (100,100)-STEP (10,-20)
```

disegna una riga da (100,100) a (110,80)

Questo esempio disegna quadrati a caso riempiti con colori casuali.

```
10 CLS
20 SCREEN 1,0: COLOR 0,0
30 LINE -(RND*319,RND*199), RND*2+1,BF
40 GOTO 30 'i quadrati eccedono
```

LINE INPUT (Specifica)

Scopo: Leggere un'intera riga (fino a 255 caratteri) dalla tastiera e metterla in una variabile a stringa ignorando i delimitatori.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: LINE INPUT [;] ["*richiesta*";] *varstringa*

Note:

"richiesta" è una costante a stringa che viene visualizzata prima che l'immissione venga accettata. Non viene stampato un punto interrogativo, a meno che non sia parte della stringa di richiesta.

varstringa è il nome della variabile a stringa o elemento di schiera al quale sarà assegnata la riga. Tutta l'immissione, dalla fine della richiesta fino alla pressione del tasto di immissione, viene assegnata a *varstringa*. Gli spazi bianchi di coda vengono ignorati.

Nel BASIC su disco e avanzato, se LINE INPUT è seguita immediatamente da un punto e virgola, premendo il tasto di immissione per terminare la riga di immissione, non si produce a video una sequenza ritorno a margine/avanzamento riga. Cioè il cursore rimane sulla stessa riga della risposta.

Si può terminare la funzione LINE INPUT premendo Ctrl-Interr. Il BASIC ritorna a livello di comando e visualizza **Ok.** Poi si può immettere CONT per riprendere l'esecuzione dalla specifica LINE INPUT.

Esempio: vedere l'esempio della sezione successiva, "Specifica LINE INPUT #".

LINE INPUT # (Specifica)

Scopo: Leggere un'intera riga (fino a 255 caratteri), ignorando i delimitatori, da un file sequenziale in una variabile a stringa.

Versioni:	Cassetta	Disco	Avanzato	Compilatore
	***	***	***	***

Formato: LINE INPUT # *numfile, varstringa*

Note:

numfile è il numero con il quale è stato aperto il file

varstringa è il nome di una variabile a stringa o elemento di schiera al quale sarà assegnata la riga.

LINE INPUT # legge tutti i caratteri del file sequenziale fino al ritorno a margine. Poi salta la sequenza ritorno a margine/avanzamento riga e la successiva LINE INPUT # e legge tutti i caratteri fino al successivo ritorno a margine. (Se si incontra una sequenza avanzamento riga/ritorno a margine, questa viene mantenuta. Cioè i caratteri di avanzamento riga, ritorno a margine, vengono ritornati come parte della stringa).

LINE INPUT ≠ è particolarmente utile se ogni riga di un file è stata spezzata in campi, oppure se un altro programma sta leggendo come dati un programma BASIC salvato in modo ASCII. Fare riferimento all'appendice A, "Immissione ed emissione su disco nel BASIC" nel manuale del BASIC.

LINE INPUT
(Specifica)

Esempio: il seguente esempio usa LINE INPUT per ottenere le informazioni dalla tastiera, informazioni che probabilmente contengono virgole o altri delimitatori. Successivamente le informazioni vengono scritte in un file sequenziale e lette ancora dal file con la specifica LINE INPUT#.

```
Ok
10 OPEN "LIST" FOR OUTPUT AS #1
20 LINE INPUT "Indirizzo? ";C$
30 PRINT #1, C$
40 CLOSE 1
50 OPEN "LIST" FOR INPUT AS #1
60 LINE INPUT #1, C$
70 PRINT C$
80 CLOSE 1
RUN
Indirizzo?
```

Si supponga di rispondere con DELRAY BEACH, FL 33345. Il programma continua:

```
Indirizzo? DELRAY BEACH, FL    33445
DELRAY BEACH,FL       33445
```

LIST (Comando)

Scopo: Listare il programma attualmente in memoria sullo schermo o su un'altra unità specificata.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	

Formato: LIST [*riga1*][-[*riga2*]] [,*specfile*]

Note:

riga1, riga2 sono numeri di riga validi compresi tra 0 e 65529. *riga1* è la prima riga da listare; *riga2*, l'ultima. Per entrambi i numeri di riga, si può usare un punto (.) per indicare la riga attuale.

specfile è una espressione a stringa per la specifica del file. Nel BASIC 2.0 e successivi livelli, può contenere un percorso. *Specfile* deve essere conforme alle regole esposte nel capitolo 3 del manuale BASIC. In caso contrario apparirà un errore.

Nella versione BASIC su cassetta, la stampa dello schermo determinata dalla omissione della specifica dall'unità può essere interrotta mediante la pressione dei tasti Ctrl-Interr. La lista indirizzata a specifiche unità non può essere interrotta. Cioè LIST *range* può essere interrotta, LIST *range,* "SCRN" no.

Nel BASIC su disco e nel BASIC avanzato, qualsiasi lista inviata a schermo o stampatrice può essere interrotta premendo Ctrl-Interr.

LIST
(Comando)

Se si omette la gamma di righe, l'intero programma viene listato.

Se si usa un trattino (-) nella gamma di righe, sono disponibili tre opzioni:

- Se si indica solo *riga1*, viene listata quella riga e tutte le altre con numero più alto.
- Se si indica solo *riga2*, vengono listate tutte le righe dall'inizio del programma fino a *riga2*.
- Se si specificano entrambi i numeri di riga, vengono listate tutte le righe da *riga1* a *riga2*, compresa.

Quando la lista viene eseguita in un file su disco o cassetta la parte di programma specificata viene salvata in formato ASCII. Successivamente questo file può essere utilizzato con MERGE. Dopo l'esecuzione di LIST il BASIC ritorna sempre a livello di comando.

LIST (Comando)

Esempio: Lista l'intero programma sullo schermo.

```
LIST
```

Lista la riga 35 sullo schermo.

```
LIST 35,"SCRN:"
```

Lista le righe dalla 10 alla 20 sulla stampatrice.

```
LIST 10-20, "LPT1:"
```

Lista tutte le righe dalla riga 100 alla fine del programma al primo adattatore per le comunicazioni a 1200 bps, senza parità, 8 bit di dati, 1 bit di arresto.

```
LIST 100- ,"COM1:1200,N,8"
```

Esegue la lista dalla prima riga alla 200 in un file denominato "BOB" su cassetta.

```
LIST -200,"CAS1:BOB"
```

LLIST
(Comando)

Scopo: Listare tutto o parte del programma attualmente in memoria sulla stampatrice (LPT1:).

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	

Formato: LLIST [*riga1*] [-[*riga2*]]

Note: I limiti del numero di riga per LLIST corrispondono a quelli per LIST.

Il BASIC ritorna sempre a livello di comando dopo l'esecuzione di LLIST.

Esempio: Stampa l'intero programma.

```
LLIST
```

Stampa la riga 35.

```
LLIST 35
```

Lista le righe dalla 10 alla 20 sulla stampatrice.

```
LLIST 10-20
```

Stampa tutte le righe dalla numero 100 fino alla fine del programma.

```
LLIST 100-
```

Stampa dalla prima riga fino alla 200.

```
LLIST -200
```

LOAD
(Comando)

Scopo: Caricare un programma in memoria dall'unità specificata e facoltativamente eseguirlo.

Versioni:	Cassetta	Disco	Avanzato	Compilatore
	***	***	***	

Formato: LOAD *specfile*[,R]

Note: *specfile* è un'espressione a stringa per la specifica del file. In BASIC 2.Ø e successivi livelli può contenere un percorso. Deve essere conforme alle regole descritte al Capitolo 3, del manuale BASIC altrimenti si verifica un errore ed il caricamento viene annullato.

LOAD chiude tutti i file aperti e cancella tutte le variabili e le righe di programma attualmente in memoria prima di caricare il programma specificato. Se si omette l'opzione **R,** il BASIC ritorna al modo diretto dopo il caricamento del programma.

Tuttavia se si usa l'opzione **R** con LOAD, il programma viene eseguito dopo il suo caricamento. In questo caso, tutti i file aperti vengono mantenuti tali. Perciò si può usare LOAD con l'opzione **R** per concatenare più programmi (o segmenti dello stesso programma). Le informazioni possono essere passate tra i programmi usando i file di dati.

LOAD *specfile,* R equivale a RUN *specfile.*

LOAD (Comando)

Se si usa il BASIC su cassetta e si omette il nome dell'unità, si assume CAS1:. CAS1: è l'unica unità disponibile per LOAD nel BASIC su cassetta.

Se si usa il BASIC su disco o il BASIC avanzato omettendo l'unità, si usa il modulo a minidischi assunto per difetto dal DOS. Se non viene fornita alcuna estensione ed il nome del file è di otto caratteri al massimo, si aggiunge l'estensione **.BAS** al nome del file.

Note nell'utilizzo di CAS1:

1. Se si immette la specifica LOAD in modo diretto, i nomi dei file sul nastro saranno visualizzati seguiti da un punto (.) e da una singola lettera che indica il tipo di file. Questa è seguita dal messaggio **"Skipped"** (Saltato), per i file che non corrispondono a quello indicato, o **"Found"** (Trovato) in caso contrario. I tipi di file e le loro lettere corrispondenti sono:

   .B per i programmi BASIC in formato interno (creati con il comando SAVE).
   .P per i programmi BASIC protetti in formato interno (creati con il comando SAVE ,P).
   .A per i programmi BASIC in formato ASCII (creati con il comando SAVE ,A).
   .M per file ad immagine di memoria (creati con il comando BSAVE).
   .D per file di dati (creati da OPEN seguito da specifiche di emissione).

Per sapere quali file si trovano sul nastro, riavvolgere lo stesso ed immettere un nome di file che si ritiene non sia presente sul nastro. Ad esempio: LOAD "CAS1: NOWHERE". Saranno visualizzati tutti i file.

Se si esegue il comando LOAD in un programma BASIC, non vengono visualizzati i nomi dei file saltati e trovati.

2. Notare che durante LOAD è possibile premere Ctrl-Interr in qualsiasi momento. Tra i file o dopo un periodo di supero tempo, il BASIC termina la ricerca e ritorna a livello di comando. I precedenti contenuti di memoria rimangono invariati.

3. Se si specifica CAS1: come unità e si omette il nome del file, viene caricato il successivo file di programma sul nastro.

Esempio: Questo esempio carica il programma MENU, ma non lo esegue.

```
LOAD "MENU"
```

Questo esempio carica ed esegue il programma INVENT.

```
LOAD "INVENT",R
```

che equivale a

```
RUN "INVENT"
```

Questo esempio carica il file VLAD .BAS dal disco del modulo B. Notare che non è necessario specificare **.BAS.**

```
LOAD "B:VLAD.BAS"
```

Questo esempio carica il successivo programma sul nastro.

```
LOAD "CAS1:"
```

LOC (Funzione)

Scopo: Fornire l'attuale posizione nel file.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	***

Formato: *v* = LOC *(numfile)*

Note: *numfile* è il numero usato all'apertura del file.

Con i file ad accesso casuale, LOC fornisce il numero dell'ultimo record letto o scritto in un file ad accesso casuale da quando il file è stato aperto.

Con i file sequenziali, LOC fornisce il numero di record letti o scritti nel file, dal momento in cui è stato aperto. (Un record per un file sequenziale è un blocco di dati di 128 byte). Quando si apre un file per l'immissione sequenziale, il BASIC legge il primo settore del file, in modo che LOC fornisca un 1 prima di qualsiasi immissione dal file.

Per un file di comunicazione, LOC fornisce il numero di caratteri della memoria di transito di immissione che attendono di essere letti. L'ampiezza per difetto della memoria di transito di immissione è 256 caratteri, ma è possibile modificarla con l'opzione /**C:** nella linea di comando del BASIC. Se nella memoria di transito ci sono più di 255 caratteri, LOC ritorna 255. Poiché una stringa è limitata a 255 caratteri, questo limite pratico elimina la necessità di provare la stringa prima di immettervi i dati. Se nella memoria di transito rimangono meno di 255 caratteri, LOC fornisce il totale reale.

LOC (Funzione)

Esempio: Questo esempio arresta il programma se si supera il 50esimo record del file.

```
100 IF LOC(1)>50 THEN STOP
```

Questo esempio può essere usato per riscrivere il record appena letto.

```
100 PUT #1,LOC(1)
```

LOCATE (Specifica)

Scopo: Posizionare il cursore sullo schermo attivo. I parametri facoltativi attivano e disattivano il lampeggiamento del cursore e definiscono l'ampiezza del cursore lampeggiante.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: LOCATE [*fila*][,[*col*] [,[*cursore*][,[*avvio*] [*arresto*]]]]

Note:

fila è un'espressione numerica compresa tra 1 e 25. Indica il numero di riga dello schermo dove si vuole posizionare il cursore.

col è un'espressione numerica compresa tra 1 e 40 o 1 e 80, a seconda dell'ampiezza dello schermo. Indica il numero della colonna sullo schermo dove si vuole posizionare il cursore.

cursore è un valore che indica se il cursore è visibile o meno. Uno 0 indica la condizione negativa, un 1 quella positiva.

avvio è la riga di scansione iniziale del cursore. Deve essere un'espressione numerica compresa tra 0 e 31.

arresto è la riga di scansione finale del cursore. Deve essere un'espressione numerica compresa tra 0 e 31.

cursore, avvio e arresto non sono validi per il modo grafici.

LOCATE (Specifica)

avvio e arresto consentono di dare al cursore l'ampiezza desiderata. Si indicano le righe di avvio e di arresto scansione. Le righe di scansione sono numerate a partire da 0. La riga di scansione finale è 7 se si possiede l'Adattatore per il video grafico a colori, 13 se si possiede l'Adattatore per il video monocromatico IBM e stampatrice parallela. Se si indica *avvio* e si omette *arresto,* quest'ultimo assume il valore di *avvio.*
Se *avvio* è maggiore di *arresto,* si ottiene un cursore in due parti. Il cursore passa dall'ultima riga dello schermo alla prima.

Dopo una specifica LOCATE, le specifiche di I/E sullo schermo iniziano a porre i caratteri alla posizione specificata.

Durante l'esecuzione di un programma, il cursore è generalmente spento. Si può utilizzare LOCATE,, 1 per riattivarlo. Normalmente il BASIC non stampa sulla riga 25. Tuttavia si può disattivare la visualizzazione dei tasti usando KEY OFF, quindi per scrivervi sopra usare LOCATE 25, 1:PRINT.... E' possibile omettere qualsiasi parametro. I parametri omessi assumono il valore corrente.

I valori immessi, che non sono compresi nei limiti indicati, producono un errore **(Illegal function call).** Vengono conservati i valori precedenti.

LOCATE
(Specifica)

Esempio: Questo esempio sposta il cursore sulla posizione iniziale, nell'angolo superiore sinistro dello schermo.

```
10 LOCATE 1,1
```

Questo esempio rende visibile il cursore lampeggiante; la sua posizione rimane invariata.

```
10 LOCATE ,,1
```

In questo esempio la posizione e la visibilità del cursore rimangono invariate. Il cursore viene visualizzato sotto il carattere, con l'Adattatore per il video grafico a colori (avvio e arresto sulla riga di scansione 7).

```
10 LOCATE ,,,7
```

In questo esempio il cursore si sposta alla riga 5, colonna 1. Rende visibile il cursore, coprendo l'intero carattere, con l'Adattatore per il video grafico a colori, con avvio alla riga di scansione 0 e arresto alla riga di scansione 7.

```
40 LOCATE 5,1,1,0,7
```

LOF (Funzione)

Scopo: Fornire il numero di byte assegnati al file (lunghezza del file).

Versioni:	Cassetta	Disco	Avanzato	Compilatore
		***	***	***

Formato: *v* = LOF*(numfile)*

Note: *numfile* è il numero usato quando è stato aperto il file.

Per i file su minidisco creati dal BASIC 1.1Ø, LOF fornisce un multiplo di 128. Ad esempio, se i dati reali nel file sono 257 byte, sarà fornito il numero 384. Per i file su minidisco non creati dal BASIC (ad esempio usando EDLIN), e per i file creati col BASIC 2.Ø, e successive versioni, LOF, fornisce il numero reale di byte allocati al file.

Per le comunicazioni, LOF fornisce la quantità di spazio libero nella memoria di transito di immissione. Cioè *amp*- LOC *(numfile)*, dove *amp* è l'ampiezza della memoria di transito di comunicazione, il cui valore assunto è 256, ma può essere modificato con l'opzione /C: nella linea di comando BASIC. LOF può essere utilizzato per sapere se la memoria di transito di immissione è piena. Praticamente la funzione LOC è adeguata a questo scopo come mostrato nell'esempio riportato nell'Appendice C "Comunicazioni".

LOF (Funzione)

Esempio: queste specifiche estraggono l'ultimo record del file denominato BIG, assumendo che BIG sia stato creato con una lunghezza record di 128 byte:

```
10 OPEN "BIG" AS #1
20 GET #1,LOF(1)/128
```

LOG (Funzione)

Scopo: Fornire il logaritmo naturale di *x*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v* = LOG(*x*)

Note: *x* deve essere un'espressione numerica maggiore di zero.

Il logaritmo naturale è il logaritmo in base *e*.

Nel BASIC 2.0 e successive versioni questo calcolo può essere eseguito a doppia precisione specificando /D nella riga di comando BASIC quando questo viene inizialmente caricato.

Esempio: il primo esempio calcola il logaritmo dell'espressione 45/7:

```
PRINT LOG(45/7)
 1.86Ø752
```

Il secondo esempio calcola il logaritmo di *e* e di e^2.

```
E=2.718282

? LOG(E)
 1

? LOG(E*E)
 2
```

LPOS (Funzione)

Scopo: Fornire la posizione attuale della testina di stampa nella memoria di transito della stampatrice per LPT1:.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v* = LPOS(*n*)

Note: *n* è un'espressione numerica che è un argomento fittizio nel BASIC su Cassetta. Nel BASIC su disco e nel BASIC avanzato, *n* indica quale stampatrice si stia provando:

0 o 1	LPT1:
2	LPT2:
3	LPT3:

Perciò si raccomanda di usare 0 o 1 nel BASIC su cassetta per mantenere la compatibilità con le altre versioni.

La funzione LPOS non indica necessariamente la posizione fisica della testina di stampa sulla stampatrice.

Esempio: In questo esempio, se la lunghezza della riga supera i 60 caratteri, un carattere di ritorno a margine viene inviato alla stampatrice in modo da saltare alla riga successiva.

```
100 IF LPOS(0)>60 THEN LPRINT CHR$(13)
```

LPRINT e LPRINT USING (Specifiche)

Scopo: Stampare i dati sulla stampatrice (LPT1:).

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: LPRINT [*lista di espressioni*] [;]

LPRINT USING *v$; lista di espressioni* [;]

Note:

lista di espressioni è una lista di espressioni numeriche e/o a stringa che devono essere stampate. Le espressioni devono essere separate da virgole o punti e virgola.

espressione è una espressione a stringa o numerica il cui valore deve essere stampato.

v$ è una costante o variabile a stringa che identifica il formato da usare per la stampa. Vedere la specifica PRINT per una descrizione dettagliata.

Queste specifiche lavorano come PRINT e PRINT USING tranne nel fatto che l'emissione va alla stampatrice. Vedere le specifiche PRINT e PRINT USING.

LPRINT assume per la stampatrice una lunghezza riga di 80 caratteri. Cioè il BASIC inserisce automaticamente un ritorno a margine/ avanzamento riga dopo la stampa di 80 caratteri. Questo produce il salto di due righe se si stampano esattamente 80 caratteri, a meno che non si termini la specifica con un punto e virgola. E' possibile modificare il valore dell'ampiezza con una specifica WIDTH "LPT1:".

LPRINT e LPRINT USING (Specifiche)

Se si effettua un avanzamento pagina (LPRINT CHR$(12);) seguito da un'altra LPRINT e la stampatrice impiega più di 10 secondi per l'avanzamento di pagina, si può verificare l'errore **Device Time out** sulla seconda LPRINT. Per evitare l'inconveniente immettere:

```
1 ON ERROR GOTO 65000
.
.
.
65000 IF ERR = 24 THEN RESUME '24=timeout
```

Si può provare ERR per assicurarsi che il supero tempo sia stato causato da una specifica LPRINT.

Esempio: Questo è un esempio di invio di caratteri di controllo speciali alla stampatrice grafica IBM da 80 CPS usando LPRINT e CHR$. Questi caratteri di controllo della stampatrice si trovano nel manuale di riferimento dell'elaboratore.

```
10 LPRINT CHR$(14);"Riga di titolo
20 FOR I=2 TO 4
30 LPRINT "Riga di dettaglio";I
40 NEXT I
50 LPRINT CHR$(15);"Stampa condensata; 132 car/riga"
60 LPRINT CHR$(18);"Ritorna alla stampa normale"
70 LPRINT CHR$(27);"E"
80 LPRINT "Questa è una stampa enfatizzata"
90 LPRINT CHR$(27);"F"
100 LPRINT "Torna di nuovo alla stampa normale"
```

LPRINT e LPRINT USING (Specifiche)

L'emissione prodotta da questo programma appare così:

```
  Riga di titolo
Riga di dettaglio 2
Riga di dettaglio 3
Riga di dettaglio 4
Stampa condensata; 132 car/riga
Ritorna normale

Questa è una stampa enfatizzata

Torna di nuovo normale
```

LSET e RSET (Specifiche)

Scopo: Spostare i dati nella memoria di transito di un file ad accesso casuale in preparazione di una specifica PUT (file).

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	***

Formato: LSET *varstringa* = *x$*

RSET *varstringa* = *x$*

Note:

varstringa è il nome di una variabile che è stata definita in una specifica FIELD.

x$ è un'espressione a stringa per le informazioni da porre nel campo identificato da *varstringa.*

Se x$ richiede meno byte di quelli specificati per *varstringa* nella specifica FIELD, LSET allinea a sinistra la stringa nel campo e RSET la allinea a destra (si usano degli spazi per riempire le posizioni in eccesso). Se *x$* è più lunga di *varstringa,* vengono omessi i caratteri a destra.

I valori numerici devono essere convertiti in stringhe prima di LSET o RSET. Vedere le funzioni MKI$, MKS$ e MKD$.

Per una spiegazione dell'uso dei file ad accesso casuale, vedere l'appendice A sul manuale del BASIC.

LSET e RSET (Specifiche)

Nota: si può utilizzare LSET o RSET con una variabile a stringa che non è stata definita in una specifica FIELD per allineare a sinistra o a destra una stringa in un determinato campo. Ad esempio, le righe di programma:

```
10 A$=SPACE$(20)
20 RSET A$=N$
```

allineano a destra la stringa N$ in un campo di 20 caratteri. Può essere utile per organizzare l'emissione stampata.

Esempio: questo esempio converte il valore numerico AMT in una stringa e la allinea a sinistra nel campo A$ in preparazione della specifica PUT (file).

```
10 LSET A$=MKS$(AMT)
```

MERGE (Comando)

Scopo: Fondere le righe di un file di un programma ASCII nel programma attualmente in memoria.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	

Formato: MERGE *specfile*

Note:

specfile è un'espressione a stringa per la specifica del file. Nel BASIC 2.0 e successivi livelli può contenere un percorso. Deve attenersi alle regole sulla "Denominazione dei file" descritte al Capitolo 3, del manuale BASIC, altrimenti si verifica un errore.

Si cerca il file indicato nell'unità. Se trovato, le righe di programma del file sull'unità sono fuse con quelle in memoria. Se qualche riga del file che si sta fondendo ha lo stesso numero delle righe del programma in memoria, le righe del file vengono sostituite dalle corrispondenti in memoria.

Dopo il comando MERGE, il programma fuso risiede in memoria ed il BASIC ritorna a livello di comando.

Nel BASIC su Cassetta, se si omette il nome dell'unità, si assume CAS1:. CAS1: è l'unica unità disponibile per MERGE con il BASIC su cassetta. Con il BASIC su disco e con il BASIC avanzato, se si omette il nome dell'unità, si assume il modulo assunto per difetto dal DOS.

Se si specifica CAS1: come nome di unità e si omette il nome del file, viene fuso il successivo file di programma ASCII del nastro.

Se il programma da fondere non è stato salvato in formato ASCII (usando l'opzione A nel comando SAVE), si verifica l'errore **BAD file mode.** Il programma in memoria rimane invariato.

Esempio: In questo esempio viene fuso il file "NUMBRS" nel modulo A con il programma in memoria.

```
MERGE "A:NUMBRS"
```

MID$
(Funzione e specifica)

Scopo: Fornire la parte richiesta di una determinata stringa. Usata come specifica, come nel secondo formato, sostituisce una porzione di una stringa con un'altra stringa.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: Come funzione:

v$* = MID$(x$,n[,m])*

Come specifica:

MID$*(v$,n[,m])* = y$

Note: **Per la funzione (v$ = MID$...):**

x$ è una qualsiasi espressione a stringa.

n è un'espressione intera compresa tra 1 e 255.

m è un'espressione intera compresa tra 0 e 255.

La funzione fornisce una stringa di caratteri di lunghezza *m* da *x$* a partire dall'*n*esimo carattere. Se si omette *m* o se vi sono meno di *m* caratteri alla destra dell'*n*esimo carattere, vengono forniti tutti i caratteri più a destra a partire dall'*n*esimo. Se *m* è uguale a zero o se *n* è maggiore di LEN*(x$)*, MID$ fornisce una stringa nulla.

Vedere anche le funzioni LEFT$ e RIGHT$.

MID$
(Funzione e specifica)

Per la specifica (MID$...=y$):

v$	è una variabile a stringa o elemento di schiera di cui si devono sostituire i caratteri.
n	è un'espressione intera compresa tra 1 e 255.
m	è un'espressione intera compresa tra 0 e 255.
y$	è un'espressione a stringa.

I caratteri di *v$*, a partire da posizione *n*, vengono sostituiti dai caratteri di *y$*. La *m* facoltativa fa riferimento al numero di caratteri di *y$* utilizzati nella sostituzione. Se si omette *m*, si usa tutta l'espressione *y$*.

Comunque senza tener conto del fatto che *m* sia incluso o meno la lunghezza di *v$* non cambia. Per esempio se *v$* è di 4 caratteri e *y$* di 5, dopo l'operazione *v$* conterrà solo i primi 4 caratteri di *y$*.

> **Nota:** se *n* o *m* non è compresa nei limiti, si verifica un errore di **Illegal function call**.

MID$
(Funzione e specifica)

Esempio: il primo esempio usa la funzione MID$ per scegliere la parte centrale della stringa **B$.**

```
10 A$="GOOD "
20 B$="MORNING EVENING AFTERNOON"
30 PRINT A$;MID$(B$,9,7)
RUN
GOOD EVENING
```

L'esempio successivo usa la specifica MID$ per accedere a più sottostringhe inserite in una stringa più ampia. Questa tecnica riduce la frammentazione dello spazio della stringa.

```
10 RECORD$ = STRING$(255,0)
20 PART1.OFF = 1
30 PART1.LEN = 5
40 PART2.OFF = 6
50 PART2.LEN = 15
   .
   .
   .
100 MID$(RECORD$,PART1.OFF,PART1.LEN) = "STRNG"
```

MKDIR (Comando)

Scopo: Creare un indirizzario sul disco specificato. Disponibile col BASIC livello 2.0 e successivi livelli.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	

Formato: MKDIR *percorso*

Note:

percorso è una espressione a stringa non superiore a 63 caratteri che identifica il nuovo indirizzario da creare. Per ulteriori informazioni sui percorsi fare riferimento al capitolo 3, paragrafi "Denominazione dei file" e "Indirizzari strutturati ad albero" del manuale BASIC.

Esempi: Per creare dall'indirizzario radice un sottoindirizzario di nome SALES immettere:

```
MKDIR "SALES"
```

Per creare dall'indirizzario radice un sottoindirizzario chiamato MIKE nell'indirizzario SALES:

```
MKDIR "SALES\MIKE"
```

Per creare dall'indirizzario radice un sottoindirizzario chiamato ALICE nell'indirizzario MIKE:

```
MKDIR "SALES\MIKE\ALICE
```

MKDIR (Comando)

Per creare dall'indirizzario radice, un sottoindirizzario chiamato ACCTING:

```
MKDIR "ACCTING"
```

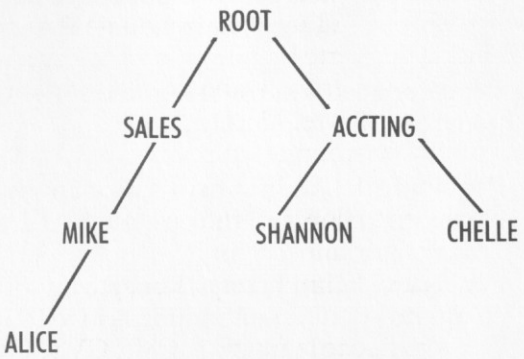
Per far diventare ACCTING l'indirizzario in uso, e poi creare due sottoindirizzari chiamati SHANNON e CHELLE:

```
CHDIR "ACCTING"
MKDIR "SHANNON"
MKDIR "CHELLE"
```

La stessa struttura può essere creata dalla radice immettendo:

```
MKDIR "ACCTING\SHANNON"
MKDIR "ACCTING\CHELLE"
```

Seguendo gli esempi precedenti si creerà un indirizzario strutturato ad albero come il seguente:



MKI$, MKS$, MKD$ (Funzioni)

Scopo: Convertire i valori di tipo numerico in valori di tipo stringa.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	***

Formato: *v$* = MKI$ *(espressione intera)*

*v$* = MKS$ *(espressione a precisione singola)*

*v$* = MKD$ *(espressione a precisione doppia)*

Note: Qualsiasi valore numerico posto nella memoria di transito di un file ad accesso casuale con una specifica LSET o RSET deve essere convertito in una stringa. MKI$ converte un intero in una stringa di 2 byte. MKS$ converte un numero a precisione singola in una stringa di 4 byte. MKD$ converte un numero a precisione doppia in una stringa di 8 byte.

Queste funzioni si differenziano da STR$ nel fatto che non modificano realmente i byte di dati, ma il modo in cui il BASIC interpreta quei byte.

Vedere anche le funzioni CVI, CVS, CVD in questo manuale e l'appendice A del manuale BASIC.

MKI$, MKS$, MKD$ (Funzioni)

Esempio: Questo esempio usa un file ad accesso casuale (#1) con campi definiti a riga 100. Il primo campo, D$, deve contenere un valore numerico, AMT. La riga 110 converte AMT in un valore a stringa usando MKS$ ed usa LSET per porre il valore reale di AMT nella memoria di transito del file ad accesso casuale. La riga 120 pone una stringa nella memoria di transito (non è necessario convertire una stringa); poi la riga 130 scrive i dati dalla memoria di transito del file ad accesso casuale nel file.

```
100 FIELD#1, 4 AS D$,20 AS N$
110 LSET D$ = MKS$(AMT)
120 LSET N$ = A$
130 PUT #1
```

MOTOR (Specifica)

Scopo: Accendere e spegnere da un programma il registratore a cassette.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	

Formato: MOTOR [*stato*]

Note:

stato è un'espressione numerica che indica l'attivazione o disattivazione.

Se *stato* non è zero, il registratore è acceso. Se *stato* è zero, il registratore è spento.

Se si omette *stato,* viene collegato il registratore. Cioè se è spento, viene acceso e viceversa.

Esempio: La seguente sequenza di specifiche accende il registratore, poi lo spegne e lo riaccende.

```
10 MOTOR 1
20 MOTOR 0
30 MOTOR
```

NAME (Comando)

Scopo: Modificare il nome di un file su disco. Il comando NAME del BASIC è simile al comando RENAME del DOS.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	***

Formato: NAME *specfile* AS *nomefile*

Note:

specfile è una espressione a stringa per la specifica del file. Nel BASIC 2.0 e successivi livelli, può contenere un percorso. *Specfile* deve essere conforme alle regole esposte nel Capitolo 3 del manuale BASIC "Denominazione dei file". In caso contrario si verifica un errore.

nomefile è il nuovo nome del file. Deve essere un nome valido come descritto nel capitolo di cui sopra.

Il file specificato da *specfile* deve esistere mentre *nomefile* non deve esistere sul minidisco. In caso contrario si verifica un errore. Se si omette il nome dell'unità, si assume l'unità per difetto del DOS. Notare che l'estensione del file non assume **.BAS**.

Dopo un comando NAME, il file si trova sullo stesso disco, nella stessa area, con il nuovo nome.

NAME (Comando)

Esempio: In questo esempio il file precedentemente denominato ACCTS.BAS nel minidisco che si trova nel modulo A viene ora denominato LEDGER.BAS.

```
NAME "A:ACCTS.BAS" AS "LEDGER.BAS"
```

NEW (Comando)

Scopo: Cancellare il programma attualmente in memoria ed eliminare tutte le variabili.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	

Formato: NEW

Nota: Si usa, normalmente, NEW per togliere un programma dalla memoria prima di immettere un nuovo programma. Il BASIC ritorna sempre a livello di comando dopo l'esecuzione di NEW. NEW chiude tutti i file e disattiva la traccia, se era attiva. Vedere anche i comandi TRON e TROFF.

Esempio: In questo esempio viene cancellato il programma in memoria.

```
NEW
```

OCT$ (Funzione)

Scopo: Fornire una stringa che rappresenta il valore ottale dell'argomento decimale.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v$* = OCT$*(n)*

Note:

n è un'espressione numerica compresa tra -32768 e 65535.

Se *n* è negativa, si usa la forma del complemento a due. Cioè, OCT$*(-n)* è uguale a OCT$ (65536-*n*).

Per la conversione esadecimale, vedere la funzione HEX$.

Esempio: Questo esempio mostra come 24 in decimale corrisponde a 30 in ottale.

```
PRINT OCT$(24)
30
```

ON COM(n) (Specifica)

Scopo: Definire un numero di riga al BASIC per eseguire l'intercettazione quando vi sono informazioni in arrivo nella memoria di transito delle comunicazioni.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
		***	(**)

Formato: ON COM*(n)* GOSUB *riga*

Note:

n è il numero dell'adattatore per le comunicazioni (1 o 2)

riga è il numero della riga iniziale del programma di intercettazione. Impostando *riga* uguale a zero, si disabilita l'intercettazione dell'attività di comunicazione per l'adattatore specificato.

E' necessario eseguire una specifica COM*(n)* ON per attivare questa specifica per l'adattatore *n*. Dopo COM*(n)* ON, se si indica un numero di riga diverso da zero nella specifica ON COM*(n)*, ogni volta che il programma avvia una nuova specifica, il BASIC controlla se sono arrivati dei caratteri nell'adattatore per le comunicazioni indicato. In caso affermativo, il BASIC esegue una GOSUB alla *riga* specificata.

Se si esegue COM*(n)* OFF, non viene eseguita alcuna intercettazione per l'adattatore. Anche se vi è stata attività di comunicazione, questa non viene registrata.

Se si esegue una specifica COM*(n)* STOP, non viene eseguita alcuna intercettazione per l'adattatore. Tuttavia, i caratteri che si ricevono vengono registrati in modo che avvenga un'immediata intercettazione quando si esegue COM*(n)* ON.

ON COM(n) (Specifica)

Quando si verifica una intercettazione viene eseguita una COM*(n)* STOP automatica in modo da evitare intercettazioni ripetitive. Il RETURN del programma di intercettazione esegue automaticamente una specifica COM*(n)* ON, a meno che, all'interno del programma di intercettazione non sia esplicitamente dichiarata una COM*(n)* OFF.

L'intercettazione di azioni non ha luogo quando il BASIC non è in fase di esecuzione di un programma. Quando si verifica l'intercettazione di un errore (che risulta da una specifica ON ERROR), vengono disabilitate automaticamente tutte le intercettazioni (comprese ON COM, ON ERROR, ON PEN, ON PLAY, ON STRIG, e ON TIMER).

Il programma di intercettazione delle comunicazioni legge un intero messaggio dalla linea di comunicazione prima di ritornare. Non è consigliabile usare l'intercettazione delle comunicazioni per messaggi di un solo carattere, poiché ad alta velocità del segnale il mantenimento dell'intercettazione e della lettura di ogni carattere singolo può causare l'eccedenza della memoria di transito di interruzione delle comunicazioni.

E' possibile usare RETURN *riga* per tornare al programma BASIC ad un determinato numero di riga. Tuttavia usare attentamente questo ritorno non locale, poiché qualsiasi GOSUB, WHILE o FOR, attiva al momento dell'intercettazione, rimane tale.

Per uscire da iterazioni attive impostare il contatore delle iterazioni su valori fuori dai limiti oppure inserire delle specifiche di condizione nella iterazione stessa. Questo assicura che ogni iterazione di FOR abbia la corrispondente NEXT ed ogni iterazione di WHILE la corrispondente WEND.

ON COM(n) (Specifica)

Esempio:

```
150 ON COM(1) GOSUB 500
160 COM(1) ON
 .
 .
 .
500     'caratteri in arrivo
 .
 .
 .
590 RETURN 300
```

Questo esempio imposta un programma di intercettazione per il primo adattatore per le comunicazioni a riga 500.

ON ERROR (Specifica)

Scopo: Consentire l'intercettazione degli errori e specificare la prima riga del sottoprogramma di gestione degli stessi.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: ON ERROR GOTO *riga*

Nota: *riga* è il numero della prima riga del programma di intercettazione degli errori. Se non esiste il numero di riga, si verifica un errore del tipo **Underfined line number.**

Dopo aver attivato l'intercettazione degli errori, tutti gli errori rilevati *compresi quelli in modo diretto* causano un salto al sottoprogramma di gestione degli errori specificato.

Per disabilitare l'intercettazione degli errori, eseguire una specifica ON ERROR GOTO 0. Gli errori successivi stamperanno un messaggio di errore e arresteranno l'esecuzione. La comparsa di una specifica ON ERROR GOTO 0 in un sottoprogramma di intercettazione degli errori arresta il BASIC e stampa il messaggio relativo all'errore che ha causato l'intercettazione. E' consigliabile che tutti i sottoprogrammi di intercettazione degli errori eseguano una specifica ON ERROR GOTO 0 se si rileva un errore che non sia possibile correggere.

Dal momento in cui avviene un errore si può considerare che il BASIC stesso sia entro un programma di intercettazione errori che lo fa saltare alla riga indicata nella specifica ON ERROR, fino a che non incontra una specifica RESUME. Si deve usare la specifica RESUME per uscire da un programma di intercettazione errori. Fare riferimento alla specifica RESUME.

ON ERROR (Specifica)

Poiché l'intercettazione degli errori non è attiva in un programma di intercettazione errori, non potrà funzionare un ON ERROR GOTO *riga* (entro un programma di intercettazione errori), con *riga* diversa da Ø.

> **Nota:** Se si verifica un errore durante l'esecuzione di un sottoprogramma di gestione degli errori, viene stampato il messaggio di errore del BASIC e l'esecuzione termina. L'intercettazione degli errori non avviene nel sottoprogramma di gestione degli errori.

Esempio: Questo esempio verifica se l'alloggiamento del disco è stato chiuso o meno quando il programma deve aprire un file.

```
1Ø ON ERROR GOTO 1ØØ
2Ø OPEN "DATA" FOR INPUT AS #1
3Ø END
   .
   .
   .
1ØØ IF ERR=71 THEN LOCATE 23,1
    PRINT "IL DISCO NON E' PRONTO"
11Ø RESUME NEXT
```

ON–GOSUB e ON–GOTO (Specifiche)

Scopo: Saltare ad uno o più numeri di riga specificati, in base al valore di un'espressione.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: ON *n* GOTO *riga*[,*riga*]...

ON *n* GOSUB *riga*[,*riga*]...

Note:

n è un'espressione numerica arrotondata all'intero, se necessario. Deve essere compresa tra Ø e 255, altrimenti si verifica l'errore **Illegal function call.**

riga è il numero di una riga a cui si vuole saltare.

Il valore di *n* determina quale numero di riga della lista sarà usato per il salto. Ad esempio, se il valore di *n* è 3, la destinazione del salto sarà la terza riga della lista.

Nella specifica ON–GOSUB, ogni numero di riga della lista deve essere il primo numero di riga di un sottoprogramma. E' necessario avere una specifica RETURN per tornare alla riga che segue ON–GOSUB.

Se il valore di *n* è zero o maggiore del numero di elementi della lista (ma minore o uguale a 255), il BASIC continua con la successiva specifica eseguibile.

ON–GOSUB e ON–GOTO (Specifiche)

Esempio: Il primo esempio salta alla riga 150 se L–1 è uguale a 1, alla riga 300 se L–1 è uguale a 2, alla riga 320 se L–1 è uguale a 3 e alla riga 390 se L–1 è uguale a 4. Se L–1 è uguale a 0 o maggiore di 4, il programma passa alla successiva specifica.

```
100 ON L-1 GOTO 150,300,320,390
```

Il successivo esempio mostra come usare una specifica ON–GOSUB.

```
100 REM display menù
110 PRINT "1. Routine 1"
120 PRINT "2. Routine 2"
130 PRINT "3. Routine 3"
140 PRINT "4. Routine 4"
150 INPUT "Your choice?"; CHOICE
160 ON CHOICE GOSUB 200, 300, 400, 500
170 GOTO 100 ' visualizza il menù dopo l'esecuzione del programma
200 REM inizio del primo sottoprogramma
    .
    .
    .
290 RETURN
300 REM inizio del secondo sottoprogramma
```

ON KEY(n) (Specifica)

Scopo: Definire un numero di riga che deve essere intercettato dal BASIC quando si preme il tasto funzionale specificato o quello di controllo cursore.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
		***	(**)

Formato: ON KEY*(n)* GOSUB *riga*

Note: *n* è un'espressione numerica compresa tra 1 e 20 che indica il tasto da intercettare, nel modo seguente:

1-10 tasti funzionali F1-F10
11 spostamento del cursore verso l'alto
12 spostamento del cursore a sinistra
13 spostamento del cursore a destra
14 spostamento del cursore verso il basso
15-20 tasti (solo in BASIC 2.0 avanzato e successivi livelli) definiti dalla formula:
KEY *n*, CHR$ *(valore)* +CHR$ *(codice tasto)*.
Per maggiori informazioni vedere la specifica KEY(n) in questo capitolo.

riga è il numero della prima riga del programma di intercettazione per il tasto specificato. Impostando *riga* uguale a zero si disattiva l'intercettazione del tasto.

Per attivare questa specifica si deve eseguire una KEY*(n)* ON. Dopo KEY*(n)* ON, se si indica un numero di riga diverso da zero nella specifica ON KEY*(n)*, ogni volta che il programma avvia una nuova specifica

ON KEY(n) (Specifica)

il BASIC controlla se è stato premuto il tasto specificato. In caso affermativo, il BASIC esegue una GOSUB alla *riga* specificata.

Se si esegue una specifica KEY*(n)* OFF non avviene alcuna intercettazione del tasto specificato. Anche se si preme il tasto, l'azione non viene registrata.

Se si esegue una specifica KEY*(n)* STOP, non avviene alcuna intercettazione del tasto specificato. Tuttavia, se si preme il tasto, l'azione viene registrata, in modo da avere un'immediata intercettazione quando si esegue KEY*(n)* ON.

Quando si verifica un'intercettazione, viene eseguita una KEY*(n)* STOP automatica in modo da impedire intercettazioni ripetitive. RETURN dal programma di intercettazione esegue automaticamente una KEY *(n)* ON, a meno che nel programma di intercettazione non sia stata eseguita una KEY*(n)* OFF

L'intercettazione delle azioni non avviene se il BASIC non sta eseguendo un programma. Quando si verifica l'intercettazione di un errore (che risulta da una specifica ON ERROR) vengono automaticamente disabilitate tutte le intercettazioni (comprese ON COM, ON ERROR, ON PEN, ON PLAY, ON STRIG, e ON TIMER).

E' possibile che non avvenga l'intercettazione di un tasto se ne vengono premuti altri prima di quello specificato. Il tasto che ha provocato l'intercettazione non può essere provato con INPUT$ o INKEY$, di conseguenza il programma di intercettazione di ogni tasto deve essere diverso se si desidera una funzione diversa.

E' possibile utilizzare RETURN *riga* se si desidera ritornare al programma BASIC ad un determinato numero di riga. Tuttavia, questo ritorno non locale deve essere usato con attenzione, poiché le altre specifiche GOSUB, WHILE o FOR, attive al momento dell'intercettazione, rimangono tali.

Le iterazioni attive possono terminare o, impostando la variabile del contatore dell'iterazione fuori dai limiti o inserendo un comando di condizione nell'iterazione stessa. Questo assicura che ogni iterazione di una FOR abbia la corrispondente NEXT ed ogni iterazione di WHILE abbia la corrispondente WEND.

La KEY (*n*) ON non ha effetto se il valore del tasto funzionale viene visualizzato nella parte inferiore dello schermo.

Speciali considerazioni se si usa il minidisco DOS in lingua nazionale.

I programmi di tastiera con il DOS in lingua nazionale hanno la possibilità di passare in ogni momento dalla tastiera nazionale alla tastiera USA.

Usare i tasti F1 oppure F2 tenendo premuto Altrn e Ctrl per ottenere la permutazione (vedere il manuale DOS per maggiori informazioni sui programmi di tastiera). Se il programma BASIC intercetta uno di questi tasti, non passa le informazioni al programma di tastiera del DOS e la permutazione della tastiera non avviene. Se il programma BASIC necessita di permutare la tastiera, deve evitare di intercettare i tasti F1 e F2.

> **Nota:** quando s'intercetta uno di questi tasti, i programmi di tastiera del DOS non tengono conto dello stato del tasto maiuscolo. L'intercettazione dei tasti F1 e F2 sia in maiuscolo che in minuscolo, evita che un semplice sfioramento del tasto passi il controllo al programma DOS.

ON KEY(n) (Specifica)

Esempio: Questo è un esempio di intercettazione per il tasto funzionale 5.

```
100 ON KEY(5) GOSUB 200
110 KEY(5) ON
   .
   .
   .
200    'premuto il tasto funzionale 5
   .
   .
   .
290 RETURN 140
```

Questo esempio intercetta Ctrl-Interr e Ctrl-Altrn-Canc. Per BASIC versione 2.0 e successivi livelli.

```
10 KEY 15,CHR$(&H04)+CHR$(70) 'Trap Ctrl-Interr
20 KEY 16,CHR$(&H04+&H08)+CHR$(83)
       'Trap Ctrl-Altrn-Canc
30 ON KEY(15) GOSUB 1000
40 ON KEY(16) GOSUB 2000
50 KEY(15) ON: KEY(16) ON
   .
   .
   .
1000 PRINT "Trapping for Ctrl-Interr"
1010 RETURN
2000 TRAPS=TRAPS+1
2010 ON TRAPS GOTO 2100,2200,2300,2400, 2500
2020 '
2100 PRINT  "First trap of System Reset":RETURN
2200 PRINT "Second trap of System Reset":RETURN
2300 PRINT "Third trap of System Reset":RETURN
2400 PRINT "Fourth trap of System Reset":RETURN
2500 KEY(16) OFF 'Disable trap of System Reset
```

ON PEN (Specifica)

Scopo: Definire un numero di riga in modo che il BASIC vi trasferisca il controllo quando si attiva la penna luminosa.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
		***	(**)

Formato: ON PEN GOSUB *riga*

Note:

riga è il numero della prima riga del programma di intercettazione della penna luminosa. Usando un numero di riga zero si disattiva l'intercettazione della penna luminosa.

Per attivare questa specifica si deve eseguire una PEN ON. Dopo PEN ON, se si indica un numero di riga diverso da zero nella specifica ON PEN, ogni volta che il programma avvia una nuova specifica il BASIC controlla se è stata attivata la penna luminosa. In caso affermativo, il BASIC esegue una GOSUB *riga*.

Se si esegue PEN OFF, non avviene alcuna intercettazione. Anche se si attiva la penna, l'azione non viene registrata.

Se si esegue una specifica PEN STOP, non avviene alcuna intercettazione, ma viene registrata l'attività della penna, in modo da avere un'immediata intercettazione quando si esegue PEN ON.

Quando si verifica l'intercettazione, viene eseguita una PEN STOP automatica, in modo da impedire intercettazioni ripetitive. Il ritorno (RETURN) dal programma di intercettazione esegue automaticamente una PEN ON, a meno che nel programma di intercettazione non sia stata eseguita una PEN OFF.

ON PEN (Specifica)

L'intercettazione delle azioni non avviene se il BASIC non sta eseguendo un programma. Quando si verifica l'intercettazione di un errore (che risulta da una specifica ON ERROR) vengono automaticamente disabilitate tutte le intercettazioni (comprese ON COM, ON ERROR, ON PEN, ON PLAY, ON STRIG e ON TIMER.

PEN(Ø) non viene impostata quando l'attività della penna causa un'intercettazione.

E' possibile utilizzare RETURN *riga* se si desidera ritornare al programma BASIC ad un determinato numero di riga. Tuttavia usare attentamente questo ritorno non locale, poiché le altre specifiche GOSUB, WHILE o FOR, attive al momento dell'intercettazione, rimangono tali.

Le iterazioni attive possono terminare o, impostando la variabile del contatore delle iterazioni fuori dai limiti o inserendo una specifica di condizione nell'iterazione stessa. Questo assicura che ogni iterazione di una FOR abbia la corrispondente NEXT ed ogni iterazione di WHILE abbia la corrispondente WEND.

Nota: Non tentare nessuna I/E su cassetta mentre PEN è ON.

Esempio: Questo esempio imposta un programma di intercettazione per la penna luminosa.

```
1Ø ON PEN GOSUB 5ØØ
2Ø PEN ON
   .
   .
   .
5ØØ 'subroutine for pen
   .
   .
   .
65Ø RETURN 3Ø
```

ON PLAY(n) (Specifica)

Scopo: Permettere di suonare continuamente della musica di fondo durante l'esecuzione di un programma. Si applica solo al BASIC versione 2.Ø e successivi livelli.

Versioni: Cassetta Disco Avanzato Compilatore
***

Formato: ON PLAY*(n)* GOSUB *riga*

Note:

n è una espressione intera compresa fra 1 e 32 e indica le note che debbono essere intercettate. Per valori al di fuori di questa gamma verrà evidenziato il messaggio di errore **Illegal function call.**

riga è il numero di riga di inizio del programma di intercettazione delle note. Se il numero di riga è zero l'intercettazione si disattiva.

La specifica PLAY ON deve essere usata per attivare la specifica ON PLAY*(n)*. Dopo PLAY ON se c'è una specifica PLAY*(n)* con *riga* diversa da zero, il programma, ogni volta che inizia una nuova specifica BASIC, controlla se la memoria di transito per la musica passa dalla nota *n* a *n-1*. In questo caso il BASIC esegue un GOSUB alla riga specificata.

Usando PLAY OFF non avviene alcuna intercettazione. Anche se avviene una attività musicale, l'evento non è ricordato.

Se viene usata la specifica PLAY STOP non avviene alcuna intercettazione, ma l'attività musicale è ricordata così che avviene una immediata intercettazione dopo l'esecuzione di un PLAY ON.

ON PLAY(n) (Specifica)

Quando avviene una intercettazione si innesca automaticamente un PLAY STOP così che non possono avvenire nel frattempo altre intercettazioni. Il RETURN da un programma di intercettazione che non contempli un esplicito PLAY OFF, esegue automaticamente un PLAY ON.

L'intercettazione dell'evento non avviene durante il tempo in cui il BASIC non elabora un programma. Quando avviene una intercettazione di errore (risultato di una specifica ON ERROR) tutte le intercettazioni sono automaticamente disabilitate (incluse ON COM, ON ERROR, ON PEN, ON PLAY ON STRIG, e ON TIMER).

Si può usare RETURN *riga* per ritornare al programma BASIC ad una riga specificata. Prestare molta attenzione a questo ritorno non locale perché ogni altro GOSUB, WHILE o FOR che erano attivi al momento della intercettazione, rimangono tali.

Le iterazioni attive possono terminare o impostando la variabile del contatore delle iterazioni fuori dai limiti o inserendo una specifica di condizione nella iterazione stessa.

Note:

- L'intercettazione di una nota è permessa quando PLAY "MB..." (musica di sottofondo) è stato usato. L'intercettazione non è permessa quando PLAY "MF.." (musica in primo piano) è stato usato.

- L'intercettazione di una nota non avviene se la memoria di transito della musica di sottofondo è già piena quando è eseguita una specifica PLAY ON.

- Fare attenzione nello scegliere il valore n. Per esempio: ON PLAY (32) causa un tale numero di intercettazioni da lasciare poco tempo per elaborare il resto del programma.

ON PLAY(n) (Specifica)

Per ulteriori informazioni fare riferimento in questo capitolo alla funzione PLAY*(n)*.

Esempio: questo esempio prepara un programma di intercettazione che è chiamato quando cinque note sono lasciate nella memoria di transito della musica di sottofondo.

```
10 ON PLAY(5) GOSUB 500
20 PLAY ON
   .
   .
   .
500    'subroutine for background music
   .
   .
   .
650 RETURN 30
```

ON STRIG(n) (Specifica)

Scopo: Definire un numero di riga per l'intercettazione da parte del BASIC quando si preme uno dei tasti della barra di comando per i giochi.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
		***	(**)

Formato: ON STRIG*(n)* GOSUB *riga*

Note: *n* può essere 0, 2, 4, 6 ed indica il tasto da intercettare nel modo seguente:

0 tasto A1

2 tasto B1

4 tasto A2

6 tasto B2

riga è il numero della prima riga del programma di intercettazione per STRIG. Se *riga* è zero, si disattiva l'intercettazione del tasto della barra di comando per i giochi.

Per attivare questa specifica per il tasto *n* si deve eseguire una STRIG*(n)* ON. Se si esegue STRIG*(n)* ON e si indica un numero di riga diverso da zero nella specifica ON STRIG*(n)*, ogni volta che il programma avvia una nuova specifica, il BASIC controlla se è stato premuto il tasto specificato. In caso affermativo, il BASIC esegue una GOSUB alla *riga* specificata.

Se si esegue STRIG*(n)* OFF, non avviene alcuna intercettazione del tasto *n*. Anche se si preme il tasto, l'azione non viene registrata.

ON STRIG(n) (Specifica)

Se si esegue una specifica STRIG*(n)* STOP, non avviene alcuna intercettazione del tasto *n,* ma l'azione viene registrata in modo da avere un'immediata intercettazione quando si esegue STRIG*(n)* ON.

Quando si verifica l'intercettazione, automaticamente viene eseguita una STRIG *(n)* STOP, in modo da impedire intercettazioni ripetitive. RETURN dal programma di intercettazione esegue automaticamente una STRIG*(n)* ON, a meno che nel programma di intercettazione non sia stata eseguita una STRIG*(n)* OFF.

L'intercettazione delle azioni non avviene se il BASIC non sta eseguendo un programma. Quando si verifica l'intercettazione di un errore (che risulta da una specifica ON ERROR) vengono automaticamente disabilitate tutte le intercettazioni (comprese ON COM, ON ERROR, ON PEN, ON PLAY, ON STRIG, e ON TIMER). Con STRIG*(n)* ON si attiva il programma di interruzione che controlla lo stato del tasto specificato sulla barra di comando dei giochi. La pressione del tasto che causa l'intercettazione non imposta le funzioni STRIG(Ø), STRIG(2), STRIG(4) o STRIG(6).

E' possibile utilizzare RETURN *riga* se si desidera ritornare al programma BASIC ad un determinato numero di riga. Tuttavia, usare attentamente questo ritorno non locale, poiché le altre specifiche GOSUB, WHILE o FOR, attive al momento dell'intercettazione, rimangono tali.

Per uscire da iterazioni attive impostare il contatore delle iterazioni su valori fuori da quelli massimi oppure inserire nelle iterazioni stesse delle specifiche di condizione. Questo assicura che ogni iterazione di FOR abbia la corrispondente NEXT ed ogni iterazione di WHILE la corrispondente WEND.

ON STRIG(n) (Specifica)

Esempio: Questo è un esempio di programma di intercettazione del tasto sulla barra di comando dei giochi.

```
1Ø ON STRIG(Ø) GOSUB 2ØØØ
2Ø STRIG(Ø) ON
   .
   .
   .
5ØØ REM subroutine for 1st button
   .
   .
   .
65Ø RETURN
```

ON TIMER (Specifica)

Scopo: Trasferire il controllo ad una data riga in un programma BASIC quando è trascorso un definito periodo di tempo. (Per BASIC 2.0 e successivi livelli).

Versioni: Cassetta Disco Avanzato Compilatore
***

Formato: ON TIMER*(n)* GOSUB *riga*

Note:

n è una espressione numerica compresa fra 1 e 86.400 (da 1 secondo e 24 ore). Valori immessi al di fuori di questo intervallo provocano il seguente errore: **Illegal function call.**

riga è il numero di riga dell'inizio del programma di intercettazione per TIMER. Un numero di riga zero non fa eseguire alcuna intercettazione.

Una specifica TIMER ON deve essere usata per iniziare la specifica ON TIMER. Dopo TIMER ON, il BASIC, elaborando una specifica ON TIMER *(n)* con numero di riga diverso da zero, tiene traccia dei secondi che passano. Quando sono trascorsi *n* secondi, il BASIC esegue un GOSUB alla riga specificata. L'evento viene intercettato ed il BASIC riparte contando da zero.

Usando TIMER OFF non avviene alcuna intercettazione. Anche se vi è stata un'azione TIMER questa non viene registrata.

Usando TIMER STOP non avviene alcuna intercettazione, ma l'attività del TIMER è registrata in modo da avere un'immediata intercettazione quando si esegue una TIMER ON.

ON TIMER (Specifica)

Quando avviene l'intercettazione, viene eseguito un TIMER STOP automatico così che altre intercettazioni non possono avvenire. Un RETURN dal programma di intercettazione esegue automaticamente un TIMER ON a meno che un TIMER OFF non sia presente nel programma di intercettazione.

L'intercettazione dell'evento non avviene nel tempo in cui il BASIC non esegue un programma. Quando avviene l'intercettazione di un errore (dopo una specifica ON ERROR) tutte le altre intercettazioni sono disabilitate (incluso ON COM, ON ERROR, ON PEN, ON PLAY, ON STRIG, e ON TIMER).

Si può usare RETURN *riga* se si vuole tornare al programma BASIC ad una riga specificata. Bisogna usare molta attenzione per questo ritorno non locale, perché ogni altro GOSUB, WHILE o FOR attivi al momento della intercettazione, rimane tale.

Per uscire da iterazioni attive impostare il contatore della iterazione su valori fuori dai limiti, oppure inserire delle specifiche di condizione all'interno della iterazione. Questo assicura che ogni iterazione di FOR abbia la corrispondente NEXT ed ogni iterazione WHILE la corrispondente WEND.

ON TIMER (Specifica)

Esempio: ON TIMER è utile nei programmi che hanno bisogno di un contatore d'intervalli. Per esempio, per vedere l'ora del giorno sulla riga 1 ogni minuto:

```
10 CLS
20 ON TIMER(60) GOSUB 10000
30 TIMER ON
   .
   .
   .
10000 OLDROW=CSPLIN'save current row
10010 OLDCOL=POS(0)'save current column
10020 LOCATE 1,1:PRINT TIME$;
10030 LOCATE OLDROW,OLDCOL 'restore row & col
10040 RETURN
```

OPEN (Specifica)

Scopo: Consentire l'I/E in un file o unità.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: prima forma:

OPEN *specfile* [FOR *modo*] AS [#] *numfile* [LEN= *lrec*]

forma alternativa:

OPEN *modo2*, [#] *numfile, specfile* [,*lrec*]

Note:

modo nella prima forma corrisponde a:

OUTPUT specifica il modo di emissione sequenziale

INPUT specifica il modo di immissione sequenziale.

APPEND specifica il modo di emissione sequenziale in cui il file è posizionato alla fine dei dati quando viene aperto.

Notare che *modo* deve essere una costante a stringa, *non* racchiusa tra apici. Se si omette *modo,* si assume l'accesso casuale.

modo2 nella forma alternativa è un'espressione a stringa, il primo carattere della quale può essere:

OPEN (Specifica)

O specifica il modo di emissione sequenziale.

I specifica il modo di immissione sequenziale

R specifica il modo di immissione/emissione casuale.

Per entrambi i formati:

numfile è un numero intero il cui valore è compreso tra uno e il massimo consentito di file. Nel BASIC su cassetta il massimo è 4. Nel BASIC su disco e nel BASIC avanzato il massimo assunto è 3, ma può essere modificato con l'opzione /F: nel comando BASIC.

specfile è un'espressione a stringa per la specifica del file, e deve essere conforme alle regole descritte al capitolo 3 "Denominazione dei file" del manuale BASIC.

In BASIC 2.Ø e successivi livelli può contenere un percorso.

lrec è un'espressione intera che, se inclusa imposta la lunghezza record dei file ad accesso casuale. Può essere compresa tra 1 e 32767. La lunghezza assunta per difetto è di 128 byte. Il valore definito dell'opzione LREC, non può superare il valore di /S: nella linea di comando del BASIC.

OPEN alloca una memoria di transito per l'I/E di un file o di un'unità e determina il modo di accesso che sarà utilizzato con la memoria di transito.

numfile è il numero associato al file da quando questo viene aperto e utilizzato da altre specifiche di I/E per far riferimento al file o unità.

OPEN (Specifica)

Una OPEN deve essere eseguita prima di qualsiasi I/E su un'unità o file con una delle seguenti specifiche, o una specifica o funzione che richiede un numero file:

PRINT #
PRINT #USING
WRITE #
INPUT$
IOCTL #
INPUT #
LINE INPUT #
GET #
PUT #

GET e PUT sono valide per i file ad accesso casuale (o file di comunicazione). Un file su disco può essere ad accesso casuale o sequenziale ed uno su stampatrice può essere aperto in modo casuale o sequenziale; tuttavia, tutte le altre unità possono essere aperte solo per operazioni sequenziali.

Normalmente il BASIC aggiunge un avanzamento di riga dopo ogni ritorno a margine (CHR$ (13)) inviato alla stampatrice. Tuttavia se si apre una stampatrice come un file ad accesso casuale (LPT1:, LPT2: o LPT3:) con ampiezza 255, questo avanzamento di riga viene soppresso.

APPEND è valido solo per file su disco. Il puntatore al file è inizialmente impostato alla fine del file ed il numero record è impostato all'ultimo record del file. PRINT # o WRITE # estendono il file.

Tuttavia non è possibile aprire un file per emissione sequenziale o aggiunta se questo è già aperto.

Se si omette il nome dell'unità quando si usa il BASIC su cassetta, si assume CAS1:. Se si usa il BASIC su disco o BASIC avanzato, si assume l'unità del DOS.

Se come unità si specifica CAS1: e si omette il nome del file, viene aperto il successivo file di dati sulla cassetta.

Nel BASIC su cassetta, possono essere aperti contemporaneamente al massimo quattro file (cassetta, stampatrice, tastiera e schermo). Notare che i file su cassetta possono essere aperti solo uno alla volta. Per il BASIC su disco o avanzato il massimo assunto per difetto è tre file. Si può sostituire questo valore con l'opzione **/F:** nel comando BASIC.

Se un file aperto per l'immissione non esiste, si verifica un errore **(File not found).** Se si apre per l'emissione, l'aggiornamento o l'accesso casuale un file inesistente, viene creato un nuovo file.

Qualsiasi valore fornito, che non sia compreso nei limiti indicati produce un errore ed il file non viene aperto.

Per una descrizione completa sull'uso delle unità minidisco e relativi file fare riferimento al capitolo 3 e all'Appendice A del manuale BASIC. Per informazioni sull'apertura dei file di comunicazioni, vedere la specifica OPEN "COM...

Esempio: Una di queste specifiche apre il file denominato "DATA" per l'emissione sequenziale sull'unità assunta per difetto (CAS1: per il BASIC su cassetta, l'unità assunta per difetto per il BASIC su disco o BASIC avanzato). Notare che l'apertura per l'emissione distrugge i dati esistenti nel file. Se non si vogliono distruggere i dati, si deve aprire il file con APPEND.

```
10 OPEN "DATA" FOR OUTPUT AS #1
  or
10 OPEN "O",#1,"DATA"
```

OPEN
(Specifica)

Le due specifiche precedenti aprono il file denominato "SSFILE" sul minidisco nel modulo B per l'immissione ed emissione casuale. La lunghezza del record è 256.

```
10 OPEN "B:SSFILE" AS 1 LEN=256
  or
10 OPEN "R",1,"B:SSFILE",256
```

Questo esempio apre il file "DATA.ART" sul minidisco nell'unità A e posiziona i puntatori del file in modo che qualsiasi emissione sul file venga posta alla fine dei dati esistenti nel file.

```
10 FILE$ = "A:DATA.ART"
20 OPEN FILE$ FOR APPEND AS 3
```

La riga 10 di questo esempio apre la stampatrice in modo casuale. Poiché l'ampiezza per difetto è 80, le righe stampate dalle righe 20 e 30 terminano con un ritorno a margine/avanzamento riga. La riga 40 modifica l'ampiezza della stampatrice in 255, quindi l'avanzamento di riga dopo il ritorno a margine viene soppresso. La riga stampata dalla riga 50 termina solo con un ritorno a margine e non con un avanzamento di riga. Questo causa una sovrapposizione della riga stampata dalla riga 70, producendone la sottolineatura. La riga 60 riporta l'ampiezza a 80 in modo che la sottolineatura e le righe successive terminino con un avanzamento di riga.

```
10 OPEN "LPT1:" AS #1' accesso casuale
20 PRINT #1,"Stampa con ampiezza 80"
30 PRINT #1,"Cambia l'ampiezza in 255"
40 WIDTH #1,255
50 PRINT #1,"Questa riga sarà sottolineata"
60 WIDTH #1,80
70 PRINT #1, STRING$(28,"_")
80 PRINT #1,"Stampa con ampiezza 80 e CR/LF"
RUN
```

Sulla stampatrice sarà stampato:

```
Stampa con ampiezza 80
Cambia l'ampiezza in 255
Questa riga sarà sottolineata
Stampa con ampiezza 80 e CR/LF
```

I seguenti esempi illustrano l'uso dei percorsi per specfile.

Tutte e due queste specifiche aprono un file chiamato "DATA" per una emissione sequenziale nell'unità assunta per difetto nell'indirizzario chiamato LVL2.

```
10 OPEN "LVL1\LVL2\DATA" FOR OUTPUT AS #1
```

```
10 OPEN "O",#1,"LVL1\LVL2\DATA"
```

Ambedue le specifiche seguenti aprono un file chiamato "RRFILE" nell'indirizzario LVL1 su un minidisco inserito nell'unità B per I/E casuale. La lunghezza dell'informazione è 256.

```
20 OPEN "B:LVL1\RRFILE" AS 1 LEN=256
```

```
20 OPEN "R" ,1,"B:LVL1\RRFILE:" ,256
```

OPEN "COM... (Specifica)

Scopo: Aprire un file per le comunicazioni.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	(**)

Valida solo con l'Adattatore per le comunicazioni asincrone.

Formato: OPEN "COM*n:* [*velocità*] [,*parità*] [,*dati*] [,*arresto*] [,RS] [,CS[*n*]] [,DS[*n*]] [,CD[*n*]] [,LF] [,PE]" AS [#] *numfile* [LEN = *numero*]

Note:

n è 1 o 2 ed indica il numero dell'Adattatore per le comunicazioni asincrone.

velocità è una costante intera che specifica la velocità di trasmissione/ricezione in bit per secondo (bps). Velocità valide sono 75, 110, 150, 300, 600, 1200, 1800, 2400, 4800 e 9600. Velocità assunta in mancanza di informazioni è 300 bps.

parità è una costante di un carattere che specifica la parità per la trasmissione e ricezione nel modo seguente:

S SPACE: bit di parità sempre trasmessi e ricevuti come spazio (bit 0).

O ODD: controllo di parità di trasmissione o ricezione.

M MARK: bit di parità sempre trasmesso e ricevuto come segno (bit 1).

OPEN "COM... (Specifica)

E EVEN: controllo di parità di trasmissione e ricezione.

N NONE: nessun controllo di parità di trasmissione e ricezione.

Il valore assunto è EVEN (E).

dati è una costante intera che indica il numero di bit di dati per la trasmissione/ricezione. I valori validi sono: 5, 6, 7 o 8. Il valore assunto per difetto è 7.

arresto è una costante intera che indica il numero di bit di arresto. I valori validi sono 1 o 2. Il valore assunto sono due bit di arresto per 75 e 110 bps, un bit per tutti gli altri. Se si usa 5 per *dati* un 2 qui significa un bit e mezzo di arresto.

numfile è un'espressione intera che valuta un numero di file valido. Il numero è associato al file finché questo è aperto e utilizzato da altre specifiche di I/E per le comunicazioni.

numero è il numero massimo di byte che possono essere letti dalla memoria di transito di comunicazione quando si utilizza GET o PUT. Il valore assunto è 128 byte.

OPEN "COM... alloca una memoria di transito per l'I/E nello stesso modo di OPEN per i file su minidisco. Supporta la comunicazione asincrona RS232 con altri elaboratori e unità periferiche.

Un'unità per le comunicazioni può essere aperta solo ad un numero di file alla volta.

OPEN "COM... (Specifica)

Le opzioni **RS, CS, DS, CD, LF** e **PE** interessano i segnali di linea nel modo seguente:

RS sopprime RTS (richiesta per invio).

CS[*n*] controlla CTS (pronto per invio).

DS[*n*] controlla DSR (gruppo dati pronto).

CD[*n*] controlla CD (rilevazione di frequenza).

LF invia un avanzamento di riga dopo ogni ritorno a margine.

PE abilita controllo di parità.

CD (rilevazione di frequenza) è conosciuto anche come RLSD (rilevazione di segnale di linea ricevuto).

> **Nota:** i parametri *velocità, parità, dati* ed *arresto* sono posizionali, a differenza di **RS, CS, DS, CD, LF** e **PE.**

La richiesta per invio (RTS) è attivata quando si esegue una specifica OPEN "COM..., a meno che non si includa l'opzione **RS.**

L'argomento *n* nelle opzioni **CS, DS** e **CD** specifica il numero di millisecondi di attesa del segnale prima di ritornare un errore di **"Device timeout"** *n* può variare tra **0** e 65535. Se *n* viene omesso o se è uguale a zero, lo stato della linea non viene controllato.

I valori assunti sono **CS1000, DS1000 e CD0.** Se si specifica **RS**, il valore assunto è **CS0**.

OPEN "COM... (Specifica)

Normalmente le specifiche di I/E su un file di comunicazione non hanno esito se i segnali CTS (pronto per invio) e DSR (gruppo dati pronto) sono disattivati. Il sistema attende un secondo prima di ritornare un errore **Device timeout.** Le opzioni **CS e DS** consentono di ignorare queste linee e di specificare l'attesa prima dello scadere del tempo impostato.

Normalmente si ignora la rilevazione di frequenza (CD o RLSD) quando si esegue una specifica OPEN "COM.. L'opzione **CD** consente di provare questa linea includendo il parametro *n* allo stesso modo di **CS e DS.** Se si omette *n* o se è uguale a zero, la rilevazione di frequenza non viene controllata (corrisponde all'omissione dell'opzione **CD).**

Il parametro **LF** è rivolto ai file di comunicazione che lo utilizzano come mezzo di stampa su una stampatrice seriale. Quando si indica **LF,** dopo ogni carattere di ritorno a margine (esadecimale ØC) viene inviato automaticamente un carattere di avanzamento riga (esadecimale ØA). (Comprende il ritorno a margine inviato come risultato dell'impostazione dell'ampiezza). Notare che INPUT # e LINE INPUT #, utilizzate per la lettura di un file di comunicazione aperto con l'opzione **LF,** si arrestano quando rilevano un carattere di ritorno a margine. L'avanzamento di riga viene sempre ignorato.

La scelta **PE** permette il controllo di parità. Per difetto non c'è controllo di parità. La scelta **PE** causa un errore, **Device I/O error,** quando c'è un errore di parità e mette in ON il bit di ordine più alto quando ci sono 7 bit di dati o meno. La scelta **PE** non ha effetto per errori di stringa o di ricoprimento. Questi errori mettono sempre in ON il bit di ordine più alto e causano un errore **Device I/O error.**

Qualsiasi errore di codifica nell'espressione a stringa che inizia con **velocità** produce un errore

OPEN "COM... (Specifica)

(Bad file name). Non viene fornita alcuna indicazione sul parametro errato.

Fare riferimento all'Appendice C "Comunicazioni" per maggiori informazioni sul controllo dei segnali di emissione ed altre informazioni tecniche sul supporto per le comunicazioni.

Se si specificano 8 bit di dati, è necessario indicare parità N. Il BASIC usa tutti gli 8 bit in un byte per memorizzare i numeri; di conseguenza se si stanno trasmettendo o ricevendo dati numerici (ad esempio, usando PUT), è necessario specificare 8 bit di dati (non è necessario se si stanno inviando dati numerici *come testo)*.

Fare riferimento alla Specifica "OPEN" se si desiderano aprire delle unità che non siano di comunicazione.

Esempio: In questo esempio il file 1 viene aperto per la comunicazione con tutti i valori assunti. La velocità è di 300 bps con parità. Vi sono 7 bit di dati ed uno di arresto.

```
10 OPEN "COM1:" AS #1
```

In questo esempio il file 2 viene aperto per l'I/E asincrona a 1200 bps; non viene prodotta o controllata nessuna parità; byte di 8 bit vengono inviati e ricevuti e sarà trasmesso un bit di arresto (1).

```
10 OPEN "COM2:1200,N,8" AS #1
```

Questo esempio apre COM1: a 9600 bps senza parità e 8 bit di dati. CTS, DSR e CD non vengono verificati.

```
10 OPEN "COM1:9600,N,8,,CS,DS,CD" AS #1
```

Questo esempio apre COM1: a 1200 bps con parità assunta 7 bit di dati ed uno di arresto. RTS viene inviato, CTS non è controllato e viene fornito il messaggio **Device Timeout** se DSR non compare entro due secondi.

Notare che le virgole sono indispensabili per indicare la posizione dei parametri *parità, avvio e arresto,* anche se non è specificato un valore. Ciò è quanto si intende per parametro posizionale.

E' possibile utilizzare una specifica OPEN con una ON ERROR per assicurarsi che il modem funzioni correttamente prima di inviare dati. Ad esempio, il seguente programma si assicura di avere la rilevazione di frequenza (CD o RLSD) dal modem prima di iniziare. La riga 20 è impostata a supero tempo dopo 10 secondi. TRIES è impostato a 6 in modo da rinunciare se non si incontra la rilevazione di frequenza entro un minuto. Una volta stabilita la comunicazione, si riapre il file con un breve ritardo fino allo scadere del tempo impostato.

```
10 TRIES=6:ON ERROR GOTO 100
20 OPEN "COM1:300,N,8,2,CS,DS,CD10000" AS#1
30 ON ERROR GOTO 0
40 CLOSE#1 ' Il lavoro può così continuare
50 GOTO 1000
   .
   .
   .
100 TRIES=TRIES-1
110 IF TRIES=0 THEN ON ERROR GOTO 0  'give up
120 RESUME
   .
   .
   .
1000 OPEN "COM1:300,N,8,2,CS,DS,CD2000" AS#1
```

OPEN "COM... (Specifica)

L'esempio successivo mostra un tipico esempio di utilizzo di un file di comunicazione per controllare una stampatrice seriale. Il parametro LF nella specifica OPEN assicura che le righe non vengano stampate una sull'altra.

```
10 WIDTH "COM1:", 132
20 OPEN "COM1:1200,N,8,,CS1000,DS1000,
     CD1000,LF" AS#1
```

OPTION BASE (Specifica)

Scopo: Dichiarare il valore minimo per subscritti di schiera.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: OPTION BASE *n*

Note: *n* è 1 oppure Ø.

La base assunta per difetto è Ø. Se si esegue la specifica:

```
OPTION BASE 1
```

il valore minimo di un elemento di schiera è 1.

La specifica OPTION BASE deve essere codificata prima di definire o usare una schiera. Se si cambia il valore di base quando una schiera esiste, si provoca un errore.

Nel BASIC 1.1Ø il programma con cui ci si vuole concatenare non può avere una OPTION BASE anche quando ambedue i programmi hanno il medesimo valore base.

OUT (Specifica)

Scopo: Inviare un byte ad una porta di uscita dell'elaboratore.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: OUT *n,m*

Note:

n è un'espressione numerica per il numero della porta compresa tra Ø e 65535.

m è un'espressione numerica per i dati da trasmettere, compresa tra Ø e 255.

Per una descrizione dei numeri delle porte valide (indirizzi I/E) vedere il manuale di riferimento dell'elaboratore.

La specifica OUT è complementare alla funzione INP.

Un uso di OUT è rivolto all'emissione su video. Su alcune unità collegate all'Adattatore per il video grafico a colori, i primi due o tre caratteri della riga non appaiono sullo schermo. Se l'unità video non possiede il controllo per la regolazione orizzontale, si possono utilizzare le seguenti specifiche per spostare la visualizzazione.

```
OUT 980,2: OUT 981,43
```

La visualizzazione si sposta di due caratteri verso destra con l'ampiezza di 40 colonne (o 16 punti in modo grafici a media risoluzione, oppure 32 punti ad alta risoluzione).

```
OUT 980,2: OUT 981,85
```

La visualizzazione si sposta a destra di cinque caratteri con l'ampiezza di 80 colonne.

Lo spostamento causato da queste specifiche OUT rimane valido finché non si esegue una specifica WIDTH o SCREEN. Si può utilizzare anche il comando MODE del DOS per spostare la visualizzazione; ha il vantaggio di rimanere valido fino ad un Ripristino del sistema.

Esempio: Questo invia il valore 100 alla porta 32.

```
100 OUT 32,100
```

PAINT
(Specifica)

Scopo: Riempire un'area sullo schermo con il colore scelto.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
		***	(**)

Solo modo grafici.

Formato: PAINT *(x,y)* [[,*colore*] [,*limite*] [,*sfondo*]].

Note:

(x,y) sono le coordinate di un punto nell'area da riempire. Le coordinate possono essere indicate in forma relativa o assoluta (vedi Capitolo 3 del manuale BASIC). Questo punto sarà utilizzato come punto di partenza. Punti che sono al di fuori dello schermo non vengono disegnati e non viene segnalato nessun errore.

colore può essere un numero o una espressione numerica. E' usato per colorare o disegnare un'area o i suoi contorni. Quando *colore* è un'espressione numerica sceglie un attributo nella gamma dei colori del modo schermo in uso A media risoluzione, il colore è il colore in uso per quell'attributo definito dalla specifica COLOR. A media risoluzione sono disponibili 4 attributi (Ø-3). Ad alta risoluzione 2 (Ø-1). Zero (Ø) è sempre il colore per lo sfondo. L'attributo assunto per difetto per lo sfondo è il massimo attributo per il modo schermo in uso: 3 a media risoluzione: 1 ad alta risoluzione.

Se *colore* è un'espressione a stringa allora la sovrapposizione del colore viene eseguita come successivamente descritto. Cio è valido per il BASIC 2.0 e livelli seguenti.

limite è un'espressione numerica costituita da un numero intero della gamma di attributi del modo schermo in uso. Definisce il colore dei contorni della figura da colorare.

sfondo è un'espressione a stringa di 1 byte usata per la sovrapposizione del colore (BASIC 2.0 e successivi livelli).

Poiché ad alta risoluzione vi sono solo due colori, *colore* non sarà differente da *limite*. Per difetto, *limite* è uguale a *colore*. Si può riempire di nero un'area bianca e viceversa.

A media risoluzione si può riempire o contornare un'area usando uno dei quattro colori a disposizione della tavolozza in uso definita mediante la specifica COLOR. Ad esempio si può riempire di rosso un cerchio verde e viceversa.

colore comincia da uno specifico punto e riempie un'area fino a che non incontra l'attributo relativo al limite. Pertanto *colore* deve sempre cominciare nell'area da riempire. Se il punto di partenza specificato è lo stesso attributo di *limite,* allora il riempimento si ferma a quel punto e non appare nient'altro. Al riguardo si può per esempio, tracciare mediante PSET un punto con lo stesso attributo di *limite* quindi usare le coordinate di quel punto mediante la specifica PAINT.

PAINT
(Specifica)

La specifica PAINT riempie ogni area senza riguardo alla forma della figura, tuttavia, più è complesso l'interno della figura, maggiore sarà lo spazio richiesto dal BASIC. In questi casi si può usare la specifica CLEAR all'inizio del programma al fine di aumentare lo spazio disponibile.

PAINT permette con poche specifiche la visualizzazione di qualsiasi figura.

Nell'esempio che segue, la specifica PAINT a riga 30 riempie il quadrato disegnato a riga 20 con il colore rappresentato dall'attributo della tavolozza in uso.

```
10 SCREEN 1
20 LINE (0,0)-(100,150),2,B
30 PAINT (50,50),1,2
```

I seguenti argomenti riguardano solo il riempimento col colore (BASIC 2.0 e successivi livelli). A tale scopo l'attributo *colore* deve essere una espressione a stringa nella forma:

```
CHR$(&Hnn)+CHR$(&Hnn)+CHR$(&Hnn)
```

La sequenza di CHR$ specifica una maschera di bit ampia 1 byte. Quando la maschera traccia in lungo e verso il basso l'area stabilita dal *limite,* viene creato un modello più che un colore solido. I due numeri esadecimali delle espressioni CHR$ corrispondono a 8 bit o ad un byte.

L'espressione a stringa può contenere sino a 64 byte. L'esempio creato dalla espressione a stringa può essere rappresentato come segue:

	x aumenta → **76543210**	
0,0	xxxxxxxx	Tile byte 0
0,1	xxxxxxxx	Tile byte 1
0,2	xxxxxxxx	Tile byte 2
.		
.		
.		
0,63	xxxxxxxx	Tile byte 63 (maximum allowed)

La sequenza dei bit viene ripetuta uniformemente sull'area definita da *limite*. Se l'area non viene definita, si intende come area designata lo schermo intero. Ogni byte della stringa maschera 8 bit lungo l'asse x quando i punti vengono tracciati. Ogni byte della stringa viene ruotato come richiesto per allineare il modello lungo l'asse y. Il BASIC sceglie il particolare byte del modello da riempire, usando la formula **y mod tile length.**

PAINT
(Specifica)

Poiché in alta risoluzione c'è solo un bit per pixel, un punto è tracciato ad ogni posizione che nella maschera ha un valore di 1. Lo schermo nell'alta risoluzione può essere colorato con delle X usando il seguente esempio:

```
10 CLS:SCREEN 1: COLOR 1: KEY OFF
20 LOCATE 12,7:PRINT "I JUST LOVE MY IBM COMPUTER"
30 PAINT(320,100),CHR$(&H81)+CHR$(&H42)+
   CHR$(&H24)+CHR$(&H18)+CHR$(&H24)+
   CHR$(&H24)+CHR$(&H42)+CHR$(&H81)
40 GOTO 40
```

La lunghezza di questa maschera è 8 byte numerati da zero a sette. In questo caso PAINT alle coordinate (320, 100) inizia a tracciare il byte 4. Questo è calcolato usando la formula **y mod tile lenght** sostituendo 100 ad y ed 8 per tile lenght. Questa sequenza appare sullo schermo come segue:

	x aumenta →	
Tile byte 0	10000001	CHR$(&H81)
Tile byte 1	01000010	CHR$(&H42)
Tile byte 2	00100100	CHR$(&H24)
Tile byte 3	00011000	CHR$(&H18)
Tile byte 4	00011000	CHR$(&H18)
Tile byte 5	00100100	CHR$(&H24)
Tile byte 6	01000010	CHR$(&H42)
Tile byte 7	10000001	CHR$(&H81)

PAINT (Specifica)

Il metodo per disegnare sullo schermo dipende dal numero di attributi di colore disponibili su ciascun tipo di schermo. Questo perché il numero di bit per pixel è strettamente correlato al numero di attributi di colore su ciascun tipo di schermo. Per ciascuno schermo, dove X è il numero totale di attributi disponibili per quello schermo, vale la relazione:

$$LOG_2(X) = Y$$

dove Y è il numero di bit per pixel. In alta risoluzione, ciascun byte della stringa è in grado di tracciare 8 punti sullo schermo (1 bit per pixel) poiché $LOG_2(2) = 1$.

Nello schermo 1, un tile byte descrive 4 pixel, dal momento che la media risoluzione ha solo 2 bit per pixel: cioè, $LOG_2(4) = 2$ bit per pixel. Ogni 2 bit del tile byte descrive 1 dei 4 possibili attributi di colore associati con ognuno dei 4 pixel che devono essere tracciati.

Il seguente schema mostra i valori binari ed esadecimali associati ad ogni attributo in media risoluzione.

PAINT
(Specifica)

Colore Tavolozza 0	Valore del colore in binario	Sequenza da disegnare in valore binario	Sequenza da disegnare in valore esadecimale
verde	01	01010101	&H55
rosso	10	10101010	&HAA
marrone	11	11111111	&HFF

Colore Tavolozza 1	Valore del colore in binario	Sequenza da disegnare in valore binario	Sequenza da disegnare in valore esadecimale
azzurro	01	01010101	&H55
magenta	10	10101010	&HAA
bianco	11	11111111	&HFF

In media risoluzione, il seguente esempio traccia un modello di quadrati con un bordo color rosso con la tavolozza 0 e magenta con la tavolozza 1.

```
10 CLS: SCREEN 1: COLOR 1: KEY OFF
20 LOCATE 12,7:PRINT "I JUST LOVE MY IBM COMPUTER"
30 PAINT (320,100),CHR$(&HAA)+CHR$(&H82)+
   CHR$(&H82)+CHR$(&H82)+CHR$(&H82)+
   CHR$(&H82)+CHR$(&H82)+CHR$(&HAA)
40 GOTO 40
```

	76543210	
Tile byte 0	10101010	CHR$(&HAA)
Tile byte 1	10000010	CHR$(&H82)
Tile byte 2	10000010	CHR$(&H82)
Tile byte 3	10000010	CHR$(&H82)
Tile byte 4	10000010	CHR$(&H82)
Tile byte 5	10000010	CHR$(&H82)
Tile byte 6	10000010	CHR$(&H82)
Tile byte 7	10101010	CHR$(&HAA)

Occasionalmente, si può voler ricoprire un'area già dipinta con lo stesso colore o modello di due byte consecutivi della maschera tile. Normalmente questo costituisce una condizione limite in quanto il punto è circondato da punti dello stesso modello di bit. (segue un esempio sull'uso dello sfondo).

L'attributo di sfondo può essere usato per annullare questa condizione di termine. Non si possono specificare più di 2 linee consecutive nel modello tile che corrisponde a questo attributo di *sfondo* perché si verificherebbe un errore di **Illegal function call.**

PAINT
(Specifica)

Esempio: Il seguente programma dimostra come sovrapporre un'area con 3 linee di rosso, 2 linee di verde ed 1 linea di rosso. La tavolozza poi cambia per mostrare che la stessa maschera tile produrrà lo stesso disegno con colori diversi.

```
10 CLS:SCREEN 1,0:KEY OFF
20 TIL$=CHR$(&HAA)+CHR$(&HAA)+CHR$(&HAA)
   +CHR$(&H55)+CHR$(&H55)+CHR$(&HFF)
30 COLOR 0,0  'choose palette 0
40 VIEW (1,1)-(150,100),0,2
50 GOSUB 1000
60 COLOR 0,1  'choose palette 1
70 GOTO 1020
1000 PAINT (125,50),TIL$,2
1010 RETURN
1020 GOTO 1020
```

PAINT (Specifica)

Il seguente esempio usa la sovrapposizione del colore con l'attributo di sfondo.

```
10 CLS: SCREEN 1 COLOR 0,1:KEY OFF
20 TIL$=CHR$(&H5F)+CHR$(&H5F)+CHR$(&H27)+CHR$(&H81)
30 VIEW (1,1)-(150,100),0,2
40 LOCATE 3,22:PRINT "<---Without back-"
50 LOCATE 4,22:PRINT "    ground tile "
60 PAINT (125,50),CHR$(&H5F)
70 PAINT (125,50),TIL$,2
80 '
90 'with background tile
100 '
110 VIEW (160,100)-(310,198),0,2
120 LOCATE 16,1:PRINT "With background-->"
130 LOCATE 17,1:PRINT "tile CHR$(&H5F)"
140 PAINT (125,50),CHR$(&H5F)
150 PAINT (125,50),TIL$,2,CHR$(&H5F)
160 LINE (1,100)-(319,100),3
170 FOR I=1 TO 2500:NEXT I
180 GOTO 180
```

PEEK (Funzione)

Scopo: Fornire il byte letto dalla posizione di memoria indicata.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v* = PEEK(*n*)

Note: *n* è un intero compreso tra Ø e 65535. *n* è la posizione relativa all'attuale segmento definito dalla specifica DEF SEG (vedere la specifica DEF SEG).

Il valore fornito sarà un intero compreso tra Ø e 255.

PEEK è la funzione complementare della specifica POKE. (vedere la specifica POKE).

Esempio: Il seguente esempio inserito in un programma ha la funzione di verificare quale adattatore video si trovi nel sistema. Dopo aver eseguito la riga 3Ø, la variabile IBMMONO avrà il valore Ø se si usa l'Adattatore per il video grafico a colori, oppure 1 se si usa l'Adattore per l'unità video IBM monocromatica e stampatrice parallela.

```
1Ø 'test display adapter
2Ø DEF SEG=Ø
3Ø IF (PEEK(&H41Ø) AND &H3Ø)=&H3Ø
      THEN IBMMONO=1
      ELSE IBMMONO=Ø
```

PEN
(Specifica e funzione)

Scopo: Interpretare i caratteri letti dalla penna luminosa.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

PEN STOP solo nel BASIC avanzato e nel compilatore.

Formato: Come specifica:

PEN ON

PEN OFF

PEN STOP

Come funzione.

v = PEN*(n)*

Note: La funzione PEN, *v* = PEN*(n)*, legge le coordinate della penna luminosa.

n è un'espressione numerica compresa tra Ø e 9 ed influenza il valore fornito dalla funzione nel modo seguente:

- Ø — Un contrassegno che indica se la penna è abbassata dall'ultima scansione. In caso affermativo fornisce -1, altrimenti Ø.
- 1 — Fornisce la coordinata x del punto in cui la penna era stata attivata l'ultima volta. I limiti sono Ø-319 a media risoluzione e Ø-639 ad alta risoluzione.

PEN
(Specifica e funzione)

2 Fornisce la coordinata y del punto in cui la penna era stata attivata l'ultima volta. I limiti sono Ø-199.

3 Fornisce il valore attuale di commutazione della penna. -1 se disattivata, Ø in caso contrario.

4 Fornisce l'ultima coordinata x valida. I limiti sono Ø-319 a media risoluzione e Ø-639 ad alta risoluzione.

5 Fornisce l'ultima coordinata y valida. I limiti sono Ø-199.

6 Fornisce la posizione della riga dove la penna è stata attivata l'ultima volta. I limiti sono 1-24.

7 Fornisce la posizione della colonna dove la penna è stata attivata l'ultima volta. I limiti sono 1-4Ø o 1-8Ø, in funzione di WIDTH.

8 Fornisce la riga dell'ultimo carattere valido conosciuto. I limiti sono 1-24.

9 Fornisce la posizione della colonna dell'ultimo carattere valido (da 1 a 4Ø o da 1 a 8Ø in dipendenza di WIDTH).

PEN ON abilita la funzione di lettura PEN, che inizialmente non è attiva. Una specifica PEN ON deve essere eseguita prima di richiamare una funzione di lettura della penna. Il richiamo di una funzione PEN mentre questa non è attiva causa l'errore **Illegal function call.**

Per migliorare la velocità di esecuzione, è consigliabile disattivare la penna con una specifica PEN OFF se questa non viene utilizzata.

PEN
(Specifica e funzione)

Nel BASIC avanzato eseguendo PEN ON si consente l'intercettazione con la specifica ON PEN. Dopo PEN ON, se nella specifica ON PEN è stato indicato un numero diverso da zero, ogni volta che il programma avvia una nuova specifica, il BASIC controlla se la penna è stata attivata. Vedere la specifica ON PEN.

PEN OFF disabilita la funzione di lettura PEN. Nel BASIC avanzato non viene eseguita alcuna intercettazione della penna e l'azione della penna luminosa stessa non viene registrata anche se ha avuto luogo.

PEN STOP è disponibile solo nel BASIC avanzato e disabilità l'intercettazione dell'attività della penna luminosa. Tuttavia se una qualsiasi attività si è verificata, questa viene registrata, in modo che venga eseguita un'immediata intercettazione quando si esegue PEN ON.

Quando la penna è abbassata sul bordo dello schermo, i valori forniti sono imprecisi.

Non si deve tentare l'I/E su cassetta se PEN è ON.

Esempio: Questo esempio stampa il valore della penna dall'ultima scansione a quella attuale.

```
10 PEN ON
20 FOR I=1 TO 500
30 X=PEN(0): X1=PEN(3)
40 PRINT X, X1
50 NEXT
600 PEN OFF
```

PLAY
(Specifica)

Scopo: Suonare la musica specificata dalla *stringa*.

Versioni: Cassetta Disco Avanzato Compilatore
*** (**)

Formato: PLAY *stringa*

Note: PLAY sviluppa un concetto simile a DRAW inserendo in una stringa carattere un "linguaggio di definizione della musica".

stringa è un'espressione a stringa che consiste in comandi per la musica di un singolo o doppio carattere.

I comandi in PLAY sono:

Da A(LA) a G(SOL) con facoltativi #, +, o −.
Esegue la nota indicata nell'ottava. Un segno di numero (#) o un segno più (+) seguenti indicano un diesis, un segno meno (–) indica un bemolle. #, + o – non sono consentiti a meno che non corrispondano ai tasti neri del pianoforte. Ad esempio, B# non è una nota valida.

O n Ottava. Definisce l'ottava per le note successive. Ci sono 7 ottave numerate, da 0 a 6. Ogni ottava va da C(DO) a B(SI). La terza ottava inizia con il C(D0) centrale. La quarta, è l'ottava assunta per difetto.

> n Sale alla successiva ottava più alta e suona la nota n. Ogni volta che la nota n viene emessa, l'ottava sale ancora fino a raggiungere la 6ª ottava. Per esempio PLAY "> A" incrementa l'ottava ed emette la nota A. Ogni volta che PLAY "> A" è eseguita l'ottava sale fino a raggiungere la 6ª ottava, poi ogni volta che PLAY "> A" è eseguita emette la nota A come 6ª ottava (BASIC 2.Ø e successivi livelli).

< n scende di una ottava e suona la nota *n*. Ogni volta che la nota n è emessa, l'ottava scende fino a raggiungere l'ottava zero. Per esempio PLAY "< A" abbassa l'ottava ed emette la nota A. Ogni volta che viene eseguito PLAY "< A" l'ottava scende fino a raggiungere l'ottava zero dopo di che ogni volta che PLAY "< A" è eseguita emette la nota A come ottava Ø (BASIC 2.Ø e successivi livelli).

N n Esegue la nota n che può essere compresa tra Ø e 84. Nelle 7 ottave possibili ci sono 84 note. *n*=Ø significa pausa. Questo è un modo alternativo per scegliere le note oltre a quello di specificare l'ottava (O *n*) ed il nome della nota (A-G = LA-SOL).

L n Imposta la lunghezza delle note che seguono. La lunghezza reale della nota è 1/n, dove n può essere compreso tra 1 e 64.

Lunghezza Equivalente a

L1	intera (semibreve)
L2	metà (minima)
L3	una nota di una terzina di tre metà (1/3 di una misura a 4 tempi)
L4	quarto
L5	una nota di una quintina (1/5 di una misura)
L6	una nota di una terzina di un quarto
.	
.	
.	
L64	1/64 di nota

PLAY
(Specifica)

La lunghezza può anche seguire la nota se si desidera modificare solo la sua lunghezza. Per esempio A16 equivale a L16A.

Pn Pausa .*n* può essere compresa tra 1 e 64 e rappresenta la lunghezza della pausa nello stesso modo di L (lunghezza).

. Punto. Dopo una nota, causa l'esecuzione di una nota come nota puntata. Un punto aumenta la durata di una nota della sua stessa metà. Ogni nota può avere più di un punto. Ognuno di questi aumenta il valore totale della nota di 1/2 del valore del punto precedente. Ad esempio una mezza nota con 2 punti sarà equivalente in durata a 1/2 nota + 1/4 + 1/8 della nota stessa. I punti possono apparire anche dopo una pausa (P) per graduare allo stesso modo la lunghezza della pausa .

T n Tempo. Imposta il numero di quarti in un minuto. *n* può essere compreso tra 32 e 255. Il valore assunto per difetto è 120. Vedere la specifica SOUND per una lista di tempi comuni e gli equivalenti battiti al minuto.

MF Musica di primo piano (creata da SOUND o PLAY). Ogni nota o suono successivo non inizia finché non termina quello precedente. Per terminare PLAY è possibile premere Ctrl-Interr. La musica di primo piano è assunta per difetto.

MB Musica di sottofondo (creata da SOUND o PLAY). Ogni nota o suono viene posto in una memoria di transito consentendo al programma BASIC di continuare l'esecuzione mentre la musica suona in sottofondo. E' possibile suonare in sottofondo, contemporaneamente, fino a 32 note.

MN Musica normale. Ogni nota suona 7/8 del tempo specificato da L (lunghezza).
Questa è la definizione assunta per difetto di **MN ML e MS.**

ML Legato. Ogni nota suona l'intero periodo impostato da L (lunghezza).

MS Staccato. Ogni nota suona 3/4 del tempo specificato da L.

X variabile;
Esegue la stringa specificata.

In tutti questi comandi l'argomento *n* può essere una costante, come 12, oppure può essere =*variabile;* dove *variabile* è il nome di una variabile. Il punto e virgola (;) è indispensabile se si usa una variabile in questo modo e se si usa il comando X. Il punto e virgola è invece facoltativo fra comandi, ma non è consentito dopo MF, MB, MN, ML o MS. Gli spazi in bianco nella *stringa* sono ignorati.

E' possibile specificare delle variabili nella forma VARPTR$ (variabile), invece di=*variabile;* e questa è la sola forma da usare in programmi compilati.
Per esempio:

Metodo	**Metodo alternativo**
PLAY "XA$;"	PLAY "X"+VARPTR$(A$)
PLAY "O=I;"	PLAY "O="+VARPTR$(I)

PLAY
(Specifica)

Una **x** può essere utilizzata per memorizzare una melodia in una stringa e richiamarla più volte con tempi e ottave diversi da un'altra stringa.

Esempi: il seguente esempio esegue una melodia.

```
10 'little lamb
20 MARY$="GFE-FGGG"
30 PLAY "MB T100 O3 L8;XMARY$;P8 FFF4"
40 PLAY "GB-B-4; XMARY$; GFFGFE-...."
```

il seguente esempio esegue una scala musicale dall'ottava Ø all'ottava 6.

```
10 'Play the scale using > octave
20 SCALE$="CDEFGAB"
30 PLAY "O0 XSCALE$;"
40 FOR I=1 TO 6
50 PLAY ">XSCALE$;"
60 NEXT
70 'Play the scale using < octave
80 PLAY "O6 XSCALE$;"
90 FOR I=1 TO 6
100 PLAY "<XSCALE$;"
110 NEXT
```

PLAY(n) (Funzione)

Scopo: Fornire il numero delle note presenti nella memoria di transito della musica di sottofondo. (BASIC 2.0 e successivi livelli).

Versioni: Cassetta Disco Avanzato Compilatore
***

Formato: *v* = PLAY(*n*)

Note: *n* è un argomento fittizio che può avere qualsiasi valore.

PLAY(n) fornisce uno zero quando il programma esegue musica in primo piano. Il massimo valore che può essere fornito è 32 che è il massimo numero di note che la memoria di transito può contenere.

PLAY(n) fornisce le note della memoria di transito, solamente quando esegue musica di sottofondo (MB).

Esempio:

```
10 'when 5 notes are left in the background music buffer
20 'go to line 1000 and play another tune
30 PLAY "MB CDEFGAB"
40 IF PLAY (1)=5 GOTO 1000
50 GOTO 2000
 .
 .
 .
1000 PLAY "MB 04 T200 L4 MS GG#GE"
2000 END
```

PMAP (Funzione)

Scopo: Convertire una coordinata fisica in una coordinata spaziale e viceversa. (BASIC 2.0 e successivi livelli).

Versioni: Cassetta Disco Avanzato Compilatore
***

Solamente in modo grafici.

Formato: *v* = PMAP(*x*,*n*)

Note: *x* coordinata del punto che deve essere convertita

n può essere un valore nella gamma da zero a tre:

0 converte la coordinata spaziale x in coordinata fisica x

1 converte la coordinata spaziale y in coordinata fisica y

2 converte la coordinata fisica x in coordinata spaziale x

3 converte la coordinata fisica y in coordinata spaziale y

PMAP è usato per convertire delle coordinate dal sistema spaziale come definito dalla specifica WINDOW al sistema fisico.

PMAP (Funzione)

PMAP(x,Ø) e PMAP(x,1) sono usati per convertire coordinate dal sistema spaziale al sistema fisico.

PMAP(x,2) e PMAP(x,3) sono usati per convertire coordinate dal sistema fisico al sistema spaziale.

Per esempio, con la seguente specifica

```
SCREEN1: WINDOW (-1,-1)-(1,1)
```

si può usare PMAP per convertire le coordinate spaziali (–1, –1) e (1, 1) nei corrispondenti punti fisici dello schermo.

PMAP(–1,Ø) fornisce il valore zero come coordinata fisica x.

PMAP(–1,1) fornisce il valore 199 come coordinata fisica y

PMAP(1,Ø) fornisce il valore 319 come coordinata fisica x

PMAP(1,1) fornisce il valore zero come coordinata fisica y.

L'informazione precedente dice che il punto (–1, –1) nell'angolo più basso a sinistra dello schermo corrisponde al punto fisico (Ø,199); che il punto (1,1) nell'angolo più alto a destra corrisponde al punto fisico (319,Ø).

POINT (Funzione)

Scopo: Nella prima forma, fornire l'attributo del punto specificato nello schermo. Nella seconda fornire il valore della coordinata grafica x o y in uso.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Solo modo grafici

Formato: *v* = POINT*(x,y)*
v = POINT*(n)*

Note:

(x,y) sono le coordinate del punto da usare. Devono essere in forma assoluta (vedere il Capitolo 3 del manuale BASIC).

Se il punto indicato non è compreso nei limiti, viene fornito il valore –1. A risoluzione media le risposte valide sono 0,1, 2, 3; ad alta risoluzione 0 e 1.

n fornisce il valore dell'attuale coordinata grafica x o y. (In BASIC 2.0 e successivi livelli) *n* può avere un valore da 0 a 3:

0 fornisce l'attuale coordinata fisica *x*

1 fornisce l'attuale coordinata fisica *y*

2 fornisce l'attuale coordinata spaziale *x* se WINDOW è attivo. Se WINDOW non è attivo, fornisce l'attuale coordinata fisica x.

3 fornisce l'attuale coordinata spaziale *y* se WINDOW è attivo. Se WINDOW non è attivo, fornisce l'attuale coordinata fisica y.

Per maggiori informazioni vedere la specifica WINDOW.

Esempio: il seguente esempio inverte lo stato attuale del punto (I,I).

```
10 SCREEN 2
20 IF POINT(I,I)<>0 THEN PRESET(I,I)
                    ELSE PSET(I,I)
   oppure
20 PSET(I,I),1-POINT(I,I)
```

POINT (Funzione)

Il seguente esempio illustra i valori forniti dalla funzione POINT. Notare il cambiamento dei valori in relazione alla WINDOW.

```
10 CLS:SCREEN 1,0:KEY OFF
20 PRINT "POINT(n) con WINDOW inattiva"
30 GOSUB 110
40 WINDOW (0,0)-(319,199)
50 PRINT "POINT(n) con WINDOW attiva"
60 GOSUB 110
70 PRINT "POINT(n) con WINDOW e SCREEN attiva"
80 WINDOW SCREEN (0,0)-(319,199)
90 GOSUB 110
100 END
110 PSET (5,15)
120 FOR I=0 TO 3
130 PRINT POINT (I);
140 NEXT
150 PRINT:PRINT
160 RETURN
RUN

POINT(n) con WINDOW inattiva
5  15  5  15
POINT(n) con WINDOW attiva
5  184  5  15
POINT(n) con WINDOW e SCREEN attiva
5  15  5  15
```

POKE (Specifica)

Scopo: Scrivere un byte in una posizione della memoria.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: POKE *n,m*

Note:

n deve essere compreso tra Ø e 65535 ed indica l'indirizzo della posizione nel segmento in uso dove devono essere scritti i dati. Il segmento in uso è definito dalla specifica DEF SEG.

m sono i dati da scrivere nella posizione specificata. Deve essere compreso tra Ø e 255.

La funzione complementare di POKE è PEEK. POKE e PEEK sono utili per memorizzare dati e caricare sottoprogrammi in linguaggio macchina. Vedere anche la funzione PEEK.

Attenzione:
Il BASIC non controlla l'indirizzo. Perciò non eseguire POKE nei livelli, nelle aree di variabile e nel programma BASIC.

Esempio: Vedere gli esempi nell'Appendice B, "Sottoprogrammi in linguaggio macchina".

POS (Funzione)

Scopo: Fornire la colonna in cui si trova attualmente il cursore.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v* = POS(*n*)

Note: *n* è un argomento fittizio.

Viene fornita l'attuale posizione orizzontale del cursore (colonna). Il valore fornito è compreso tra 1 e 40 o 1 e 80, in base all'impostazione di WIDTH. Si può utilizzare CSRLIN per trovare la posizione verticale (riga) del cursore.

Vedere anche la funzione LPOS.

Esempio: Questo esempio genera un ritorno a margine (sposta il cursore all'inizio della riga successiva) se il cursore va oltre la posizione 60 sullo schermo.

```
IF POS(0)>60 THEN PRINT CHR$(13)
```

PRINT (Specifica)

Scopo: Visualizzare i dati sullo schermo.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: PRINT [*lista di espressioni*] [;]

? [*lista di espressioni*] [;]

Note: *lista di espressioni*
è una lista di espressioni a stringa e/o numeriche, separate da virgole, spazi in bianco o punti e virgola. Le costanti a a stringa della lista devono essere racchiuse tra apici.

Se si omette la lista di espressioni, viene visualizzata una riga bianca. Se si include la lista, vengono visualizzati i valori delle espressioni.

Posizioni di stampa

La posizione di ogni elemento stampato viene determinata dalla punteggiatura usata per separare gli elementi nella lista. Il BASIC divide la riga in zone di stampa di 14 spazi ognuna.

Nella lista delle espressioni:

- Una virgola tra le espressioni causa la stampa del valore successivo all'inizio della zona successiva.

- Un punto e virgola fa stampare il valore successivo immediatamente dopo l'ultimo valore.

- L'immissione di uno o più spazi tra le espressioni ha lo stesso effetto del punto e virgola.

Se la lista di espressioni termina con una virgola, con un punto e virgola o con la funzione SPC o TAB, la successiva specifica PRINT inizia la stampa sulla stessa riga, eseguendo la spaziatura corretta. Se la lista di espressioni termina senza una virgola, un punto e virgola, una funzione SPC o TAB, viene effettuato un ritorno a margine alla fine della riga; cioè, il BASIC sposta il cursore all'inizio della riga successiva.

Se la lunghezza del valore da stampare supera il numero di posizioni carattere che rimangono nella riga corrente, il valore viene stampato all'inizio di quella successiva. Se il valore da stampare è più lungo della WIDTH definita, il BASIC stampa quanto possibile sulla riga corrente e continua la stampa del resto del valore sulla successiva riga fisica.

Lo scorrimento si verifica come descritto nella sezione "Modo Testo" del Capitolo 3 del manuale BASIC.

I numeri stampati sono sempre seguiti da uno spazio. I numeri positivi sono preceduti da uno spazio, quelli negativi da un segno meno. I numeri a precisione singola, che possono essere rappresentati con 7 o meno cifre in formato a punto decimale fisso così come in formato a punto decimale mobile, vengono emessi usando il formato a punto decimale fisso o intero. Ad esempio, 10 ^ (-7) viene emesso come .0000001 e 10 ^ (-8) viene emesso come 1E-8.

Il BASIC inserisce automaticamente un ritorno a capo/avanzamento di riga dopo la stampa dei caratteri *ampiezza,* dove *ampiezza* può essere 40 o 80, come definito dalla specifica WIDTH.

Questo provoca il salto di due righe se si stampano esattamente 40 (o 80) caratteri, a meno che la specifica PRINT non termini con un punto e virgola (;).

Per stampare le informazioni sulla stampatrice si usa LPRINT. Vedere le specifiche LPRINT e LPRINT USING.

PRINT (Specifica)

Esempio: In questo esempio, le virgole nella specifica PRINT fanno sì che ogni valore sia stampato all'inizio della successiva zona di stampa.

```
10 X=5
20 PRINT X+5, X-5, X*(-5)
RUN
 10            0          -25
```

In questo esempio il punto e virgola alla fine della riga 20 causa la stampa di entrambe le specifiche PRINT sulla stessa riga.

```
10 INPUT X
20 PRINT X;"AL QUADRATO E'";X^2;"E";
30 PRINT X;"AL CUBO E'";X^3
RUN
? 9
 9 AL QUADRATO E' 81 E 9 AL CUBO E' 729
```

PRINT USING (Specifica)

Scopo: Stampare stringhe o numeri usando uno specifico formato.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: PRINT USING *v$; lista di espressioni*[;]

Note:

v$ è una costante o variabile a stringa di caratteri di formato speciale. Questi caratteri determinano il campo ed il formato delle stringhe o dei numeri stampati.

lista di espressioni
consiste in espressioni a stringa o numeriche che devono essere stampate, separate da punti e virgola o virgole.

Campi a stringa

Se si usa PRINT USING per stampare le stringhe, è possibile utilizzare uno dei tre caratteri di organizzazione del formato per formattare il campo a stringa.

! Specifica che deve essere stampato solo il primo carattere della stringa indicata.

\n spazi Specifica che devono essere stampati 2 +n caratteri della stringa. Se si immettono le barre al contrario senza spazi, vengono stampati due caratteri; con uno spazio 3 caratteri e così via.

Se la stringa è più lunga del campo, vengono ignorati i caratteri in eccesso. Se il campo è più lungo della stringa, questa viene allineata a sinistra nel campo e riempita di spazi in bianco sulla destra.

Esempio: Questo esempio mostra come usare *!* e \\ per stampare dei campi a stringa.

```
10 A$="LOOK":B$="OUT"
20 PRINT USING "!";A$;B$
30 PRINT USING "\  \";A$;B$
RUN
LO
LOOKOUT
```

& Specifica un campo a stringa di lunghezza variabile. Se si specifica il campo con "&", la stringa viene emessa esattamente come l'immissione. Esempio:

```
10 A$="LOOK": B$="OUT"
20 PRINT USING "!";A$;
30 PRINT USING "&";B$
RUN
LOUT
```

Campi numerici

Se si usa PRINT USING per stampare i numeri, è possibile utilizzare i seguenti caratteri speciali per formattare il campo numerico:

Si usa un segno di numero per rappresentare la posizione di ogni cifra. Le posizioni delle cifre sono sempre riempite. Se il numero da stampare contiene meno cifre delle posizioni specificate, il numero viene allineato a destra nel campo (preceduto da spazi in bianco).

•

E' possibile inserire un punto decimale in qualsiasi posizione del campo. Se il formato specifica che una cifra deve sempre precedere il punto decimale, la cifrà sarà sempre stampata (come Ø se necessario). I numeri vengono arrotondati se necessario.

```
PRINT USING "##.##";.78
 Ø78
```

Esempio: In questo esempio sono stati inseriti tre spazi alla fine della stringa di formato per separare i valori stampati sulla riga.

```
PRINT USING "##.##   ";1Ø.2,5.3,66.789,.234
1Ø.2Ø    5.3Ø   66.79    Ø.23
```

+

Un segno più all'inizio o alla fine della stringa di formato provoca la stampa del segno del numero (più o meno) prima o dopo di esso.

```
PRINT USING "+## .##    ";-68.95,2.4,55.6,-.9
-68.95    +2.4Ø   +55.6Ø     -Ø.9Ø
```

–

Un segno meno alla fine del campo del formato provoca la stampa dei numeri negativi con un segno meno finale.

```
PRINT USING "##.##-   ";-68.95,22.449,-7.Ø1
68.95-   22.45     7.Ø1-
```

**

Due asterischi all'inizio della stringa di formato provocano il riempimento con asterischi degli spazi iniziali del campo numerico. I due asterischi ** specificano inoltre le posizioni per altre due cifre.

PRINT USING (Specifica)

```
PRINT USING "**#.#   ";12.39,-0.9,765.1
*12.4   *-0.9* 765.1
```

$$ Un segno di dollaro doppio genera la stampa di un segno di dollaro a sinistra del numero. $$ specifica altre due posizioni per le cifre, una delle quali è il segno di dollaro. Con $$ non è possibile usare il formato esponenziale.

```
PRINT USING "$$##.##   ";456.78,0.9,-765.1
 $456.78    $0.90      -$765.10
```

**\$* All'inizio di una stringa di formato l'insieme di questi caratteri combina gli effetti dei due simboli precedenti. Gli spazi iniziali sono riempiti con asterischi e davanti al numero viene stampato il segno di dollaro. **$ specifica altre tre posizioni numeriche, una delle quali è il segno di dollaro.

```
PRINT USING "**$##.##";2.34
***$2.34
```

, Una virgola alla sinistra del punto decimale in una stringa di organizzazione del formato genera la stampa di una virgola alla sinistra di ogni tre cifre che si trovano alla sinistra del punto decimale. Una virgola alla fine della stringa viene stampata come parte della stringa. Una virgola specifica un'altra posizione numerica. La virgola non ha effetto se usata col formato esponenziale (^^^^).

PRINT USING (Specifica)

```
PRINT USING "####,.##";1234.5
1,234.5Ø

PRINT USING "####.##,";1234.5
1234.5Ø,
```

^^^^ Dopo i caratteri di posizione numerica, è possibile porre quattro segni di omissione per specificare il formato esponenziale. I quattro segni procurano spazio per la stampa di E±nn o D±nn.
Può essere specificato qualsiasi posizione del punto decimale. Le cifre significative sono allineate a sinistra e viene regolato l'esponente. A meno che non si specifichi un + iniziale o un + o − finale, si usa una posizione numerica alla sinistra del punto decimale per stampare uno spazio o un segno meno.

```
PRINT USING "##.##^^^^";234.56
 2.35EØ2

PRINT USING ".###^^^^-";-88888
.889EØ5-

PRINT USING "+.##^^^^";123
+.12EØ3
```

_ Un carattere di sottolineatura nella stringa di formato provoca l'emissione del successivo carattere come costante.

```
PRINT USING "_!##.##_!";12.34
!12.34!
```

Il carattere costante può essere a sua volta una sottolineatura ponendo "__" nella stringa di formato.

PRINT USING (Specifica)

Se il numero da stampare è maggiore del campo numerico specificato, davanti al numero viene stampato un segno di percentuale (%). Se con l'arrotondamento il numero eccede il campo, il segno di percentuale viene stampato davanti al numero arrotondato.

```
PRINT USING "##.##";111.22
%111.22

PRINT USING ".##";.999
%1.00
```

Se il numero di cifre specificate supera 24 si verifica un errore di **Illegal function call.**

Esempio: Questo esempio mostra come includere costanti a stringa nella stringa di formato.

```
PRINT USING "QUESTO E' L'ESEMPIO _##";1
QUESTO E' L'ESEMPIO #1
```

PRINT # e PRINT # USING (Specifiche)

Scopo: Scrivere sequenzialmente i dati in un file.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: PRINT # *numfile,* [USING *x$;*] *lista di espr.* [;]

Note:

numfile è il numero usato, quando il file è stato aperto, per l'emissione.

x$ è un'espressione a stringa costituita da caratteri per l'organizzazione del formato, come descritto nella precedente specifica PRINT USING.

lista di espr. è una lista delle espressioni numeriche e/o a stringa che saranno scritte nel file.

PRINT # non comprime i dati nel file. Una immagine dei dati viene scritta nel file così come viene visualizzata sullo schermo con una specifica PRINT. Per questa ragione, è consigliabile delimitare con cura i dati nel file, in modo che l'immissione sia corretta.

Nella lista di espressioni, quelle numeriche devono essere delimitate da punti e virgola. Ad esempio:

```
PRINT #1,A;B;C;X;Y;Z
```

(Se si usano le virgole come delimitatori, vengono scritti nel file anche gli spazi in eccesso inseriti tra i campi di stampa).

PRINT # e PRINT # USING (Specifiche)

Le espressioni a stringa devono essere separate da punti e virgola nella lista. Per formattare correttamente nel file le espressioni a stringa, usare i delimitatori espliciti nella lista di espressioni.

Ad esempio, se A$="CAMERA" e B$="93604-1" La specifica

```
PRINT #1,A$;B$
```

scrive nel file CAMERA93604-1. Poiché non vi sono delimitatori, non è possibile l'immissione in due stringhe separate. Per correggere l'errore, inserire dei delimitatori espliciti nella specifica PRINT # nel modo seguente:

```
PRINT #1,A$;",";B$
```

l'immagine scritta nel file è:

```
CAMERA, 93604-1
```

che può essere letta in due variabili a stringa.

Se le stringhe stesse contengono virgole, punti e virgola, spazi bianchi iniziali significativi, ritorni a margine o avanzamenti di riga, scriverli nel file con CHR$(34) delimitati da apici espliciti.

Ad esempio, se A$="CAMERA,AUTOMATIC" e B$="93604-1". La specifica:

```
PRINT #1,A$;B$
```

scrive nel file:

```
CAMERA, AUTOMATIC  93604-1
```

e la specifica:

```
INPUT #1,A$,B$
```

PRINT # e PRINT # USING (Specifiche)

immette la stringa "CAMERA" in A$ e "AUTOMATIC 93604-1" in B$.

Per separare correttamente nel file queste stringhe, scrivere degli apici doppi per l'immagine del file con CHR$(34).
La specifica:

```
PRINT #1,CHR$(34);A$;CHR$(34);CHR$(34);B$;CHR$(34)
```

scrive nel file:

```
"CAMERA, AUTOMATIC""93604-1"
```

e la specifica:

```
INPUT #1,A$,B$
```

immette "CAMERA,AUTOMATIC" in A$ e "93604-1" in B$.

La specifica PRINT # può essere utilizzata anche con l'opzione USING per controllare il formato del file. Ad esempio:

```
PRINT# 1,USING"$$###.##,";J;K;L
```

PRINT # e PRINT # USING (Specifiche)

Esempio: Poiché i dati scritti nel file contengono il segno di dollaro, usando le variabili a stringa come mostrato nell'esempio riportato, possono essere a loro volta riletti.

```
10 A=123
20 B=6789
30 C=22,33
40 OPEN "DATA" FOR OUTPUT AS #1
50 PRINT #1,USING "$$###.##,";A;B;C
60 CLOSE
70 OPEN "DATA" FOR INPUT AS #1
80 INPUT #1,A$,B$,C$
90 CLOSE
100 PRINT A$,B$,C$
```

PSET e PRESET (Specifiche)

Scopo: Disegnare un punto nella posizione specificata sullo schermo.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Solo modo grafici.

Formato: PSET *(x,y)* [,*colore*]

PRESET *(x,y)* [,*colore*]

Note:

(x,y) sono le coordinate del punto da impostare. Possono essere in forma relativa o assoluta come descritto nel Capitolo 3 del manuale BASIC.

colore è un'espressione intera che sceglie un attributo dalla gamma disponibile per il modo schermo in uso. A risoluzione media, *colore* è il colore in uso per quell'attributo così come è stato definito dalla specifica COLOR. A media risoluzione sono disponibili 4 attributi (Ø-3); ad alta risoluzione 2 (Ø-1). Zero (Ø) è sempre l'attributo per lo sfondo. L'attributo assunto per difetto per il primo piano è sempre il massimo attributo per quel modo schermo; 3 a media risoluzione, 1 ad alta risoluzione.

PSET e PRESET (Specifiche)

PRESET è quasi identica a PSET. L'unica differenza è che se a PRESET non viene fornito il parametro *colore,* si *sceglie* il colore di fondo Ø. Se si indica *colore* PRESET è identica a PSET. La riga 7Ø dell'esempio potrebbe essere:

```
7Ø PSET(I,I),Ø
```

Le coordinate fuori dall'intervallo vengono troncate.

Esempio: Le righe 2Ø-4Ø di questo esempio disegnano una linea diagonale dal punto (Ø, Ø) al punto (1ØØ, 1ØØ). Le righe 6Ø-8Ø cancellano la linea impostando ogni punto al colore Ø.

```
1Ø CLS:SCREEN 1:KEY OFF
2Ø FOR I=Ø TO 1ØØ
3Ø PSET (I,I)
4Ø NEXT
5Ø 'cancella la riga
6Ø FOR I=1ØØ TO Ø STEP -1
7Ø PRESET(I,I)
8Ø NEXT
```

PUT
(Specifica File)

Scopo: Scrivere un record da una memoria di transito ad accesso casuale in un file ad accesso casuale.

Versioni:	Cassetta	Disco	Avanzato	Compilatore
		***	***	***

Formato: PUT [#] *numfile* [,*numero*]

Note: *numfile* è il numero con il quale è stato aperto il file.

numero è il numero del record da scrivere, compreso tra 1 e 16 megabyte.

Se si omette *numero* il record assume il successivo numero disponibile (dopo l'ultima PUT).

PRINT #, PRINT #USING, WRITE #, LSET e RSET possono essere utilizzate per immettere i caratteri nella memoria di transito del file ad accesso casuale prima di una specifica PUT. Nel caso di WRITE # il BASIC riempie la memoria di transito con spazi in bianco fino al ritorno a margine.

Qualsiasi tentativo di lettura o scrittura oltre la fine della memoria di transito causa un errore di **Field overflow.** Vedere nell'Appendice A del manuale BASIC la sezione I/E su disco BASIC.

Poiché il BASIC e il DOS bloccano più record possibili in settori da 512 byte, la specifica PUT non esegue necessariamente una scrittura fisica sul disco per ogni record.

PUT
(Specifica file)

PUT può essere utilizzata per un file di comunicazioni. In questo caso *numero* corrisponde al numero di byte da scrivere nel file di comunicazioni. Questo numero deve essere inferiore o uguale al valore impostato dall'opzione LEN della specifica OPEN "COM...

Esempio: vedere l'appendice A del manuale BASIC.

PUT
(Specifica grafici)

Scopo: Tracciare immagini su una specifica area dello schermo.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
		***	***

Solo modo grafici.

Formato: PUT *(x,y), schiera* [,*azione*]

Note:

(x,y) sono le coordinate dell'angolo superiore sinistro dell'immagine da trasferire.

schiera è il nome di una schiera numerica contenente le informazioni da trasferire. Per maggiori informazioni vedere la specifica GET.

azione è una di queste:

PSET
PRESET
XOR
OR
AND

XOR è quella assunta per difetto.

PUT è l'opposto di GET nel senso che prende i dati della schiera e li porta sul video. Tuttavia fornisce anche l'opzione per interagire con i dati già sullo schermo.

PSET memorizza semplicemente i dati provenienti dalla schiera sullo schermo, quindi è l'opposto di GET.

PUT
(Specifica grafici)

PRESET equivale a PSET tranne nel fatto che produce un'immagine complementare. Cioè, a media risoluzione che ha un attributo massimo di 3, un valore Ø nella schiera assegna al punto corrispondente il colore 3 e viceversa; un valore 1 nella schiera assegna al punto corrispondente il colore 2 e viceversa.

AND, OR, e XOR specificano le operazioni logiche sui bit di ogni immagine. AND si usa quando un'immagine già esiste nell'area in cui l'immagine è trasferita.

Si usa OR per sovrapporre l'immagine trasferita a quella esistente.

XOR è un modo speciale che può essere utilizzato per l'animazione. XOR ha una caratteristica esclusiva; quando si pone (PUT) un'immagine *due volte* su uno sfondo complesso, lo sfondo viene ripristinato invariato. Ciò consente lo spostamento di un oggetto senza cancellare lo sfondo.

A media risoluzione AND, XOR e OR hanno sul colore i seguenti effetti:

AND

Schermo	Valore della schiera 0	1	2	3
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

OR

Schermo	Valore della schiera 0	1	2	3
0	0	1	2	3
1	1	1	3	3
2	2	3	2	3
3	3	3	3	3

PUT
(Specifica grafici)

XOR

Schermo	Valore della schiera			
	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

L'animazione di un oggetto può essere eseguita nel modo seguente:

1. Porre (PUT) l'oggetto sullo schermo (con XOR).
2. Ricalcolare la nuova posizione dell'oggetto.
3. Porre (PUT) l'oggetto sullo schermo (con XOR) una seconda volta nella vecchia posizione per eliminare la vecchia immagine.
4. Ripetere il passo 1, questa volta ponendo l'oggetto nella nuova posizione.

Il movimento eseguito in questo modo lascia invariato lo sfondo. Il tremolio può essere ridotto minimizzando il tempo tra i passi 4 e 1 e assicurandosi che non esistano ritardi tra i passi 1 e 3. Se si deve animare più di un oggetto, elaborarne uno alla volta, un passo per volta.

Se non è importante mantenere lo sfondo, l'animazione può essere eseguita con l'azione PSET. Ricordarsi di avere un'area di immagine che contenga le immagini dell'oggetto "prima" e "dopo". In questo modo l'area in eccesso cancellerà realmente la vecchia immagine.

PUT
(Specifica grafici)

Questo metodo può essere più veloce di quello che utilizza XOR, poiché è necessaria solo una specifica PUT per spostare un oggetto (anche se si deve portare sullo schermo un'immagine più grande).

Se l'immagine da trasferire supera le dimensioni dello schermo, si verifica un errore di **Illegal function call**.

Esempio: Questo esempio mostra come spostare un cerchio attraverso lo schermo con XOR.

```
10 CLS:DEFINT A-Z:SCREEN 1:KEY OFF
20 DIM A(404)
30 CIRCLE (160,100), 20,3
40 PAINT (160,100),2,3
50 GET (140,80)-(180,120),A:CLS
60 X=30: Y=50
70 FOR I=1 TO 20
80 PUT (X,Y),A,XOR
90 PUT (X,Y),A,XOR
100 X=X + 10
110 NEXT
```

RANDOMIZE (Specifica)

Scopo: Ripristinare il generatore di numeri casuali.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: RANDOMIZE [*n*]

RANDOMIZE TIMER

Note: *n* è un'espressione intera con precisione singola o doppia utilizzata come generatore di numeri casuali.

Se si omette *n*, il BASIC sospende l'esecuzione del programma e richiede un valore prima di eseguire RANDOMIZE, visualizzando:

```
Random Number Seed (-32768 to 32767)?
```

Se il generatore di numeri casuali non è ripristinato, la funzione RND fornisce la stessa sequenza di numeri casuali ogni volta che viene eseguito il programma. Per modificare questa sequenza ad ogni esecuzione del programma, porre una specifica RANDOMIZE all'inizio del programma stesso e modificare il generatore ad ogni esecuzione.

Nel BASIC su disco e nel BASIC avanzato, l'orologio interno può essere utile per avere un generatore di numeri casuali. E' possibile utilizzare VAL per modificare le ultime due cifre di TIME$ in un numero ed utilizzare questo numero per *n*.

RANDOMIZE (Specifica)

Nel BASIC 2.0 e successivi livelli si può avere eliminando la richiesta un nuovo numero casuale. Per far questo usare nell'espressione la funzione TIMER.

Esempio:

```
10 RANDOMIZE
20 FOR I=1 TO 4
30 PRINT RND;
40 NEXT I

RUN
Random Number Seed (-32768 to 32767)?
```

Si supponga di rispondere 3. Il programma continua:

```
Random Number Seed (-32768 to 32767)? 3
 .7655695  .3558607  .3742327  .1388798

RUN
Random Number Seed (-32768 to 32767)?
```

Questa volta, si supponga di rispondere 4. Il programma continua:

```
Random Number Seed (-32768 to 32767)? 4
 .1719568  .5273236  .6879686  .713297

RUN
Random Number Seed (-32768 to 32767)?
```

Se si ritenta con il numero 3 si avrà la stessa sequenza della prima esecuzione.

```
Random Number Seed (-32768 to 32767)? 3
 .7655695  .3558607  .3742327  .1388798
```

RANDOMIZE (Specifica)

Ogni volta che il seguente programma, che usa TIMER, viene eseguito, si avrà una diversa sequenza di numeri.

```
10 RANDOMIZE TIMER
20 FOR I=1 TO 4
30 PRINT RND;
40 NEXT
RUN
 .9590051  .1036786  .1464037  .7754918

RUN
 .8261163  .17422  .9191545  .5041142
```

RANDOMIZE (Specifica)

Nel BASIC 2.0 e successivi livelli si può avere eliminando la richiesta un nuovo numero casuale. Per far questo usare nell'espressione la funzione TIMER.

Esempio:

```
10 RANDOMIZE
20 FOR I=1 TO 4
30 PRINT RND;
40 NEXT I

RUN
Random Number Seed (-32768 to 32767)?
```

Si supponga di rispondere 3. Il programma continua:

```
Random Number Seed (-32768 to 32767)? 3
 .7655695   .3558607   .3742327   .1388798

RUN
Random Number Seed (-32768 to 32767)?
```

Questa volta, si supponga di rispondere 4. Il programma continua:

```
Random Number Seed (-32768 to 32767)? 4
 .1719568   .5273236   .6879686   .713297

RUN
Random Number Seed (-32768 to 32767)?
```

Se si ritenta con il numero 3 si avrà la stessa sequenza della prima esecuzione.

```
Random Number Seed (-32768 to 32767)? 3
 .7655695   .3558607   .3742327   .1388798
```

READ (Specifica)

Scopo: Leggere i valori di una specifica DATA e assegnarli a delle variabili.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: READ *variabile* [,*variabile*]...

Note: *variabile* è una variabile numerica o a stringa o un elemento di schiera che deve ricevere il valore letto dalla tabella DATA.

Una specifica READ deve essere sempre usata congiuntamente a DATA. Le specifiche READ assegnano i valori di DATA alle variabili di READ, sulla base di una-a-una.

Le variabili della specifica READ possono essere numeriche o a stringa ed i valori letti devono corrispondere ai tipi di variabile specificati. In caso negativo si verifica un **Syntax error.**

Un'unica specifica READ può accedere a più specifiche DATA (in ordine), oppure più specifiche READ possono accedere alla stessa specifica DATA. Se il numero di variabili nella lista supera il numero di elementi delle specifiche DATA, si verifica un errore **Out of data.** Se il numero delle variabili specificate è inferiore a quello degli elementi nelle specifiche DATA, le successive READ inizieranno leggendo i dati al primo elemento non letto. Se non vi sono specifiche READ successive, i dati in eccesso vengono ignorati.

READ
(Specifica)

Per rileggere i dati di una riga nella lista di specifiche DATA, usare RESTORE.

Esempio: Questo segmento di programma legge i valori delle specifiche DATA nella schiera A. Dopo l'esecuzione, il valore di A(1) è 3.Ø8 e così via.

```
1Ø FOR I=1 TO 1Ø
2Ø READ A(I)
3Ø NEXT I
4Ø DATA 3.Ø8,5.19,3.12,3.98,4.24
5Ø DATA 5.Ø8,5.55,4.ØØ,3.16,3.37
```

Questo programma legge dati a stringa o numerici dalla specifica DATA a riga 3Ø. Notare che non sono necessari gli apici ai limiti di COLORADO, poiché non ha virgole, punti e virgole o spazi in bianco iniziali e finali significativi. Tuttavia sono necessari gli apici per "DENVER," a causa della virgola.

```
1Ø PRINT "CITTA'", "STATO", "CAP"
2Ø READ C$,S$,Z
3Ø DATA "DENVER,", COLORADO, 8Ø211
4Ø PRINT C$,S$,Z
RUN
CITTA'     STATO          CAP
DENVER,    COLORADO       8Ø211
```

REM (Specifica)

Scopo: Inserire note esplicative in un programma.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: REM *nota*

Note: *nota* può essere una qualsiasi sequenza di caratteri.

Le specifiche REM non vengono eseguite ma, quando viene listato il programma, vengono visualizzate esattamente come erano state immesse. Tuttavia rallentano il tempo di esecuzione e occupano spazio in memoria.

E' possibile saltare nelle specifiche REM (da una specifica GOTO o GOSUB) e l'esecuzione continua con la prima specifica eseguibile dopo REM.

E' possibile aggiungere delle note alla fine di una riga facendo precedere alla nota un unico punto interrogativo invece di :REM. Se si immette una nota su una riga con altre specifiche BASIC, la nota deve essere *l'ultima* specifica della riga.

Non è consigliabile usare un singolo apice, per aggiungere commenti, alla fine di una specifica DATA in quanto il BASIC prenderebbe il commento come parte della stringa. A tale scopo si può invece usare :REM.

REM
(Specifica)

Esempio: Questo esempio illustra i due modi di inserire delle note in un programma.

```
10 ' calcola la velocità media
20 SUM=0: REM inizializza SUM
30 FOR I=1 TO 20
40 SUM=SUM + V(I)
```

La riga 20 può essere scritta anche:

```
20 SUM=0 ' inizializza SUM
```

RENUM (Comando)

Scopo: Rinumerare le righe di programma.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	

Formato: RENUM [*nuovonum*] [,[*vecchionum*] [,*incremento*]]

Note:

nuovonum è il primo numero di riga da utilizzare nella nuova sequenza. Il valore assunto è 10.

vecchionum è la riga del programma dalla quale deve iniziare la numerazione. Il valore assunto è la prima riga del programma.

incremento è l'incremento da usare nella nuova sequenza. Il valore assunto è 10.

RENUM modifica anche tutti i riferimenti al numero di riga che seguono GOTO, GOSUB, THEN, ELSE, ON...GOTO, ON...GOSUB, RESTORE, RESUME e ERL per riflettere i nuovi numeri di riga. Se appare un numero di riga inesistente dopo una di queste specifiche, viene stampato il messaggio di errore **Undefined line number xxxxx in yyyyy**. Il riferimento al numero di riga errato non viene modificato da RENUM, ma può essere cambiato il numero di riga.

> **Nota:** Non è possibile usare RENUM per modificare l'ordine delle righe di programma (ad esempio, RENUM 15,30 quando il programma ha tre righe numerate 10, 20 e 30) o per creare numeri di riga maggiori di 65529. Si verificherebbe l'errore **Illegal function call.**

RENUM (Comando)

Esempio: rinumera l'intero programma. Il primo nuovo numero di riga è 10. L'incremento delle righe è 10.

```
RENUM
```

Questo esempio rinumera l'intero programma. Il primo nuovo numero di riga è 300. L'incremento delle righe è 50.

```
RENUM 300,,50
```

Questo esempio rinumera le righe a partire dalla 900, in modo da iniziare dal numero 1000 e l'incremento è 20.

```
RENUM 1000,900,20
```

RESET (Comando)

Scopo: Chiudere tutti i file su minidisco e vuotare la memoria di transito del sistema.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	***

Formato: RESET

Note: Se tutti i file aperti si trovano sul disco, RESET corrisponde a CLOSE senza numeri di file.

RESTORE (Specifica)

Scopo: Consentire di rileggere le specifiche DATA da una determinata riga.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: RESTORE [*riga*]

Nota: *riga* è il numero di riga di una specifica DATA del programma.

Dopo l'esecuzione di una specifica RESTORE, la successiva READ accede al primo elemento della prima specifica DATA del programma. Se si specifica riga, la successiva READ accede al primo elemento della specifica DATA indicata.

Esempio: La specifica RESTORE a riga 20 ripristina il puntatore DATA all'inizio, in modo che i valori letti a riga 30 siano 57, 68 e 79.

```
10 READ A,B,C
20 RESTORE
30 READ D,E,F
40 DATA 57, 68, 79
50 PRINT A;B;C;D;E;F
RUN
 57  68  79  57  68  79
```

RESUME (Specifica)

Scopo: Continuare l'esecuzione del programma dopo una procedura di ripristino dell'errore.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: RESUME [0]

RESUME NEXT

RESUME [*riga*]

Note: E' possibile utilizzare uno qualsiasi dei formati illustrati, a seconda del punto in cui deve riprendere l'esecuzione.

RESUME o RESUME 0

L'esecuzione riprende dalla specifica che ha causato l'errore.

> **Nota:** se si tenta di rinumerare un programma contenente una specifica RESUME 0, si verifica l'errore **Undefined line number**. La specifica indicherà ancora RESUME 0, che è corretto.

RESUME NEXT

L'esecuzione riprende dalla specifica immediatamente successiva a quella che ha causato l'errore.

RESUME riga

L'esecuzione riprende dal numero di riga specificato.

RESUME (Specifica)

Una specifica RESUME che non si trova in un programma di intercettazione dell'errore causa un messaggio **RESUME without error.**

Esempio: La riga 1000 è l'inizio del programma di intercettazione degli errori. La specifica RESUME fa ritornare il programma alla riga 80 quando a riga 90 si verifica l'errore 230.

```
10 ON ERROR GOTO 1000
 .
 .
 .
1000 IF (ERR=230) AND(ERL=90)  THEN PRINT
          "PROVARE DI NUOVO": RESUME 80
```

RETURN (Specifica)

Scopo: Per ritornare da un sottoprogramma. Vedere le specifiche GOSUB e RETURN.

Versioni:	Cassetta	Disco	Avanzato	Compilatore
	***	***	***	***

Formato: RETURN [*riga*]

Nota: *riga* è il numero della riga di programma alla quale si vuole ritornare. E' utilizzabile solo nel BASIC avanzato e nel BASIC compilatore.

Benché si possa ritornare da un sottoprogramma con RETURN *riga,* questa miglioria è stata aggiunta per consentire ritorni non locali dai programmi di intercettazione delle azioni. Da uno di questi programmi si ritorna al programma BASIC ad un determinato numero di riga eliminando ancora l'elemento GOSUB creato dall'intercettazione. Tuttavia, usare attentamente questo ritorno (RETURN) non locale, poiché le altre GOSUB, WHILE o FOR, attive al momento dell'intercettazione, rimangono tali.

RIGHT$ (Funzione)

Scopo: Fornire i caratteri *n* più a destra della stringa *x$*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v$* = RIGHT$(*x$*,*n*)

Note:

x$ è un'espressione a stringa.

n è un'espressione intera che specifica il numero di caratteri che devono trovarsi nel risultato.

Se *n* è maggiore o uguale a LEN(*x$*) viene fornita *x$*. Se *n* è zero, viene fornita la stringa nulla (lunghezza zero).

Vedere anche le funzioni MID$ e LEFT$.

Esempio: Vengono forniti i sette caratteri più a destra della stringa A$.

```
10 A$="BOCA RATON, FLORIDA"
20 PRINT RIGHT$(A$,7)
RUN
FLORIDA
```

RMDIR (Comando)

Scopo: Cancellare un indirizzario da uno specifico disco. (BASIC 2.0 e successivi livelli).

Versioni:

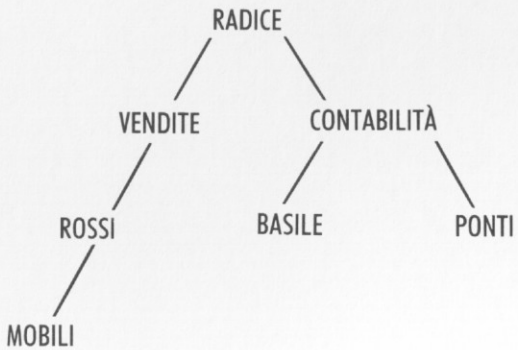
Cassetta	Disco	Avanzato	Compilatore
	***	***	

Formato: RMDIR *percorso*

Note: *percorso* è una espressione a stringa e non può superare 63 caratteri. Identifica il sottoindirizzario che deve essere cancellato nell'indirizzario esistente. Riferirsi a "Denominazione dei file" e "Indirizzari strutturati ad albero" nel Capitolo 3 del manuale BASIC per maggiori informazioni.

L'indirizzario deve essere vuoto di tutti i file e dei sottoindirizzari prima di poter essere cancellato, con l'eccezione delle entrate '.' e '..' altrimenti si verifica l'errore **Path/file access error.**

Esempio:



RMDIR (Comando)

I seguenti esempi si riferiscono alla struttura ad albero precedente.

Trovandosi nell'indirizzario RADICE e volendo rinnovare MOBILI, immettere:

```
RMDIR "VENDITE\ROSSI\MOBILI"
```

Se si vuole rendere attuale l'indirizzario CONTABILITA' e cancellare quello chiamato PONTI:

```
CHDIR "CONTABILITA'"
RMDIR "PONTI"
```

Un'altra maniera per cancellare l'indirizzario PONTI è:

```
CHDIR "\"
RMDIR "CONTABILITA' PONTI"
```

L'indirizzario che precede l'attuale non può essere cancellato. Usando la struttura ad albero descritta, supponiamo che ROSSI sia l'attuale. Provando a cancellare VENDITE si provoca l'errore **Path/file access.**

Se si prova ad usare il comando KILL per cancellare un indirizzario, si provoca ancora lo stesso errore.

RND (Funzione)

Scopo: Fornire un numero casuale tra Ø e 1.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: $v = \text{RND}\,[(x)]$

Note:

x è un'espressione numerica che interessa i valori forniti, come descritto sotto.

La stessa sequenza di numeri casuali viene generata ad ogni esecuzione del programma, a meno che non venga ripristinato il generatore di numeri casuali. Per questo utilizzare la specifica RANDOMIZE. E' possibile ripristinare il generatore anche quando si richiama la funzione RND usando *x*, dove *x* è negativa. Questo genera sempre una particolare sequenza per la *x*. Tale sequenza non è interessata da RANDOMIZE, in modo che se si vuole generare una sequenza diversa ad ogni esecuzione del programma, si deve usare ogni volta un valore diverso per *x*.

Se *x* è positiva o non è inclusa, RND*(x)* genera il successivo numero casuale della sequenza.

RND(Ø) ripete l'ultimo numero generato.

Per avere numeri casuali nella gamma Ø-*n*, usare la formula:

```
INT (RND * (n+1))
```

RND (Funzione)

Esempio: In questo esempio la prima riga orizzontale dei risultati mostra tre numeri casuali, generati usando una *x* positiva.

A riga 40, si utilizza un numero negativo per ripristinare il generatore di numeri casuali. I numeri prodotti dopo il ripristino si trovano nella seconda fila dei risultati.

A riga 80, il generatore dei numeri casuali viene ripristinato con la specifica RANDOMIZE; a riga 90 è ripristinato richiamando RND con lo stesso valore negativo usato a riga 40. Questo annulla l'effetto della specifica RANDOMIZE; la terza riga dei risultati è uguale alla seconda.

A riga 130 si richiama RND con un argomento zero, in modo che l'ultimo numero stampato sia uguale a quello precedente.

```
10 FOR I=1 TO 3
20 PRINT RND(I);          ' x>0
30 NEXT I
40 PRINT: X=RND(-6)       ' x<0
50 FOR I=1 TO 3
60 PRINT RND(I);          ' x>0
70 NEXT I
80 RANDOMIZE 853 'randomize
90 PRINT: X=RND(-6)       ' x<0
100 FOR I=1 TO 3
110 PRINT RND;            ' come x>0
120 NEXT I
130 PRINT: PRINT RND(0)
RUN
 .6291626   .1948297   .6305799
 .6818615   .4193624   .6215937
 .6818615   .4193624   .6215937
 .6215937
```

RND (Funzione)

Il ripristino con la funzione RND (numero negativo) avviene mediante la modifica dell'ultimo punto decimale mobile. Poiché nessun calcolo viene eseguito in questo esempio, la nuova **generazione** è sempre la stessa.

```
10 DEFINT A-Z
20 FOR J=1 TO 5
30 X=RND(-J)
40 FOR I=1 TO 5:PRINT RND;NEXT:PRINT
50 NEXT J
```

Se la riga 20 viene modificata in:

```
20 FOR J=1 TO 2 STEP .1
```

ogni volta verrà effettuata una nuova **generazione.**

RUN (Comando)

Scopo: Iniziare l'esecuzione di un programma.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: RUN [*riga*]

RUN *specfile* [,R]

Note:

riga è il numero della riga del programma in memoria dalla quale si vuole iniziare l'esecuzione.

specfile è un'espressione a stringa per la specifica del file, nel BASIC 2.Ø e successivi livelli può contenere un percorso. *specfile* deve essere conforme alle regole esposte nella sezione "Denominazione dei File" nel Capitolo 3 del manuale BASIC. In caso contrario si verifica un errore.

RUN o RUN *riga* inizia l'esecuzione del programma in memoria. Se si specifica *riga*, l'esecuzione inizia con il numero di riga indicato. Altrimenti l'esecuzione inizia dal numero di riga più basso.

RUN *specfile* carica un file dal minidisco o cassetta in memoria e lo elabora. Chiude tutti i file aperti e cancella l'attuale contenuto della memoria prima di caricare il programma scelto. Tuttavia con l'opzione R, tutti i file di dati rimangono aperti. Vedere l'Appendice A del manuale BASIC.

RUN (Comando)

L'esecuzione di un comando RUN disattiva l'eventuale suono e ripristina la musica di fondo. Anche PEN e STRIG vengono ripristinate a OFF.

Esempio: In questo esempio si usa la prima forma di RUN su due programmi molto piccoli. Il primo programma è eseguito dall'inizio. Nel secondo esempio si è usata l'opzione RUN *riga* per eseguire il programma da riga 20. In questo caso, la riga 10 non viene eseguita, così PI non riceve il valore corretto. Viene stampato uno 0 perché tutte le variabili numeriche hanno un valore iniziale zero.

```
10 PRINT 1/7
RUN
 .1428571
Ok
10 PI=3.141593
20 PRINT PI
RUN 20
 0
```

L'esempio seguente carica il programma "NEWFIL" dal nastro e lo esegue, mantenendo aperti i file.

```
RUN "NEWFIL",R
```

SAVE (Comando)

Scopo: Salvare un file di programma BASIC su un disco o cassetta.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	

Formato: SAVE *specfile* [,A]

SAVE *specfile* [,P]

Note: *specfile* è un'espressione a stringa per la specifica del file. In BASIC 2.0 e successivi livelli può contenere un percorso. *specfile* deve essere conforme alle regole descritte nella sezione "Denominazione di File" del Capitolo 3 del manuale BASIC.

Il programma BASIC viene scritto sull'unità specificata. Salvando su CAS1:, viene acceso il registratore ed il file viene scritto immediatamente sul nastro.

Per i file su disco, se il nome del file è di otto caratteri o meno e non viene fornita l'estensione, al nome si aggiunge .BAS. Se sul minidisco esiste già un file con lo stesso nome, quest'ultimo verrà ricoperto.

Utilizzando il BASIC su cassetta, se si omette il nome dell'unità, si assume CAS1:, che è l'unica unità disponibile per il salvataggio.

Per il BASIC su disco e avanzato, si assumono le unità del DOS.

L'opzione A salva il programma in formato ASCII. Altrimenti, il BASIC salva il file in un formato binario compresso. I file ASCII occupano più spazio, ma alcuni tipi di accesso richiedono che il file sia in formato ASCII. Ad esempio, un file sul quale si intende eseguire la fusione deve essere salvato in formato ASCII. I programmi salvati in ASCII devono essere letti come file di dati.

L'opzione P salva il programma in un formato binario codificato. Quando successivamente si esegue (o si carica) un programma protetto, qualsiasi tentativo di listarlo (LIST) o editarlo (EDIT) causa un errore di **Illegal function call.** Non vi è alcun modo per togliere la protezione al programma.

> **Nota:** l'indirizzario del disco per un file di programma BASIC non fornisce indicazioni se il file è protetto o memorizzato in formato ASCII. In ogni caso si usa l'estensione .BAS

Vedere anche l'Appendice A "I/E su disco BASIC". nel manuale BASIC.

Esempio: Questo esempio salva il programma in memoria come INVENT con l'estensione .BAS

```
SAVE "INVENT"
```

Questo esempio salva PROG.BAS nell'unità B: in formato ASCII, in modo da poter eseguire successivamente la fusione.

```
SAVE "B:PROG",A
```

Questo esempio salva SECRET.BOZ nell'unità A: in forma protetta per impedire che venga alterato.

```
SAVE "A:SECRET.BOZ",P
```

SCREEN (Funzione)

Scopo: Fornire il codice ASCII (0-255) per il carattere sullo schermo attivo alla riga e colonna specificata.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v* = SCREEN(*riga*,*col*[,*z*])

Nota:

riga è un'espressione numerica compresa tra 1 e 25.

col è un'espressione numerica compresa tra 1 e 40 o 1 e 80 in base all'impostazione di WIDTH.

z è un'espressione numerica che calcola un valore falso o vero. *z* è valida solo in modo testo.

Vedere l'Appendice D per una lista dei codici ASCII.

In modo testo, se si include *z* ed è vera (non zero), viene fornito l'attributo del colore per il carattere, invece del codice. L'attributo del colore è un numero compreso tra 0 e 255. Questo numero, *v*, può essere decifrato nel modo seguente:

(*v* MOD 16) è il colore di primo piano.

(((*v-primopiano*)/16) MOD128) è il colore di fondo, dove primo piano è calcolato come sopra.

($v > 127$) è vera (–1) se il carattere lampeggia, falsa (Ø) in caso contrario.

Fare riferimento alla specifica COLOR per una lista dei colori e relativi numeri.

In modo grafici, se la posizione specificata contiene informazioni grafiche (punti o righe, l'opposto di un carattere), la funzione SCREEN ritorna zero.

Qualsiasi valore immesso che non sia compreso nei limiti indicati produce un errore di **Illegal function call.**

Vedere la specifica SCREEN nella sezione successiva.

Esempio: In questo esempio il carattere a posizione 1Ø,1Ø è A, allora X è 65.

```
1ØØ X = SCREEN (1Ø,1Ø)
```

Questo esempio fornisce l'attributo del colore per il carattere nell'angolo superiore sinistro dello schermo.

```
11Ø X = SCREEN (1,1,1)
```

SCREEN (Specifica)

Scopo: Definire gli attributi dello schermo che devono essere utilizzati dalle specifiche successive.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Ha significato solo con l'Adattatore per il video grafico a colori.

Formato: SCREEN [*modo*] [,[*colore*] [,[*paginaa*] [,*paginav*]]]

Note:

modo è un'espressione numerica che dà come risultato un valore intero 0, 1 o 2. I modi validi sono:

- **0** Modo testo ad ampiezza attuale (40 o 80).
- **1** Modo grafici media risoluzione (320x200). Da usare solo con l'Adattatore per il video grafico a colori.
- **2** Modo grafici alta risoluzione (640x200). Da usare solo con l'Adattatore per il video grafico a colori.

colore è un'espressione numerica che dà come risultato un valore falso o vero. Abilita o disabilita il colore. Su un monitor RGB il colore è sempre ON, mentre su un monitor composito può essere ON o OFF.
In modo testo (*modo* = 0), un valore falso (zero) disabilita il colore (vengono visualizzate solo immagini in bianco e e nero) ed un valore vero (non zero) abilita il colore (consente immagini colorate).

In modo grafici a risoluzione media (*modo* = 1), un valore vero (non zero) disabilita il colore ed uno falso (zero) lo abilita. Poiché il bianco e il nero sono gli unici colori ad alta risoluzione, questo parametro non ha effetto ad alta risoluzione.

paginaa (pagina attiva) è una espressione intera compresa tra Ø e 7 con ampiezza 4Ø, oppure Ø e 3 con ampiezza 8Ø. Sceglie la pagina su cui scrivere con specifiche di emissione sullo schermo; è valida solo in modo testo (*modo* = Ø).

paginav (pagina di visualizzazione) sceglie quale pagina deve essere visualizzata, nello stesso modo di *paginaa*. La pagina di visualizzazione può essere diversa da quella attiva. *paginav* è valida solo in modo testo (*modo* = Ø); se si omette, si assume *paginaa*.

Se sono validi tutti i parametri, si memorizza il nuovo modo dello schermo ed il precedente viene cancellato. Il colore di primo piano è impostato al bianco ed i colori di fondo e dei bordi sono impostati al nero.

Se il nuovo modo dello schermo corrisponde al precedente, non cambia nulla.

Se il modo è testo e sono specificate solo *paginaa* e *paginav*, l'effetto è quello di modificare le pagine di visualizzazione per esaminarle.
Inizialmente sia la pagina attiva che quella di visualizzazione assumono per difetto Ø. Manipolando queste pagine, è possibile visualizzare una pagina mentre se ne costruisce un'altra. Poi è possibile passare alle pagine di visualizzazione istantaneamente.

SCREEN (Specifica)

Se si mischiano i grafici al testo nel modo grafici di 40 o 80 colonne e non si utilizza una tastiera americana fare riferimento al comando GRAFTABL nel manuale DOS per ulteriori informazioni sul supporto di caratteri addizionali ed altre tastiere da parte del video grafico a colori.

> **Nota:** esiste un solo cursore condiviso tra tutte le pagine. Se si passa da una pagina attiva all'altra, si deve salvare la posizione del cursore sulla pagina attiva attuale (usando POS (0) e CSRLIN), prima di passare ad un'altra pagina attiva. Poi, quando si ritorna alla pagina originale, si può ripristinare la posizione del cursore con la specifica LOCATE.

Può essere omesso qualsiasi parametro. I parametri omessi, eccetto *paginav,* assumono il vecchio valore.

I valori immessi che non sono compresi nei limiti indicati, producono un errore di **Illegal function call.** Vengono conservati i valori precedenti.

Esempio: Questo esempio sceglie il modo testo con il colore e imposta la pagina attiva e di visualizzazione a 0.

```
10 SCREEN 0,1,0,0
```

In questo esempio modo e colore rimangono invariati. La pagina attiva è impostata a 1 e quella di visualizzazione a 2.

```
10 SCREEN ,,1,2
```

SCREEN (Specifica)

Questo esempio permette di passare al modo grafici ad alta risoluzione;

```
1Ø SCREEN 2,,Ø,Ø
```

Questo esempio attiva il colore per i grafici a risoluzione media;

```
1Ø SCREEN 1,Ø
```

Questo esempio disattiva il colore per i grafici a risoluzione media.

```
5Ø SCREEN ,1
```

SGN (Funzione)

Scopo: Fornire il segno di *x*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v* = SGN*(x)*

Note: *x* è un'espressione numerica.

SGN*(x)* è la funzione matematica del segno:

- Se *x* è positiva, SGN*(x)* fornisce 1.
- Se *x* è zero, SGN*(x)* fornisce Ø.
- Se *x* è negativa, SGN*(x)* fornisce –1.

Esempio: Questo esempio salta a 1ØØ se X è negativa, a 2ØØ se X è zero e a 3ØØ se X è positiva.

```
ON SGN(X)+2 GOTO 1ØØ,2ØØ,3ØØ
```

SHELL (Specifica)

Scopo: Ricaricare ed eseguire un altro file programma (file con estensioni .COM, .BAT, e .EXE). Ogni programma eseguito sotto il BASIC riguarda una elaborazione secondaria. Quando questa finisce la sua esecuzione il controllo ritorna al programma BASIC principale alla specifica successiva alla specifica SHELL.

(SHELL non è disponibile per BASIC di livello antecedente al 3.Ø).

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	

Formato: SHELL [*string com*]

string com è un'espressione a stringa contenente il nome del programma che deve essere eseguito e facoltativamente ogni parametro che deve essere inviato alla elaborazione secondaria (dipendente).

Le elaborazioni secondarie sono eseguite da SHELL caricando ed elaborando una copia di COMMAND. COM con l'interruttore /C. Usando COMMAND in questo modo, ogni parametro viene correttamente passato ai blocchi di controllo dei file assunti per difetto. L'immissione e l'emissione standard possono essere reindirizzate ed i comandi incorporati come DIR, PATH e SORT possono essere eseguiti. Se SHELL viene immesso senza *string com,* una copia di COMMAND. COM è caricata, appare una richiesta DOS, e può essere immesso qualsiasi comando valido per il DOS (DIR, COPY, SORT, ecc.). Per ritornare al BASIC è sufficiente battere la parola EXIT. Mediante la specifica SHELL è possibile anche richiamare file di riserva.

SHELL (Specifica)

Per ritornare al programma principale BASIC, il proprio file di riserva deve contenere EXIT come ultima specifica.

Per eseguire elaborazioni secondarie usando la specifica SHELL è necessario conoscere e seguire alcune particolari procedure e regole. In caso contrario, l'applicazione che si vuole eseguire può non riuscire o dare risultati imprevedibili.

Per garantirsi un appropriato ritorno al BASIC, e nel modo schermo desiderato, è necessario effettuare una delle due seguenti operazioni:

- Usare BIOS Interrupt 10H, richiamo della funzione 15, per salvare il modo video in uso. Quando il programma secondario ritorna al DOS usare il richiamo della funzione 0 per ripristinare il modo video.

- Eseguire, dal programma BASIC, immediatamente dopo SHELL una specifica SCREEN seguita da una specifica CLS.

Prima di eseguire una specifica SHELL, il BASIC, salva ogni vettore di interruzione usato; questo non garantisce tuttavia che ogni interruzione usata dal proprio sottoprogramma, ma non dal BASIC, venga ripristinata. Assicurarsi quindi di salvare ogni vettore di interruzione che il proprio sottoprogramma usa al fine di prescrivere l'appropriata interfaccia al DOS. Alcune opzioni devono essere lasciate integre per assicurare che siano esattamente come il DOS e il BASIC si aspetta che siano. Queste opzioni sono l'8259, l'8253, l'8237, l'8255, e l'8250. Ulteriori descrizioni su queste opzioni possono essere trovate nel manuale tecnico del PC IBM.

SHELL (Specifica)

Una elaborazione secondaria che modifica alcuni dei file aperti dall'applicazione BASIC, può avere dei risultati imprevedibili. Se questi file devono essere aggiornati assicurarsi di chiuderli dall'applicazione BASIC. Prima di eseguire SHELL tener presente che i file aperti a seguito di un reindirizzamento di immissione ed emissione standard fanno parte di una OPEN e che non possono essere modificati da una elaborazione SHELL.

Prima di eseguire una specifica SHELL, il BASIC libera ogni parte di memoria non in uso a meno che BASIC non sia stato richiamato mediante /M:. In questo caso BASIC assume che un sottoprogramma in linguaggio macchina verrà caricato subito dopo il segmento di dati BASIC. Questo per prevenire una compressione dell'area di lavorazione BASIC prima di eseguire una SHELL e conseguentemente di evitare usando /M: il verificarsi di un errore **Out of memory.**

Il metodo migliore per una esecuzione di applicazioni che usano una specifica SHELL è quello di caricare il sottoprogramma in linguaggio macchina prima di eseguire il BASIC. Questo comporta l'inserimento di particolari codici nel sottoprogramma che una volta richiamati dal DOS permettano di terminare l'elaborazione e di restare residenti (INT27H). Per ulteriori informazioni far riferimento all'Appendice B "Sottoprogrammi in linguaggio macchina".

Qualsiasi sottoprogramma eseguito da una specifica SHELL non dovrebbe mai aver termine e rimanere residente. Questo potrebbe non lasciare sufficiente spazio al BASIC per ripristinare la propria area di lavoro. Tutti i file vengono chiusi, viene stampato il messaggio di errore **Can't continue after SHELL** ed il BASIC và dal DOS. Il BASIC rimane in memoria mentre l'elaborazione secondaria è in esecuzione. Quando questa finisce, il BASIC continua.

SHELL (Specifica)

Un nome di programma in una *string com* può avere qualsiasi estensione scelta. Se non viene fornita nessuna estensione il DOS ricerca una estensione .COM, poi un'estensione .EXE ed infine una estensione .BAT. Se il nome del file non viene localizzato durante questa ricerca, COMMAND emetterà un messaggio d'errore.

Qualsiasi testo separato da un nome di programma fornito nella *string com* con almeno uno spazio in bianco, viene processato come un parametro di programma. Quando il BASIC è in esecuzione, il suo ambiente viene ereditato dal DOS.

> **Nota:** Per ulteriori informazioni sull'ambiente fare riferimento alla sezione "Descrizioni dettagliate sui comandi del DOS avanzato" del manuale DOS.

Se si tenta di eseguire il BASIC come una elaborazione secondaria si verificherà un errore. Dopo che questo messaggio è stato visualizzato il controllo ritorna al BASIC primario.

Esempio: Questo esempio crea un file, và al programma di utilità DOS SORT e successivamente ritorna al BASIC.

```
10 OPEN "SORTIN.DAT" FOR OUTPUT AS #1
20 'scrive i dati da selezionare
   .
   .
   .
100 CLOSE 1
110 SHELL "SORT <SORTIN.DAT >SORTOUT.DAT"
120 OPEN "SORTOUT.DAT" FOR INPUT AS #1
130 'elabora i dati selezionati
```

SHELL (Specifica)

Il seguente esempio visualizza un indirizzario su disco dal BASIC.

```
SHELL
A>DIR   (type DIR command at DOS prompt)
A>EXIT  (type EXIT to return to BASIC)
```

Lo stesso risultato può essere ottenuto con:

```
SHELL "DIR".
```

SIN (Funzione)

Scopo: Calcolare la funzione trigonometrica del seno.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: $v = \text{SIN}(x)$

Note: *x* è un angolo in radianti.

Se si desidera convertire i gradi in radianti, moltiplicare per PI/180 dove PI=3.141593. In BASIC 2.0 e successivi livelli questo calcolo può essere eseguito a precisione doppia specificando /D sulla riga di comando BASIC, quando il BASIC è inizialmente caricato. Vedere la sezione "Opzioni nel Comando BASIC" nel manuale BASIC.

Esempio: Questo esempio calcola il seno di 90 gradi, dopo aver prima convertito i gradi in radianti.

```
10 PI=3.141593
20 DEGREES = 90
30 RADIANS=DEGREES * PI/180 ' PI/2
40 PRINT SIN(RADIANS)
RUN
 1
```

SOUND (Specifica)

Scopo: Generare il suono attraverso l'altoparlante.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: SOUND *freq, durata*

Note:

freq è la frequenza desiderata in Hertz (cicli per secondo). Deve essere un'espressione numerica compresa tra 37 e 32767.

durata è la durata desiderata in battiti dell'orologio. Il battito si verifica 18.2 volte per secondo. *durata* deve essere un'espressione numerica compresa tra .Ø27 e 65535 in BASIC su disco, e tra .ØØ15 e 65535 in BASIC avanzato.

Quando la specifica SOUND produce un suono, il programma continua l'esecuzione fino ad un'altra SOUND. Se la *durata* della nuova specifica SOUND è zero, viene disattivata quella attualmente in esecuzione. Altrimenti, il programma attende il completamento del primo suono prima di eseguire la nuova specifica SOUND.

Se si utilizza il BASIC avanzato, è possibile immettere i suoni nella memoria di transito in modo che l'esecuzione non si arresti alla nuova specifica SOUND. Vedere il comando MB descritto nella specifica PLAY.

Se non vi sono specifiche SOUND in esecuzione, SOUND *x*,Ø non ha effetto.

SOUND (Specifica)

La nota di accordo, LA, ha una frequenza di 440 Hz. La seguente tabella mette in correlazione le note con le rispettive frequenze per due ottave su entrambi i lati del DO (C) centrale.

Nota	Frequenza	Nota	Frequenza
C(DO)	130.810	C(DO)	523.250
D(RE)	146.830	D(RE)	587.330
E(MI)	164.810	E(MI)	659.260
F(FA)	174.610	F(FA)	698.460
G(SOL)	196.000	G(SOL)	783.990
A(LA)	220.000	A(LA)	880.000
B(SI)	246.940	B(SI)	987.770
C(DO)*	261.630	C(DO)	1046.500
D(RE)	293.660	D(RE)	1174.700
E(MI)	329.630	E(MI)	1318.500
F(FA)	349.230	F(FA)	1396.900
G(SOL)	392.000	G(SOL)	1568.000
A(LA)	440.000	A(LA)	1760.000
B(SI)	493.880	B(SI)	1975.500

*DO centrale (C). Le note più alte (o più basse) possono essere approssimate raddoppiando (o dimezzando) la frequenza della nota corrispondente nella precedente (successiva) ottava.

Per creare periodi di silenzio, usare SOUND 32767, *durata*.

La durata di un battito può essere calcolata dai battiti per minuto, dividendoli per 1092 (il numero dei battiti dell'orologio in un minuto).

SOUND (Specifica)

La seguente tabella mostra i tempi tipici in termini di battiti dell'orologio.

	Tempo	Battiti/ minuto	Tic/ battito
molto lento	Larghissimo		
	Largo	40-60	27.3-18.2
	Larghetto	60-66	18.2-16.55
	Grave		
	Lento		
	Adagio	66-76	16.55-14.37
lento	Adagietto		
	Andante	76-108	14.37-10.11
medio	Andantino		
	Moderato	108-120	10.11-9.1
veloce	Allegretto		
	Allegro	120-168	9.1-6.5
	Vivave		
	Veloce		
	Presto	168-208	6.5-5.25
molto veloce	Prestissimo		

Esempio: il seguente programma crea un suono che va dall'alto verso il basso lungo la scala musicale.

```
10 FOR I=440 TO 1000 STEP 5
20 SOUND I,0.5
30 NEXT
40 FOR I=1000 TO 440 STEP -5
50 SOUND I,7 0.5
60 NEXT
```

SPACE$ (Funzione)

Scopo: Fornire una stringa di *n* spazi.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v$*=SPACE$*(n)*

Note: *n* deve essere compreso tra Ø e 255.

Vedere anche la funzione SPC.

Esempio: Questo esempio usa la funzione SPACE$ per stampare ogni numero I su una riga preceduto da I spazi. Viene inserito un ulteriore spazio perché il BASIC pone uno spazio davanti ai numeri positivi.

```
1Ø FOR I = 1 TO 5
2Ø X$ = SPACE$(I)
3Ø PRINT X$;I
4Ø NEXT I
RUN
 1
  2
   3
    4
     5
```

SPC (Funzione)

Scopo: Saltare *n* spazi in una specifica PRINT.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: PRINT SPC *(n)*

Note: *n* deve essere compreso tra Ø e 255.

Se *n* è maggiore dell'ampiezza definita per l'unità, il valore usato è *n* MOD *ampiezza*. SPC può essere usata solo per le specifiche PRINT, LPRINT e PRINT #.

Se la funzione SPC si trova alla fine della lista di dati il BASIC non aggiunge un ritorno a margine, come nel caso in cui dopo la funzione SPC si trovi un punto.e virgola.

Vedere anche la funzione SPACE$.

Esempio: Questo esempio stampa OVER e THERE separati da 15 spazi.

```
PRINT "OVER" SPC(15) "THERE"
OVER               THERE
```

SQR (Funzione)

Scopo: Fornire la radice quadrata di *x*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v* = SQR(*x*)

Note: *x* deve essere maggiore di o uguale a zero.

In BASIC 2.Ø e successivi livelli questo calcolo può essere ottenuto a precisione doppia specificando /D sulla riga di comando BASIC quando il BASIC viene inizialmente caricato. Fare riferimento alla sezione "Opzioni nel comando BASIC" nel manuale BASIC.

Esempio: Questo esempio calcola la radice quadrata dei numeri 1Ø, 15, 2Ø e 25.

```
1Ø FOR X = 1Ø TO 25 STEP 5
2Ø PRINT X, SQR(X)
3Ø NEXT
RUN
 1Ø            3.162278
 15            3.872984
 2Ø            4.472136
 25            5
```

STICK (Funzione)

Scopo: Fornire le coordinate x e y delle due barre di comando per i giochi.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v* = STICK *(n)*

Note: *n* è un'espressione numerica compresa tra 0 e 3 che interessa il risultato nel modo seguente:

- **0** fornisce la coordinata x per la barra A.
- **1** fornisce la coordinata y per la barra A.
- **2** fornisce la coordinata x per la barra B.
- **3** fornisce la coordinata y per la barra B.

Nota: STICK(0) reperisce i quattro valori per le coordinate e fornisce il valore per STICK(0). STICK(1), STICK(2) e STICK(3) prendono i valori precedentemente reperiti da STICK(0).

La gamma dei valori per x e y dipende dalle particolari barre di comando per giochi.

STICK
(Funzione)

Esempio: Questo programma stampa 100 esempi delle coordinate della barra B.

```
10 PRINT "Joystick B"
20 PRINT "x coordinate","y coordinate"
30 for J=1 TO 100
40 TEMP=STICK(0)
50 X=STICK(2): Y=STICK(3)
60 PRINT X,Y
70 NEXT
```

STOP
(Specifica)

Scopo: Terminare l'esecuzione del programma e ritornare a livello di comando.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: STOP

Note: Le specifiche STOP possono essere utilizzate ovunque in un programma per terminare l'esecuzione. Quando il BASIC incontra una specifica STOP, visualizza il seguente messaggio:

```
Break in nnnnn
```

dove nnnnn è il numero della riga in cui si trova STOP.

A differenza della specifica END, STOP non chiude i file.

Il BASIC ritorna sempre a livello di comando dopo aver eseguito una specifica STOP. E' possibile riprendere l'esecuzione del programma immettendo un comando CONT. A riguardo vedere il comando CONT.

STOP
(Specifica)

Esempio: Questo esempio calcola il valore di TEMP, poi si arresta. Durante l'arresto del programma, è possibile controllare il valore di TEMP. Poi, è possibile utilizzare CONT per riprendere l'esecuzione del programma a riga 40.

```
10 INPUT A, B
20 TEMP= A*B
30 STOP
40 FINAL = TEMP+200: PRINT FINAL
RUN
? 26, 2.1
Break in 30
PRINT TEMP
 54.6
CONT
 254.6
```

STR$ (Funzione)

Scopo: Fornire una rappresentazione a stringa del valore di *x*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v$*=STR$(*x*)

Note: *x* è una qualsiasi espressione numerica.

Se x è positiva, la stringa fornita da STR$ contiene uno spazio iniziale (riservato al segno più). Per esempio:

```
? STR$(321); LEN(STR$(321))
 321  4
```

La funzione VAL è complementare a STR$. Vedere la funzione VAL.

Esempio: Questo esempio salta a diverse sezioni del programma in base al numero di cifre di un numero immesso. Le cifre del numero sono contate usando STR$ per convertire il numero in una stringa; poi il programma salta in base alla lunghezza della stringa.

```
10 INPUT "BATTERE UN NUMERO";N
20 ON LEN(STR$(N))-1 GOSUB 30,100,200,300
```

STRIG
(Specifica e funzione)

Scopo: Fornire lo stato dei tasti sulle barre di comando per i giochi.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: Come specifica:

STRIG ON

STRIG OFF

Come funzione:

v = STRIG *(n)*

Note: *n* è un'espressione numerica compresa tra 0 e 3. Interessa il valore fornito dalla funzione nel seguente modo:

0 Fornisce –1 se il tasto A1 è stato premuto dopo l'ultimo richiamo della funzione STRIG(0); in caso contrario fornisce 0.

1 Fornisce –1 se si sta premendo il tasto A1; in caso contrario fornisce 0.

2 Fornisce –1 se il tasto B1 è stato premuto dopo l'ultimo richiamo della funzione STRIG(2); in caso contrario fornisce 0.

3 Fornisce –1 se si sta premendo il tasto B1; in caso contrario fornisce 0.

STRIG (Specifica e funzione)

Nel BASIC avanzato e nel compilatore BASIC, è possibile leggere quattro tasti dalle barre di comando per i giochi. Gli ulteriori valori per *n* sono:

4 Fornisce –1 se è stato premuto il tasto A2 dopo l'ultimo richiamo della funzione STRIG(4); in caso contrario fornisce Ø.

5 Fornisce –1 se si sta premendo il tasto A2; in caso contrario fornisce Ø.

6 Fornisce –1 se è stato premuto il tasto B2 dopo l'ultimo richiamo della funzione STRIG(6); in caso contrario fornisce Ø.

7 Fornisce –1 se si sta premendo il tasto B2; in caso contrario fornisce Ø.

E' necessario eseguire STRIG ON prima di richiamare una funzione STRIG*(n)*. Dopo STRIG ON, ogni volta che il programma avvia una nuova specifica, il BASIC controlla se è stato premuto un tasto.

Se STRIG è OFF, non si esegue alcuna prova.

Vedere anche la specifica STRIG*(n)* per miglioramenti alla funzione STRIG nel BASIC avanzato.

STRIG(n) (Specifica)

Scopo: Abilitare e disabilitare l'intercettazione dei tasti sulle barre di comando per i giochi.

Versioni:	Cassetta	Disco	Avanzato	Compilatore
			***	(**)

Formato: STRIG*(n)* ON
STRIG*(n)* OFF
STRIG*(n)* STOP

Note: *n* può essere 0, 2, 4, o 6 ed indica il tasto da intercettare:

0	tasto A1
2	tasto B1
4	tasto A2
6	tasto B2

E' necessario eseguire STRIG*(n)* ON per consentire l'intercettazione tramite la specifica ON STRIG*(n)*. Dopo STRIG*(n)* ON, ogni volta che il programma avvia una nuova specifica, il BASIC controlla se è stato premuto il tasto specificato.

Se si esegue STRIG*(n)* OFF, non avviene alcuna intercettazione o prova. Anche se si preme il tasto, l'azione non viene registrata.

Se si esegue una specifica STRIG*(n)* STOP, non avviene alcuna intercettazione. Tuttavia, se si preme il tasto, l'azione viene registrata in modo che si verifichi un'immediata intercettazione quando si esegue STRIG*(n)* ON.

STRING$ (Funzione)

Scopo: Fornire una stringa di lunghezza *n* i cui caratteri hanno il codice ASCII *m*, oppure il primo carattere di *x$*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v$* = STRING$*(n,m)*

v$* = STRING$(n,x$)*

Note: *n,m* sono compresi tra 0 e 255.

x$ è un'espressione a stringa.

Esempio: Questo esempio ripete un valore ASCII di 45 per stampare una stringa di trattini.

```
10 X$ = STRING$(10,45)
20 PRINT X$ "MONTHLY REPORT" X$
RUN
----------MONTHLY REPORT----------
```

Questo esempio ripete il primo carattere della stringa "ABCD".

```
10 X$="ABCD"
20 Y$+STRING$(10,X$)
30 PRINT Y$
RUN
AAAAAAAAAA
```

SWAP (Specifica)

Scopo: Scambiare i valori di due variabili.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: SWAP *variabile1, variabile2*

Note: *variabile1, variabile2,*
sono i nomi delle due variabili o elementi di schiera.

E' possibile scambiare qualsiasi tipo di variabile (intero, precisione singola o doppia, stringa), ma le due variabili devono essere dello stesso tipo, altrimenti si verifica un errore di **Type mismatch.**

Esempio: Dopo l'esecuzione della riga 30, A$ ha il valore "ALL" e B$ il valore "ONE".

```
10 A$=" ONE " : B$=" ALL " : C$="FOR"
20 PRINT A$;C$;B$
30 SWAP A$,B$
40 PRINT A$;C$;B$
RUN
 ONE FOR ALL
 ALL FOR ONE
```

SYSTEM (Comando)

Scopo: Uscire dal BASIC e ritornare al DOS.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	***

Formato: SYSTEM

Note: SYSTEM chiude tutti i file prima di ritornare al DOS. Il programma BASIC viene perso.

Se si entra nel BASIC tramite un file di lavoro dal DOS, il comando SYSTEM ritorna al file di lavoro che continua l'esecuzione dal punto in cui si era arrestato.

TAB (Funzione)

Scopo: Eseguire delle tabulazioni fino alla posizione *n*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: PRINT TAB*(n)*

Note: *n* deve essere compreso tra 1 e 255.

Se l'attuale posizione di stampa si trova già oltre lo spazio *n*, TAB va alla posizione *n* della riga successiva. Spazio 1 è la posizione più a sinistra, mentre quella più a destra è l'ampiezza (WIDTH) definita.

E' possibile usare TAB solo nelle specifiche PRINT, LPRINT e PRINT #.

Se la funzione TAB si trova alla fine della lista di dati, il BASIC non aggiunge un ritorno a margine, come nel caso in cui la funzione TAB sia seguita da un punto e virgola.

Esempio: Nel seguente esempio si usa TAB per allineare in colonne le informazioni sullo schermo.

```
10 PRINT "NAME" TAB(25) "AMOUNT" : PRINT
20 READ A$,B$
30 PRINT A$ TAB(25) B$
40 DATA "L. M. JACOBS","$25.00"
RUN
NAME                    AMOUNT

L. M. JACOBS            $25.00
```

TAN (Funzione)

Scopo: Calcolare la tangente trigonometrica di *x*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v* = TAN(*x*)

Note:

x è l'angolo in radianti. Per convertire i gradi in radianti, moltiplicare per PI/180, dove PI=3,141593.

Nel BASIC 2.0 e successivi livelli, questo calcolo può essere eseguito a precisione doppia specificando /D nella riga di comando BASIC quando il BASIC viene inizialmente caricato. Vedere le sezioni "Opzioni nel comando BASIC" nel manuale BASIC.

Esempio: Questo esempio calcola la tangente di 45 gradi.

```
10 PI=3.141593
20 DEGREES=45
30 PRINT TAN(DEGREES*PI/180)
RUN
 1
```

TIME$
(Variabile e specifica)

Scopo: Imposta e fornisce l'ora.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	***

Formato: Come variabile:

*v$* = TIME$

Come specifica:

TIME$ = *x$*

Note: **Per la variabile (v$ = TIME$):**

Viene fornita l'ora attuale come stringa di 8 caratteri. La stringa è nella forma *hh:mm:ss,* dove *hh,* è l'ora (da ØØ a 23), *mm* sono i minuti (da ØØ a 59) e *ss* sono i secondi (da ØØ a 59). L'ora può essere impostata dal DOS prima di richiamare il BASIC.

Per la specifica (TIME$ = x$):

Viene impostata l'ora. *x$* è una espressione a stringa che indica l'ora da impostare. *x$* può essere indicata in una delle seguenti forme:

hh Imposta l'ora da Ø a 23. Per i minuti ed i secondi si assume ØØ.

hh:mm Imposta l'ora ed i minuti. Questi devono essere compresi tra Ø e 59. Per i secondi si assume ØØ.

hh:mm:ss Imposta l'ora, i minuti ed i secondi. I secondi devono essere compresi tra Ø e 59.

TIME$
(Variabile e specifica)

Lo zero iniziale può essere omesso nei precedenti valori, ma si deve includere almeno una cifra. Ad esempio, se si vuole impostare l'ora a mezzanotte e trenta, si può immettere

TIME$ = "Ø:3Ø"

ma non

TIME$ = ":3Ø"

Se uno dei valori non è compreso nei limiti, si verifica un errore. Viene mantenuta l'ora precedente. Se *x$* non è una stringa valida, si verifica un errore di **Type mismatch.**

Esempio: Il seguente programma visualizza costantemente l'ora sullo schermo.

```
1Ø CLS
2Ø LOCATE 1Ø,15
3Ø PRINT TIME$
4Ø GOTO 2Ø
```

TIMER (Funzione)

Scopo: Fornire un numero in precisione singola che rappresenti il numero dei secondi passati dalla mezzanotte o dall'ultimo ripristino del sistema. (BASIC 2.Ø e successivi livelli).

Versioni:

Cassetta	Disco	Avanzato	Compilatore
	***	***	

Formato: *v*=TIMER

Note: Le frazioni di secondo vengono calcolate col massimo dell'approssimazione. TIMER è una funzione di sola lettura.

Esempio: Questo esempio mostra come TIMER venga ripristinato dopo la mezzanotte. A seconda del sistema che si usa i valori possono essere leggermente differenti.

```
1Ø TIME$="23:59:59"
2Ø FOR I=1 TO 2Ø
3Ø PRINT "TIME$= ";TIME$;"TIMER=";TIMER
4Ø NEXT
RUN

TIME$= 23:59:59       TIMER= 86399.Ø6
TIME$= 23.59:59       TIMER= 86399.11
TIME$= 23.59.59       TIMER= 86399.18
                        .
                        .
                        .
TIME$= 24:ØØ:ØØ       TIMER= Ø
TIME$= ØØ:ØØ:ØØ       TIMER= .Ø5
TIME$= ØØ:ØØ:ØØ       TIMER= .16
TIME$= ØØ:ØØ:ØØ       TIMER= .21
```

TRON e TROFF (Comandi)

Scopo: Eseguire la traccia dell'esecuzione delle specifiche di programma.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: TRON

TROFF

Note: Come aiuto per la rilevazione degli errori, il comando TRON (che può essere immesso in modo indiretto) consente l'esecuzione di una traccia che stampa ogni numero di riga del programma al momento dell'esecuzione. I numeri appaiono racchiusi tra parentesi quadre. La funzione di traccia viene disattivata dal comando TROFF.

Esempio: Questo esempio usa TRON e TROFF per la traccia dell'esecuzione di un'iterazione. I numeri tra parentesi quadre sono i numeri di riga; quelli fuori parentesi alla fine di ogni riga sono i valori J, K e L che vengono stampati dal programma.

```
10 K=10
20 FOR I=1 TO 2
30 L=K + 10
40 PRINT J;K;L
50 K=K+10
60 NEXT
70 END
TRON
RUN
 [10] [20] [30] [40] 0 10 20
 [50] [60] [30] [40] 0 20 30
 [50] [60] [70]
TROFF
```

USR (Funzione)

Scopo: Richiamare il sottoprogramma in linguaggio macchina indicato con l'argomento *arg*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: *v* = USR [*n*] (*arg*)

Note:

n è compresa tra Ø e 9 e corrisponde alla cifra fornita con la specifica DEF USR per il programma desiderato. Se si omette *n*, si assume USRØ. Fare riferimento alla specifica DEF USR.

arg è un'espressione numerica o variabile a stringa che sarà l'argomento per il sottoprogramma in linguaggio macchina.

Se il sottoprogramma non richiede un argomento, *arg* deve essere tuttavia fornito come un argomento fittizio.

La specifica CALL è un altro modo per richiamare un sottoprogramma in linguaggio macchina. Per informazioni sull'uso dei sottoprogrammi in linguaggio macchina, vedere l'Appendice B.

Quando la funzione USR è richiamata, il registro AL contiene un valore che specifica il tipo di argomento fornito. Il valore di AL può essere uno dei seguenti:

Valore in AL	Tipo di argomento
2	intero di 2 byte (complemento a due)
3	stringa
4	numero a precisione singola
8	numero a precisione doppia

Se l'argomento è una stringa, il registro DX punta al descrittore a stringa di 3 byte. Per informazioni al riguardo vedere l'Appendice B "Sottoprogrammi in linguaggio macchina".

Se l'argomento è un numero e non una stringa il valore dell'argomento è messo nel FAC (Floating point accumulator), che è un'area di 8 byte situata nell'area di dati BASIC. In questo caso il registro BX contiene la posizione relativa, nell'area di dati BASIC, al quinto degli otto byte del FAC. Per i seguenti esempi assumiamo che il valore di FAC sia da 49F a 4A6 esadecimali e che quindi BX contenga 4A3:

Se l'argomento è un numero intero:

- L'esadecimale 4A4 contiene gli otto bit superiori dell'argomento.

- L'esadecimale 4A3 contiene gli otto bit inferiori dell'argomento.

Se l'argomento è un numero a precisione singola:

- L'esadecimale 4A6 contiene l'esponente meno 128 e il punto binario è alla sinistra del più significativo bit della mantissa.

USR (Funzione)

L'esadecimale 4A5 contiene i 7 bit più alti della mantissa con l'1 di testa soppresso (sottinteso). Il bit 7 è il segno del numero (Ø = positivo; 1 = negativo).

- L'esadecimale 4A4 contiene gli 8 bit intermedi della mantissa.

- L'esadecimale 4A3 contiene gli 8 bit superiori della mantissa.

Se l'argomento è a precisione doppia

- Gli esadecimali da 4A3 a 4A6 sono gli stessi descritti nel paragrafo precedente circa il numero a precisione singola.

- Gli esadecimali da 49F e 4A2 contengono 4 ulteriori byte della mantissa (l'esadecimale 49F contiene gli 8 bit inferiori).

Di solito il valore fornito dalla funzione USR è dello stesso tipo dell'argomento che gli è stato passato (intero, a precisione singola o doppia). Tuttavia un argomento numerico della funzione, senza riguardo al suo tipo, può essere forzato in un valore intero mediante il sottoprogramma FRCINT che genera il numero intero equivalente all'argomento del registro BX.

Se il valore che la funzione restituisce deve essere un intero, posizionare il valore risultante nel registro BX. Quindi richiamare MAKINT prima del ritorno dell'intersegmento. Questo restituisce il valore al BASIC immettendolo nel FAC.

Esempio: Il metodo per accedere a FRCINT e a MAKINT è spiegato nel seguente esempio:

```
10 DEFINT A-Z
20 OPTION BASE 1
30 X=0: Y=0: Z=0:
40 DIM ARRAY (&H14)
50 Z = VARPTR(ARRAY(1))
60 BLOAD"SUBRT.COM",Z
70 DEF USR0= Z
80 X=5
90 Y=USR0(X)
100 PRINT Y
```

Il seguente sottoprogramma in linguaggio macchina è stato caricato in una schiera di interi. Questo sottoprogramma raddoppia l'argomento passato e restituisce un risultato intero.

USR (Funzione)

```
          PAGE,132        ;
          TITLE SUBRT.ASM
;                         BLOAD header
AAA       SEGMENT PARA PUBLIC 'CODE'
          DB      ØFDH
          DW      Ø,Ø
          DW      TRAILER-HEADER
AAA       ENDS
;
;
RSEG        SEGMENT AT ØF6ØØH ;base of BASIC ROM
            ORG   3       ;offset to force integer
FRCINT      LABEL FAR
            ORG   7       ;offset to make integer
MAKINT      LABEL FAR
RSEG        ENDS
;
;
CSEG        SEGMENT BYTE PUBLIC 'CODE'
HEADER      EQU    $
USRPRG      PROC  FAR     ;entry point
;call FRCINT and force argument
;in FAC into [BX]
            DB    9AH
            DW    FRCINT ;offset
            DW    ØF6ØØH ;segment id
            ADD   BX,BX  ; [BX] = [BX] *2
;call MAKINT and put integer result
;in [BX] into FAC
            DB    9AH
            DW    MAKINT ;offset
            DW    ØF6ØØH ;segment id
            RET          ;intersegment return
USRPRG      ENDP
TRAILER     EQU   $
CSEG        ENDS
            END
```

USR (Funzione)

Nota: FRCIN e MAKINT eseguono ritorni di intersegmenti. Assicurarsi che il richiamo delle funzioni FRCIN e MAKINT sia definito dalla procedura FAR.

VAL (Funzione)

Scopo: Fornire il valore numerico della stringa *x$*.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v*=VAL*(x$)*

Note: *x$* è un'espressione a stringa.

La funzione VAL elimina gli spazi in bianco, le tabulazioni e gli avanzamenti di riga dalla stringa argomento in modo da determinare il risultato. Ad esempio:

```
VAL("  -3")
```

fornisce -3.

Se il primo carattere di *x$* non è numerico, VAL*(x$)* fornisce zero.

Vedere la funzione STR$ per la conversione da numerico a stringa.

Esempio: In questo esempio, VAL si usa per estrarre il numero di casa dall'indirizzo.

```
PRINT VAL("3408 SHERWOOD BLVD.")
 3408
```

VARPTR (Funzione)

Scopo: Fornire l'indirizzo in memoria della variabile.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: *v*=VARPTR(*variabile*)

Note: *variabile* è il nome di una variabile numerica o a stringa o dell'elemento di schiera del programma. E' necessario assegnare un valore alla *variabile* prima di richiamare VARPTR, altrimenti si verifica un errore di **Illegal function call.**

VARPTR fornisce un indirizzo compreso tra 0 e 65535. Questo numero è la posizione relativa nel segmento di dati del BASIC del primo byte del dato identificato con *variabile*. Il formato di questo dato è descritto nell'Appendice B "Informazioni sulla memoria" del manuale BASIC.

> **Nota:** è necessario assegnare tutte le variabili semplici prima di richiamare VARPTR per una schiera, poiché gli indirizzi delle schiere cambiano ogni volta che si assegna una nuova variabile semplice.

VARPTR è usato solitamente per ottenere l'indirizzo di una variabile o di una schiera da passare a sottoprogrammi in linguaggio macchina.

VARPTR (Funzione)

Esempio: Questo esempio usa VARPTR per estrarre i dati da una variabile. A riga 30, P estrae l'indirizzo dei dati. Gli interi sono memorizzati in due byte, il primo dei quali è quello meno significativo. Il valore reale memorizzato a posizione P è calcolato alla riga 40. I byte vengono letti con la funzione PEEK ed il secondo byte è moltiplicato per 256 perché contiene i bit più alti.

```
10 DEFINT A-Z
20 DATA1=500
30 P=VARPTR(DATA1)
40 V=PEEK(P) + 256*PEEK(P+1)
50 PRINT V
```

VARPTR$ (Funzione)

Scopo: Fornire l'indirizzo di una variabile in memoria sottoforma di carattere. Viene utilizzata principalmente con PLAY e DRAW nei programmi che saranno compilati.

Versioni:	Cassetta	Disco	Avanzato	Compilatore
		***	***	***

Formato: *v$*=VARPTR$*(variabile)*

Note: *variabile* è il nome di una variabile esistente nel programma.

Nota: Tutte le variabili semplici devono essere assegnate prima di richiamare VARPTR$ per un elemento di schiera, poiché gli indirizzi delle schiere cambiano ogni volta che si assegna una nuova variabile semplice.

VARPTR$ fornisce una stringa di 3 byte nella forma:

Byte 0	**Byte 1**	**Byte 2**
Tipo	byte inferiore dell'indirizzo della variabile	byte superiore dell'indirizzo della variabile

tipo indica il tipo di variabile:

2 intero
3 stringa
4 precisione singola
8 precisione doppia

VARPTR$ (Funzione)

Il valore fornito corrisponde a:

```
CHR$(tipo)+MKI$(VARPTR(variabile))
```

Si può utilizzare VARPTR$ per indicare un nome di variabile nella stringa di comando per PLAY o DRAW. Ad esempio:

Metodo

```
PLAY "XA$;"
PLAY "O=I;"
```

Metodo alternativo

```
PLAY "X"+VARPTR$(A$)
PLAY "O="+VARPTR$(I)
```

VIEW (Specifica)

Scopo: Definire una porzione di schermo chiamata quadrante tramite il quale viene visualizzato il contenuto delle finestre (window). (BASIC livello 2.Ø e successivi livelli).

Versioni: Cassetta Disco Avanzato Compilatore
***

Esclusivamente in modo Grafici.

Formato: VIEW [[SCREEN] [(*x1,y1*)—(*x2,y2*) [,[*colore*] [,[*limite*]]]]]

Note: *(x1,y1)—(x2,y2)*
sono le coordinate dei punti in alto a sinistra e in basso a destra del quadrante definito. Le coordinate devono trovarsi all'interno dei limiti dello schermo in uso altrimenti appare l'errore **Illegal function call.** Per ulteriori informazioni consultare le sezioni "Come specificare le coordinate" e "Modo Grafici" nel Capitolo 3 del manuale BASIC.

colore permette di colorare il quadrante definito. Se il parametro è omesso il quadrante non viene riempito. Colore è un'espressione intera che sceglie un attributo nella gamma di attributi disponibile per il modo schermo al momento in uso. A media risoluzione il colore è quello corrispondente alla tavolozza definita mediante la specifica COLOR.

VIEW (Specifica)

A media risoluzione gli attributi disponibili sono quattro (Ø-3); ad alta risoluzione due (Ø-1). Zero (Ø) è sempre l'attributo dello sfondo. L'attributo relativo allo sfondo assunto per difetto è sempre il massimo attributo per quel modo schermo: 3 a media risoluzione; 1 ad alta.

limite permette di tracciare una linea di bordatura intorno al quadrante (se c'è spazio disponibile). Se il parametro è omesso non viene tracciato alcun bordo. Può assumere i valori descritti per il parametro *colore*.

Notare che VIEW ordina le coppie di x e y in modo da porre per primi i valori più bassi. Per esempio:

```
VIEW (100, 100)-(5,5)
```

diventa

```
VIEW (5,5)-(100,100)
```

Altro esempio:

```
VIEW (310,100)-(200,150)
```

diventa

```
VIEW (200,100)-(310,150)
```

Sono permesse tutte le possibili combinazioni di x e y. L'unica restrizione è che x1 non può essere uguale a x2 e y1 non può essere uguale a y2. Il quadrante non può essere più grande dello schermo.

VIEW (Specifica)

Se l'attributo SCREEN viene omesso, tutti i punti tracciati sono relativi al quadrante. Cioè x1 e y1 vengono sommate rispettivamente alle coordinate x e y prima di tracciare il punto sullo schermo.
Per esempio se:

```
10 VIEW (10,10)-(200,100)
```

viene eseguita, il punto tracciato da PSET (Ø,Ø),3 si troverà sullo schermo nella posizione effettiva 1Ø,1Ø.

Se è presente l'attributo SCREEN tutti i punti tracciati sono assoluti e possono trovarsi all'interno o all'esterno dello schermo. Comunque solo i punti entro i limiti del quadrante saranno visibili. Per esempio se:

```
10 VIEW SCREEN (10,10)-(200,100)
```

viene eseguita, il punto tracciato da PSET (Ø,Ø),3 non apparirà sullo schermo perchè Ø,Ø è fuori dal quadrante. PSET (1Ø,1Ø),3 è all'interno dello schermo e traccerà un punto nell'angolo in alto a sinistra.

VIEW senza parametri definisce come quadrante l'intero schermo.

Si possono definire diversi quadranti ma solo uno per volta è attivo. RUN e SCREEN eliminano i quadranti.

VIEW usato con WINDOW permette di riprodurre in scala. Vedere il secondo esempio e la specifica WINDOW.

> **Nota:** Usando VIEW, per cancellare solo il quadrante in uso si impiega la specifica CLS. Per pulire l'intero schermo, prima usare VIEW per eliminare i quadranti poi usare CLS per cancellare lo schermo.

VIEW
(Specifica)

Se si usa CLS con i quadranti al termine dell'operazione il cursore non si troverà in alto a sinistra. Battere Ctrl-Ritcurs per pulire lo schermo e portare il cursore in alto a sinistra.

Esempio: Con il seguente esempio si definiscono quattro quadranti:

```
10 SCREEN 1:VIEW:CLS:KEY OFF
20 VIEW (1,1)-(151,91),,1
30 VIEW (165,1)-(315,91),,2
40 VIEW (1,105)-(151,195),,2
50 VIEW (165,105)-(315,195),,1
60 LOCATE 2,4:PRINT "Viewport 1"
70 LOCATE 2,25:PRINT "Viewport 2"
80 LOCATE 15,4:PRINT "Viewport 3"
90 LOCATE 15,25:PRINT "Viewport 4"
100 VIEW (1,1)-(151,91):GOSUB 1000
200 VIEW (165,1)-(315,91):GOSUB 2000
300 VIEW (1,105)-(151,195):GOSUB 3000
400 VIEW (165,105)-(315,195):GOSUB 4000
900 END
1000 CIRCLE (65,50),30,2
1010 'Traccia un cerchio nella prima finestra
1020 RETURN
2000 LINE (45,50)-(90,75),1,B
2010 'Traccia una figura nella seconda finestra
2020 RETURN
3000 FOR D=0 TO 360:DRAW "ta=d;nu20":NEXT
3010 'Traccia dei raggi nella terza finestra
3020 RETURN
4000 PSET(60,50),2:DRAW "e15;f15;130"
4010 'Traccia un triangolo nella quarta finestra
4020 RETURN
```

VIEW (Specifica)

Questo esempio illustra l'impiego di VIEW per riprodurre in scala.

```
10 KEY OFF:CLS:SCREEN 1,0:COLOR 0,0
20 WINDOW SCREEN(320,0)-(0,200)
30 GOTO 80
40 C=1
50 CIRCLE (160,100),60,C,,,5/18
60 CIRCLE (160,100),60,C,,,1
70 RETURN
80 GOSUB 460:FOR I=1 TO 1500:NEXT I: CLS
90 VIEW (1,1)-(160,90),,2:GOSUB 40
100 GOTO 100
```

WAIT (Specifica)

Scopo: Sospendere l'esecuzione del programma durante il controllo dello stato di una porta del sistema.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: WAIT *porta n* [,*m*]

Note:

porta è il numero della porta compreso tra Ø e 65535.

n,m sono espressioni intere comprese tra Ø e 255.

Vedere il manuale di riferimento tecnico del PC IBM per una descrizione degli indirizzi di porta validi.

WAIT sospende l'esecuzione fino a quando uno specifico punto di immissione della macchina non sviluppi una determinata struttura di bit.

Sui dati letti si esegue la funzione XOR con l'espressione intera *m* e la funzione AND con *n*. Se il risultato è zero, il BASIC torna indietro e legge di nuovo i dati. Se il risultato non è zero, l'esecuzione continua con la successiva specifica. Se si omette *m*, si assume che sia zero.

La specifica WAIT consente di provare una o più posizioni di bit ad un porta di immissione. Si può provare la posizione di bit per 1 o Ø. Le posizioni di bit da provare sono specificate impostando degli 1 nelle posizioni di *n*.

Se non si indica *m,* vengono provati, alla ricerca di 1, i bit del punto di immissione. In caso contrario, con un 1 in una posizione di bit di *m* (per i quali c'è un bit 1 in *n),* WAIT prova quel bit di immissione alla ricerca di Ø.

Al termine dell'esecuzione, la specifica WAIT continua a provare i bit di immissione specificati da 1 in *n.* Se uno di quei bit è 1 (o Ø se il corrispondente bit di *m* è 1), il programma continua con la specifica successiva. Dunque WAIT non attende l'apparire di un'intera struttura di bit, ma solo uno di essi.

> **Nota:** è possibile entrare in un'iterazione infinita con la specifica WAIT. Per uscire dall'iterazione premere Ctrl-Interr o eseguire un ripristino del sistema.

Esempio: Per sospendere l'esecuzione del programma fino a quando la porta 32 riceve un bit 1 nella seconda posizione di bit:

```
10Ø WAIT 32,2
```

WHILE e WEND (Specifiche)

Scopo: Eseguire una serie di specifiche in un'iterazione purché una determinata condizione sia vera.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato:

WHILE *espressione*
.
.
.
Specifiche di iterazione
.
.
.
WEND

Note: *espressione* è un'espressione numerica.

Se *espressione* è vera (non zero), le *specifiche di iterazione* vengono eseguite fino a quando si incontra WEND. Poi il BASIC ritorna alla specifica WHILE e controlla *l'espressione*. Se è ancora vera, il processo si ripete. In caso contrario, l'esecuzione riprende con la specifica successiva a WEND.

Le iterazioni WHILE... WEND possono essere nidificate a qualsiasi livello. Ogni WEND deve corrispondere alla più recente WHILE. Una WHILE senza una specifica corrispondente causa un errore "WHILE without WEND" e viceversa, una WEND senza la corrispondente specifica causa un errore "WEND without WHILE".

WHILE e WEND (Specifiche)

Esempio: In questo esempio gli elementi della schiera A vengono ordinati alfabeticamente. A è stata definita con gli elementi J.

```
10 'seleziona la matrice A
20 FLIPS=1 'forza il primo passo nella iterazione
30 WHILE FLIPS
40 FLIPS=0
50 FOR I=1 TO J-1
60 IF A(I)>A(I+1) THEN
        SWAP A(1),A(I+1): FLIPS=1
70 NEXT I
80 WEND
```

WIDTH (Specifica)

Scopo: Definire l'ampiezza della riga di emissione in numeri di caratteri. Dopo aver emesso il numero caratteri indicato, il BASIC aggiunge un ritorno a margine.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	(**)

Formato: WIDTH *ampiezza*

WIDTH *unità, ampiezza*

WIDTH # *numfile, ampiezza*

Note:

ampiezza è un'espressione numerica compresa tra 0 e 255. Questa è la nuova ampiezza. WIDTH 0 corrisponde a WIDTH 1.

unità è un'espressione a stringa per l'identificazione dell'unità. Le unità valide sono: SCRN:, LPT1:, LPT2:, LPT3:, COM1: o COM2:.

numfile è un'espressione numerica compresa tra 1 e 15. Questo è il numero di un file aperto per una delle unità listate sotto.

In base all'unità specificata, sono possibili le seguenti azioni:

WIDTH ampiezza o WIDTH "SCRN:"ampiezza

Imposta l'ampiezza dello schermo. Sono consentite solo 40 o 80 colonne di ampiezza. WIDTH 40 non è valido per lo schermo monocromatico IBM.

Se lo schermo è in modo grafici a media risoluzione (come con una specifica SCREEN 1), WIDTH 80 forza lo schermo ad alta risoluzione (come con la specifica SCREEN 2).

E' valido anche l'inverso: se lo schermo è in modo grafici ad alta risoluzione (come con una specifica SCREEN 2), WIDTH 40 forza lo schermo a media risoluzione (come con la specifica SCREEN 1).

Nota: cambiando l'ampiezza dello schermo si cancella il video e si impostano al nero i bordi dello schermo.

WIDTH *unità, ampiezza*

Utilizzata come assegnazione di ampiezza ritardata per l'unità. Questa forma di ampiezza memorizza il nuovo valore per l'ampiezza senza modificare realmente quello attuale. Una successiva OPEN sull'unità userà questo valore per l'ampiezza mentre il file è aperto. L'ampiezza non cambia immediatamente se l'unità è già aperta.

Nota: LPRINT, LLIST e LIST,"LPTn:" eseguono una OPEN implicita e sono perciò interessate da questa specifica.

WIDTH #*numfile, ampiezza*

L'ampiezza dell'unità associata a *numfile* è modificata immediatamente in quella nuova specificata. Questo consente di modificare l'ampiezza mentre il file è aperto. Questa forma di WIDTH ha significato solo per LPT1: nel BASIC su cassetta. Il BASIC su disco e avanzato consente anche LPT2:, LPT3:, COM1: e COM2:. Notare che è richiesto il simbolo #.

WIDTH (Specifica)

Qualsiasi valore immesso che non sia compreso nei limiti indicati causa un errore. Viene conservato il valore precedente. WIDTH non ha effetto per la tastiera (KYBD:) o la cassetta (CAS1:).

Quando si avvia il BASIC si assume l'ampiezza 80 per ogni stampatrice. L'ampiezza massima per la stampatrice grafica IBM di 80 CPS è 132. Tuttavia non si verifica un errore per i valori compresi tra 132 e 255.

Spetta all'utente impostare l'appropriata ampiezza fisica della stampatrice. E' possibile impostare le stampatrici inviando codici speciali, o con gli interruttori. Con la stampatrice grafica IBM di 80 CPS si deve usare la specifica LPRINT CHR$ (15); per scegliere un tipo di carattere condensato, se si stampa con ampiezze maggiori di 80. Usare LPRINT CHR$ (18); per tornare alle dimensioni normali. La stampatrice IBM aggiunge automaticamente un ritorno a margine se si supera la lunghezza massima della riga.

Specificando un'ampiezza 255 si disabilita il ritorno a margine, producendo l'effetto di un'ampiezza "infinita". WIDTH 255 è il valore assunto per i file di comunicazione.

Modificando l'ampiezza per un file di comunicazione non si altera la memoria di transito di ricezione o trasmissione; il BASIC invia, però, un ritorno a margine dopo i caratteri dell'*ampiezza*.

Modificando il modo schermo si modifica anche l'ampiezza solo se ci si sposta tra SCREEN 2 SCREEN 1 o SCREEN 0. Vedere la specifica SCREEN.

Esempio: In questo esempio la riga 10 memorizza un'ampiezza per la stampatrice di 75 caratteri per riga. La riga 20 apre il file # 1 alla stampatrice ed imposta l'ampiezza a 75 per le successive specifiche PRINT # 1,... La riga 6020 cambia l'attuale ampiezza della stampatrice in 40 caratteri per riga.

```
10 WIDTH "LPT1:",75
20 OPEN "LPT1:" FOR OUTPUT AS #1
   .
   .
   .
6020 WIDTH   1,40
```

Notare come il valore di WIDTH deve essere indicato prima della specifica OPEN.

Questo esempio modifica il modo schermo e l'ampiezza.

```
SCREEN 1,0  'Imposta la media risoluzione
WIDTH 80    'Imposta l'alta risoluzione
WIDTH 40    'Ritorna in media risoluzione
SCREEN 0,1  'Imposta il modo testo 40x25
WIDTH 80    'Imposta il modo testo 80x25
```

WINDOW (Specifica)

Scopo: Ridefinire le coordinate dello schermo. (BASIC 2.Ø e successivi livelli).

Versioni: Cassetta Disco Avanzato Compilatore
***

Solo modo grafici.

Formato: WINDOW [[SCREEN] *(x1,y1)—(x2,y2)]*

Note: *(x1,y1), (x2,y2)*
Sono coordinate definite dal programmatore chiamate *coordinate spaziali.* Queste coordinate sono numeri a virgola mobile in precisione singola. Esse definiscono le coordinate spaziali che verranno tracciate all'interno delle coordinate fisiche definite dalla specifica VIEW.

WINDOW permette di disegnare oggetti nello spazio ("sistema di coordinate spaziali"), non restringendo il campo ai limiti dello schermo ("sistema di coordinate fisiche"), specificando le coppie di coordinate spaziali (x1,y1) e (x2,y2).

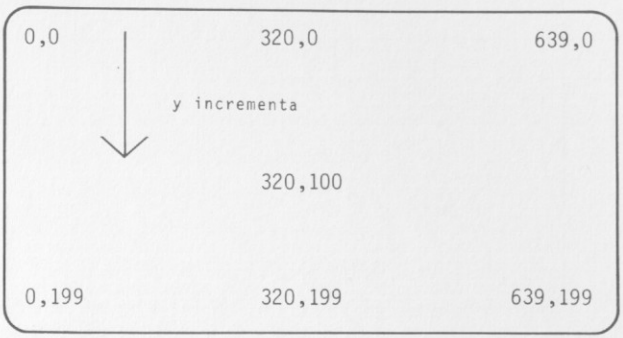
Il BASIC converte le coppie di coordinate spaziali nelle corrispondenti coppie di coordinate per permetterne la visualizzazione sullo schermo.

Per rendere questa informazione possibile il BASIC deve necessariamente conoscere quale porzione dell'area delle coordinate spaziali contiene le informazioni che si vogliono visualizzare. Questa porzione rettangolare dell'area è chiamata finestra.

Se si eseguono le seguenti specifiche nel sistema di coordinate fisiche:

```
NEW
SCREEN 2
```

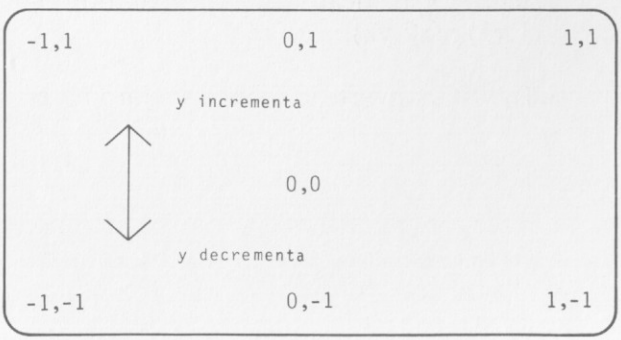
lo schermo con coordinate standard apparirà come segue:



Se l'attributo SCREEN è omesso lo schermo viene visualizzato nelle effettive coordinate cartesiane. Per esempio immettendo:

```
WINDOW (-1,-1)-(1,1)
```

Lo schermo apparirà come segue



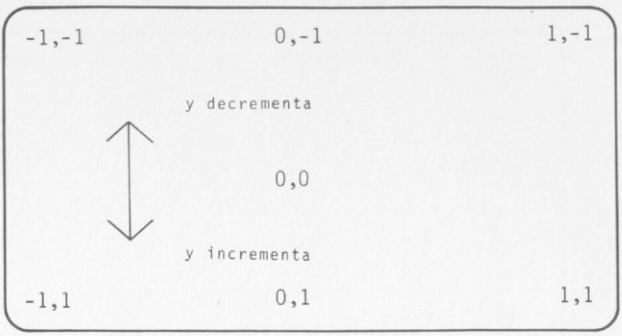
Notare che la coordinata y è invertita in modo che *(x1,y1)* sia la coordinata del punto in basso a sinistra; e *(x2,y2)* la coordinata del punto in alto a destra.

WINDOW (Specifica)

Se l'attributo SCREEN è presente le coordinate non sono invertite e perciò (x1,y1) è il punto in alto a sinistra e (x2,y2) il punto in basso a destra. Per esempio:

```
WINDOW SCREEN (-1,-1)-(1,1)
```

definisce lo schermo come segue:



E' importante notare che la WINDOW ordina le coppie di x e y mettendo i valori più bassi per primi. Per esempio:

```
WINDOW (100,100)-(5,5)
```

diventa

```
WINDOW (5,5)-(100,100)
```

Inoltre:

```
WINDOW (-4,4)-(4,-4)
```

diventa

```
WINDOW (-4,-4)-(4,4)
```

E' permesso qualsiasi accoppiamento di x e y. L'unica restrizione è che x1 non può essere uguale a x2 e y1 non può essere uguale a y2.

WINDOW permette anche di ingrandire e ridurre. Usando una finestra con campo di coordinate più grande di una immagine, questa verrà visualizzata interamente e ci sarà dello spazio vuoto ai margini dello schermo. Scegliendo una finestra con campo di coordinate più piccolo dell'immagine le linee al di fuori del campo verranno tagliate e solo una parte dell'immagine verrà visualizzata ingrandita. Variando le dimensioni delle finestre si può ingrandire un oggetto fino a fargli assumere le dimensioni dell'intero schermo o ridurlo fino a farlo diventare un punto.

RUN, SCREEN, e WINDOW senza attributi eliminano qualsiasi definizione di WINDOW e riportano il video al sistema di coordinate fisiche.

Esempio: Il seguente esempio illustra il procedimento del taglio delle linee con la WINDOW:

```
10 SCREEN 2:CLS
20 WINDOW (-6,-6)-(6,6)
30 CIRCLE (4,4),5,1
40 'Il cerchio è ampio e se ne vede solo una parte
50 WINDOW (-100,-100)-(100,100)
60 CIRCLE (4,4),5,1 'Il cerchio è molto piccolo
70 END
```

WINDOW
(Specifica)

Il seguente esempio illustra come ingrandire usando la WINDOW.

```
10 KEY OFF:CLS:SCREEN 1,0
20 '
30 GOTO 160
40 '====================
50 'procedure display
60 '
70 LINE (X,0)-(-X,0),,,&HAA00 'crea l'asse x
80 LINE (0,X)-(0,-X),,,&HAA00 'crea l'asse y
90 '
100 CIRCLE (X/2,X/2),R 'il cerchio ha raggio r
110 FOR P=1 TO 50:NEXT P 'ciclo di ritardo
120 '
130 RETURN
140 '====================
150 '
160 X=1000:WINDOW (-X,-X)-(X,X):R=20
170 'crea un grafico con ampie scale di coordinate
180 GOSUB 50:FOR P=1 TO 1000:NEXT P:CLS
190 '
200 X=60:WINDOW (-X,-X)-(X,X):R=20
210 'rimpicciolendo la scala delle coordinate il cerchio si ingrandisce
220 GOSUB 50:FOR P=1 TO 1000:NEXT P:CLS
230 '
240 X=100:WINDOW (-5,-5)-(X,X):R=20
250 'modifica la scala per visualizzare solo una porzione delle assi
260 GOSUB 50:FOR P=1 TO 1000:NEXT P:CLS
270 '
280 PRINT "...an illustration of zooming..."
290 FOR P=1 TO 1500:NEXT P
300 CLS:T=-50:U=100:X=U
310 FOR I=1 TO 45
320     T=T + 1:U=U - 1:X=X-1:R=20
330     WINDOW (T,T)-(U,U):CLS:GOSUB 50
340 NEXT I
350 END
```

WRITE (Specifica)

Scopo: Visualizzare i dati sullo schermo.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: WRITE [*lista di espressioni*]

Note: *lista di espressioni*
è una lista di espressioni numeriche e/o a stringa, separate da virgole o punti e virgola.

Se si omette la lista di espressioni, viene visualizzata una riga bianca. Se si include la lista, vengono emessi sullo schermo i valori delle espressioni.

Ogni elemento di queste viene separato dall'ultimo da una virgola. Le stringhe sono delimitate da apici. Dopo la visualizzazione dell'ultimo elemento della lista, il BASIC aggiunge un ritorno a margine/avanzamento riga.

WRITE è simile a PRINT. La differenza tra WRITE e PRINT è che WRITE inserisce delle virgole tra gli elementi quando vengono visualizzati e delimita le stringhe con apici. Inoltre i numeri positivi non sono preceduti da spazi in bianco.

WRITE (Specifica)

Esempio: Questo esempio illustra come WRITE visualizzi i valori a stringa e numerici.

```
10 A=80: B=90: CS="THAT'S ALL
20 WRITE A,B,CS
RUN
80,90,"THAT'S ALL"
```

WRITE # (Specifica)

Scopo: Scrivere i dati in un file sequenziale.

Versioni:

Cassetta	Disco	Avanzato	Compilatore
***	***	***	***

Formato: WRITE # *numfile, lista di espressioni*

Note: *numfile* è il numero con il quale è stato aperto il file per l'emissione.

lista di espressioni
è una lista di espressioni a stringa e/o numeriche separate da virgole o punti e virgola.

La differenza tra WRITE # e PRINT # è che WRITE # inserisce le virgole tra gli elementi quando vengono scritti e delimita le stringhe con apici, perciò non è necessario porre dei delimitatori espliciti nella lista. Inoltre WRITE # non inserisce uno spazio in bianco davanti al numero positivo. Dopo che è stato scritto l'ultimo elemento della lista si inserisce un ritorno a margine/avanzamento riga.

Esempio: Si supponga A$ = ".CAMERA" e B$ = ".93604-1". La specifica:

```
WRITE #1,A$,B$
```

scrive la seguente immagine nel file.

```
",CAMERA",".93604-1"
```

WRITE
(Specifica)

Una successiva specifica INPUT #, come:

```
INPUT #1,A$,B$
```

immette ".CAMERA in A$ e ".936Ø4-1" in B$.

Appendici

Indice

Appendice A. Messaggi di errore A-3

Appendice B. Sottoprogrammi in linguaggio macchina B-1
Riferimenti B-1

Dove caricare in memoria i sottoprogrammi B-2
Come parte residente del DOS B-2
Considerazioni B-3
In un segmento di dati BASIC B-3
Considerazioni B-4
Oltre al segmento di dati del BASIC B-5
Considerazioni B-5

Come caricare e chiamare i sottoprogrammi B-6
Immettere o assegnare un sottoprogramma in memoria B-6
Considerazioni B-6

Caricamento di un sottoprogramma da un file . . . B-10
Considerazioni B-11

Esempio di sottoprogramma B-12
Spiegazione dell'esempio del sottoprogramma. . B-14

Caricamento del sottoprogramma come estensione residente del DOS B-19
Considerazioni B-19
Come il BASIC supporti i sottoprogrammi in linguaggio macchina B-24

La specifica CALL B-26

Mappa della memoria B-29

Appendice C. Comunicazioni C-1
Come aprire un file per comunicazioni C-1
I/E della comunicazione C-1
GET e PUT per file di comunicazione C-2
Funzioni di I/E C-2
Funzioni INPUT$ C-3
Esempio di programma C-4
Note sul programma C-5

Operazione dei segnali di controllo C-6
Controllo dei segnali di emissione con OPEN . C-6
Uso dei segnali di controllo di immissione . . . C-6
Prova dei segnali di controllo del modem C-7
Controllo diretto dei segnali di controllo di emissione C-8
Errori di comunicazione C-9

Appendice D. Codici carattere ASCII D-1

Codici estesi D-6

Appendice E. Codici di scansione E-1

Appendice A. Messaggi di errore

Se il BASIC trova un errore che determina l'arresto dell'esecuzione di un programma, viene visualizzato un messaggio. E' possibile intercettare ed interpretare gli errori di un programma BASIC usando la specifica ON ERROR e le variabili ERR o ERL. Al riguardo fare riferimento anche alle variabili ERDEV e ERDEV$ in questo manuale.

Questa appendice lista in ordine alfabetico tutti i messaggi di errore BASIC con relativa loro spiegazione, e suggerimenti per superarli. La lista di consultazione rapida separata da questo manuale li elenca in ordine numerico.

Numero	Messaggio
73	**Advanced Feature** **(Dispositivo avanzato)** Il programma ha utilizzato un dispositivo del BASIC avanzato mentre era in uso il BASIC su disco. Avviare il BASIC avanzato e rieseguire il programma.
54	**Bad file mode** **(Modo file errato)** Si è tentato di usare PUT o GET con un file sequenziale o chiuso; oppure di eseguire una OPEN con un modo file diverso da immissione, emissione, aggiunta o casuale. Assicurarsi che la specifica OPEN sia stata immessa ed eseguita correttamente. GET e PUT richiedono un file ad accesso casuale. Questo errore si verifica anche se si tenta di fondere un file non in formato ASCII.

In questo caso, assicurarsi che si stia eseguendo la fusione del file corretto. Se necessario, caricare il programma e salvarlo di nuovo usando l'opzione A.

64 **Bad file name**
(Nome file errato)
Con i comandi KILL, NAME e FILE si è usato per il nome del file un formato non valido.

Vedere nel Capitolo 3 del manuale BASIC la sezione "Denominazione dei File" per informazioni sui nomi dei file validi. Correggere quindi il nome errato.

52 **Bad file number**
(Numero file errato)
Una specifica usa un numero di un file che non è aperto, oppure il numero non è compreso nella gamma specificata all'inizializzazione del file. Oppure, il nome di unità o il nome del file sono troppo lunghi o invalidi.

Assicurarsi che il file desiderato sia stato aperto e immesso correttamente nella specifica. Controllare di avere una specifica del file valida. Vedere nel Capitolo 3 del manuale BASIC la sezione "Denominazione dei File" per informazioni al riguardo.

63 **Bad record number**
(Numero record errato)
In una specifica PUT e GET, il numero del record è uguale a zero oppure è maggiore del massimo consentito (32767 per versioni del BASIC antecedenti al livello 2.Ø).
GET e PUT sono state successivamente ampliate per permettere numeri di record da 1 a 16.777.215.

Correggere la specifica PUT o GET per usare un numero di record valido.

17 **Can't continue**
(Impossibile continuare)
Si è tentato di usare CONT per continuare un programma che:

- Si è arrestato a causa di un errore.
- E' stato modificato durante un'interruzione dell'esecuzione.
- Non esiste.

Assicurarsi che il programma sia caricato e usare RUN per eseguirlo.

— **Can't continue after SHELL**
(Impossibile continuare dopo SHELL)
E' stata usata una specifica SHELL in un programma secondario che ha avuto termine e che è rimasto residente. Un programma secondario non può rimanere residente in quanto impedisce al BASIC l'uso della propria area di lavoro, causando un arresto di programma.

69 **Communication buffer overflow**
(Eccedenza memoria di transito comunicazioni)
E' stata eseguita una specifica di immissione delle comunicazioni, ma la memoria di transito di immissione era già piena

Quando si verifica questa condizione usare una specifica ON ERROR per ritentare l'immissione. Le successive immissioni tentano di eliminare questo errore, a meno che non si continuino a ricevere caratteri molto velocemente ed il programma non sia in grado di elaborarli. Se si verifica ciò, vi sono diverse possibili soluzioni:

- Aumentare l'ampiezza della memoria di transito per le comunicazioni usando quando si avvia il BASIC l'opzione /C :.

- Sviluppare un protocollo "personale" con l'altro elaboratore per informarlo di arrestare l'invio finché non è possibile riceverlo. Vedere l'esempio nell'Appendice C "Comunicazioni".

- Usare una velocità inferiore per trasmettere e ricevere.

25 **Device fault**
(Errore unità)
Un'indicazione di errore macchina è stata ritornata da un adattatore di interfaccia.

Nel BASIC su cassetta, ciò si verifica solo quando l'adattatore di interfaccia della stampatrice ritorna uno stato di errore.

Questo messaggio può apparire anche nella trasmissione di dati ad un file di comunicazioni. In questo caso, indica che uno o più segnali da provare (indicati nella specifica OPEN "COM...) non sono stati trovati nel periodo di tempo specificato.

57 **Device I/O Error**
(Errore unità I/E)
Si è verificato un errore in un'operazione di I/E sull'unità. Il DOS non può correggere l'errore.

Durante la ricezione di dati per la comunicazione, questo errore può verificarsi a causa di una perdita di dati, di una interruzione o a seguito di errori di parità. Se si ricevono dati con 7 bit al massimo, l'ottavo bit viene attivato nel byte errato.

24 **Device timeout**
(Supero tempo unità)
Il BASIC non ha ricevuto le informazioni dall'unità di immissione/emissione nel tempo predeterminato. Nel BASIC su cassetta, questo si verifica solamente mentre il programma sta tentando di leggere da una cassetta o scrivere su una stampatrice.

Per i file di comunicazione, questo messaggio indica che, nel periodo di tempo specificato non è stato trovato uno o più dei segnali provati con OPEN "COM...

Ritentare l'operazione.

68 **Device unvailable**
(Unità non disponibile)
Si è tentato di aprire un file su un'unità che non esiste oppure non si possiede il componente fisico per supportare l'unità (come gli adattatori per la seconda o terza stampatrice), oppure l'unità è stata disabilitata.

Ad esempio, si è usata l'opzione /C:0 nel comando BASIC per avviare il BASIC su disco. Questo disabilita le unità per le comunicazioni.

Assicurarsi che l'unità sia stata installata correttamente. Se necessario, ritornare al DOS dal quale è possibile reimmettere il comando BASIC.

66 **Direct Statement in file**
(Specifica diretta nel file)
Si è trovato una specifica diretta durante il caricamento o concatenamento ad un file in formato ASCII. La specifica LOAD o CHAIN è terminata.

Il file ASCII deve contenere solo specifiche precedute da numeri di riga. Questo errore può verificarsi a causa di un carattere di avanzamento riga nel flusso di immissione.

61 **Disk full**
(Disco pieno)
Tutto lo spazio di memoria del disco è in uso. Quando si verifica questo errore i file vengono chiusi.

Se sul minidisco vi sono dei file non più necessari, cancellarli; oppure usare un nuovo minidisco. Poi ritentare l'operazione o rieseguire il programma.

72 **Disk media error**
(Errore disco)
La scheda di collegamento del dispositivo di controllo ha rilevato un errore hardware. Generalmente significa che il minidisco è rovinato.

Copiare i file esistenti su un nuovo minidisco e inizializzare nuovamente quello errato. Se l'inizializzazione non ha esito positivo, scartare il minidisco.

71 **Disk not ready**
(Disco non pronto)
Lo sportello dell'unità a minidischi è aperto, oppure nell'unità non vi sono minidischi.

Inserire il minidisco corretto e continuare il programma.

70 **Disk write protect**
(Disco protetto da scrittura)
Si è tentato di scrivere su un minidisco protetto da scrittura.

Assicurarsi di usare il minidisco corretto. In caso affermativo eliminare la protezione da scrittura e ritentare l'operazione.

Questo errore può verificarsi anche a causa di un malfunzionamento hardware.

11 **Division by zero**
(Divisione per zero)
In un'espressione, si è tentato di dividere per zero o di elevare zero ad una potenza negativa.

Non è necessario correggere questa condizione poiché il programma continua l'esecuzione. Il risultato della divisione è l'infinito di macchina con il segno del numero da dividere; oppure l'infinito di macchina positivo è il risultato dell'elevazione a potenza. Questo errore non può essere intercettato da ON ERROR.

10 **Duplicate definition**
(Definizione doppia)
Si è tentato di definire l'ampiezza della stessa schiera due volte. Ciò può essere avvenuto in più modi:

- La stessa schiera è stata definita in due specifiche DIM.
- Il programma incontra una specifica DIM per una schiera dopo che per quest'ultima è stata stabilita la dimensione 10 assunta per difetto.

- Il programma vede una specifica OPTION BASE dopo il dimensionamento di una schiera tramite specifica DIM o per difetto.

Spostare la specifica OPTION BASE per assicurarsi che venga eseguita prima di usare una schiera; oppure, correggere il programma in modo che ogni schiera venga definita solo una volta.

50 **FIELD overflow**
(Eccedenza di FIELD)
Una specifica FIELD ha tentato di allocare più byte di quanti ne erano stati indicati nella specifica OPEN per la lunghezza record di un file ad accesso casuale. Oppure, si è raggiunta la fine della memoria di transito di FIELD durante l'I/E sequenziale (PRINT #, WRITE #, INPUT #) in un file ad accesso casuale.

Controllare le specifiche OPEN o FIELD per assicurarsi che corrispondano. Se si sta eseguendo una I/E sequenziale in un file ad accesso casuale, assicurarsi che la lunghezza dei dati letti o scritti non superi quella del record del file ad accesso casuale.

58 **File already exits**
(Il file esiste già)
Il nome del file indicato nella specifica NAME corrisponde ad un altro già in uso sul disco.

Ritentare il comando NAME usando un nome diverso.

55 **File already open**
(File già aperto)
Si è tentato di aprire un file già aperto per l'emissione o aggiunta sequenziale oppure si è tentato di usare KILL su un file aperto.

Assicurarsi di usare il minidisco corretto. In caso affermativo eliminare la protezione da scrittura e ritentare l'operazione.

Questo errore può verificarsi anche a causa di un malfunzionamento hardware.

11 **Division by zero**
(Divisione per zero)
In un'espressione, si è tentato di dividere per zero o di elevare zero ad una potenza negativa.

Non è necessario correggere questa condizione poiché il programma continua l'esecuzione. Il risultato della divisione è l'infinito di macchina con il segno del numero da dividere; oppure l'infinito di macchina positivo è il risultato dell'elevazione a potenza. Questo errore non può essere intercettato da ON ERROR.

10 **Duplicate definition**
(Definizione doppia)
Si è tentato di definire l'ampiezza della stessa schiera due volte. Ciò può essere avvenuto in più modi:

- La stessa schiera è stata definita in due specifiche DIM.
- Il programma incontra una specifica DIM per una schiera dopo che per quest'ultima è stata stabilita la dimensione 10 assunta per difetto.

- Il programma vede una specifica OPTION BASE dopo il dimensionamento di una schiera tramite specifica DIM o per difetto.

Spostare la specifica OPTION BASE per assicurarsi che venga eseguita prima di usare una schiera; oppure, correggere il programma in modo che ogni schiera venga definita solo una volta.

50 **FIELD overflow**
(Eccedenza di FIELD)
Una specifica FIELD ha tentato di allocare più byte di quanti ne erano stati indicati nella specifica OPEN per la lunghezza record di un file ad accesso casuale. Oppure, si è raggiunta la fine della memoria di transito di FIELD durante l'I/E sequenziale (PRINT #, WRITE #, INPUT #) in un file ad accesso casuale.

Controllare le specifiche OPEN o FIELD per assicurarsi che corrispondano. Se si sta eseguendo una I/E sequenziale in un file ad accesso casuale, assicurarsi che la lunghezza dei dati letti o scritti non superi quella del record del file ad accesso casuale.

58 **File already exits**
(Il file esiste già)
Il nome del file indicato nella specifica NAME corrisponde ad un altro già in uso sul disco.

Ritentare il comando NAME usando un nome diverso.

55 **File already open**
(File già aperto)
Si è tentato di aprire un file già aperto per l'emissione o aggiunta sequenziale oppure si è tentato di usare KILL su un file aperto.

Assicurarsi di eseguire solo una OPEN su un file su cui si sta scrivendo sequenzialmente. Chiudere un file prima di usare KILL.

53 **File not found**
(File non trovato)
Una LOAD, KILL, NAME, FILES o OPEN fa riferimento ad un file che non esiste nel disco del modulo specificato.

Verificare che nell'unità specificata si trovi il minidisco corretto e che la specifica del file sia stata immessa correttamente. Ritentare l'operazione.

26 **FOR without NEXT**
(FOR senza NEXT)
E' stata trovata una specifica FOR senza la corrispondente NEXT. Cioè, un'iterazione FOR era attiva quando si è raggiunta la fine fisica del programma.

12 **Illegal direct**
(Modo diretto non valido)
Si è tentata l'immissione in modo diretto di una specifica che non può essere immessa in questo modo (come DEF FN).

La specifica deve essere immessa come parte di una riga di programma.

5 **Illegal function call**
(Richiamo funzione non valido)
Viene passato ad una funzione di sistema un parametro che è fuori limiti. L'errore può anche essere causato da:

- Un subscritto negativo o troppo largo.
- Il tentativo di elevare un numero negativo ad una potenza che non è un intero.

- Il richiamo di una funzione USR prima di definire l'indirizzo iniziale con DEF USR.

- Un numero di record negativo in GET o PUT (file).

- Un argomento improprio in una funzione o specifica.

- Il tentativo di listare o editare un programma BASIC protetto.

- Il tentativo di cancellare dei numeri di riga che non esistono.

Vedere la sezione "Funzioni, specifiche e comandi BASIC" di questo manuale per informazioni sulla particolare specifica o funzione.

— **Incorrect DOS version**
(Versione DOS non corretta)
E' stato immesso un comando non permesso dalla versione DOS in uso.

62 **Input past end**
(Immissione dopo la fine)
Questo è un errore di fine file. Viene eseguita una specifica di immissione per un file nullo (vuoto) o dopo che sono già stati immessi tutti i dati di un file sequenziale.

Per evitare l'errore, usare la funzione EOF per rilevare la fine del file.

Questo errore si verifica anche se si tenta la lettura di un file aperto per l'emissione o l'aggiunta. Se si vuole leggere un file di emissione sequenziale (o di aggiunta), è necessario chiuderlo e poi riaprirlo per l'immissione.

51 **Internal error**
(Errore interno)
Si è verificato un malfunzionamento interno nel BASIC.

Rieseguire la copia del minidisco. Controllare l'hardware e ritentare l'operazione. Se si verifica ancora l'errore, riferire al punto di vendita le condizioni in cui questo si è verificato.

23 **Line buffer overflow**
(Eccedenza memoria di transito della riga).
Si è tentata l'immissione di una riga composta da troppi caratteri.

Separare le specifiche multiple in modo che vengano a trovarsi su più di una riga. Si possono utilizzare anche le variabili a stringa invece delle costanti, ove possibile.

22 **Missing operand**
(Manca l'operando)
Un'espressione contiene un operatore, tipo * o OR, senza nessun operando.

Assicurarsi di includere tutti gli operandi necessari nell'espressione.

1 **NEXT without FOR**
(NEXT senza FOR)
La specifica NEXT non ha la corrispondente FOR. E' possibile che una variabile della specifica NEXT non corrisponda ad una variabile della specifica FOR precedentemente eseguita e non confrontata.

Correggere il programma in modo che NEXT abbia la corrispondente FOR.

19 **No RESUME**
(Nessuna RESUME)
Il programma è saltato ad un programma attivo di intercettazione degli errori, come risultato di una condizione di errore o di una specifica ERROR. Il programma non contiene una specifica RESUME (si è incontrata la fine fisica del programma nel sottoprogramma di intercettazione dell'errore).

Assicurarsi di includere RESUME nel sottoprogramma di intercettazione degli errori per continuare l'esecuzione. E' possibile aggiungere una specifica ON ERROR GOTO 0 al programma di intercettazione degli errori in modo che il BASIC visualizzi un messaggio per ogni errore non intercettato.

4 **Out of data**
(Mancano dati)
Una specifica READ sta tentando di leggere più dati di quanti se ne trovano nelle specifiche DATA.

7 **Out of memory**
(Manca memoria)
Un programma è troppo grande, ha troppe iterazioni FOR o GOSUB, troppe variabili, espressioni complicate o disegni complessi.

Si può utilizzare CLEAR all'inizio del programma per riservare più spazio o area di memoria.

27 **Out of paper**
(Manca la carta)
La stampatrice ha finito la carta, oppure non è accesa.

Inserire la carta (se necessario), verificare che la stampatrice sia collegata correttamente e che l'alimentazione sia fornita; poi continuare il programma.

14 **Out of string space**
(Manca spazio per la stringa)
Il BASIC colloca dinamicamente lo spazio per la stringa finché non esaurisce la memoria. Questo messaggio significa che le variabili a stringa hanno fatto superare al BASIC la quantità di memoria libera che rimane dopo il riordino.

6 **Overflow**
(Eccedenza)
Un numero è troppo ampio per essere rappresentato numericamente nel formato BASIC. L'eccedenza di intero causa l'arresto dell'esecuzione. Altrimenti, viene fornito come risultato l'infinito di macchina con l'appropriato segno e l'esecuzione continua. L'eccedenza di intero è l'unico tipo di eccedenza che può essere intercettata.

Per correggere l'eccedenza di intero è necessario usare numeri più piccoli o modificare le variabili in precisione singola o doppia.

> **Nota:** Se un numero è troppo piccolo per essere rappresentato numericamente nel formato BASIC si verifica la condizione opposta. In questo caso, il risultato è zero e l'esecuzione continua senza errori.

75 **Path/file access error**
(Errato accesso a percorso/file)
Durante un'operazione di OPEN, RENAME, MKDIR, CHDIR, o RMDIR si è tentato di usare un percorso o un nome di file che fa riferimento a un file non accessibile. Per esempio: si è tentato di fare la OPEN su un indirizzario o un nome di volume; si è tentato di fare la OPEN per scrivere su un file di sola lettura, o si è tentato di eliminare l'indirizzario in uso. L'operazione non viene portata a termine.

76 **Path not found**
(Percorso non trovato)
Durante un'operazione di OPEN, MKDIR, CHDIR o RMDIR il DOS non è in grado di trovare il percorso nel modo specificato. L'operazione non viene portata a termine.

74 **Rename across disks**
(Ridenominazione su altro disco)
Si è tentato di ridenominare un file ma non è stato specificato il disco corretto. L'operazione non viene portata a termine.

Usando RENAME il disco specificato deve essere lo stesso sia per il vecchio che per il nuovo nome del file a meno che non sia attivo il comando di DOS ASSIGN. In questo caso il nome del disco può essere differente, ma deve essere specificata la stessa unità.

20 **RESUME without error**
(RESUME senza errore)
Il programma ha incontrato una specifica RESUME senza avere intercettato un errore. Il programma di intercettazione degli errori deve essere immesso solo quando si verifica un errore e si esegue una specifica ERROR.

Probabilmente è necessario includere una specifica STOP o END prima del programma di intercettazione degli errori per impedire al programma di "cadere" nel sottoprogramma di intercettazione dell'errore.

3 **RETURN without GOSUB**
(RETURN senza GOSUB)
Una specifica RETURN richiede una precedente specifica GOSUB.
Correggere il programma. Probabilmente è necessario porre una specifica STOP o END prima del sottoprogramma, in modo che il programma non "cada" nel sottoprogramma.

16 **String formula too complex**
(Formula di stringa troppo complessa)
Un'espressione a stringa è troppo lunga o complessa.

L'espressione deve essere divisa in espressioni più piccole.

15 **String too long**
(Stringa troppo lunga)
Si è tentato di creare una stringa superiore ai 255 caratteri.

Spezzare la stringa in stringhe più piccole.

9 **Subscript out of range**
(Elemento fuori limiti)
E' stato utilizzato un elemento di schiera che supera le dimensioni della schiera o una schiera con un numero errato di elementi.

Controllare l'uso della schiera. Può essere stato immesso un elemento in una variabile che non è una schiera, oppure aver codificato erroneamente una funzione incorporata.

2 **Syntax error**
(Errore di sintassi)
Una riga contiene una sequenza di caratteri errata,ad esempio come una sola parentesi, un comando scritto in modo errato, oppure la specifica o la punteggiatura errata. Oppure i dati della specifica DATA non corrispondono al tipo (numerico o stringa) della variabile della specifica READ. Oppure è stata usata come nome della variabile una parola riservata.

Quando si verifica questo errore, l'editore di programmi BASIC visualizza automaticamente la riga errata. Correggere la riga o il programma.

67 **Too many files**
(Troppi file)
Si è tentato di creare un nuovo file (con SAVE o OPEN) quando tutte le voci dell'indirizzario del disco sono piene, oppure quando la specifica del file non è valida.

Se la specifica del file è valida, usare un nuovo minidisco inizializzato e ritentare l'operazione.

13 **Type mismatch**
(Tipo non corrispondente)
Si è fornito un valore a stringa quando ne era richiesto uno numerico o viceversa. Questo errore può essere causato anche dal tentativo di scambiare (SWAP) variabili di tipo diverso, come ad esempio a precisione singola o a precisione doppia.

8 **Undefined line number**
(Numero di riga indefinito)
Una riga di una specifica o comando fa riferimento ad una riga inesistente nel programma.

Controllare i numeri di riga del programma ed utilizzare il numero corretto.

18 **Undefined user function**
(Funzione dell'utente indefinita)
E' stata richiamata una funzione prima di definirla con la specifica DEF FN.

Assicurarsi che il programma esegua la specifica DEF FN prima di usare la funzione.

— **Unprintable error**
(Errore non previsto)
Non è disponibile un messaggio per l'errore verificatosi. Questo normalmente è causato da una specifica ERROR priva di codice. Controllare il programma per assicurarsi di aver utilizzato appropriatamente tutti i codici di errore.

30 **WEND without WHILE**
(WEND senza WHILE)
E' stata trovata una WEND prima che fosse eseguita la corrispondente WHILE.

Correggere il programma in modo che vi sia una WHILE per ogni WEND.

29 **WHILE without WEND**
(WHILE senza WEND)
Una specifica WHILE non ha la corrispondente WEND. Cioè, una WHILE era ancora attiva quando è stata raggiunta la fine del programma.

Correggere il programma in modo che vi sia una WHILE per ogni WEND.

— **You cannot SHELL to BASIC**
(Funzione SHELL non utilizzabile sul BASIC).
Il BASIC in uso non può eseguire un programma secondario mediante la funzione SHELL.

Appendice B. Sottoprogrammi in linguaggio macchina.

Questo paragrafo descrive come il BASIC esegua l'interfaccia con i sottoprogrammi in linguaggio macchina. In particolare descrive:

- Dove allocare i sottoprogrammi in memoria.
- Come il BASIC esegua l'interfaccia con i sottoprogrammi.
- Come caricare e richiamare i sottoprogrammi.

Questa appendice si intende debba essere usata da programmatori esperti in linguaggio macchina.

Dove allocare in memoria i sottoprogrammi

Il BASIC normalmente usa tutta la memoria disponibile dalla posizione iniziale fino ad un massimo di 64K byte. Questa area contiene il programma BASIC e relativi dati, l'area di lavoro del programma di interpretazione e lo stack del BASIC. E' possibile allocare spazio di memoria per i sottoprogrammi in linguaggio macchina sia all'interno che all'esterno dell'area di lavoro di 64 K byte del BASIC. Per stabilire dove caricare il sottoprogramma occorre considerare:

- L'ampiezza della memoria del proprio elaboratore.
- L'ampiezza del programma BASIC.
- Le caratteristiche del programma stesso.

Vi sono inoltre diversi metodi per installare i sottoprogrammi. I tre che seguono sono particolarmente consigliabili:

- Come parte residente del DOS caricato prima che il BASIC venga richiamato.
- In una schiera di interi inserita nel segmento del BASIC.
- Oltre il segmento dati del BASIC.

Come parte residente del DOS

Questo è il metodo maggiormente raccomandato in quanto assicura una sufficiente compatibilità presente e futura tra i propri sottoprogrammi, il DOS e il BASIC, riducendo inoltre il margine di possibili errori. Questo metodo richiede una addizionale pianificazione nel codificare i sottoprogrammi. Vedere al riguardo l'esempio di sottoprogramma riportato più avanti in questo capitolo.

Caratteristiche

- Questo metodo permette sottoprogrammi multisegmenti (sino a 64K) e sottoprogrammi non autocaricanti che sarebbe impossibile usare propriamente con le specifiche BLOAD o DATA.

- I sottoprogrammi saranno parte residente del DOS non necessitando quindi da parte del DOS e del BASIC di spazi protetti o riservati.

Considerazioni

- I sottoprogrammi caricati DOS rimangono residenti e vengono rimossi solo a seguito di un ripristino del sistema. Se richiamate più volte anche copie multiple possono diventare residenti. Questo è maggiormente gestibile definendo un richiamo alla fine dell'applicazione che ripristina il sistema rappresentato dal BIOS e rinizializzando poi tutta la memoria.

- Nel richiamare mediante INT 27 i sottoprogrammi residenti, verrà creata nella stessa porzione d'area residente una copia dell'ambiente DOS (minimo 128 byte) ed una PSP (program segment prefix) di 256 byte.

In un segmento di dati BASIC

Il sistema più facile e sicuro di allocare dei sottoprogrammi in linguaggio macchina relativamente corti, consiste nel caricarli in una schiera intera nel BASIC.

Caratteristiche

- Poiché si sta lavorando in un segmento dati del BASIC non è necessario assegnare un segmento esterno per il proprio sottoprogramma mediante una specifica DEF SEG.

- Caricando i sottoprogrammi nel segmento dati in un'area dati assegnata e gestita dal BASIC, non si corre nessun rischio che i propri sottoprogrammi vengano inseriti in spazi critici dentro o fuori dal BASIC.

Considerazioni

- Se l'applicazione BASIC utilizza la maggior parte dell'area disponibile nel segmento dati del BASIC, il metodo fin'ora considerato non è il migliore in quanto verrà comunque usato dello spazio addizionale per assegnare i sottoprogrammi ad una schiera intera.

- In alcune configurazioni, questo metodo è il solo possibile. Prima di scegliere il metodo da seguire è necessario calcolare la quantità di memoria libera disponibile. I fattori da considerare sono: l'ampiezza del DOS, l'ampiezza del BASIC, se l'applicazione necessita di un BASIC su disco o avanzato ed infine l'ampiezza di ogni sottoprogramma residente tipo MODE, PRINT o GRAPHICS.

  L'ampiezza del DOS e del BASIC dipende dal livello di codifica che si sta usando. Su un piccolo sistema, quando il BASIC viene richiamato, potrebbe non esserci in cima alla memoria nessuno spazio disponibile. In questo caso è consigliabile usare il metodo della schiera intera.

- La codifica assemblatore deve essere auto-caricante; cioè non deve contenere nessun riferimento a numeri di paragrafi o a segmenti

i cui valori dipendano dallo spazio in cui sono stati caricati. Ciò significa che tutte le posizioni relative devono essere espresse in termini relativi in modo che il sottoprogramma sia in grado di risolvere ogni riferimento nel suo ambito.

Oltre il segmento dati del BASIC

Per questa procedura è necessario avere nel proprio sistema almeno 128K byte. Inoltre parte della memoria può essere dedicata alla gestione dei sottoprogrammi per garantirne la loro integrità. Per questo motivo il caricare dei sottoprogrammi oltre il BASIC è un metodo non consigliabile.

Suggerimenti

- Non usare spazio del segmento dati del BASIC per i propri sottoprogrammi.
- Con i sistemi che posseggono una maggior quantità di memoria, possono essere caricati programmi che altrimenti non potrebbero stare nel segmento dati del BASIC.

Considerazioni

Quando viene caricato un sottoprogramma fuori dal segmento dati del BASIC, allora si è direttamente responsabili della gestione dell'area occupata. Per assicurare l'integrità del programma occorre:

- Restringere l'ampiezza dello spazio totale indirizzabile del BASIC immettendo nella riga di comando BASIC la specifica CLEAR o /M:. Questo assicura che i dati del BASIC non verranno a sovrapporsi al proprio programma.
- Calcolare nel proprio programma il nuovo segmento ID che sarà passato alla specifica DEF SEG.

- Nel calcolare l'ampiezza della memoria disponibile permettere ad ogni programma residente al momento, di poter chiamare il sottoprogramma (per esempio PRINT, MODE o un programma disco virtuale).

Come chiamare e caricare i sottoprogrammi.

Gli esempi che seguono mostrano come dei sottoprogrammi in linguaggio macchina vengano caricati in memoria. Vi sono essenzialmente tre modi:

- Immetterlo o assegnarlo in memoria dal proprio programma BASIC.
- Mediante il comando BLOAD da un file su disco o cassette.
- Caricarlo da un file su disco come parte residente del DOS.

Immettere o assegnare un sottoprogramma in memoria.

Un sottoprogramma in linguaggio macchina può essere assegnato o immesso (mediante la specifica POKE) in memoria direttamente dal proprio programma. In questo modo il sottoprogramma diventa parte integrante del programma BASIC.

Considerazioni

- Con questo metodo non è necessario usare un assemblatore.
- Tutto il codice è contenuto in un file. Non bisogna quindi preoccuparsi della creazione e della manutenzione di due o più file separati.

- Codificare ogni specifica del proprio sottoprogramma in esadecimale può diventare noioso e possono essere commessi degli errori. Questo metodo è particolarmente utile per sottoprogrammi di limitate dimensioni.

- Se il sottoprogramma in questione viene caricato in memoria fuori dal segmento di dati del BASIC, al fine di assicurare l'integrità del proprio programma, è necessario eseguire una adeguata gestione della memoria per preservare o proteggere lo spazio del sottoprogramma dal BASIC o dal DOS.

Assegnazione di un sottoprogramma in memoria.

Questa è la procedura da usare per assegnare un codice macchina ad una schiera di interi e richiamare il sottoprogramma:

1. Determinare il codice macchina per il proprio sottoprogramma. Al riguardo fare riferimento a "DEBUG" nel manuale del DOS.

2. Dimensionare la schiera all'ampiezza desiderata. Tenere presente che un numero intero usa 2 byte di memoria, cosicché l'ampiezza della schiera sarà metà del numero di byte indicato nel sottoprogramma.

   Diverse sono le ragioni che consigliano l'utilizzo di una schiera di interi (2 byte per elemento) invece di una schiera a precisione singola (4 byte per elemento) o precisione doppia (8 byte per elemento). Nell'assegnare un sottoprogramma ad una schiera non bisognerà lasciare più di un byte di memoria inutilizzato; a precisione doppia il numero di byte inutilizzati può essere 7.

   Sarebbe facile commettere un errore assegnando quattro parole per ogni elemento di schiera, mentre la CPU nel sistema si aspetta di ricevere un codice macchina nel formato low-byte e high-byte.

l'esempio che segue mostra come il codice macchina nella specifica di assegnazione sia impostato dall'ordine dei byte così come appaiono nella lista assemblata.

3. Definire tutti gli scalari che dovranno essere usati nell'applicazione. Dopo aver dimensionato una schiera, ogni scalare creato inserisce o riassegna la schiera in memoria. Per ulteriori informazioni vedere la mappa della memoria nell'Appendice D.

4. Assegnare il codice di linguaggio macchina agli elementi della schiera, una parola alla volta. E' opportuno ricordare che il sistema legge i byte aspettando di ricevere prima il byte inferiore così da conservare i dati in modo low-byte, high-byte.

5. Determinare la posizione relativa del sottoprogramma nell'ambito del segmento dati del BASIC utilizzando la funzione VARPTR.

6. Richiamare il sottoprogramma mediante la specifica CALL o la funzione USR.

Esempio:

```
10 DEFINT A-Z: OPTION BASE 1: DIM ARRAY (3)
20 DATA &HCD55              :REM 55H Push BP
30 DATA &H5D05              :REM CD05H INT 5
40                          :REM 5DH POP BP
50 DATA &H90CB              :REM 90H NOP
70 FOR I = 1 TO 3: READ ARRAY(I): NEXT I
80 PRINT "This is an example of how "
90 PRINT "a BASIC routine can cause "
100 PRINT "a hard copy to be dumped "
110 PRINT "to a printer by using the "
120 PRINT "shift-print screen function "
130 PRINT "called via an assembly language "
140 PRINT "subroutine that is built into "
150 PRINT "a BASIC integer array"
160 SUBRT = VARPTR(ARRAY(1)): CALL SUBRT
```

Notare come la specifica di lettura legga i dati nella schiera una parola alla volta. La lista del programma listerà i byte nell'ordine in cui essi appaiono nella sezione di commento del programma. E' consigliabile mettere la funzione VARPTR e la specifica CALL nella stessa linea in modo che la specifica CALL non venga eseguita senza aver localizzato la schiera contenente il sottoprogramma.

Assegnare un sottoprogramma in memoria.

L'esempio successivo utilizza lo stesso sottoprogramma in linguaggio macchina, con la differenza che la specifica BASIC POKE viene utilizzata per assegnare i byte nella schiera. La specifica POKE tiene conto dell'ordine dei byte cosicché l'utente non è tenuto ad avere l'ordine dei byte così come apparirebbero nella lista assemblata.

```
10 DEFINT A-Z:OPTION BASE 1:P=0:I=0:J=0:
   DIM ARRAY(3)

20 DATA &H55            :REM 55H Push BP
30 DATA &HCD,&H05       :REM CD05H INT 5
40 DATA &H5D            :REM 5DH POP BP
50 DATA &HCB            :REM CBH RET FAR
60                      :REM 90H NOP
70 P=VARPTR(ARRAY(1)):FOR I=0 TO 4:READ J:
70 P=VARPTR(ARRAY(1):FOR I=0 TO 4:READ J:
   POKE(P+I),J:NEXT I
80 PRINT "This is an example of how "
90 PRINT "a BASIC routine can cause "
100 PRINT "a hard copy to be dumped "
110 PRINT "to a printer by using the "
120 PRINT "shift-print screen function "
130 PRINT  "called via an assembly language "
140 PRINT "subroutine that is built into "
150 PRINT "a BASIC integer array"
160 SUBRT = VARPTR(ARRAY(1)): CALL SUBRT
```

La specifica POKE può essere usata per immettere qualsiasi tipo di dati in memoria. Per immettere questo sottoprogramma in un altro segmento la riga 70 dovrebbe essere:

```
DEF SEG = SEGMENT: FOR I=0 TO 4: READ J
POKE(I,J): NEXT
```

Poiché il sottoprogramma viene immesso in un'area fuori dal segmento dati del BASIC, non vi è nessun riferimento VARPTR per ottenere la posizione relativa del sottoprogramma stesso.

Caricamento di un sottoprogramma da un file

Il comando di BASIC BLOAD viene usato per caricare un file ad immagine della memoria direttamente in memoria. Il file può essere un file ad immagine della memoria salvato usando il comando BASIC BSAVE; oppure può essere un sottoprogramma in linguaggio macchina assemblato mediante un comando BLOAD iniziale, compilato e convertito in un file.COM usando EXE2BIN.

Per informazioni su EXE2BIN fare riferimento al manuale DOS. I sottoprogrammi in linguaggio macchina possono essere caricati in una schiera di interi oppure oltre il segmento dati del BASIC.

Cosiderazioni

- Quando un sottoprogramma viene caricato nell'area dati del BASIC, il BASIC esegue la gestione di tutta la memoria e si assume la responsabilità della sua integrità che a questo punto non è più dell'utente.

- Caricando un sottoprogramma fuori dall'area di lavoro del BASIC è possibile costruire un sottoprogramma di ampiezza superiore ai 64K limite del BASIC.

- Per applicare questo metodo è necessario avere un assemblatore.

- Affinché un sottoprogramma in linguaggio macchina venga caricato in memoria, bisogna che sia stato creato mediante una BSAVE. Questo si ottiene includendo una intestazione BSAVE di 7 byte come primo segmento del sottoprogramma. Questi 7 byte funzionano come un elemento di caricamento che il comando BLOAD ricerca e quindi rilascia. Il sottoprogramma in memoria comincia al primo byte successivo a questa intestazione.

L'esempio che segue mostra come un sottoprogramma venga caricato in una schiera di interi con il BASIC e dal BASIC stesso poi richiamato. Il sottoprogramma somma nel terzo parametro il contenuto del primo e del secondo.

L'esempio illustra anche un codice auto caricante. Segue una spiegazione passo per passo di questa procedura.

1. Assembla e collega il sottoprogramma.

2. Converte mediante EXE2BIN il file in un file .COM.

3. Dimensiona dal BASIC, secondo le necessità, una schiera di interi (tener presente che l'ampiezza del subscritto della schiera di interi è metà dei byte nel sottoprogramma).

4. Dichiara tutti gli scalari usati nell'applicazione. Questo include ogni parametro che può essere passato attraverso la specifica CALL.

5. Crea la posizione relativa della schiera nello spazio dati del BASIC mediante l'uso della funzione VARPTR.

6. Carica il sottoprogramma nella schiera.

7. Richiama il sottoprogramma mediante questa posizione relativa.

Esempio di sottoprogramma

La funzione di questo sottoprogramma è quella di addizionare il contenuto del primo e del secondo parametro e di muovere il risultato di questa operazione nel terzo. L'esempio illustra l'unico modo per passare i parametri al sottoprogramma generato nella schiera. I parametri vengono trasmessi caricandoli come elementi di una schiera là dove è presente il sottoprogramma.

I commenti inseriti nella lista del programma hanno la funzione di aiutare l'utente a capire meglio i punti più importanti di questa operazione e la struttura del sottoprogramma. Una più approfondita analisi segue la lista riportata.

```
AAAA                    SEGMENT PARA PUBLIC 'CODE'
            DB          ØFDH            ;BSAVE id
            DW          Ø,Ø
            DW          TRAILER-HEADER ;Module size
AAAA        ENDS
            PAGE
            BASDATSEG SEGMENT BYTE PUBLIC 'CODE'
            ASSUME CS:BASDATSEG,DS:BASDATSEG
HEADER EQU    $     ;Start of memory image
SUBRT  PROC   FAR
       NOP          ;Adjusts for proper
;                    boundary also allows
;                    replacement by ØCCH,
;                    the INT 3, wich is
;                    DEBUG's breakpoint.
       CALL   SKI   ;This pushes offset
;                    of 'BASE' into stack
BASE:               ;local workarea
WORKA  DW Ø ;First input parameter,
;                    BASIC's ARRAY(3)
WORKB  DW Ø ;Second input parameter,
;                    BASIC's ARRAY(4)
WORKC  DW Ø ;Result, BASIC'sARRAY(5)
SKIP:  POP    BX    ;Now BX knows at what
;                    offset into BASIC's
;                    data space is 'BASE'
       MOV    AX,WORKA - BASE[BX] ;Get 1st parm
       ADD    AX,WORKB - BASE[BX] ;Add 2nd parm
;                                    to ACC
       MOV    WORKC-BASE[BX],AX ;Pass to 3rd parm
       RET
SUBRT     ENDP
TRAILER   EQU      $        ;End of memory image
BASDATSEG ENDS
          END
```

Spiegazione dell'esempio del sottoprogramma

Questo esempio genera una testata come se fosse stata creata con una BSAVE. Deve essere collegata con l'inizio del modulo caricato usando un nome di segmento che lo precede alfabeticamente.

```
AAAA            SEGMENT PARA PUBLIC 'CODE'
        DB      0FDH            ;BSAVE id
        DW      0,0
        DW      TRAILER-HEADER ;Module size
AAAA    ENDS
        PAGE
```

Questo valore per l'ampiezza del modulo diviso per 2, è il valore che deve essere sostituito ad N nel programma BASIC riportato in seguito.

L'allineamento BYTE relativo alla seguente specifica SEGMENT è necessario in quanto l'immagine della memoria verrà caricata come successivo byte che segue la testata fittizia BSAVE senza nessun byte intermedio di riempimento.

```
BASDATSEG SEGMENT BYTE PUBLIC 'CODE'
```

Questo codice viene caricato in una schiera di interi BASIC nel segmento dati del BASIC. Le posizioni relative riportate nella lista dell'assemblatore sono relative all'inizio del sottoprogramma, e non all'inizio del segmento di dati del BASIC a cui realmente puntano CS e DS. Il sottoprogramma deve determinare la sua posizione relativa là dove è stata caricata nel segmento dati del BASIC e successivamente da questa base modificare le posizioni relative dell'assemblatore.

```
          ASSUME CS:BASDATSEG,DS:BASDATSEG
HEADER EQU    $    ;Start of memory image
SUBRT  PROC   FAR
       NOP          ;Adjusts for proper
;                    boundary also allows
;                    replacement by ØCCH,
;                    the INT 3,wich is
;                    DEBUG's breakpoint.
       CALL   SKI  ;This pushes offset
;                    of  'BASE' into stack
BASE:               ;local workarea
WORKA  DW Ø ;First input parameter,
;                    BASIC's ARRAY(3)
WORKB  DW Ø ;Second input parameter,
;                    BASIC's ARRAY(4)
WORKC  DW Ø ;Result,BASIC's ARRAY(5)
SKIP:  POP    BX    ;Now BX knows at wath
;                    offset into BASIC's
;                    data space is 'BASE'
       MOV    AX,WORKA - BASE[BX] ;Get 1st parm
       ADD    AX,WORKB - BASE[BX] ;Add 2nd parm
;                                   to ACC
```

Notare che la memoria locale deve essere indirizzata. Aggiungendo BX si effettua la riassegnazione automatica, cambiando le posizioni relative locali in posizioni relative dal DS (segmento dati del BASIC).

```
       MOV     WORKC-BASE[BX],AX ;Pass to 3rd parm
       RET
SUBRT     ENDP
TRAILER   EQU       $       ;End of memory image
BASDATSEG ENDS
          END
```

Il sottoprogramma di richiamo dal BASIC dovrebbe apparire come segue:

```
10 OPTION BASE 1
20 DEFINT A-Z
30'Definisce gli scalari prima di DIM
40 SUBR1 = 0
50 DIM ARRAY(10)
60'Carica il sottoprogramma nella schiera
70 BLOAD "SUB2.BLO",SUBRT
80'Passa i parametri alla schiera
90 ARRAY(3)=2: ARRAY(4)=3
100'Trova dove inizia la schiera
110 SUBRT=VARPTR(ARRAY(1))
120'I parametri sono nella schiera
130 CALL SUBRT
140 Visualizza il risultato
150 PRINT ARRAY(5)
```

L'esempio successivo ha la stessa funzione del precedente. Tuttavia, i parametri vengono passati mettendo fra parentesi dei nomi di variabile nella specifica CALL.

```
10 OPTION BASE 1
20'Definisce gli scalari prima di DIM
30 SUBRT%=0: PARMC%=0
40 DIM ARRAY%(12)
50'Trova l'inizio della schiera
60 SUBRT%=VARPTR(ARRAY%(1))
70 'Carica il programma nella schiera
80 BLOAD "SUB1.BL0", SUBRT%
90'Passa i parametri al sottoprogramma
100 PARMA% = 2: PARMB% = 3
110'Memorizza il sottoprogramma
120 SUBRT% = VARPTR(ARRAY%(1))
130'Parametro in CALL
140 CALL SUBRT%(Parma%,PARMB%, PARMC%)
150'Visualizza i risultati
160 PRINT PARMC%
```

Il sottoprogramma denominato SUB1 .BLO appare come segue:

```
PARM    STRUC       ;Description of parameter list
SAVEBP  DW      Ø   ;Saves BASIC's BP register
RETOFF  DW      Ø   ;Offset of where to RET to
RETSEG  DW      Ø   ;Segment to return to
PARMC   DW      Ø   ;Offset to third parameter
PARMB   DW      Ø   ;Offset to second parameter
PARMA   DW      Ø   ;Offset to first parameter
PARM    ENDS

AAA     SEGMENT PARA
        DB      ØFDH            ;BSAVE ID
        DW      Ø,Ø
        DW      TRAILER-HEADER  ;Module size
AAA     ENDS

BASDATSEG  SEGMENT  BYTE  PUBLIC 'CODE'
        ASSUME  CS:BASDATSEG,DS:BASDATSEG

HEADER  EQU  $    ;Start of memory image
SUBRT   PROC FAR
        NOP       ;For debugging, can be
;                  replaced by ØCCH (INT 3)
;                  which is the DEBUG
;                  breakpoint
        PUSH BP   ;Save BASIC's BP register
        MOV  BP,SP   ;Set adressability to
;                    parm area on stack
        MOV  SI,[BP].PARMA   ;Get offset to parm1
        MOV  AX,[SI]  ;Get value of parm1
        MOV  SI,[BP].PARMB   ;Get offset to parm2
        MOV  AX,[SI]  ;Add value of parm2 to acc
        MOV  DI,[BP].PARMC   ;Get offset to parm3
        MOV  [DI],AX  ;Pass result to parm3
        POP  BP   ;Restore BASIC's BP
        RET  6    ;Return to BASIC,
;                  throwing away the 3 parms
SUBRT   ENDP
TRAILER EQU  $    ;End of memory image
BASDATASEG ENDS
        END
```

Carica il programma in linguaggio macchina nell'area dati esterna usando il comando BASIC BLOAD. Il sottoprogramma può anche essere messo in memoria mediante la specifica POKE all'inizio della posizione relativa Ø del primo paragrafo dopo il segmento dati del BASIC. Questo segmento ID può essere calcolato sommando al valore del segmento ID del BASIC il massimo numero di paragrafi che il BASIC può indirizzare.

Il BASIC non viene necessariamente assegnato nello stesso segmento di ogni macchina, cosicché per calcolare dove la specifica BLOAD carica il programma può essere usata la seguente procedura:

- Impostare il segmento in uso al segmento Ø con la specifica DEF SEG. Il segmento ID del segmento dati del BASIC è generato nella parte bassa della memoria, segmento Ø, posizione relativa &H510 e &H511. Il tracciato di quest'area della memoria si trova nella sezione 3 del manuale Technical Reference dell'IBM Personal Computer.

- Esegue la funzione PEEK su questi due valori ed assegna il risultato ad una variabile.

- Somma a questo valore il numero massimo di paragrafi che il BASIC può indirizzare. Per difetto questo valore sarà 4Ø96 oppure &H1ØØØ e potrà essere modificato mediante la specifica CLEAR o il parametro /M:.

- Assegna questo valore al nuovo segmento ID con la specifica DEF SEG.

- Carica il sottoprogramma nella posizione relativa Ø di questo segmento.

La codifica BASIC al riguardo sarà:

```
1Ø DEF SEG = Ø  'Look at low memory
2Ø 'Get segment ID
3Ø V = PEEK(&H51Ø) + (256 * PEEK(&H511))
4Ø SEGID = V + &H1ØØØ'Add 4Ø96 paragraphs
5Ø 'Def seg to next segment after basic
6Ø DEF SEG = SEGID
7Ø 'Load into this segment at offset Ø
8Ø BLOAD "SUBROUT.COM" , Ø
```

Caricamento del sottoprogramma come estensione residente del DOS

Il sottoprogramma assemblato può essere caricato sotto DOS come se fosse un comando e richiamato dal BASIC quando necessario.

Considerazioni

- L'area che contiene il sottoprogramma è protetta dal DOS. Non è quindi necessario effettuare la gestione della memoria per garantire la sua integrità.

- Non è necessario che il proprio codice assemblato sia autocaricante.

- Mediante questo metodo possono essere caricati dei moduli di ampie dimensioni. Il DOS riserva la porzione di memoria necessaria al suo caricamento e al caricamento delle sue estensioni, prima di assegnare lo spazio al BASIC.

- Poiché il sottoprogramma, una volta caricato dal BASIC, rimane residente, si può anche compilare un sottoprogramma che possa essere utilizzato da vari programmi. Il sottoprogramma deve essere quindi caricato una sola volta.

- Per usare questo metodo di caricamento del sottoprogramma è necessario avere un assemblatore.

- I sottoprogrammi caricati DOS rimangono residenti fino a che il sistema non venga ripristinato. Se lo spazio in memoria è insufficiente è consigliabile codificare la propria applicazione in modo che esegua un ripristino automatico.

I moduli caricati DOS si compongono essenzialmente di due parti: il sottoprogramma BASIC e la sezione di caricamento. Il modulo viene richiamato come un comando esterno DOS (battere il nome del file senza estensione). Nella operazione di caricamento del sottopogramma devono verificarsi i seguenti passi:

- Assegnazione in memoria della doppia parola chiave che punta al sottoprogramma.

- Informazione al DOS del tempo totale di residenza del modulo (solo la porzione relativa al sottoprogramma).

- Ritorno al DOS mediante INT 27H, che lascia il sottoprogramma residente.

Con il modulo caricato DOS si presenta il problema di dover far conoscere al BASIC l'indirizzo del caricamento in memoria del sottoprogramma. E' possibile selezionare un vettore nel primo K di memoria riservato agli utenti. L'area di comunicazione tra applicazioni (16 byte nelle posizioni &H4F0-4FF nel segmento 0) può inoltre essere usata per contenere un vettore di quattro byte che punti al sottoprogramma. Vedere le relative tavole esplicative nella Sezione 3 del manuale di riferimento dell'IBM Personal Computer.

Il dispositivo di caricamento del sottoprogramma dovrebbe conservare del sottopogramma stesso sia la posizione relativa che il segmento ID in una locazione fissa del segmento 0. Il BASIC può successivamente eseguire, per individuare la posizione del sottoprogramma, una DEF SEG = 0 ed una PEEK nei 4 byte seguenti la posizione relativa &H4F0. Questo è come ricercare il valore di DEF SEG assunto dal BASIC cui si è fatto riferimento precedentemente.

Al fine di evitare copie multiple del sottoprogramma residente, è necessario definire un file .BAT che lo richiami al fine di caricarlo e che richiami il BASIC e le sue applicazioni. Una volta completato il caricamento, l'applicazione dovrebbe richiamare la funzione di ripristino BIOS del sistema e lasciare il sottoprogramma in memoria.

Il file .BAT dovrebbe apparire come segue:

```
Subroutine-name
BASIC application-name
```

La porzione di caricamento del sottoprogramma posiziona il puntatore dell'area di comunicazione e lascia residente il sottoprogramma:

```
SEGZERO SEGMENT AT Ø
        ORG 4FØH    ;Intra-application
;                          communication area
COMAREA_OFF  DW  ?      ;Vector pointing to
COMAREA_SEG  DW  ?      ;permanetn resident
;                          BASIC subroutine.
SEGZERO ENDS

GRP     GROUP    AAAA,ZLOADER

AAAA    SEGMENT 'CODE' ;Identifies the start
;                              of load module
AAAA    ENDS           ;Forces class "code"
;                          to load before "data".

SUBDAT  SEGMENT PARA PUBLIC 'DATA'
;Insert local data to be used by subroutine here
SUBDAT ENDS
```

```
SUBSEG  SEGMENT PARA PUBLIC 'CODE'
        ASSUME  CS:SUBSEG
SUBRT   PROC    FAR     ;BASIC entry point
        PUSH    BP      ;Save BASIC's registers
        MOV     BP,SP   ;Get
;                        addressability
;                        to parms on stack
        PUSH    ES      ;Save BASIC'S
;                        registers
        MOV     AX,SUBDAT
        MOV     ES,AX   ;Set
;                        addressability
;                        to local data
        ASSUME  ES:SUBDAT ;DS still points
;                        to BASIC's data area
;insert the BASIC subroutine here
        POP     ES      ;Restore BASIC's
        POP     BP      ;Registers
        RET     n       ;Return to BASIC,
;                        discard parms
;                        from stack
SUBRT   ENDP
SUBSEG  ENDS
ZLOADER SEGMENT BYTE 'ZLOAD'
        ASSUME  CS:ZLOADER,SS:ZSTACK
;                        Stack not available
;                        to subroutine
ENDRES  EQU     $           ;End BASIC subroutine.
;                            The rest
;                            of this subroutine
;                            does not stay resident
LOADER  PROC    FAR         ;Entry from DOS
        PUSH    ES          ;Set up stack to
;                            contain vector so
        XOR     AX,AX       ;'RET' will return
;                            to INT 27H
;                            instruction at
;                            offset Ø in PSP.
```

```
        PUSH    AX
        MOV     BYTE PTR ES:1,27H
;                       Change INT 20H
;                       to INT 27H at start
;                       of PSP
        XOR     AX,AX ;Clear a register
        MOV     ES,AX ;Set extra segment
                       to 0
        ASSUME  ES:SEGZERO
;                      ;Seg reg points to
;                       low memory
        MOV     COMAREA_OFF, OFFSET SUBRT
;                       Pass offset to entry
;                       point of subroutine
        MOV     COMAREA-SEG,SEG SUBRT
;                       Pass subroutine
;                       segment
        MOV     DX,OFFSET GRP:ENDRES+100H
;                       Pass offset of end
;                       of resident code.
;                       At exit, CS will
;                       point to the psp
;                       which has the exit
;                       instruction 27H.
;                       This sets DX to the
;                       offset from the CS
;                       that is the end of
;                       the resident code.
        RET            ;Return to DOS, via
;                       INT 27H in PSP
;                       leaving subroutine
;                       resident.
LOADER ENDP
ZLOADER ENDS
ZSTACK  SEGMENT PARA STACK 'STACK'
        DB      16 DUP ("STACK")
                       ;Stack used during
;                       procedure only
ZSTACK  ENDS
        END     LOADER
```

Il sottoprogramma viene assemblato e collegato ed è ora pronto per l'esecuzione con il comando esterno DOS .EXE. Una volta che l'applicazione BASIC è caricata il sottoprogramma può essere richiamato con:

```
20 DEFINT A-Z
30 DEF SEG=0 'Look at low memory at
40 'inter application communication area.
50 SUBOFF=PEEK(&H4F0)+(256*PEEK(&H4F1))
60 SUBSEG=PEEK(&H4F2)+(256*PEEK(&H4F3))
```

Il sottoprogramma può quindi venire richiamato con:

```
100 DEF SEG = SUBSEG
110 CALL SUBOFF(PARMA,PARMB,PARMC)
120 DEF SEG
```

Quando l'esecuzione è finita, l'applicazione può ripristinare il sistema, rimuovendo di conseguenza il sottoprogramma:

```
1000 DEF SEG = &HF000: SUBOFF =     &HFFF0:
     CALL SUBOFF
```

Come il BASIC supporti i sottoprogrammi in linguaggio macchina

Il BASIC ha due possibilità di richiamare i sottoprogrammi in linguaggio macchina dall'applicazione dell'utente: mediante la specifica CALL e la funzione USR. La specifica CALL è il procedimento maggiormente raccomandato. Fare riferimento alla funzione USR per ulteriori ragguagli.

Quando il sottoprogramma viene richiamato si verifica che:

- Inizialmente i registri di segmento DS, ES, e SS vengono impostati sullo stesso valore, mentre il segmento ID viene impostato al segmento dati del BASIC. Questo è il valore assunto per difetto per DEF SEG.

- Il registro CS contiene il segmento ID dell'ultimo valore specificato per DEF SEG. Per difetto questo è il segmento ID del BASIC.

- Inizialmente il registro (SP) che punta alla memoria indica che si possiede un minimo di 8 parole disponibili per essere utilizzate dal sottoprogramma. Ulteriore spazio è fornito in memoria per l'uso delle interruzioni (TIMER) e riferimenti DOS e BIOS provenienti da sottoprogrammi in Assembler. Se il

sottoprogramma necessita di maggior spazio in memoria è consigliabile creare una propria area di 128 byte riservata al sistema prima delle specifiche del sottoprogramma stesso.

Prima del ritorno al BASIC il segmento (SS) ed il puntatore (SP) dell'area devono avere lo stesso valore che avevano al momento in cui il salto programma è stato richiamato dal BASIC. Fare attenzione al mantenimento di questi valori se si vuole attivare l'area di memoria (stack).

Se il parametro di immissione è una stringa, il valore passato è la posizione relativa nel segmento dati del BASIC di un'area di 3 byte chiamata descrittore della stringa.

Byte Ø contiene la lunghezza della stringa (da Øa 255).

Byte 1 contiene gli 8 bit inferiori della posizione relativa della stringa nell'area del segmento dati del BASIC.

Byte 2 contiene gli 8 bit superiori della posizione relativa della stringa nell'area del segmento dati del BASIC.

> **Attenzione.** Il sottoprogramma non deve modificare il contenuto di nessuno dei 3 byte del descrittore della stringa.

Il sottoprogramma può modificare il contenuto della stringa ma non la sua lunghezza.

Se il sottoprogramma modifica una stringa, questo può indirettamente modificare il programma. L'esempio che segue modifica la stringa "TEST" nel programma BASIC.

```
A$ = "TEXT"
CALL SUBRT(A$)
```

Tuttavia il successico esempio non modifica il programma, poiché la concatenazione della stringa fa sì che il BASIC copi la stringa stessa in una area in cui la stringa possa essere modificata senza interessare il testo originale.

```
A$ = "BASIC" + ""
   CALL SUBRT(A$)
```

Per ritornare dal sottoprogramma, bisogna:

- Abilitare ogni interruttore disabilitato dal sottoprogramma.
- Usare una istruzione RET poiché ogni sottoprogramma richiamato dal BASIC è un richiamo FAR.
- Ripristinare tutti i registri di segmento ed il puntatore (SP). Tutti gli altri registri possono rimanere alterati.

La specifica CALL

L'esecuzione di una specifica CALL genera quanto segue:

- L'indirizzo di ogni variabile presente nella lista viene immesso in memoria. L'indirizzo viene specificato mediante 2 byte nel segmento dati del BASIC. I valori possono essere ritornati al BASIC attraverso i parametri modificando i valori delle variabili a cui si punta mediante la lista dei parametri.

Il sottoprogramma richiamato da CALL deve sapere quanti parametri sono passati. I parametri sono determinati sommando a BP una posizione relativa positiva dopo che il sottoprogramma richiamato ha mosso il puntatore di memoria in BP. La prima istruzione del sottoprogramma potrebbe essere:

```
PUSH BP          ;SAVE BP
MOV BP, SP       ;MOVE SP TO BP
```

La posizione relativa in memoria di qualsiasi parametro viene calcolata come segue:

```
OFFSET from BP = 2 *(N - M) + 6
```

dove N è il numero totale di parametri passati ed M è la posizione di quel parametro specifico nella lista di parametri di CALL. M può essere un valore tra 1 ed N.

Nell'Assemblatore Macro l'operazione "STRUC" è un modo convenzionale di definire il contenuto della memoria, mettendo in evidenza la posizione dei parametri.

```
FRAME     STRUC
SAVE BP   DW      ?   ;Saved BASIC's bp
RET OFF   DW      ?   ;Offset to RET to BASIC
RET SEG   DW      ?   ;Seg to RET to BASIC
PARMN     DW      ?   ;Offset to Nth parameter
          .
          .
          .
PARM1     DW      ?   ;Offset to first parameter
FRAME     ENDS

PARMSIZE EQU OFFSET PARM1-OFFSET RETSEG ;
  (To be used as operand on RET)
```

Attenzione: Assicurarsi che il parametro nella specifica CALL corrisponda come numero, tipo e lunghezza al parametro che il sottoprogramma si aspetta di ricevere; per esempio, un valore a precisione doppia (8 byte).

- Il controllo viene trasferito al sottoprogramma in linguaggio macchina usando il segmento indicato nella prima specifica DEF SEG e la posizione relativa nella specifica CALL.

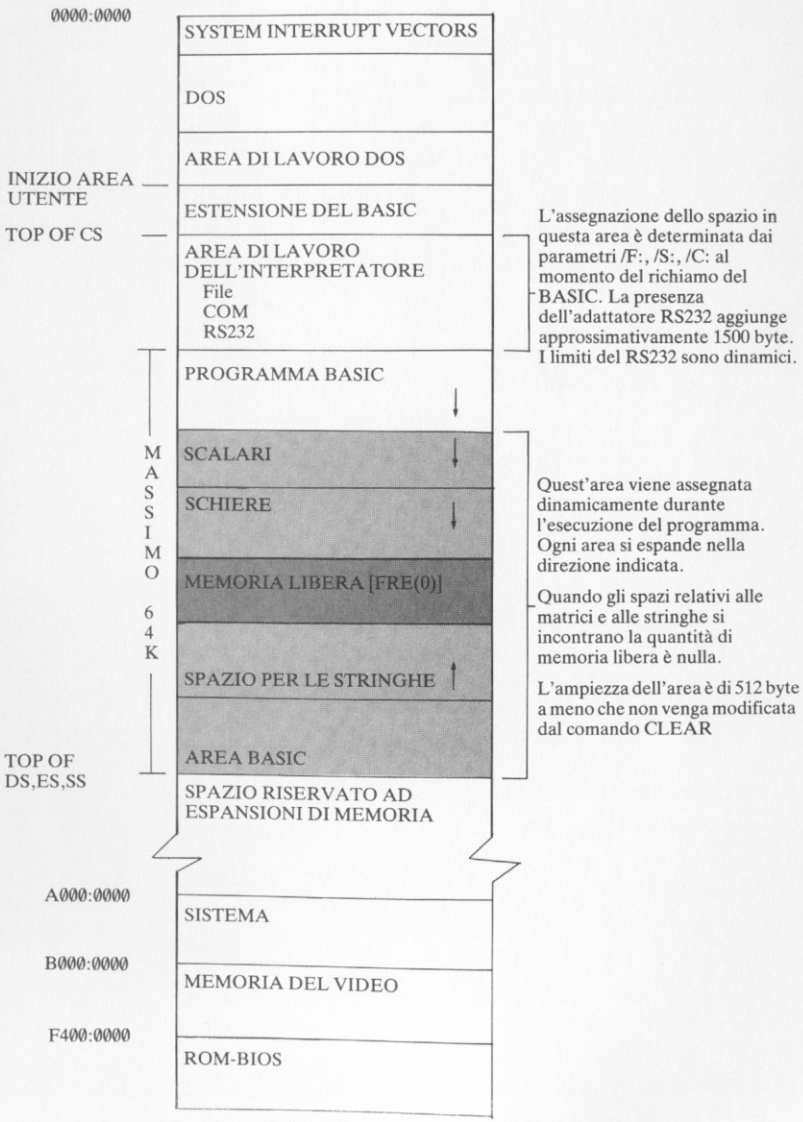
- Il valore fornito dal segmento ID specificato nel registro CS e la posizione relativa vengono immessi nella memoria.

- Quando un sottoprogramma viene richiamato usando una specifica CALL che ne indica i parametri, il sottoprogramma deve ritornare mediante una RET N, dove N è un valore pari a due volte il numero di parametri della lista. Questo imposta la memoria nella condizione di inizio della sequenza di richiamo.

- La specifica CALL non ha sempre necessariamente dei parametri. Inoltre se il sottoprogramma è generato in una schiera di interi, è possibile passare i parametri alla matrice stessa invece che mediante la specifica CALL.

Mappa della memoria

Questa e la mappa della memoria per il BASIC avanzato ed il BASIC su disco. Il DOS e le estensioni del BASIC non sono presenti nel BASIC su cassetta. Gli indirizzi sono in forma esadecimale SEGMENT:OFFSET.



Note:

Appendice C. Comunicazioni

Questa appendice descrive le specifiche BASIC necessarie per le comunicazioni asincrone RS232 con altri elaboratori e unità periferiche.

Come aprire un file per le comunicazioni

La specifica OPEN "COM... alloca una memoria di transito per l'I/E nello stesso modo di OPEN per i file su disco. Vedere la specifica OPEN "COM...

I/E della comunicazione

Poiché ogni adattatore per le comunicazioni viene aperto come un file, tutte le specifiche di immissione/emissione valide per i file su disco sono valide anche per le comunicazioni.

Le specifiche di immissione sequenziale per le comunicazioni corrispondono a quelle per i file su disco e sono:

INPUT #
LINE INPUT #
INPUT $

Le specifiche di emissione sequenziale per le comunicazioni corrispondono a quelle per i file su disco e sono:

PRINT #
PRINT # USING
WRITE #

Fare riferimento alle sezioni INPUT e PRINT per dettagli sulla sintassi della codifica e l'uso.

GET e PUT per i file di comunicazione

Le specifiche GET e PUT sono leggermente diverse per i file di comunicazione rispetto a quelle per i file su disco. Si usano per l'I/E a lunghezza fissa. Invece di specificare il numero di record da scrivere o leggere, si indica il numero di byte da trasferire nella o dalla memoria di transito del file. Questo numero non può superare il valore impostato dall'opzione LEN della specifica OPEN "COM... Per ulteriori informazioni fare riferimento alle specifiche GET e PUT.

Funzioni di I/E

L'aspetto più difficile delle comunicazioni asincrone è quello di riuscire ad elaborare i caratteri alla stessa velocità con cui vengono ricevuti. A velocità di 1200 bps o superiori, può rendersi necessario dover sospendere la trasmissione dei caratteri dall'altro elaboratore in modo da poterli elaborare. Ciò si ottiene inviando XOFF (CHR$(19)) all'altro elaboratore e XON (CHR$(17)) quando si è pronti per continuare. XOFF informa l'altro elaboratore di sospendere l'invio, mentre XON di riprenderlo.

> **Nota:** Questa è una convenzione comune, ma non universale. Se valida, ciò dipende dal protocollo esistente con l'altro elaboratore o con la periferia.

Il BASIC avanzato o su disco fornisce tre funzioni che aiutano a determinare il verificarsi di una condizione di accumulo di lavoro. Esse sono:

LOC(f)
LOF(f)
EOF(f)

> **Nota:** Può verificarsi nella memoria di transito per le comunicazioni un errore del tipo:
> **Comunication buffer overflow** se si tenta la lettura quando la memoria di transito di immissione è già piena (cioè, quando LOF(f) ritorna Ø).

Funzione INPUT$

E' preferibile utilizzare la funzione INPUT$ invece delle specifiche INPUT# e LINE INPUT # per la lettura dei file di comunicazione, poiché tutti i caratteri ASCII hanno un significato nelle comunicazioni. INPUT # è meno consigliabile in quanto l'immissione si arresta quando incontra una virgola o un ritorno a margine. LINE INPUT # arresta l'immissione quando trova un ritorno a margine.

INPUT$ consente che tutti i caratteri letti vengano assegnati ad una stringa. INPUT$*(n,f)* ritorna *n* caratteri dal file #f. Le seguenti specifiche sono valide per la lettura di un file di comunicazione:

```
10 WHILE NOT EOF(1)
20 A$=INPUT$(LOC(1),#1)
   .
   .
   .
(process data returned in A$)
   .
   .
   .
100 WEND
```

Quando nella memoria di transito vi sono dei caratteri, la riga 20 li assegna a A$ e vengono elaborati. Se nella memoria vi sono più di 255 caratteri, i caratteri ritornati sono solo 255 per evitare un'eccedenza di stringa. Poiché in questo caso EOF(1) è (0), l'immissione continua fino a quando la memoria di transito di immissione è vuota. Questa procedura è semplice, concisa e veloce.

Per elaborare i caratteri velocemente, si deve evitare, se possibile, di esaminare ogni carattere quando lo si riceve. Se si cercano dei caratteri speciali (come quelli di controllo) si può utilizzare la funzione INSTR per trovarli nella stringa di immissione.

Esempio di programma

Il seguente programma consente l'utilizzo del Personal Computer IBM come un terminale convenzionale, con velocità doppia di trasmissione. Questo programma assume una linea di 300 bps ed una memoria di transito di immissione di 256 byte (non è stata usata l'opzione /C: nel comando BASIC).

```
10 REM    dumb terminal example
20 'set screen to monochrome text mode
30 '    and set width to 40
40 SCREEN 0,0: WIDTH 40
50 'turn off key display; clear screen
60 '   make sure all files are closed
70 KEY OFF: CLS: CLOSE
80 'define numeric variables as integer
90 DEFINT A-Z
100 'define true and false
110 FALSE=0: TRUE=NOT FALSE
120 'define the XON, XOFF characters
130 XOFF$=CHR$(19): XON$=CHR$(17)
140 'open communications to file number 1,
150 ' 300bps, EVEN parity, 7 data bits
160 OPEN "COM1:300,E,7" AS #1
170 'use screen as a file, just for fun
180 OPEN "SCRN:" FOR OUTPUT AS #2
190 'turn cursor on
200 LOCATE ,,1
400 PAUSE=FALSE: ON ERROR GOTO 9000
490 '
500 'send keyboard input to com line
510 B$=INKEY$: IF B$<>"" THEN PRINT #1,B$;
520 'if com buffer is empty, check key in
530 IF EOF(1) THEN 510
540 'if buffer more than 1/2 full, then
550 'set PAUSE flag to say input suspended,
560 'send XOFF to host to stop transmission
570 IF LOC(1)>128 THEN PAUSE=TRUE
575 PRINT #1,XOFF$
```

```
580 'read contents of com buffer
590 A$=INPUT$(LOC(1),#1)
600 'remove linefeeds to avoid double spaces
610 'when input displayed on screen
620 LFP=0
625 'look for linefeed
630 LFP=INSTR(LFP+1,A$,CHR$(10))
640 IF LFP>0 THEN MID$(A$,LFP,1)=" "
645 GOTO 630
650 'display com input, and check for more
660 PRINT#2,A$;: IF LOC(1)>0 THEN 570
670 'if trasmission suspended by XOFF,
680 'resume by sending XON
690 IF PAUSE THEN PAUSE=FALSE
695 PRINT#1,XON$;
700 'check for keyboard input again
710 GOTO 510
8999 'if error, print its number and retry
9000 PRINT "ERROR NO.";ERR: RESUME
```

Note sul programma

- Le comunicazioni asincrone implicano l'I/E di carattere, contrapposta a quella di riga o blocco. Perciò, tutte le specifiche PRINT (per i file di comunicazione o su video) terminano con un punto e virgola. Questo arresta il ritorno a margine emesso normalmente alla fine della lista di valori da stampare.

- La riga 90, dove tutte le variabili numeriche sono definite come interi, viene codificata in quanto qualsiasi programma che tenti l'ottimizzazione della velocità dovrà usare, se possibile, nelle iterazioni, dei contatori per interi.

- Notare a riga 510 che INKEY$ fornisce una stringa nulla se non vi sono caratteri in sospeso.

Operazione dei segnali di controllo

L'emissione mediante l'Adattatore per le comunicazioni asincrone è conforme allo stadard EIA RS232-C per l'interfaccia tra gli strumenti del terminale dati (DTE) e quelli della comunicazione dati (DCE). Questo standard definisce un numero di segnali di controllo trasmessi o ricevuti dal Personal Computer IBM per controllare lo scambio di dati con un altro elaboratore o periferia. Questi segnali hanno un voltaggio DC e possono essere ON (maggiori di +3 volt) o OFF (minori di –3 volt). Per ulteriori dettagli, vedere il manuale di riferimenti tecnici del PC.

Controllo dei segnali di emissione con OPEN

Quando si avvia il BASIC sul Personal Computer IBM, le linee RTS (Request to Send) e DTR (Data Terminal Ready) sono OFF. Quando si esegue una specifica OPEN "COM..., entrambe le linee vengono attivate. Tuttavia, è possibile specificare l'opzione **RS** nella specifica OPEN "COM... per sopprimere il segnale RTS. Le linee restano attive fino alla chiusura del file di comunicazione (con CLOSE, END, NEW, RESET, SYSTEM o RUN senza l'opzione **R).** Anche se la specifica OPEN "COM... ha esito negativo a causa di un errore, la linea DTR (e la linea RTS, se applicabile) viene attivata e resta attiva. Ciò consente una nuova esecuzione di OPEN, senza dover usare una specifica CLOSE.

Uso dei segnali di controllo di immissione

Normalmente, se una delle linee CTS o DSR è disattivata, la specifica OPEN "COM... non viene eseguita. Dopo un secondo il BASIC produce un errore **Device timeout** (codice di errore 24). La rilevazione di segnale di ricezione può essere ATTIVA o NON ATTIVA; non ha effetti sull'operazione del programma.

Tuttavia, è possibile indicare come queste linee devono essere testate, usando le opzioni RS, CS, DS e CD nella specifica OPEN "COM....

Se uno dei segnali da provare viene disattivato durante l'esecuzione del programma, le specifiche I/E associate al file di comunicazione non funzionano. Ad esempio, se si esegue una specifica PRINT≠ dopo che la linea CTS o DSR è stata disattivata, si verifica un errore **Device fault** (codice 25) o **Device Timeout** (codice 24). Le linee RTS e DTR restano attive anche se si verifica un errore.

Si può eseguire una prova di scollegamento di linea usando la funzione INP per leggere i bit del registro di stato del MODEM nell'Adattatore per le comunicazioni asincrone.

Prova dei segnali di controllo del modem

L'Adattatore per le comunicazioni asincrone rileva quattro segnali di controllo dell'immissione. Sono i segnali CTS e DSR (descritti precedentemente), la rilevazione del segnale di linea ricevuta (piedino 8), l'indicatore di suono (piedino 22). E' possibile indicare come si vuole eseguire la prova delle linee CTS, DSR e CD con la specifica OPEN "COM... L'indicatore di suono non viene utilizzato dalla funzione di comunicazione nel BASIC.

Se è necessario eseguire la prova di questi segnali in un programma, controllare i bit corrispondenti a questi segnali nel registro di stato del modem dell'Adattatore per le comunicazioni asincrone. Per leggere gli otto bit di questo registro si usa la funzione INP. Usare INP (&H3FE) per leggere il registro di un adattatore per le comunicazioni non modificato e INP (&H2FE) per leggere quello di un adattatore modificato.

Vedere il manuale di riferimenti tecnici del PC per una descrizione di quali bit del registro di stato corrispondano ai segnali di controllo. E' anche possibile utilizzare i bit Delta di questo registro per determinare se sono apparsi dei segnali transienti su una delle linee di controllo. Notare che affinché un segnale di controllo abbia significato, il piedino corrispondente a quel segnale deve essere collegato mediante cavo al modem o all'altro elaboratore.

E' anche possibile eseguire una prova dei bit del registro di stato della linea dell'Adattatore per le comunicazioni asincrone. Usare INP(&H3FD) per accedere a questo registro di un adattatore per le comunicazioni non modificato e INP(&H2FD) per accedervi se si tratta di un adattatore modificato. I bit vengono descritti nel manuale di riferimenti tecnici del PC. E' possibile utilizzare questi bit per determinare quali tipi di errore si siano verificati nella ricezione di caratteri dalla linea di comunicazione o se sia stato rilevato un segnale di interruzione.

Controllo diretto dei segnali di controllo di emissione

E' possibile controllare direttamente i segnali di controllo RTS o DTR da un programma BASIC con una specifica OUT. Lo stato (attivato o disattivato) di questi segnali è controllato dai bit del registro di controllo del MODEM nell'Adattatore per le comunicazioni asincrone. L'indirizzo di questo registro è &H3FC in un adattatore per le comunicazioni non modificato e &H2FC in un adattatore modificato. Il manuale di riferimenti tecnici del PC descrive quale di questi bit corrisponda ai segnali.

Inoltre è possibile modificare i bit del registro di controllo della linea nell'Adattatore per le comunicazioni asincrone. Prestare molta attenzione se si modificano questi bit, poiché la maggior parte dei bit di questo registro è stata impostata dal BASIC all'esecuzione della specifica OPEN e la modifica di un bit potrebbe causare un errore nelle comunicazioni.

Il registro di controllo della linea ha indirizzo &H3FB in un adattatore per le comunicazioni non modificato e &H2FB in un adattatore modificato.

Se si modificano i bit nel registro di controllo del MODEM o in quello della linea è necessario prima leggere il registro (con una funzione INP) e poi riscriverlo con solo il bit pertinente o i bit modificati.

Un bit da verificare nel registro di controllo della linea è il bit 6, quello che definisce l'interruzione. Questo bit consente di produrre un segnale di interruzione sulla linea di invio della comunicazione. Spesso si usa un'interruzione per segnalare ad un elaboratore remoto di arrestare la trasmissione. Generalmente dura mezzo secondo. Per produrre questo segnale è necessario attivare l'interruzione, attendere il segnale di interruzione e poi disattivare il bit. Le seguenti specifiche BASIC producono un segnale di interruzione della durata di circa mezzo secondo in un adattatore per le comunicazioni non modificato.

```
10 IC%=INP(&H3FB)
20 'get contents of modem register
30 IZ%=IC% OR &H4040 'turn ON the Set Break bit
50 OUT &H3FB,IZ%
60 'transmit to modem control register
70 FOR I=1 TO 500: NEXT I
80 'delay half a second
90 OUT &H3FB,IC% 'turn Set Break bit OFF in register
100 'turn Set Break bit OFF in register
```


Errori di comunicazione

Nei file di comunicazione, gli errori si verificano nel seguente ordine:

1) All'apertura del file:

   a. **Device timeout** se manca uno dei segnali da provare (CTS, DSR e CD).

2) Durante la lettura dei dati:

a. **Com buffer overflow** se si verifica un accumulo di lavoro.

b. **Device I/O error** per errori di accumulo lavoro, interruzione, parità o frammentazione

c. **Device fault** se si perdono i segnali DSR o CD.

3) Durante la scrittura dei dati:

a. **Device fault** se si perdono i segnali CTS, DSR o CD in un'interruzione dello stato del modem mentre il BASIC sta eseguendo qualcos'altro.

b. **Device timeout** se si perdono i segnali CTS DSR o CD nell'attesa di immettere i dati nella memoria di transito di emissione.

Appendice D. Codici carattere ASCII

La seguente tabella elenca tutti i codici ASCII (in decimali) ed i relativi caratteri. E' possibile visualizzare questi caratteri con PRINT CHR$*(n)* dove *n* è il codice ASCII. La colonna "Carattere di controllo" elenca le interpretazioni standard dei codici ASCII dallo Ø al 31 (usati generalmente per le funzioni di controllo o le comunicazioni).

E' possibile immettere ognuno di questi caratteri da tastiera premendo e tenendo premuto il tasto Altrn, poi premendo le cifre del codice ASCII sulla tastiera numerica. Notare che alcuni di questi codici hanno un significato speciale nell'editore di programmi BASIC, poiché questo usa la propria interpretazione dei codici e non visualizza i caratteri speciali qui elencati.

ASCII Value	Character	Control Character	ASCII Value	Character
000	(null)	NUL	032	(space)
001	☺	SOH	033	!
002	☻	STX	034	"
003	♥	ETX	035	#
004	♦	EOT	036	$
005	♣	ENQ	037	%
006	♠	ACK	038	&
007	(beep)	BEL	039	'
008	◘	BS	040	(
009	(tab)	HT	041	)
010	(!ine feed)	LF	042	*
011	(home)	VT	043	+
012	(form feed)	FF	044	,
013	(carriage return)	CR	045	-
014	♫	SO	046	.
015	☼	SI	047	/
016	►	DLE	048	0
017	◄	DC1	049	1
018	↕	DC2	050	2
019	!!	DC3	051	3
020	¶	DC4	052	4
021	§	NAK	053	5
022	▬	SYN	054	6
023	↨	ETB	055	7
024	↑	CAN	056	8
025	↓	EM	057	9
026	→	SUB	058	:
027	←	ESC	059	;
028	(cursor right)	FS	060	<
029	(cursor left)	GS	061	=
030	(cursor up)	RS	062	>
031	(cursor down)	US	063	?

ASCII Value	Character	ASCII Value	Character
064	@	096	`
065	A	097	a
066	B	098	b
067	C	099	c
068	D	100	d
069	E	101	e
070	F	102	f
071	G	103	g
072	H	104	h
073	I	105	i
074	J	106	j
075	K	107	k
076	L	108	l
077	M	109	m
078	N	110	n
079	O	111	o
080	P	112	p
081	Q	113	q
082	R	114	r
083	S	115	s
084	T	116	t
085	U	117	u
086	V	118	v
087	W	119	w
088	X	120	x
089	Y	121	y
090	Z	122	z
091	[	123	{
092	\	124	|
093	]	125	}
094	^	126	~
095	_	127	⌂

ASCII Value	Character
128	Ç
129	ü
130	é
131	â
132	ä
133	à
134	å
135	ç
136	ê
137	ë
138	è
139	ï
140	î
141	ì
142	Ä
143	Å
144	É
145	æ
146	Æ
147	ô
148	ö
149	ò
150	û
151	ù
152	ÿ
153	Ö
154	Ü
155	¢
156	£
157	¥
158	Pt
159	ƒ

ASCII Value	Character
160	á
161	í
162	ó
163	ú
164	ñ
165	Ñ
166	ª
167	º
168	¿
169	⌐
170	¬
171	½
172	¼
173	¡
174	«
175	»
176	▒
177	▒
178	▓
179	│
180	┤
181	╡
182	╢
183	╖
184	╕
185	╣
186	║
187	╗
188	╝
189	╜
190	╛
191	┐

ASCII Value	Character
192	└
193	┴
194	┬
195	├
196	─
197	┼
198	╞
199	╟
200	╚
201	╔
202	╩
203	╦
204	╠
205	═
206	╬
207	╧
208	╨
209	╤
210	╥
211	╙
212	╘
213	╒
214	╓
215	╫
216	╪
217	┘
218	┌
219	■
220	▄
221	▌
222	▐
223	▀

ASCII Value	Character
224	α
225	ß
226	Γ
227	π
228	Σ
229	σ
230	µ
231	τ
232	Φ
233	Θ
234	Ω
235	δ
236	∞
237	∅
238	∈
239	∩
240	≡
241	±
242	≥
243	≤
244	⌠
245	⌡
246	÷
247	≈
248	°
249	∙
250	·
251	√
252	n
253	2
254	■
255	(blank 'FF')

Codici estesi

Per alcuni tasti o combinazioni di tasti che non possono essere rappresentati con il codice ASCII standard, la variabile di sistema INKEY$ fornisce un codice esteso. Come primo carattere di una stringa di due caratteri viene fornito un carattere nullo (codice ASCII 000). Se INKEY$ riceve una stringa di due caratteri, tornare indietro ed esaminare il secondo carattere per determinare quale tasto sia stato realmente premuto. Generalmente, ma non sempre, questo secondo codice è di scansione per il tasto primario premuto. I codici ASCII (in decimali) per questo secondo carattere ed i tasti associati vengono elencati alla pagina seguente.

Secondo codice	Significato
3	(carattere nullo) NUL
15	(prima tabulazione)
16-25	Altrn Q, W, E, R, T, Y, U, I, O, P
30-38	Altrn A, S, D, F, G, H, J, K, L
44-50	Altrn Z, X, C, V, B, N, M
59-68	Tasti funzionali F1-F10 (se disabilitati come tasti di programma)
71	Ritorno cursore (Rit Curs)
72	Cursore verso l'alto
73	Rit Pg.
75	Cursore verso sinistra
77	Cursore verso destra
79	Fine
80	Cursore verso il basso
81	Av Pag
82	Ins
83	Canc
84-93	F11-F20 (maiuscola F1-F10)
94-103	F21-F30 (Ctrl F1-F10)
104-113	F31-F40 (Altrn F1-F10)
114	Ctrl-Stampa
115	Ctrl-cursore a sinistra (parola prec.)
116	Ctrl-cursore a destra (parola succ.)
117	Ctrl-Fine
118	Ctrl-Av Pag
119	Ctrl-Rit Curs
120-131	Altrn - 1,2,3,4,5,6,7,8,9,0-,=
132	Ctrl-Rit Pag

Note:

Appendice E. Codici di Scansione

Tasto	Codice in esadecimale	Tasto	Codice in esadecimale
ESC	Ø1	$\|\leftarrow$ $\rightarrow\|$	OF
! 1	02	Q	10
@ 2	03	W	11
# 3	04	E	12
$ 4	05	R	13
% 5	06	T	14
^ 6	07	Y	15
& 7	08	U	16
* 8	09	I	17
(9	0A	O	18
) 0	0B	P	19
— -	OC	{[	1A
+ =	OD	}]	1B
←	OE	↵	1C

Tasto	Codice in esadecimale	Tasto	Codice in esadecimale
Ctrl	1D	| \	2B
A	1E	Z	2C
S	1F	X	2D
D	20	C	2E
F	21	V	2F
G	22	B	30
H	23	N	31
J	24	M	32
K	25	< ,	33
L	26	> .	34
: ;	27	? /	35
" '	28	↑ Right	36
~ `	29	Stampa*	37
↑ Left	2A	Altrn	38

Tasto	Codice in esadecimale	Tasto	Codice in esadecimale
Sp bar	39	7 Rit Curs	47
Bloc maius	3A	8 ↑	48
F1	3B	9 Rit Pag	49
F2	3C	–	4A
F3	3D	4 ←	4B
F4	3E	5	4C
F5	3F	6 →	4D
F6	40	+	4E
F7	41	1 Fine	4F
F8	42	2 ↓	50
F9	43	3 Av Pag	51
F10	44	0 Ins	52
Bloc Num	45	Canc	53
Bloc Scorr	46		

Note:

Glossario

Questa parte del libro spiega molti dei termini tecnici che si incontrano programmando in BASIC.

abilitato. Lo stato dell'unità di processo che permette certi tipi di interruzioni (interrupts).

accesso sequenziale. Un metodo di accesso nel quale i records sono ricercati nello stesso ordine con il quale sono stati scritti.

adattatore. E' un dispositivo per collegare parti di macchina fra loro.

aggiornamento. Modifica, usualmente un file principale con informazioni attuali.

alfanumerico. Indica un insieme di caratteri composto da numeri e lettere.

algoritmo. E' un insieme di calcoli definito per risolvere un problema con un numero finito di passi.

allineare. Allineare orizzontalmente o verticalmente in modo da produrre un determinato formato.

allocare. Indica l'assegnazione di una risorsa ad un programma (un file su dischetto, una parte di memoria, ecc.).

analisi degli errori. Cercare di eliminare gli errori di programma.

area visualizzata. Nei grafici, un definito sottoinsieme o piccola area della superficie visibile.

argomento. Indica un valore trasferito da un programma ad una funzione.

ASCII. American National Standard Code for Information Interchange. Codice standard usato per scambiare informazioni fra sistemi di elaborazione dati. Consiste di caratteri di controllo e caratteri grafici.

asincrono. Dicesi di un qualsiasi dispositivo che non usa alcuna particolare relazione temporale rispetto alla esecuzione di istruzioni di programma.

attributo. Un valore numerico che descrive il colore di un punto sullo schermo.

avanzamento carrello (LF). Indica il carattere che causa sulla stampante o schermo l'avanzamento alla corrispondente posizione di stampa sulla riga successiva.

avanzamento pagina (FF). Il carattere che causa l'avanzamento pagina sulla stampante.

avvolgente. La tecnica per visualizzare elementi le cui coordinate giacciono al di fuori dell'area visualizzabile.

baud. Unità di velocità per la trasmissione dati. Indica il numero di segnali elementari per secondo.

binario. Pertinente ad una condizione che ha due valori possibili o stati; oppure ad una numerazione in base 2.

bit. Un numero binario.

bps. Bit per secondo.

byte. Rappresentanzioni di un carattere con 8 bit.

caduta. Condizione accidentale che impedisce ad un dispositivo di funzionare.

cambiamento di rappresentazione. La traduzione dei valori di coordinate da coordinate assolute (definite dalla specifica WINDOW).

campo. In un record indica un'area usata per una particolare categoria di dati.

canale. Il cammino attraverso il quale i segnali possono essere inviati (per es. un canale di dati o di emissione).

carattere. Una lettera o numero o simbolo usato come parte della organizzazione, controllo o rappresentazione dei dati. Una sequenza organizzata di dati dicesi stringa.

carattere alfabetico. Indica una lettera dell'alfabeto.

carattere di controllo. Carattere la cui presenza in un particolare contesto inizia, modifica o conclude una operazione di controllo. L'operazione di controllo è una azione che interessa la registrazione, l'elaborazione, la trasmissione o l'interpretazione dei dati; esempio: ritorno carrello, fine trasmissione ecc.

carattere visualizzato. La parte di area visualizzata che è il carattere stesso.

caratteri. Insieme di caratteri di specificata ampiezza e stile.

chiamare. Attivare una procedura.

chiamata. Caricare un programma o sottoprogramma specificando le condizioni iniziali e lanciandone l'esecuzione.

coda. Lista di elementi in attesa di essere serviti; il primo elemento accodato è anche il primo ad essere servito.

codifica. Qualsiasi organizzazione di dati o specifiche di programmazione che possono essere interpretate da un elaboratore; anche: codificare un programma; oppure, in modo meno preciso, indica uno o più programmi o parte di essi.

colore ricoprente. Metodo di colorazione entro un'area limitata con un colore ricoprente.

commento. Una specifica usata per documentare un programma ossia per rendere più facilmente leggibile la codifica.

compattare. Dicesi del BASIC quando comprime le stringhe di spazi conservando solo i dati validi.

complemento. Indica un numero ottenuto da un altro sottraendo un determinato valore; indica anche "l'opposto".

complemento al due. Un metodo per rappresentare i numeri negativi nel sistema di numerazione binaria.

compressione. Manipolare i dati in modo che occupino il minor spazio possibile.

comunicazione. Trasmissione e ricezione dei dati.

concatenazione. Operazione che unisce due stringhe insieme, secondo un ordine specificato, per formare una singola stringa di lunghezza uguale alla somma delle due.

conversione. Una tecnica per convertire dati in una desiderata forma; per esempio le lettere minuscole possono essere convertite in maiuscole.

coordinate. Numeri che identificano una posizione sullo schermo.

coordinate assolute. In modo grafico serve a specificare la posizione di un punto rispetto alla origine delle coordinate.

coordinate fisiche. Il limite logico dello schermo (Vedi le specifiche VIEW e WINDOW).

coordinate relative. Nei grafici, valore che identifica la locazione di un punto specificandone la distanza da un altro punto.

costante. Un valore fisso o un elemento di dati.

cursore. Indicatore mobile per indicare la posizione di lettura o scrittura sullo schermo

diagnostici. Riguardano la ricerca e l'isolamento di un malfunzionamento dell'elaboratore.

dichiarazione implicita. Lo stabilire la dimensione in una schiera senza dichiarare esplicitamente la specifica DIM.

dinamico. Che si realizza al momento dell'esecuzione.

disabilitato. Uno stato che impedisce ad un dispositivo di lanciare interruzioni (interrupts).

dischetto flessibile. Un minidisco (non rigido, flessibile).

doppia precisione. Indica 8 bytes di memoria che rappresentano un numero. E' usata per aumentare la precisione delle operazioni matematiche.

DOS. Disk Operating System. In questo manuale si riferisce solo al DOS del PC.

duplex. Nelle comunicazioni indica il modo di trasmissione simultaneo nelle due direzioni. Stesso concetto di full-duplex.

eccedenza. Dicesi del risultato di un'operazione quando questa eccede i valori ad essa assegnati.

eccedenza aritmetica. Vedi eccedenza.

eco. Rimanda i dati al dispositivo che li ha inviati o ad un altro dispositivo. Per esempio il video è l'eco della tastiera.

editare. Immettere, modificare e cancellare dati.

elemento. Un membro di un insieme; in particolare un elemento di una schiera.

errore. Errore di programma.

errore di sintassi. Viene segnalato quando si contravviene alle regole della sintassi (errori di digitazione, mancanza di punti, parentesi ecc.).

esecuzione. L'esecuzione di una specifica di programma.

estensione. Spazio continuo su dischetto occupato o riservato per un file.

evento. Un avvenimento o evento; nel BASIC del PC si riferisce a particolari eventi controllati dalle specifiche COM(n)ON), ON KEY(n), ON PEN, ON PLAY(n), ON STRIG(n), ON TIMER.

file. Un insieme di record correlati.

fine file (EOF). Il carattere, che segue l'ultimo record di un file, segnalante la fine del file.

finestra. Nei grafici, una precisa area in coordinate assolute.

fittizio. Dicesi di un elemento (fisico o di programma) che si comporta formalmente come quello che sta rappresentando, senza però averne le capacità funzionali. Per esempio: un argomento fittizio di una funzione.

formato. Una particolare disposizione di dati sul proprio supporto (es: lo schermo, il dischetto, ecc.).

funzione. Una procedura che genera dei valori in funzione dell'impostazione di una o più variabili. Più generalmente lo specifico scopo di un programma o di un dispositivo.

gerarchia. Struttura a molti livelli organizzata in forma di albero. "Gerarchia delle operazioni" indica la priorità con cui deve essere eseguito un insieme di operazioni.

grafico. Un simbolo prodotto da una elaborazione come una stampa, un disegno, ecc.

gruppo (stack). Metodo di salvataggio temporaneo dei dati in modo che l'ultimo elemento salvato sia anche il primo ad essere elaborato.

half duplex. Nella trasmissione di dati si riferisce a trasferimenti alternati, una volta in un senso ed una volta nel senso opposto.

hertz (Hz). Unità di frequenza che corrisponde ad un ciclo al secondo.

identificatore di elemento. Identificatore di elemento di schiera.

immagine rovesciata. Scambiare il colore di un carattere con quello dello sfondo e viceversa.

immunità da errore. La potenzialità per una macchina di essere immune da errori. Essa è misurata come rapporto fra la quantità di errori rilevati in un arco di tempo ed un valore di riferimento memorizzato nella macchina.

incremento. Un valore usato per alterare un contatore (+/–).

indirizzario. Tabella di identificatori e riferimenti agli elementi di dati. Per esempio l'indirizzario di un dischetto contiene i nomi dei file sul dischetto (identificatori) insieme alle informazioni che mettono in grado il DOS di ricercare un file sul dischetto.

indirizzario corrente. L'indirizzario, assunto in mancanza di informazioni, per ciascun dischetto. Questo è l'indirizzario che il BASIC cerca se si immette un nome di file senza altre informazioni.

indirizzario strutturato ad albero. Un gruppo di file correlati ed indirizzari residenti sullo stesso disco o minidisco ed organizzati in struttura gerarchica.

indirizzo. Può essere il contenuto di un registro, una locazione di memoria ecc. oppure riferito ad un dispositivo indica la locazione del dispositivo stesso.

indirizzo relativo. Numero di locazioni fra un certo indirizzo ed un altro (in un record, blocco di controllo o memoria). Per esempio nel BASIC l'indirizzo attuale di una locazione di memoria è dato dalla sua posizione relativa in byte dalla posizione definita con la specifica DEF SEG.

infinito macchina. Il più grande numero che può essere rappresentato e su cui può operare l'elaboratore.

ingrandimento. Il processo che permette di usare una finestra con coordinate più piccole dell'intera immagine, causando l'ingrandimento dell'immagine nell'area visualizzabile.

iniziale. La prima parte di qualcosa, per esempio si può fare riferimento a zeri o spazi iniziali in una stringa di caratteri.

inizializzare. Impostare contatori, interruttori, indirizzi, contenuti di memoria, valori di partenza ecc. all'inizio di un programma o sottoprogramma.

integrità. Salvataggio dei dati per i loro specifici scopi, volto a preservarli da perdite accidentali o alterazioni.

intercettazione. Un insieme di condizioni che descrive un evento che deve essere intercettato e le azioni che devono essere successivamente intraprese.

interfaccia. Un limite condiviso in cui due sistemi interagiscono.

intero. Uno dei numeri interi (±1, ±2 ecc.).

interprete. Traduce ed esegue una specifica in linguaggio sorgente in una o più istruzioni in linguaggio macchina.

interruttore. Si riferisce a tutto ciò che ha due stati stabili con possibilità di passare da uno stato all'altro.

interruzione. Segnale volto a interrompere un processo (in modo che possa essere ripreso successivamente) per eseguirne un altro.

intervallo. L'insieme dei valori che una quantità o una funzione può avere.

istruzione. In un programma indica qualsiasi espressione significativa che indichi una operazione ed i suoi operandi.

iterazione. Insieme di istruzioni che possono essere eseguite ciclicamente se una certa condizione è vera.

K. Misura della quantità di memoria: vale 2 alla decima (1024).

lampeggiante. Una intenzionale alternanza di luminosità di un carattere sullo schermo.

leva di comando. Leva che può spostarsi in qualsiasi direzione, usato come indicatore di posizione.

limitatore. Carattere che raggruppa o separa parole e valori in una riga di immissione.

locazione. Qualsiasi supporto su cui può essere memorizzato un dato.

lunghezza fissa. Si riferisce a qualcosa la cui lunghezza non varia; per esempio i file ad accesso casuale hanno record di lunghezza fissa.

M. Mega; 1 milione. Riferito alla memoria indica 2 alla ventesima (1.Ø48.576).

mantissa. Riferito ad un numero espresso in forma di virgola mobile, indica la parte numerale che non è l'esponente.

maschera. Un campione di caratteri che controlla l'assunzione o l'eliminazione di un altro campione di caratteri.

matrice. Indica un insieme di elementi a 1 o più dimensioni.

memoria. Un dispositivo, o parte di esso, atto a conservare dati.

memoria ad accesso diretto (RAM). Memoria che può essere letta o scritta in qualsivoglia locazione.

menu. Lista di operazioni disponibili.

minifloppy. Dischetti da 5-1/4 pollici.

nidificare. Incorporare una struttura in un'altra. Si possono, per esempio, nidificare iterazioni entro altre iterazioni o chiamate a sottoprogrammi da altri sottoprogrammi ecc.

notazione. Insieme di simboli e regole di uso per la rappresentazione dei dati.

nullo o vuoto. Vuoto, senza significato; in particolare una stringa senza caratteri.

obiettivo. Nell'assegnazione di una specifica rappresenta la variabile il cui valore deve essere impostato.

operando. Dato pertinente ad una istruzione eseguibile che ne richieda la presenza.

operando mancante. Richiede dati mancanti di una istruzione che non può essere eseguita.

operazione. Una precisa azione che applicata a qualsiasi possibile combinazione di entità note, produce una nuova entità.

orologio. Dispositivo che genera segnali periodici usati per sincronizzare eventi; ciascun segnale, da esso generato, dicesi impulso.

ospitante. Calcolatore primario o di controllo in una installazione composta da molti calcolatori.

ottale. Si riferisce ad una numerazione in base 8.

pagina. La parte di memoria video che può essere visualizzata o scritta.

pagina attiva. E' la pagina video che il BASIC usa per operazioni di immissione/emissione. Essa può essere diversa dalla pagina le cui informazioni devono essere visualizzate.

pagina visualizzata. La pagina video corrente che il BASIC visualizza sullo schermo.

panoramica. Il processo che usa una finestra con coordinate più grandi dell'immagine che deve essere visualizzata.

parametro. Un nome in una procedura che è usato per referenziare un argomento passato alla procedura.

parola chiave. Determinata parola che permette l'accesso ad un programma; oppure parola riservata.

parola riservata. Una parola che è definita in BASIC per particolare scopo e che non può essere usata come nome di variabile.

penna luminosa. Dispositivo sensibile alla luce usato per selezionare un punto o carattere sullo schermo.

percorso. Il modo usato per trovare un file. Usato con l'indirizzario o qualsiasi altro comando o specifica riguardante i files.

pixel. Un punto o locazione sullo schermo usato per comporre un'immagine. Si riferisce anche ai bits che contengono informazioni per il suddetto punto.

porta. Un punto di accesso per l'immissione o emissione dei dati.

posizione. Indica in una stringa ciascuna locazione che può essere occupata da un carattere e che può essere identificata da un numero.

precisione. Una misura della possibilità di distinguere tra valori molto vicini.

programma applicativo. Indica un programma che svolge una particolare funzione (per esempio un programma di paghe e stipendi).

protetto. Dicesi di un dispositivo il cui accesso può essere limitato in tutto o in parte.

prova di parità. Una tecnica usata per validare i dati trasmessi. Un bit viene aggiunto ad un gruppo di bit in modo tale che la somma di tutti i bit sia sempre pari o sempre dispari.

punto indirizzabile. In modo grafico indica un qualsiasi punto sul video che può essere indirizzato. Tali punti sono in numero finito e formano una griglia sul video.

radice. Una forma primitiva di un programma usata per stabilire un contatto con altre parti di programma. Si può vedere anche come un programma che carica se stesso.

radice dell'indirizzario. L'indirizzario che viene creato su disco o dischetto quando lo si formatta. Indirizzario "principale" o "base".

record. Un insieme di informazioni correlate trattate come un'unità.

record a lunghezza variabile. Un record che può avere una sua lunghezza indipendente dalla lunghezza degli altri record.

record di testa. Indica un record contenente valori comuni, costanti o identificatori per un gruppo di record che seguono.

richiesta. Una domanda che il calcolatore pone quando ha bisogno di informazioni.

ricorrente. Si riferisce ad un processo nel quale ogni passo usa il risultato del passo precedente; come quando una funzione chiama se stessa.

riempire. Riempire un blocco con dati fittizi, usualmente zeri o spazi.

riga. Se riferito a stampante o schermo indica uno o più caratteri allineati. Se riferito ad un ingresso indica una stringa di caratteri accettati dal sistema come singolo blocco; per esempio tutti i caratteri immessi prima di premere il tasto IMMIS. Se riferito ai grafici indica una linea. Se riferito alla trasmissione dati indica il supporto fisico (filo, microonde ecc.) usato per la trasmissione.

riga tagliata fuori. Processo in base al quale punti referenziati fuori dello schermo non possono essere visualizzati. Qualsiasi immagine che giace in parte dentro ed in parte fuori dello schermo si dice "tagliata fuori" dalla zona visibile.

riserva. Riferito ad un sistema, dispositivo, file ecc. indica un sistema alternativo da usare in caso di malfunzionamento del primo.

risoluzione. Nei grafici, una misura della nitidezza dell'immagine, espressa in numero di punti per pollice quadrato.

ritorno carrello (CR). Indica il carattere che causa, su video o stampante il ritorno del cursore sulla prima posizione della stessa riga.

scalare. Un valore o una variabile che non siano una schiera.

scansione a rastrello. Una tecnica di generazione delle immagini sullo schermo, ottenuta spazzolando linea per linea tutta la superficie dello schermo.

scorrimento. Muovere tutta la parte visibile dello schermo in alto o in basso in modo che possano apparire nuove informazioni.

scrivere. Memorizzare dati in un dispositivo capace di accoglierli.

segmento. Una particolare area di memoria da 64K byte.

segnale. Qualsiasi tipo di indicatore che segnala un evento.

seguito. Posizionato alla fine di una stringa o numero. Per esempio nel numero 1000, 1 è seguito da 3 zeri.

selezione. Una particolare tecnica per selezionare una lista di elementi secondo una determinata sequenza. Coppie di elementi vengono paragonate e, se il caso, scambiate fra di loro; il processo è ripetuto fino al completamento della selezione.

singola precisione. Descrive un metodo di memorizzazione di un numero in 4 bytes di memoria.

sintassi. Le regole che governano la struttura del linguaggio.

sistema di coordinate assolute. Un sistema di coordinate non limitato; nei grafici "spazio" illimitato.

sistema operativo. Programmi che controllano l'esecuzione di altri programmi; spesso riferito al DOS.

solo lettura. Un tipo di accesso ai dati che permette la sola lettura e non l'alterazione dei dati.

sotto la condizione. Un evento che può causare una interruzione in seguito alla rilevazione di un errore o al realizzarsi di un altro evento che può accadere in un non prevedibile istante.

sottoindirizzario. Qualsiasi indirizzario contenuto nell'indirizzario radice o in un altro sottoindirizzario.

sfondo. Indica l'area che contorna il soggetto, in particolare la parte di video attorno ad un carattere.

sottoprogramma. Parte di un programma o sequenza di istruzione chiamate da un altro programma che possono avere uso generale o frequente.

sovrapposizione. Metodo che usa le stesse aree di memoria per differenti parti di programma in momenti diversi.

sovrascrittura. Scrivere in un'area di memoria cancellando i dati precedentemente immagazzinati.

spazio (blank). Una parte destinata ai dati nella quale non sono registrati caratteri.

spazzolamento. Esaminare sequenzialmente.

specifica. Una espressione significativa che serva a descrivere una certa operazione nel contesto del linguaggio BASIC.

stampa. Indica la stampa eseguita da una stampante.

stampante a matrice. Stampatrice che rappresenta i caratteri tramite campioni di punti.

stop bit. Un segnale che segue un carattere o blocco e prepara il sistema alla ricezione del successivo.

stringa. Sequenza di caratteri.

tasto automatico. Un tasto che ripete il carattere finché viene premuto.

tasto funzionale. Uno dei 10 tasti indicati con F1-F10 sulla tastiera.

tavola. Insieme di dati organizzati in righe e colonne.

tavolozza. Un insieme di colori.

telecomunicazioni. Sinonimo di trasmissione di dati.

terminale. Dispositivo, usualmente munito di video e tastiera, con il quale è possibile inviare e ricevere dati.

tipo di accesso. Una tecnica usata per leggere o scrivere un record da o su un file.

troncato. Il rimuovere gli elementi finali di una stringa.

valore assunto. Valore scelto in mancanza di specificazioni.

valore booleano. Un valore numerico interpretato come vero (diverso da 0) o falso (uguale a 0).

variabile. Un campo che può assumere un dato insieme di valori.

variazione di scala. Nei grafici, rappresenta il processo di cambiamento della grandezza dell'immagine variando la dimensione dell'area visibile (Vedi specifica VIEW).

vettore. Nei grafici, segmento con direzione. Più generalmente un ordinato insieme di numeri o anche una matrice monodimensionale.

zona di transito. E' una zona di memoria usata per compensare differenze di velocità nel trasferimento dei dati o differenze temporali fra eventi diversi. Usualmente si riferisce ad un'area riservata per le operazioni di I/E dove i dati possono essere letti o scritti.

Note:



IBM United Kingdom International Products Limited
PO Box 41, North Harbour, Portsmouth PO6 3AU, England

Printed in Great Britain by Ben Johnson & Co. Ltd., York.