PHILIPS

MSX-BASIC

MSX™

QUESTO MANUALE
ILLUSTRA IN MANIERA
DETTAGLIATA TUTTI
I COMANDI
DELL' MSX-BASIC

A. SICKLER

New Media Systems

MSX-BASIC

Edizione Philips

Albert Sickler

MSX-BASIC

Edizione Philips

Edizioni tecniche Kluwer
(Deventer – Paesi Bassi)

1 edizione 1986
2 edizione 1986
3 edizione 1986
4 edizione 1987
6 edizione 1987

Traduzione: Expertrans Zoetermeer – Paesi Bassi

PREMESSA

Dietro alle lettere MSX si nasconde un mondo intero. Un mondo che viene definito da una enorme quantità di diverse marche di computer, che prima di tutto stupiscono curiosamente più per le loro affinità che per le loro diversità.
Per la prima volta viene introdotto nel mondo dei computer uno standard, e senz'altro le conseguenze saranno enormi.
L'introduzione al grande pubblico dei computer MSX susciterà un grande interesse intorno alla domanda 'che cosa sono di preciso i computer MSX, e che si può fare con l'MSX BASIC?'
Scegliendo il computer MSX, avete almeno scelto un computer Philips. Una buona scelta in quanto ognuno sa come questo gigante dell'elettronica si sia distinto nel campo della qualità.
Per molti lettori questo significherà il primo incontro con il mondo dei computer, un incontro da cui sperano di ricevere un aiuto per superare gli ostacoli che dovranno affrontare. Questo manuale è stato scritto per questi lettori.

INDICE

Introduzione al Basic ... I1

1. I primi passi: istruzione PRINT e operazioni matematiche ... I1
2. E ora un programma BASIC! ... I3
3. Le variabili e ancora sui calcoli matematici ... I7
4. INPUT, READ, e DATA ... I12
5. Numeri grossi e piccoli e in tutte le forme ... I16
6. PRINT, TAB, LOCATE, PRINT USING e REM ... I21
7. Le istruzioni di controllo: GOTO, IF...THEN, FOR...NEXT e ON...GOTO ... I24
8. Il vettore (o array) ... I30
9. Le stringhe...giochiamo con le lettere ... I32
10. Le subroutine: GOSUB...RETURN e DEF FN ... I35

Ampliamenti per l'MSX-BASIC ... U1

1. Suono: BEEP, PLAY e SOUND ... U1
2. Rappresentazioni grafiche: SCREEN, PSET, COLOR e PRESET ... U8
3. Rappresentazioni grafiche: LINE, DRAW, CIRCLE e PAINT ... U12
4. Sprites ... U16
5. Archivi ... U21

Appendici ... A1

A Uso del registratore a cassette ... A1
B Uso della stampante ... A3
C Uso dei connettori joystick ... A4
D Uso del diskdrive ... A5
E Uso dell'interfaccia RS232C ... A13
F Segni e espressioni speciali ... A16
G Messaggi di errore ... A18
H Sequenze di escape ... A22
I I codici di controllod ... A23
J Costruzione dei simboli ... A24
K Nomi riservati ... A27

Panorama sulle istruzioni dell'MSX-BASIC ... O1

Indice analitico

INTRODUZIONE AL BASIC

1 I PRIMI PASSI: ISTRUZIONE PRINT E OPERAZIONI MATEMATICHE

Istruzione print per la riproduzione di calcoli diretti

Se accendiamo il computer, questo permette immediatamente di lavorare in BASIC.
Per dimostrarlo, digitiamo:

```
PRINT 2 + 3
```

Sullo schermo appare nello stesso momento: PRINT 2+3. Non appena premiamo il tasto RETURN, il computer reagirà a questa istruzione. Il tasto RETURN viene indicato sia con la parola stessa RETURN che con il simbolo ↵.
PRINT: significa 'stampa' o meglio 'rappresenta' e subito dopo aver premuto RETURN ci accorgiamo che qualcos'altro viene rappresentato, e cioè.

```
5
```

e al termine:

```
Ok
```

E' chiaro che 5 è la risposta all'addizione data 2+3. Il termine Ok indica che il computer ha svolto il suo compito e che noi possiamo perciò dargliene un altro.

Osservazioni:

- Con il computer non possiamo assolutamente scrivere 2+3= (e dopo per esempio premere RETURN). Se lo facciamo, appare una nota - naturalmente in inglese -che ci dice che abbiamo sbagliato.
- Con il computer, per ottenere immediatamente un risultato sullo schermo, dobbiamo sempre battere l'istruzione PRINT prima di ogni altro inserimento.

Moltiplicazione e divisione

Per la moltiplicazione viene usato il segno * e per la divisione la barra obliqua. Perció PRINT 5*36 dá come risultato il numero 180 e PRINT 180/36 il valore 5.

Le parentesi

Possiamo servirci delle parentesi così come abbiamo imparato ad usarle sui banchi di scuola: perciò l'operazione

$$\frac{5+15}{23(17+103)}$$

viene rappresentata dal computer come segue:

```
PRINT (5+15)/(23*(17+103))
```

Notate prima di tutto che tutta l'espressione viene collocata su una riga. Il numeratore e il denominatore sono entrambi messi tra parentesi. Se non l'avessimo fatto, l'istruzione PRINT apparirebbe così:

```
PRINT 5+15/23*(17+103)
```

Questa istruzione dà un risultato diverso da quello desiderato (questo vi diventerà immediatamente chiaro tra breve, non appena avremo parlato delle regole di priorità nelle operazioni matematiche).

Vediamo inoltre che nel denominatore è stato inserito il segno di moltiplicazione tra il numero 23 e 'aperta parentesi', questo perché nell'espressione originale del denominatore '23(17+103)' è implicito che si intende 23 moltiplicato per (17+103).

Ordine secondo il quale le operazioni vengono effettuate

Con l'MSX BASIC, invece, bisogna sempre rispettare le seguenti regole di priorità:

1. prima di tutto vengono effettuate le operazioni dentro le parentesi;
2. le moltiplicazioni e le divisioni hanno uguale priorità ma vengono svolte prima delle addizioni e delle sottrazioni che hanno anche uguale priorità fra loro;
3. l'ordine di svolgimento tra operazioni con uguale priorità va da sinistra verso destra. Per esempio:

```
PRINT 15/3*5  diventa    5*5       e quindi   25
```

Perciò il risultato è 25 (e non 1!).
Nell'appendice F viene data una visione generale di tutte le regole di priorità.

Testo

Il testo che deve essere rappresentato viene sempre messo tra virgolette. Osservate l'esempio:

```
PRINT "QUESTO E' UN ESEMPIO"
```

dà come risultato

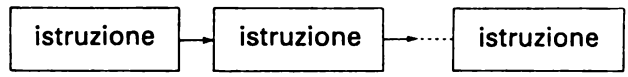
```
QUESTO E' UN ESEMPIO
```

Notate che le virgolette non appaiono nel risultato. Più avanti verranno trattate altre possibilità.

2 E ORA UN PROGRAMMA BASIC!

Un programma BASIC: uso dei numeri di riga

Un programma non è altro che una serie di istruzioni elaborate automaticamente dal computer dopo un particolare commando iniziale.
Osservate il seguente schema:



La cosa più facile è l'inserimento del programma, ad esempio una parte del testo, nel computer, che avviene semplicemente usando la tastiera. Questo vuol dire che un certo programma, cioè una certa serie di istruzioni, viene come prima cosa inserita in memoria.
Come possiamo far sí che le istruzioni di cui è formato il programma vengano come prima cosa inserite in memoria e non immediatamente realizzate? La risposta è semplice:

battendo dei numeri davanti alle istruzioni.

Con l'esempio seguente cercheremo di illustrare meglio ció che abbiamo detto.

Un esempio

Supponete che vogliamo far realizzare al computer come programma le 3 istruzioni seguenti, in quest'ordine successivo:

```
PRINT "QUESTO E' "
PRINT "IL PRIMO ESEMPIO"
PRINT "DI UN PROGRAMMA"
```

Allora quando iniziamo a battere il testo di questo programma – che si tratta in fin dei conti di una serie di istruzioni – dobbiamo inserire un numero prima di ogni istruzione.
Le numeriamo semplicemente con 10, 20, 30, ecc.
Adesso battiamo:

```
10 PRINT "QUESTO E' "
20 PRINT "IL PRIMO ESEMPIO"
30 PRINT "DI UN PROGRAMMA"
```

Alla fine di ogni riga premiamo RETURN per indicare che sono state completamente inserite (N.B. non dimenticatevi di premere RETURN anche dopo l'ultima riga!)

Quando tutte le righe sono state battute, il programma è stato inserito completamente in memoria. Del resto lo vediamo anche sullo schermo.

Andiamo a eseguire il programma!

Se ora battiamo:

```
RUN    (seguito da RETURN)
```

allora sullo schermo appare:

```
QUESTO E'
IL PRIMO ESEMPIO
DI UN PROGRAMMA
```

Evviva, siamo riusciti a realizzare un programma col computer! In questo caso il programma era formato solo da istruzioni PRINT con un testo da riprodurre, ma ció nonostante si trattava di una serie di istruzioni, in altre parole di un programma.

Annotiamo le seguenti ossevazioni:

- Dopo che il programma è stato eseguito, rimane ancora in memoria. Il programma si perde solo se diamo il comando di cancellare e anche se togliamo la corrente al computer.
- Nello stesso tempo i numeri danno l'ordine secondo il quale le istruzioni devono essere eseguite; si inizia sempre dal numero più basso fino al numero più alto.
  Avremmo anche potuto battere:

```
10 PRINT "QUESTO E' "
30 PRINT "DI UN PROGRAMMA"
20 PRINT "IL PRIMO ESEMPIO"
```

Il risultato sarebbe stato uguale proprio perché la successione è indicata dai numeri di riga. Del resto il computer sistema le istruzioni nell'ordine giusto subito dopo il loro inserimento.

- Parliamo sempre di numeri di riga e mai solo di numeri. Come numeri di riga possono essere usati solo numeri interi (da 0 a 65529).
- Il fatto di numerare le righe con 10, 20, 30 ecc., ha come vantaggio la possibilità di poter inserire più tardi altre istruzioni. Come questo funzioni di preciso, lo vedremo nel capitolo 5.

RUN è un comando, ciò significa che il computer deve eseguire l'azione richiesta immediatamente dopo che questo comando è stato battuto.
Altri comandi molto conosciuti sono:

LIST per visualizzare il programma stesso. Provate a battere LIST seguito dal tasto RETURN!

AUTO per generare automaticamente i numeri di riga.

RENUM per rinumerare i numeri di riga.

DELETE per cancellare parti del programma.

NEW per cancellare la memoria.

Questi comandi vengono descritti dettagliatemente nel Panorama sulle istruzioni dell'MSX-BASIC. Provateli uno per uno!

Aggiungere, inserire e modificare le righe

Succede spesso nella pratica di dover modificare un programma. Ve lo dimostriamo con il sequentè programma:

```
10 PRINT "QUESTO E' UN"
20 PRINT "PROGRAMMA"
30 PRINT "PER ILLUSTRARE"
```

Inseriamo ora il programma e non dimentichiamoci di cancellare con NEW un eventuale programma precedente.

Aggiungere

Scegliamo un numero di riga superiore al più grande presente nel programma. Per esempio:

```
40 PRINT "COME SI LAVORA"
50 PRINT "CON I NUMERI DI RIGA"
```

Diamo il comando LIST per osservare che queste righe siano state veramente aggiunte.

Inserire

Per inserire una riga si sceglie un numero di riga compreso tra i due numeri all' interno dei quali vogliamo che venga riprodotta la nuova riga. Per esempio:

```
15 PRINT "CORTO E SEMPLICE"
```

Controlliamo con il comando LIST se questa riga, dopo essere stata battuta, si trova veramente al posto giusto.

Cancellare

Una riga può essere cancellata semplicemente battendo il numero di riga corrispondente seguito dal tasto RETURN, per esempio:

```
15 (tasto RETURN)
```

Controlliamo nuovamente questa azione con il comando LIST. Eventualmente si può usare anche il comando DELETE (vedere Panorama sulle istruzioni dell'MSX-BASIC).

Correzione degli errori

Abbiamo già parlato nell'introduzione al capitolo di alcune possibilità di modificare un programma. In questo paragrafo vi mostriamo cosa si può fare se è stato fatto un errore.
La soluzione apparentemente più semplice sarebbe di riscrivere la riga sbagliata, naturalmente senza errori.

Battiamo:

```
30 PRINT "CORDO E SEMPLICE"
```

Dopo aver battuto LIST vediamo che nel programma è stata riprodotta la riga con questo errore evidente.

Possiamo quindi correggere l'errore riscrivendo questa riga senza errori.
Una soluzione più elegante ci viene offerta dal lavoro dei cursori. Questi sono i tasti con le freccette, alla destra della tastiera. Provate a premere questi tasti e vediamo sullo schermo dove si colloca il quadratino bianco.
Questo quadratino viene chiamato cursore e per questo i tasti con le freccette vengono chiamati tasti movimento cursore. Ora muovete il cursore finchè non si trovi precisamente sul nostro errore, così:

```
30 PRINT "CORDO E SEMPLICE"
                ↑
            indica il cursore
```

Adesso premiamo i tasti T e RETURN. Poi rimettiamo il nostro cursore in basso.....ed adesso l'errore è corretto!

Inoltre possiamo tener premuto uno dei cursori, in modo da ottenere un effetto di ripetizione, come se il tasto fosse premuto ripetutamente. Questo effetto 'ripetitivo' vale inoltre anche per gli altri tasti.

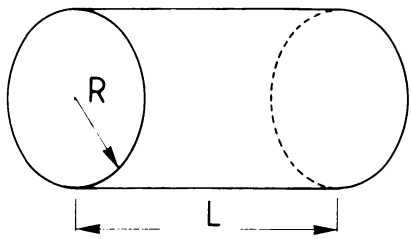
Tasti speciali

Infine concludiamo dicendo che anche durante l'inserimento del testo questo può essere di nuovo cancellato premendo il tasto BS. Con il tasto INS possiamo inserire in una riga un qualsiasi carattere della tastiera e con il tasto DEL esso può essere cancellato. Questi ultimi due tasti sono importanti soprattutto quando operiamo con i cursori. Prima portiamo il cursore nel posto dove è stato inserito l'errore e poi possiamo cambiare, cancellare o aggiungere il carattere che desideriamo.

3 LE VARIABILI E ANCORA SUI CALCOLI MATEMATICI

Presentazione Iniziamo con una introduzione

Per determinare il volume di una barra cilindrica, dobbiamo conoscere la superficie della sezione, e moltiplicarla per la lunghezza. Osservate il disegno:



superficie della sezione = 3,14159×R×R

volume della barra = superficie della sezione × L = 3,14159×R×R×L

Vediamo come il volume di una barra dipenda in generale da due grandezze: il raggio R e la lunghezza L.

Il programma seguente calcola il volume di una barra cilindrica, ma questa volta con i valori:
L=5
e R=7
Il programma:

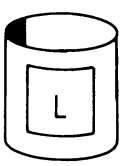
```
10 L=5
20 R=7
30 PI=3.14159
40 SUP=PI*R*R
50 V=SUP*L
60 PRINT V
```

Dopo aver premuto RUN appare:

```
769.68955
```

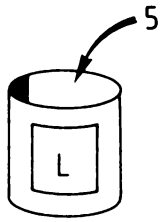
Andiamo ad osservare questo programma, riga per riga. Nella riga 10 si legge:

```
L=5
```

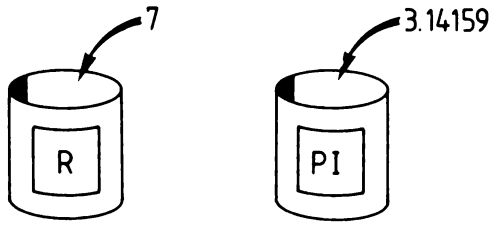


Il significato di questa espressione è che il computer mette da parte un pezzettino di memoria e gli assegna l'etichetta L. Questo pezzettino di memoria possiamo considerarlo alla stregua di un contenitore.

L'espressione L=5 significa che in questo contenitore è stato memorizzato il numero 5. Osservate il disegno:



Se il programma rimanda a L (riga 50), il computer effettivamente andrà a guardare nel contenitore. Rappresentiamo anche le righe 20 e 30 nel disegno seguente:



Osservate che nel contenitore PI viene memorizzato il numero 3.14159 e non 3,14159. In poche parole la virgola che noi adoperiamo comunemente con i numeri decimali, con BASIC viene sempre indicata da un punto. Questo 'punto decimale' appare anche nel risultato del programma.
Chi ha alle spalle qualche conoscenza di matematica, avrà già riconosciuto nel numero 3.14159 il numero π (pronuncia PI greco).
La riga 40 ci mostra un'operazione: nel contenitore verrà memorizzato il valore che si ottiene moltiplicando il PI (greco) per R*R.
Perciò nel contenitore SUP verrà memorizzato il valore 3.14159x7x7.
Il valore così trovato viene usato nella riga 50 per determinare finalmente il valore del volume che viene memorizzato nel contenitore contrassegnato dall'etichetta V.
Alla fine usiamo l'istruzione PRINT per visualizzare il contenuto di questo ultimo contenitore:

```
          PRINT V
          ↗     ↖
```

rappresenta — *il contenuto del contenitore contrassegnato dall' etichetta V*

Scriviamo ora le seguenti osservazioni che riguardano questo primo programma:

- Al posto di

```
10 L=5
```

avremmo anche potuto scrivere:

```
10 LET L=5
```

Letteralmente questo sta per 'Dai L uguale a 5' o meglio ancora 'L diventa 5'.

Con quest'ultima frase mettiamo in evidenza che il computer esegue un'azione: viene messo qualcosa nel contenitore con l'etichetta L. Questa azione vi diventerà completamente chiara con l'esempio seguente:

```
10 LET X=3
20 LET X=X+4
```

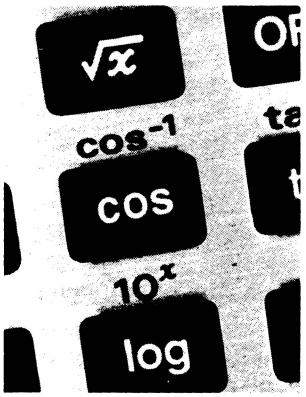
Nella seconda riga viene messo in X il vecchio valore di X, in altre parole nel contenitore verrà messo 3+4. Così alla fine il contenitore conterrà il valore 7. Notate bene che nel BASIC un'espressione come X=X+4 è perfettamente corretta, mentre in matematica non sarebbe possibile.

- Il fatto che possiamo mettere in un certo contenitore un valore arbitrario, indica che il contenuto è variabile. Questa è anche la ragione per cui comunemente si parla di variabili invece che di contenitori. Il nostro programma conosce perciò le variabili L, R, PI, SUP, e V.
  D'ora in poi parleremo perciò di 'assegnazione di valori ad una variabile' al posto di 'collocazione di una variabile in un contenitore'.

- Esistono delle regole precise sul nome da dare alle variabili, e cioè:
  - Soltanto le prime due lettere di un nome vengono riconosciute dal computer come il nome. Perciò i nomi QUARTO e QUATTRO per il computer sono uguali: soltanto le due lettere QU vengono riconosciute.
  - Non possiamo usare come nome parole che sono già state riservate, ad esempio, IF non può essere usato come nome, in quanto è già stato riservato.
    I termini 'riservati' si possono leggere in appendice.

Funzioni standard

L'esempio che abbiamo usato era molto semplice. Possiamo appena immaginare che, in questo modo, possano venire eseguiti dei calcoli complicatissimi. Parlando però delle cosiddette funzioni standard ampliamo considerevolmente la visione su ciò che possiamo calcolare usando BASIC.



Per spiegare le funzioni standard ritorniamo un attimo indietro alla calcolatrice, ma ad una calcolatrice con funzioni matematiche. Se dobbiamo calcolare la radice quadrata di un numero su una di queste calcolatrici, inseriamo prima un numero e poi premiamo il tasto della radice. In questo modo, semplicemente premendo un tasto, si può calcolare la funzione desiderata.
Con BASIC possiamo calcolare i valori delle funzioni persino più semplicemente. Chiamiamo la funzione con un determinato termine e scriviamo il numero di cui vogliamo calcolare il valore di funzione tra parentesi, così:

termine(valore)

dà la funzione — *dà il valore della funzione che vogliamo determinare*

Esempi:

```
10 A=SQR(4)
20 PRINT A
```

Risultato:

```
2
```

Nella riga 10 ad A viene attribuito il valore della radice quadrata di 4. Il termine, che in questo esempio indica la funzione, è SQR. Tra parentesi si trova il valore di cui si vuole estrarre la radice quadrata, in questo caso (per semplicità) 4.
Facciamo notare che tra parentesi può essere scritta anche una variabile e perfino un'espressione aritmetica.
Esempi:

`B=SQR(A)` : qui il valore della funzione viene indicato dalla variabile A

`C=SQR(SQR(5)+4+A)` : qui il valore della funzione viene indicato da SQR(5)+4+A. Notate che questa espressione contiene essa stessa un'altra funzione.

Nella tabella sottostante vengono mostrate le funzioni più conosciute. Nel Panorama sulle istruzioni dell'MSX-BASIC sono riportate tutte le funzioni.

Definizione dell'MSX BASIC	*Notazione algebrica*	*Significato*
ABS(X)	$\|x\|$	per calcolare il valore assoluto di X.
ATN(X)	arctg(x)	per calcolare l'arcotangente di X. Soluzioni tra -pi greco/2 e pi greco/2.
COS(X)	cos(x)	per calcolare il coseno di X:X radianti.
EXP(X)	e^x	per calcolare e^x.
INT(X)	nessuno	per calcolare il massimo intero non maggiore di X. Perciò INT(3.8) dà come valore 3 e INT(-3.1) dà come valore -4. Con INT(X+0,5) possiamo arrotondare un numero a numero intero.
LOG(X)	$^e\log(x)$	per calcolare il logaritmo naturale di X.
SGN(X)		che assume i valori di -1, 0, o 1, a secondo che X sia negativo, o uguale a 0, o positivo.
SIN(X)	sen(x)	per calcolare il seno di X:X in radianti.
SQR(X)	$\sqrt{x}$	per calcolare la radice quadrata di X
TAN(X)	tg(x)	per calcolare la tangente di X:X in radianti.

L'appendice E dà un panorama di tutte le funzioni.
Possiamo notare che in questa tabella non vengono dati i valori elevati a potenza. Per questo adoperiamo il simbolo ^. Quindi PRINT 2^3 dà come risultato 8 (2^3).

Più instruzioni in una riga

L'MSX BASIC ci offre la possibilità di scrivere più di una instruzione in una riga.
Dovete usare il segno: come segno di separazione. Il programma seguente ce lo dimostra:

senza segno di separazione:

```
10 A=10
20 B=5
30 C=A+B
40 PRINT C
```

con i segni di separazione

```
10 A=10:B=5:C=A+B:PRINT C
```

4 INPUT, READ E DATA

INPUT La prima istruzione di cui parleremo è l'istruzione INPUT. Questa istruzione ci dà la possibilità di assegnare valori alle variabili durante l'esecuzione del programma. Prima di tutto guardiamo ancora il programma del capitolo precedente.

```
10 L=5
20 R=7
30 PI=3.14159
40 SUP=PI*R*R
50 V=SUP*L
60 PRINT V
```

Al posto di questo programma avremmo naturalmente potuto scrivere:

```
PRINT 3.14159*7*7*5
```

per ottenere lo stesso risultato.
Se non l'abbiamo fatto è perché volevamo mostrare in che modo venivano memorizzati i calcoli nel computer, mentre per determinare il risultato bastava solo inserire i valori di R e di L.
Per calcolare per esempio il volume di una barra cilindrica con L=34 e R=10, possiamo inserire le righe 10 e 20:

```
10 L=34
20 R=10
```

e dopo aver premuto RUN appare la risposta desiderata. Ora, modificare un programma per far eseguire una operazione non è proprio raccomandabile. Potremmo sempre commettere degli errori.....
Il programma potrebbe migliorare molto se dopo aver premuto RUN venisse richiesto l'inserimento dei valori L e R.
Questa possibilità viene realmente offerta proprio dall'istruzione INPUT, che si presenta così:

```
INPUT 'testo da stampare', nome di variabile
```

Ad esempio

```
INPUT "INSERISCI LA LUNGHEZZA"; L
```

L'effetto che crea questa istruzione è che il computer rappresenta il testo compreso un punto interrogativo e subito dopo interrompe il programma. A questo punto bisogna inserire prima di tutto un numero, per esempio:

```
5 (e RETURN)
```

Immediatamente dopo aver premuto RETURN viene assegnato alla variabile L questo numero e il computer andrà avanti con l'esecuzione del programma. Il nostro programma dopo l'introduzione dell'istruzione INPUT si presenta così:

```
10 INPUT "INSERISCI LA LUNGHEZZA "; L
20 INPUT "INSERISCI IL RAGGIO ";R
30 PI=3.14159
40 SUP=PI*R*R
50 V=SUP*L
60 PRINT "IL VOLUME=";V
```

Ecco un esempio del procedimento:

```
INSERISCI LA LUNGHEZZA ? 5
INSERISCI IL RAGGIO ? 7
IL VOLUME = 769.68955
```

Nell'istruzione INPUT di questo programma vediamo che viene incluso anche il testo, che viene collocato tra virgolette. Viene usato anche il segno; come segno di separazione.

Il programma rispetto al nostro primo programma è considerevolmente migliorato. La struttura è salda. Non dobbiamo apportare nessun cambiamento nel programma stesso se vogliamo calcolare il volume di qualsiasi altra barra cilindrica.

Osservazioni

A proposito dell'istruzione INPUT scriviamo ancora alcune osservazioni.

- Una istruzione INPUT può accettare più di una variabile, per esempio:

```
INPUT "INSERISCI A,B e C:"; A,B,C
```

Dopo l'interruzione del programma il computer mostra il testo con un punto interrogativo; a questo punto dobbiamo inserire tre numeri separati da una virgola. Il testo nell'istruzione INPUT può anche non essere scritto; per esempio:

```
INPUT A
```

Dopo l'interruzione il computer mostra ancora un punto interrogativo per indicare che dobbiamo inserire un numero.

READ e DATA

La terza possibilità di cui parleremo ora, riguarda la capacità di poter assegnare dei valori ad una serie relativamente lunga di variabili. Ciò significa che questi valori in genere non saranno sottoposti a cambiamenti programma per programma.

In una tale situazione potremmo anche inserire naturalmente una serie di istruzioni LET, ma, come vedremo, la possibilità con READ e DATA è molto più funzionale. Mostriamo subito un esempio:

```
10 READ A,B,C,D
20 F=A+B+C+D
30 PRINT F
40 DATA 3,7,4,8
```

L'effetto dell'istruzione della riga 10 è questo: il computer 'sa' che vengono nominate un certo numero di variabili, i cui valori vengono enumerati l'uno dopo l'altro nella riga che inizia con il termine DATA.
Così ad A sarà assegnato il valore 3, a B il valore 7, a C il valore 4 e a D il valore 8.
Il risultato del programma lo conferma:

```
22
```

Notate che la riga con DATA non è un'istruzione che deve essere rappresentata. E' semplicemente un foglietto di brutta copia, sul quale il computer ritrova i valori che ha cercato. Tra l'altro avremmo potuto dividere l'istruzione READ in:

```
10 READ A,B
15 READ C,D
```

e altrettanto avremmo potuto fare con l'istruzione DATA, in questo modo per esempio:

```
40 DATA 3,7,4
50 DATA 8
```

Al computer tutto questo non fa differenza; legge i dati come se fossero una riga continua. Il meccanismo si può immaginare meglio in questa maniera: come se il computer mettesse in precedenza una freccetta sotto il primo valore dell'elenco DATA. Questa freccetta si sposta di una posizione ogni volta che, tramite l'istruzione READ, viene assegnato un valore alla variabile:

```
40 DATA 3,7,4,8
           ↑
```

la freccetta si sposta sempre di una posizione.

Se la freccetta raggiunge l'ultimo valore dell'elenco DATA, il computer la sposta verso il primo valore di un elenco DATA successivo, nel caso che esista.
Possiamo anche riportare la freccetta nella sua posizione originaria, grazie alla seguente istruzione:

```
RESTORE
```

Esempio:

```
10 READ A,B
20 RESTORE
30 READ C,D
40 F=A+B+C+D
50 PRINT F
60 DATA 3,7,4,8
```

Risultato:

```
20
```

Notate che ora il risultato è 20. Questo avviene per efetto dell'istruzione RESTORE; vengono ora assegnati infatti sia ad A e B che a C e D i valori 3 e 7.

5 NUMERI GROSSI E PICCOLI E IN TUTTE LE FORME

Introduzione Prima di continuare, diamo un'occhiata alla tabella sottostante.

Dalla tabella possiamo dedurre che, per esempio, 10^4 equivale a 10000, in altre parole la quantità di zeri corrisponde direttamente alla potenza (la cifra che si trova in alto a destra del numero 10).

a parole	*notazione*	*significato*	*valore*
10 alle potenza di 1	10^1	10	10
10 alla potenza di 2	10^2	10x10	100
10 alla potenza di 3	10^3	10x10x10	1000
10 alla potenza di 4	10^4	10x10x10x10	10000
10 alla potenza di −1	10^{-1}	1/10	0.1
10 alla potenza di −2	10^{-2}	1/(10x10)	0.01
10 alla potenza di −3	10^{-3}	1/(10x10x10)	0.001
10 alla potenza di 0	10^0	10/10	1

Esempio:
A quale potenza corrisponde 1000 000 000?
Risposta: contate il numero degli zeri, sono 9 e di conseguenza abbiamo 10^9.

Dalla tabella risulta tra l'altro che 0.001 corrisponde a 10^{-3}. Anche con i numeri più piccoli di 1, per determinare la potenza, si può contare il numero degli zeri.

Esempio:
A quale potenza corrisponde il numero riportato qui sotto?

0.000 000 000 000 000 000 1

Risposta:contate il numero degli zeri, sono 19 e perciò abbiamo 10^{-19}.
Potete anche scriverlo in questa maniera: 1×10^{-19}

Il primo numero si chiama mantissa e il secondo indica l'esponente.
Con il computer indichiamo il numero dato sopra secondo la seguente notazione:

1E−19

In altre parole, usiamo la lettera E per distinguere la mantissa dall'esponente.
Per ottenere un po' di esperienza con queste notazioni si può sperimentare il programma seguente:

```
10 INPUT A
20 INPUT B
30 C=A*B
40 PRINT C
```

Esempi:

A=987654	B=456789	dà: 451149483006
A=98765434	B=987654321	dà: 9.754610765554E+16
A=.0000000008	B=.0000000007	dà: 5.6E−19
A=8000000000	B=7000000000	dà: 5.6E+19

Singola precisione e doppia precisione

Per la rappresentazione dei numeri il nostro computer usa un numero limitato di registri di memoria (parole). Ciò significa anche che i calcoli hanno un'esattezza limitata. L'MSX BASIC offre la possibilità di operare sia con la cosiddetta doppia precisione che con quella singola. Per i numeri in doppia precisione si usano sempre 8 parole per la loro memorizzazione, mentre per i numeri in singola precisione se ne usano solo 4. Normalmente il nostro computer usa sempre i numeri in doppia precisione. Per operare in singola precisione collochiamo sempre un punto esclamativo(!) dopo il numero.
Le variabili alle quali abbiamo assegnato i numeri in singola precisione si distinguono dalle variabili in doppia precisione per la collocazione del simbolo! subito dopo il nome. Gli esempi seguenti illustrano la differenza tra operazioni in singola precisione e quelle in doppia precisione:

```
10 A=10/3                    10 A!=10!/3!
20 PRINT A                   20 PRINT A!

risultato                    risultato

3.3333333333333              3.33333
```

Aggiungiamo qui le seguenti osservazioni:

- Alla fine dei numeri in doppia precisione possiamo collocare il simbolo #. Allo stesso modo possiamo porre questo simbolo come ultimo carattere del nome. Così indichiamo esplicitamente che lavoriamo in doppia precisione.

- Usando la lettera D anziché E (p.es. 23.45156321D39) per i numeri in doppia precisione, indichiamo in questo modo esplicitamente che operiamo in doppia precisione.

- Dobbiamo chiederci sempre se abbia senso voler rappresentare i risultati con tante cifre. Supponiamo che un falegname debba segare un'asse in tre parti uguali. Secondo il nostro computer ciascuna parte misurerebbe 3.3333333333333 m...ma quale falegname potrebbe lavorare così esattamente?

- L'uso dei numeri in singola precisione è soprattutto importante se vogliamo far economia di memoria. La problematica della memoria verrà discussa più avanti quando verranno trattate i vettori o array.

Numeri interi

In un certo numero di casi vogliamo indicare che nel programma si parla di numeri interi.
La rappresentazione dei numeri in un programma non crea problemi: un numero senza il punto decimale è sempre un numero intero. Dalle variabili invece non si può vedere se rappresentano numeri interi o numeri con il punto decimale. Bene, per far vedere chiaramente questa differenza, si colloca il simbolo % dopo il nome della variabile.
Esempio:

```
10 A%=20
20 B%=3
30 C%=A%/B%
40 PRINT C%
```

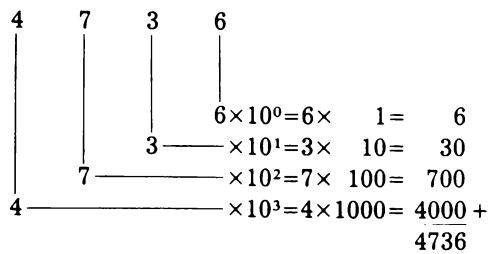
Risultato:

```
6
```

Notate ancora che il numero viene arrotondato, sempre per difetto. Nel nostro caso il numero 6.6666... è stato arrotondato a 6.

Numeri binari

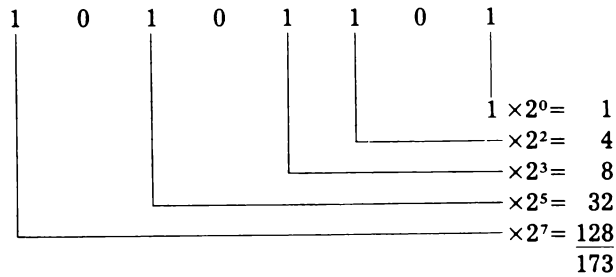
Per spiegare i numeri binari diamo un'occhiata, per prima cosa, alla formazione dei nostri numeri 'normali', cioè ai numeri decimali. Questi numeri sono sempre formati dalle potenza di 10 (vedere tabella all'inizio di questo capitolo)
Esempio:
Il numero 4736 è composto dalle potenze di 10, come potete vedere dallo schema seguente:



Vediamo come ogni cifra corrisponde ad una potenza di 10. Partendo da destra leggiamo la posizione zero e poi, la prima, la seconda, la terza posizione; queste posizioni corrispondono precisamente alle potenze di 10.

Bene, i numeri binari sono composti soltanto dalle cifre 0 e 1 e le posizioni di tali cifre corrispondono di nuovo ad una potenza, e questa volta alle potenze di 2.
Esempio:

A quale numero decimale corrisponde il numero binario 10101101?
Sviluppiamo:



Con BASIC possiamo inserire i numeri binari anche direttamente, ma dobbiamo collocare il simbolo &B prima del numero.
Esempio:

```
10 A%=&B10101101
20 PRINT A%
```

Risultato:

```
173
```

Numeri ottali e esadecimali

Dopo quello che abbiamo detto sinora, ci risulterà semplice spiegare che cosa sono i numeri ottali e esadecimali.
I numeri ottali sono i numeri in cui ogni cifra, in maniera analoga all'esposizione dei numeri binari, corrisponde ad una potenza di 8. Con i numeri esadecimali la posizione di una cifra corrisponde ad una potenza di 16.
E' utile notare che con la numerazione decimale per rappresentare questi numeri abbiamo precisamente 10 cifre (da 0 fino a 9). Con la numerazione binaria ne abbiamo solo due (0 e 1) e con la numerazione ottale ne abbiamo 8 (da 0 a 7).
Fin qui nessum problema. Con la numerazione esadecimale abbiamo naturalmente 16 cifre... Ma conosciamo solo le cifre da 0 fino a 9. Come indichiamo allora nella numerazione esadecimale le cifre rimanenti? Bene, in questo caso, usiamo le lettere da A fino a F.
La tabella seguente mostra i numeri decimali da 0 fino a 15 e allo stesso tempo mostra anche i numeri in forma binaria, ottale ed esadecimale.

decimali	*binari*	*ottali*	*esadecimali*
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Ciò di cui abbiamo parlato, non sembra essere molto utile, per il momento. Eppure vedremo (tra l'altro nel capitolo sugli sprites) come queste cose verranno a proposito.
Le numerazioni ottale e esadecimale possono essere inserite, così come sono, direttamente nei programmi. Davanti ad un numero ottale scriviamo il simbolo &0 e davanti a quello esadecimale il simbolo &H.
Esempi:

```
PRINT &017 dà come risultato 15
PRINT &HF  dà come risultato 15
PRINT &HFF dà come risultato 255
```

6 PRINT, TAB, LOCATE, PRINT USING E REM

Segni di separazione

Nei precedenti capitoli abbiamo già visto che potevamo collocare più di una cosa in una istruzione PRINT.

Diamo un esempio:

```
10 A=12
20 PRINT "A=";A
30 PRINT "A=",A
```

Risultato:

```
A=12
A=              12
```

La differenza tra il risultato della riga 20 e quello della riga 30 sta nei segni di separazione. Nella riga 20 usiamo il punto e virgola come segno di separazione e nella riga 30 la virgola.
Usando il punto e virgola il risultato che segue viene raffigurato subito dopo il precedente. Se si adopera invece la virgola, il computer elabora automaticamente una divisione in colonne. Questa divisione in colonne viene sempre calcolata con un intervallo di 14 posizioni.
Tra l'altro questi segni di separazione possono essere anche collocati alla fine di una istruzione PRINT. Guardate l'esempio seguente:

```
10 PRINT "ABC";"DEFG";
20 PRINT "H";"IJ";"KLM"
```

Risultato:

```
ABCDEFGHIJKLM
```

Se si vuole omettere una riga nella rappresentazione del testo, si usa PRINT senza ulteriori indicazioni.

Esempio:

```
10 PRINT "A"
20 PRINT
30 PRINT "B"
```

Risultato:

```
A

B
```

Si nota che tra A e B è stata aggiunta una riga bianca.

TAB Con la funzione TAB possiamo anche indicare in quale colonna una grandezza dovrà essere rappresentata.
L'esempio riportato qui sotto spiega l'uso di TAB.

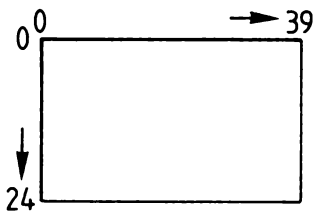
```
10 PRINT TAB(1);"MSX"
20 PRINT TAB(3);"MSX"
30 PRINT TAB(5);"MSX"
```

Risultato:

```
MSX
  MSX
    MSX
```

Vediamo che dopo TAB viene sempre indicata, tra parentesi, la posizione della colonna a partire dalla quale verrà rappresentata la cosa voluta (in questo caso il testo).

LOCATE La funzione TAB regola sulla riga la posizione del termine da rappresentare. LOCATE indica invece sia la posizione orizzontale che quella verticale. E' un'istruzione molto efficace, con la quale possiamo ottenere una bella composizione del testo da rappresentare.



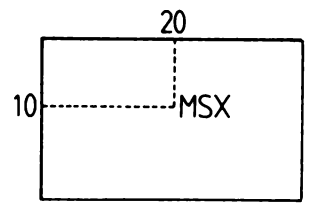
L'istruzione LOCATE si presenta come segue:

```
LOCATE    posizione orizzontale, posizione verticale
```

Lo schermo viene organizzato in modo qui accanto.

Esempio di un programma:

```
10 CLS
20 LOCATE 20,10
30 PRINT "MSX"
```



Il programma dà il risultato qui accanto.

Il programma inizia con un'istruzione di cui ancora non abbiamo parlato, e precisamente l'istruzione CLS (istruzione CLEAR SCREEN). Questa fa sì che lo schermo venga pulito e che il cursore venga collocato nell'angolo in alto a sinistra. L'istruzione seguente LOCATE muove il cursore nella posizione (20,10). Ciò vuol dire che il testo MSX viene rappresentato a partire da quella posizione.

WIDTH Osserviamo anche che con l'istruzione WIDTH si può definire il numero di colonne dello schermo. Per una descrizione completa si vedano le instruzioni SCREEN e WIDTH in Panorama sulle istruzioni dell'MSX-BASIC.

PRINT USING In moltissime applicazioni si desidera sempre che i risultati appaiano in una determinata forma. Come esempio possiamo pensare a dei calcoli matematici dove spesso abbiamo delle cifre che compaiono dopo la virgola.
Usando la cosiddetta istruzione PRINT USING possiamo indicare in quale formato un numero debba essere raffigurato. Questa istruzione si costruisce sempre così:

```
PRINT USING   "informazioni sulla rappresentazione"; numero
```

Diamo un esempio:

```
PRINT USING "##.##";32.7
```

Sullo schermo appare:

```
32.70
```

L'espressione scritta fra le virgolette, cioè i simboli ##.##, indica in quale formato deve essere rappresentato il numero. In questo caso con due cifre prima del punto decimale, e due dopo il punto decimale.

```
                ## . ##
               ↗   ↑   ↖
        due cifre  il punto  due cifre
prima del punto              dopo il punto
```

Le possibilità offerte dal PRINT USING sono eccezionalmente grandi.
Un panorama completo viene dato in Panorama sui compiti dell'MSX-BASIC.

REM L'ultima istruzione di cui parleremo in questo capitolo è l'istruzione REM. Il termine REM deriva da 'remark' che significa 'osservazioni'. Con questa istruzione possiamo aggiungere dei commenti all'interno di un programma. Queste osservazioni devono essere scritte sempre dopo il termine REM.
Un esempio:

```
10 REM TANTO PER FARE UN ESEMPIO
20 PRINT "FINE"
```

Il risultato dopo il comando RUN sarà:

```
FINE
```

Vediamo che le osservazioni inserite dopo REM non vengono rappresentate dopo che abbiamo premuto il comando RUN. Le possiamo rivedere soltanto se premiamo nuovamente il comando LIST.

7 LE ISTRUZIONI DI CONTROLLO: GOTO, IF...THEN, FOR... NEXT E ON...GOTO

GOTO

L'istruzione GOTO ha una formulazione molto semplice:

```
GOTO      numero di riga
```

Se il computer in contra questa istruzione, salterà al numero di riga indicato. Poiché questo salto viene sempre eseguito, si parla in questo caso di salto incondizionato.
Un esempio molto semplice:

```
10 PRINT "QUESTO VIENE STAMPATO"
20 GOTO 40
30 PRINT "QUESTO NO"
40 PRINT "VIENE STAMPATO ANCHE QUESTO"
```

Risultato:

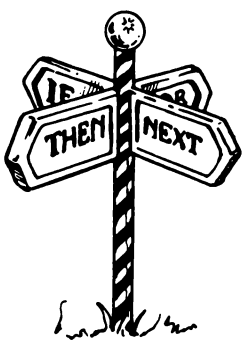
```
QUESTO VIENE STAMPATO
VIENE STAMPATO ANCHE QUESTO
```

La riga 20 fa saltare alla riga 40 e perciò la riga 30 viene sempre omessa.

IF...THEN

La più semplice formulazione dell'istruzione IF...THEN è la seguente:

```
IF condizione THEN istruzione
```



Vediamo che tra i termini IF e THEN viene posta una condizione da verificare. SE (IF) questa condizione viene soddisfatta, ALLORA (THEN) l'istruzione data può essere eseguita. La cosa migliore da fare è chiarire questa formulazione con un esempio molto semplice:

```
10 INPUT "INSERISCI UN NUMERO";K
20 IF K=3 THEN PRINT "IL NUMERO ERA TRE"
30 PRINT "FINE PROGRAMMA"
```

Il programma inizia con un'istruzione INPUT, che indica di inserire un numero. Supponiamo di aver inserito il valore 3; quindi a K viene assegnato il valore 3. Nella riga dopo si trova l'istruzione IF...THEN.
La condizione è:

è K uguale a 3?

Notate che qui abbiamo scritto l'espressione in forma interrogativa. Bene, una tale espressione può essere solo o giusta o sbagliata; nell'esempio o K è veramente uguale a 3 oppure non lo è.

Diciamo allora che una condizione data è soddisfatta o no, oppure in maniera più distinta che l'espressione logica è soddisfatta o no. Bene, a K era stato assegnato il valore 3, la conclusione perciò è che la supposta condizione è anche soddisfatta.
In questo caso il computer eseguirà l'istruzione dopo THEN, e verrà visualizzata la scritta 'IL NUMERO ERA TRE'. Se K fosse stato diverso da 3, cioè se la condizione data non fosse stata soddisfatta, allora dopo THEN l'istruzione sarebbe stata semplicemente omessa. Una volta che il computer abbia elaborato l'istruzione. IF...THEN, esso prosegue con l'istruzione successiva.

Nell'esempio seguente mostriamo un programma nel quale dopo THEN appare un'istruzione GOTO:

```
10 PRINT "QUANTO FA 2+5?"
20 INPUT K
30 IF K=7 THEN GOTO 80
40 PRINT "NO"
50 PRINT "LA RISPOSTA DI 2+5"
60 PRINT "E 7 NATURALMENTE"
70 GOTO 90
80 PRINT "QUESTA E VERAMENTE LA RISPOSTA GIUSTA"
90 END
```

Qui vediamo che dopo THEN è stata inserita l'istruzione GOTO.
Se a K viene dato il valore 7, allora si salta alla riga 80.
Se a K viene dato un valore diverso da 7, allora questa istruzione GOTO viene omessa e così si passa alle righe 40, 50, 60 e 70. Notate che anche la riga 70 contiene un'istruzione GOTO. Anche questa è necessaria, altrimenti dopo il testo delle righe 40, 50 e 60 apparirebbe il testo della riga 80...e questo potrebbe sembrare perlomeno un po 'curioso.
L'ultima istruzione di questo programma è la cosiddetta istruzione END che ci indica che il programma è finito. Questa istruzione eventualmente potrebbe anche essere omessa. Infatti nei programmi precedenti l'abbiamo sempre fatto.
Scriviamo ancora qualche osservazione su questo programma:

- al posto di IF K=7 THEN GOTO 80
si sarebbe anche potuto scrivere
IF K=7 THEN 80
o omettere THEN
IF K=7 GOTO 80

- dopo THEN eventualmente si sarebbero potute mettere altre istruzioni, divise dal segno ':'. La condizione è che l'intera espressione di IF...THEN sia contenuta in una riga BASIC (massimo 255 segni).

- nell'istruzione IF...THEN, K viene confrontato con 7 attraverso il segno =. In una condizione data questo segno viene chiamato segno di relazione. Possiamo usare oltre al segno = anche i seguenti segni:

segno	*significato*
<	minore di
>	maggiore di
< =	minore o uguale a
= <	lo stesso
> =	maggiore o uguale a
= >	lo stesso
< >	diverso da
> <	lo stesso

- Le espressioni logiche possono anche essere combinate. Queste possibilità vengono discusse nell'Appendice F.

IF...THEN...ELSE Nell' BASIC esiste anche l'istruzione IF...THEN ampliata dal termine ELSE. Cerchiamo di spiegarla meglio con un facile esempio:

```
10 INPUT K
20 IF K=7 THEN PRINT "K=7" ELSE
               PRINT "K È DIVERSO DA 7"
30 END
```

L'istruzione dopo THEN viene eseguita soltanto se è verificata la condizione posta, mentre in tutti gli altri casi viene eseguita l'istruzione dopo ELSE. Anche dopo questa istruzione, come per THEN, si possono aggiungere più istruzioni, divise tra loro dal segno dei due punti.

FOR...NEXT Dobbiamo considerare l'istruzione FOR...NEXT come uno strumento del programmatore eccezionalmente pratico con il quale possiamo indicare che una determinata serie di istruzioni deve essere eseguita (ripetuta) un certo numero di volte.

L'esempio seguente dovrebbe rendere le cose più chiare:

```
10 FOR A=1 TO 5
20 PRINT A; "QUESTA E UNA DIMOSTRAZIONE"
30 NEXT A
```

Risultato:

```
1 QUESTA E UNA DIMOSTRAZIONE
2 QUESTA E UNA DIMOSTRAZIONE
3 QUESTA E UNA DIMOSTRAZIONE
4 QUESTA E UNA DIMOSTRAZIONE
5 QUESTA E UNA DIMOSTRAZIONE
```

Le istruzioni che devono essere ripetute, devono essere sempre racchiuse tra la riga iniziale con FOR e la riga finale con NEXT. Oservate lo schema:

```
.. FOR nome della variabile=...
   ...
.. ...   } queste istruzioni vengono ripetute
   ...
.. NEXT nome della variabile
```

Nel nostro caso si tratta solo di una istruzione e precisamente l'istruzione della riga 20. Grazie a ciò che è scritto nella riga che inizia con FOR, possiamo già determinare completamente quante volte le istruzioni devono essere eseguite.

Con

```
FOR A=1 TO 5
```

L'istruzione verrà ripetuta 5 volte, e A assume successivamente i valori di 1, 2, 3, 4 e 5. Le variabili che vengono indicate dopo FOR vengono chiamate molto appropriatamente le 'variabili contatore'.
In questo caso A assume un incremento sempre uguale a 1. Possiamo anche darle un incremento maggiore usando la riga che inizia con FOR e aggiungendo STEP:

```
10 FOR=1 TO 6 STEP 2
20 PRINT A; "QUESTA E UNA DIMOSTRAZIONE"
30 NEXT A
40 PRINT A
```

Risultato:

```
1 QUESTA E UNA DIMOSTRAZIONE
3 QUESTA E UNA DIMOSTRAZIONE
5 QUESTA E UNA DIMOSTRAZIONE
7
```

Vediamo come la variabile di controllo A assuma successivamente i valori 1, 3, e 5 e come per questi valori il 'ciclo' venga sempre eseguito.
Se A è uguale a 7 viene oltrepassato il limite dato nella riga 10 (6) e il computer prosegue il programma con la riga 40. Se facciamo stampare ancora una volta il valore di A, vien così stampato il valore 7. La morale che possiamo trarre da questo piccolo pro-

gramma è chiara... fate attenzione ad usare più avanti nel programma una variabile di controllo dopo un'istruzione FOR...NEXT; questa infatti non indica necessariamente il valore ultimo che viene dato nella riga con FOR!

Ancora qualche esempio:

- `FOR A=10 TO 5 STEP-1`
  la serie di istruzioni viene eseguita per
  A=10, 9, 8, 7, 6 e 5
- `FOR A=-15 TO 15 STEP 3`
  la serie di istruzioni viene eseguita per
  A=−15, −12, −9, −6, −3, 0, 3, 6, 9, 12 e 15
- `FOR A=1.4 TO 1.7 STEP .05`
  la serie di istruzioni viene eseguita per
  A=1.4, 1.45, 1.50, 1.55, 1.60, 1.65 e 1.70
- `FOR A=B TO C STEP K/L`

Vediamo qui come sia l'intervallo di incremento che i valori iniziali e limite vengono indicati dalle variabili. L'intervallo viene persino determinato da una espressione matematica (una divisione di 2 variabili). E' possibile infatti indicare, con espressioni del genere, i valori iniziali, limite e di intervallo.

Le istruzioni FOR...NEXT possono contenere anche delle altre istruzioni FOR...NEXT. La condizione è però che una istruzione contenga completamente l'altra. Perciò la costruzione:

```
FOR K=1 TO 10
...
FOR J=1 TO 5
...
NEXT J
NEXT K
```

il ciclo interno viene completamente contenuto dal ciclo esterno

è certamente possibile e la costruzione:

```
FOR K=1 TO 10
...
FOR J=1 TO 5
...
NEXT K
...
NEXT J
```

'ciclo K'

'ciclo J'

invece non è possibile. Qui il ciclo interno non viene completamente contenuto dal ciclo esterno.

ON...GOTO L'ultima istruzione di cui parleremo in questo capitolo è l'istruzione ON...GOTO. Questa istruzione fa pensare ad un selezionatore di scelte.
In base al valore scritto tra ON e GOTO viene eseguito un determinato salto.
Un esempio chiarirà meglio:

```
10 INPUT K
20 ON K GOTO 30,50
30 PRINT "K = 1"
40 GOTO 70
50 PRINT "K = 2"
60 GOTO 70
70 END
```

Se K è uguale a 1, il computer salterà al primo numero di riga indicato che segue. Se K è uguale a 2 il computer salterà al secondo numero di riga.
In questo esempio dopo GOTO (riga 20) sono stati dati solo due numeri di riga, in realtà potrebbero essere anche di più. Osservate inoltre che le diverse parti del programma dove il computer è saltato, vengono di nuovo chiuse da un'istruzione GOTO.
In questo caso l'uso delle istruzioni GOTO è di nuovo indispensabile.

8 IL VETTORE (O ARRAY)

Introduzione

In molti casi esiste la necessità in un programma di disporre di un grosso numero di variabili.
Potremmo pensare per esempio ad un programma riguardante la gestione di un magazzino. Se per ogni articolo che abbiamo dovessimo introdurre una singola variabile, il programma diventerebbe veramente molto lungo.
Ora mostreremo perciò come il BASIC abbia risolto questo problema tramite un'istruzione che offre la possibilità di poter introdurre un grosso numero di variabili.

L'istruzione DIM

L'istruzione DIM è l'istruzione grazie alla quale possono essere introdotte un grosso numero di variabili. Si presenta sempre secondo questa formulazione:

```
DIM nome (quantità), per esempio:
DIM A(100)
```

Con questo esempio il BASIC introduce una serie di 101 variabili. Il nome di ognuna di queste variabili è composto quindi dal nome che appare nell'istruzione DIM, e dall'indicazione del numero messo tra parentesi.
Perciò queste:

```
A(0) A(1) A(2) A(3)... A(99) A(100)
```

formano precisamente le 101 variabili che appartengono all'istruzione DIM A(100). In questo caso si può dire anche che parliamo del vettore A, che in questo esempio è composto dagli elementi A(0) fino a A(100).
Tutte queste variabili possono essere usate alla stessa maniera delle variabili 'normali'.
Un esempio:

```
10 DIM A(100)
20 A(3)=6
30 A(27)=5
40 A(98)=A(3)+A(27)
50 PRINT A(98)
```

Risultato:

```
11
```

Questo programma un po' singolare mostra che abbiamo usato gli elementi del vettore A(3), A(27) e A(98) come se fossero state delle variabili 'normali', chiamate per esempio A, B e C.

Uno dei vantaggi principali delle variabili 'vettore' sta nel fatto che il numero tra parentesi (spesso chiamato indice) possa essere anche assegnato indirettamente.

`A(93)` qui il numero viene assegnato direttamente da una cifra.

`A(K)` qui il numero viene assegnato indirettamente da una variabile.

`A(K+3)` qui il numero viene assegnato indirettamente da un'espressione.

Il programma seguente ci offre un facile esempio:

```
10 DIM B (20)
20 FOR K=1 TO 20
30 B(K)=K
40 NEXT K
50 FOR K=20 TO 1 STEP -1
60 PRINT B(K);
70 NEXT K
```

Risultato:

```
20 19 18 17 16 15 14 13 12
11 10 9 8 7 6 5 4 3 2 1
```

Nella riga 10 viene introdotto il vettore B. Le righe da 20 a 40 indicano che deve essere assegnato a B (1) il valore 1, a B (2) il valore 2 ecc. Che i valori siano stati veramente assegnati, lo dimostrano le righe da 50 a 70, e di conseguenza vengono stampati i valori delle variabili B(20), B(19), B(18), ecc. Notiamo tra l'altro come possa essere pratico usare l'istruzione FOR...NEXT in combinazione con i vettori.

Annotiamo anche le seguenti osservazioni sui vettori:

- se in un programma viene usato un vettore, per esempio A(6), senza essere stato indicato da un'istruzione DIM, il computer accetta un vettore con un limite massimo di 10. In questo esempio il computer considera che la variabili A(6) appartiene al vettore A che ha come limite massimo 10 (DIM A(10)).

- in un programma possiamo anche usare i cosiddetti vettori a più dimensioni, cioè vengono dati più numeri tra parentesi. Perciò l'istruzione:

  ```
  DIM A(3,3)
  ```

  introduce le variabili:

  ```
  A(0,0)    A(0,1)    A(0,2)    A(0,3)
  A(1,0)    A(1,1)    A(1,2)    A(1,3)
  A(2,0)    A(2,1)    A(2,2)    A(2,3)
  A(3,0)    A(3,1)    A(3,2)    A(3,3)
  ```

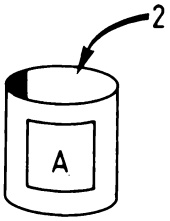
- In una riga possiamo introdurre più di un vettore.
  Per esempio:

  ```
  10 DIM A(100), P(300), Z(50), P!(30), B%(5)
  ```

- In un secondo tempo, se vogliamo, possiamo liberare lo spazio riservato usando l'istruzione ERASE.
  Per esempio: `ERASE A, P`

9 LE STRINGHE... GIOCHIAMO CON LE LETTERE

Introduzione



Fino ad ora abbiamo visto che possiamo assegnare un numero ad una variabile. Per spiegare come questo funzionasse, abbiamo usato, come esempio, la rappresentazione di un contenitore. In questo modo, secondo l'illustrazione data a lato, era stata rappresentata con chiarezza l'istruzione A=2.

Alle variabili possiamo assegnare anche una serie di lettere. Una tale serie di lettere viene chiamata una stringa e questa è la ragione per cui simili variabili vengono chiamate variabili stringa.

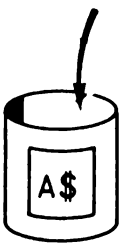
Se mettiamo immediatamente dopo il nome di una variabile il segno del dollaro, parliamo di variabili stringa.

Perciò A$, PRIMO$ e TESTO$ sono degli esempi evidenti di nomi che si riferiscono alle variabili stringa. Il testo che possiamo assegnare ad una simile variabile stringa, deve essere sempre sistemato fra virgolette.
Il programma seguente ci offre un esempio:

QUESTA E UNA STRINGA

testo che viene messo in A$

```
10 LET A$="QUESTA E UNA STRINGA"
20 PRINT A$
```



La riga 10 può essere nuovamente spiegata con un disegno.

Notate che nella riga 10 il testo si trova effettivamente tra virgolette. Se ora guardiamo il risultato del programma, vedremo che queste virgolette non sono state stampate, infatti il risultato appare così:

```
QUESTA E UNA STRINGA
```

Possiamo perciò concludere che le virgolette non costituiscono alcuna parte della stringa stessa.

Qui sotto vediamo ancora una serie di esempi di stringhe che possono essere assegnate ad una variabile stringa.

`"CIAO PAOLO"` Questa è una stringa composta di 10 caratteri, naturalmente conta anche lo spazio tra CIAO e PAOLO!

`"164"` Questa è una stringa composta di tre caratteri. Il fatto che qui si tratti di cifre rende le cose più complicate, in quanto il computer vede solo i caratteri e non le cifre! Tra breve vedremo come questi tipi di numeri speciali, o 'numeri apparenti' possano essere trasformati in un vero numero.

`""` Questa è una stringa che non è composta di alcun carattere, le virgolette sono sistemate direttamente una dopo l'altra. Ve-

dremo tra breve degli esempi in cui verrà mostrato che anche una simile 'stringa vuota' può tuttavia offrire dei vantaggi. Questo tipo di stringa viene anche indicato con il termine stringa nulla.

Concatenare (unire insieme più stringhe)

Grazie al segno dell'addizione + possiamo indicare che due stringhe devono essere unite tra loro. Gli esperti di computer parlano di 'concatenare'.

Un esempio illustrerà meglio:

```
10 A$="MSX-"
20 B$="BASIC"
30 C$=A$+B$
40 PRINT C$
```

Risultato:

```
MSX-BASIC
```

Vediamo come nelle righe 10 e 20 le stringhe 'MSX-' e 'BASIC' vengono assegnate a A$ e B$. Nella riga successiva queste righe vengono 'sommate' e il risultato assegnato a C$.

Funzioni stringa

Fino ad ora non abbiamo potuto fare molto con le stringhe, al massimo abbiamo potuto unire insieme due o più stringhe per ottenere in questa maniera una stringa più lunga. Fortunatamente l'MSX BASIC conosce un certo numero di funzioni con cui possiamo eseguire qualsiasi tipo di operazione che è in relazione alle stringhe.
Possiamo dividere queste funzioni stringa in due gruppi:

gruppo 1: le funzioni che danno nuovamente una stringa come risultato. I nomi di queste funzioni terminano sempre con $.

gruppo 2: le funzioni che forniscono un numero come risultato. La maggior parte delle funzioni appartenenti a questo gruppo usa determinate regole. Queste regole si riferiscono all'assegnazione di numeri a lettere e ad altri caratteri. I nomi delle funzioni di questo gruppo non finiscono mai con il segno del dollaro.

Le funzioni di entrambi i gruppi sono estremamente facili. Una descrizione particolareggiata si trova nel Panorama sulle istruzioni dell'MSX-BASIC in cui viene dato anche un esempio per ogni funzione. Qui diamo soltanto un breve riassunto delle funzioni più importanti.

LEN determina il numero dei caratteri di una stringa.

LEFT$ indica una sottoparte della stringa; la cosiddetta sottostringa (la parte sinistra).

RIGHT$ indica una sottostringa (la parte destra).
MID$ indica una sottostringa (generica).
ASC$ indica il valore ASCII del primo carattere.
CHR indica il carattere secondo il valore ASCII specificato.
VAL converte una stringa numerica nel numero corrispondente.
STR$ converte un numero nella stringa corrispondente.
SPACE$ per stampare un certo numero di spazi.
INSTR per ricercare una sottostringa all'interno di una stringa data.

INKEY$ e un giochetto

INKEY$ non è infatti una funzione ma una variabile. Mentre il programma sta 'girando' se si incontra il termine INKEY$, il computer 'guarda' per un attimo la tastiera. Se in quel momento viene premuto un tasto, questo carattere viene assegnato a INKEY$. Se in quel momento invece non viene premuto alcun tasto, la stringa vuota "" verrà assegnata a INKEY$.
Il programma seguente mostra come INKEY$ possa essere usato in un semplice giochetto di reazione:

```
10 PRINT "PREMI UN TASTO"
20 PRINT "SE LO SCHERMO"
30 PRINT "MOSTRA IL VALORE 25"
40 FOR K=1 TO 50
50 PRINT K
60 A$=INKEY$
70 IF A$ <>"" THEN GOTO 90
80 NEXT K
90 PRINT "FINE"
```

Ancora qualche osservazione

In questo capitolo abbiamo parlato delle funzioni più importanti che riguardano le stringhe. Una visione globale di tutte le istruzioni possibili si trova nel Panorama sulle istruzioni dell'MSX-BASIC.
Infine annotiamo le seguenti osservazioni:

- E' possibile introdurre, oltre ai numeri, anche vettori stringa, per esempio:

  ```
  DIM A$(100)
  ```

- Quando il computer viene acceso, ha già riservato un certo spazio di memoria alle stringhe. Possono essere memorizzati, di norma, circa 200 caratteri. Possiamo aumentare questo spazio con la cosiddetta istruzione CLEAR, per esempio:

  ```
  CLEAR 500
  ```

  Con questa istruzione riserviamo uno spazio di memoria di 500 caratteri.

- La stringa che possiamo assegnare ad una variabile stringa ha una lunghezza massima di 255 caratteri.

10 SUBROUTINE: GOSUB...RETURN E DEF FN

GOSUB

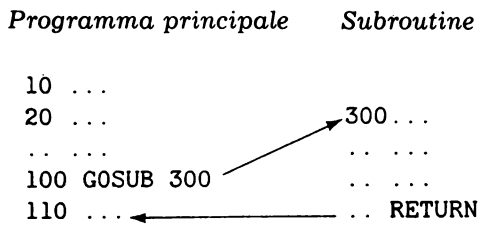
Una subroutine è una parte di programma che può essere richiamato da un punto qualsiasi del programma. Il salto alla subroutine si verifica sempre in base all'istruzione:

```
GOSUB numero di riga
```

Se il computer trova questa istruzione, continuerà il programma dal numero di riga specificato. Dopo aver riconosciuto il termine:

```
RETURN
```

il computer tornerà di nuovo al programma originale. Lo schema seguente chiarirà meglio la situazione.



Alla riga 100 vediamo come viene richiamata la subroutine. La subroutine inizia alla riga 300.
Nel momento in cui incontra RETURN, il computer ritorna alla riga 110, cioè il computer continua di nuovo con il programma originale. Abbiamo con ciò parlato della struttura completa in cui indichiamo ancora che da una subroutine si può compiere nuovamente un salto ad un'altra subroutine.
Il programma seguente offre un esempio. Grazie a questo programma possiamo giocare al famoso giochetto dei '13 fiammiferi'. A turno si possono prendere 1, 2 o 3 fiammiferi dal mucchietto. Chi prende l'ultimo fiammifero ha perso.

```
 10 PRINT "INIZIA"
 20 L=13
 30 GOSUB 80
 40 GOSUB 80
 50 GOSUB 80
 60 PRINT "HA HA, HO VINTO"
 70 END
 80 REM START SUBROUTINE
 90 INPUT K: IF K<0 OR K>3 THEN GOTO 90 ELSE A =4-K
100 PRINT "NE PRENDO";A
110 L=L-K-A
120 PRINT "NUMERO DI FIAMMIFERI RIMASTI=";L
130 PRINT "TOCCA DI NUOVO A TE"
140 RETURN
```

Notate come con END la subroutine venga separata dal programma principale. In questa maniera evitiamo che si possa entrare nelle subroutine senza l'istruzione GOSUB. Per chiarezza: si può entrare nelle subroutine soltanto con l'istruzione GOSUB.

DEF FN Grazie all'istruzione DEF FN, possiamo definire una funzione da noi stessi. L'istruzione DEF FN si presenta sempre con questa formulazione:

DEF FN nome (serie di variabili divise dalle virgole)
= espressione in cui appaiono queste variabili.

Per esempio:

```
DEF FNA (X,Y)=X^2+Y^2
```

Il nome della funzione è composto da una lettera (nel nostro esempio dalla lettera A). In questo esempio la funzione A viene definita da:

X^2+Y^2

Questa funzione viene perciò richiamata dall'espressione FNA, in altre parole dal termine FN seguito dal nome della funzione. Il programma seguente ve lo illustra:

```
10 DEF FNA(X,Y)=X^2+Y^2
20 A=3
30 B=4
40 C=FNA(A,B)
50 PRINT C
```

Risultato:

```
25
```

AMPLIAMENTI PER L'MSX-BASIC

1

SUONO: BEEP, PLAY E SOUND

BEEP BEEP significa produrre un brevissimo suono tipo 'bip'.
Il programma sottostante illustra questa istruzione:

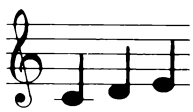
```
10 PRINT "INIZIA"
20 FOR K=1 TO 1000
30 PRINT K
40 NEXT K
50 BEEP
60 PRINT "FINITO"
```

Vediamo come l'istruzione BEEP sia stata collocata alla fine del programma. Quando il computer ha finito il suo calcolo si ode un brevissimo suono.

PLAY L'istruzione PLAY viene usata soltanto per riprodurre, facilmente, delle melodie:
E' una istruzione veramente molto efficace che può contenere, tra l'altro, le seguenti cose:

- il tempo in cui la melodia deve essere suonata.
- l'ottava a cui la nota si riferisce.
- la durata di una nota
- la nota stessa (suono)
- pausa di battuta
- il volume con cui la nota deve essere riprodotta.
- determinati effetti speciali.

PLAY può essere usata sia come istruzione che come comando.
Scrivete:



```
PLAY "CDE"
```

Dopo aver premuto RETURN, sentiamo:

Cioè le note C, D e E che appartengono all'ottava in chiave G (*).
D'ora in poi questa ottava verrà indicata come 'ottava 4'.

Il seguente programma usa l'istruzione PLAY per riprodurre, con il nostro computer MSX, il suono di un piccolo organo.

```
10 A$=INKEY$
20 PLAY A$
30 GOTO 10
```

Nella riga 10 viene assegnato ad A$ il tasto premuto che successivamente viene usato nell'istruzione PLAY, ecc.
Il nostro organino non è sicuramente perfetto, in quanto non possiamo, tra l'altro, regolare il tempo. Fortunatamente però ci vengono offerte molte altre possibilità.

* N.B. Le lettere corrispondono a queste note:
C=Do D=Re E=Mi F=Fa
G=Sol A=La B=Si

Più di un canale sonoro

Se noi inseriamo tre stringhe, dopo PLAY, separate da una virgola, il computer MSX userà ogni stringa per indicare un canale sonoro diverso.

Per esempio con:

```
PLAY "CDE"
```

verrà sempre usato un solo canale e con

```
PLAY "CDE", "EFG"
```

verranno usati due canali contemporaneamente e con:

```
PLAY "CDE", "EFG", "GAC"
```

verrano usati tre canali contemporaneamente. Ciò significa che le note C, E e G e poi D, F e A e alla fine E, G e C vengono suonate contemporaneamente.

Tempo

Il tempo viene indicato con il codice T.
Provate a dare il seguente comando:

```
PLAY "T200CDE"
```

Le note, C, D e E vengono suonate ora con un tempo più veloce del precedente. Questo è giusto, perché se non diamo alcuna indicazione, il computer partirà dalla 'posizione' T120 mentre T200 indica di conseguenza un tempo più veloce. Il valore di T può variare tra 32 e ~~256~~. 255

Volume

Il volume viene indicato dal codice V.
Osservate l'esempio:

```
PLAY "T200V13CDE"
```

Ora sentiamo la stessa musichetta dell'esempio precedente solo che il volume (V) si trova in una posizione più alta. Anche questo è giusto perché se non diamo ulteriori indicazioni, il computer assume la 'posizione' V8 mentre V13 indica un volume più alto. Il codice può variare tra 0 (minimo) e 15 (massimo).

Le note e l'ottava

Le note vengono indicate dalle lettere C, D, E, F, G, A e B. Con i mezzi toni, per esempio con CIS (cioè DO diesis), usiamo il simbolo # (o il simbolo +). Per esempio con:

```
PLAY "CC#" oppure "CC+"
```

sentiamo prima la nota C e poi successivamente il mezzo tono CIS. La domanda ora è la seguente: 'come facciamo a indicare il tono più alto di C?' Bene, il tono più alto appartiene all'ottava successiva. Abbiamo già detto che se non diamo delle ulteriori informazioni, il computer assume l'ottava 4(04). Per riprodurre la nota C con un tono più alto dobbiamo perciò inserire il codice 05.
Per esempio:

```
PLAY "CDE05CDE"
```

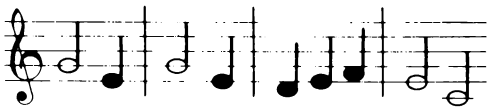
possiamo sentire due volte CDE, ma l'ultimo gruppo di note viene suonato in un'ottava superiore.
Il codice che indica l'ottava (codice 0) può variare tra un minimo di 1 ed un massimo di 8, di conseguenza la musica del computer MSX riproduce i suoni fino a 8 ottave.

La durata delle note

Fino ad ora le note avevano la stessa durata. Ma chi apre un libro di musica vedrà che le note possono avere una durata diversa.
In musica ciò viene indicato colorando il pallino della nota o lasciandolo in bianco o disegnando delle codine alla fine della stanghetta.

segni							
numero di quarti	4	2	1	1/2	1/4	1/8	1/16
codice L	1	2	4	8	16	32	64

La durata di una nota viene indicata dal codice L. Se non diamo delle ulteriori informazioni, il computer assume il valore L4 che corrisponde al valore di una semibreve. Nell'esempio seguente mostriamo il codice L grazie ad una canzoncina:



```
PLAY "L2GL4EL2GL4EDEFL2EL4C"
```

Questo comando può anche essere scritto in maniera più breve, indicando la sua durata subito dopo la nota; quindi il nostro esempio diventerà:

```
PLAY "G2EG2EDEFE2C"
```

I segni di pausa

Nei quaderni di musica si incontrano spesso i segni di pausa. Questi indicano che per un periodo determinato, o piuttosto per un certo numero di quarti, non si deve suonare alcuna nota. Per indicare le pause usiamo il codice R.
L'illustrazione seguente raffigura questi segni così come si trovano nei quaderni di musica e che qui vengono usati insieme al codice R.

segni							
numero di quarti	4	2	1	1/2	1/4	1/8	1/16
codice R	1	2	4	8	16	32	64

L'esempio seguente ci dà una dimostrazione:

```
PLAY "CDER4DECDECR2": GOTO 10
```

Dopo la prima E c'è una pausa di un quarto e dopo l'ultima C c'è una pausa di due quarti. In questo modo introducendo PLAY in un programma si possono studiare meglio le battute musicali.
La 'bella melodia' può essere interrotta con CTRL/STOP. Osserviamo infine in questo paragrafo sulle pause che possiamo anche collocare un punto dopo la nota. La durata di questa nota verrà prolungata di una volta e mezzo.

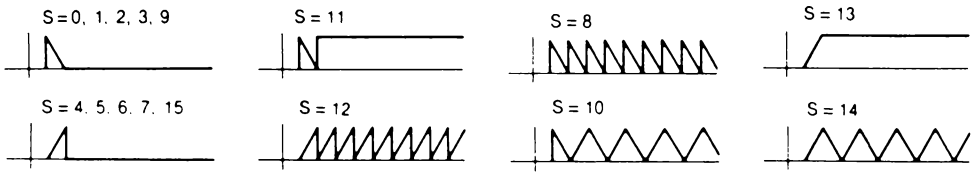
```
PLAY "CD.E"
```

Perciò anche in questo caso possiamo determinare il ritmo della nostra musica.

I codici S e M

Gli ultimi codici che possono essere usati con l'istruzione PLAY sono i codici S e M. Il codice S si riferisce alla possibilità di specificare un determinato timbro, e il codice M indica invece quante volte questo tipo di timbro deve essere ripetuto.
Il codice S indica come il suono possa essere rafforzato o dimi-

nuito secondo un determinato modello. Mostriamo le seguenti figure:



Il codice M riproduce la lunghezza del ciclo in cui il nostro modello S prescelto dovrà essere ripetuto. Se non diamo delle ulteriori indicazioni, il computer assegnerà ad S il valore 1 e a M il valore 255 (M può variare tra 1 e 65535).
Esempi:

```
PLAY "S10M510CDE"
```

e

```
PLAY "S8M6000CDE"
```

Col primo esempio sentiamo per ogni nota un suono improvviso e violento. Con il secondo esempio abbiamo l'impressione che le note vengano suonate su un piano. Provate ora da voi a scoprire le diverse combinazioni usando le diverse possibilità offerte dai codici S e M.

Un esempio già sviluppato

Con l'esempio seguente il computer riproduce la canzone 'Fra Martino' a più voci. Si può vedere come le stringhe, usate nell'istruzione PLAY, vengano costruite precedentemente con le variabili stringa.

```
 10 CLEAR 500
 20 A$="L4CDECR64CDEC"
 30 B$="EFG2EFG2"
 40 C$="L8GAGFL4ECL8GAGFL4EC"
 50 D$="C03G04C2C403G04C2"
 60 W$=A$+B$+C$+D$
 70 W1$=W$
 80 W2$="R1R1"+W$
 90 W3$="R1R1"+W2$
100 PLAY W1$,W2$,W3$
```

(N.B. nella stringa della riga 50 vengono usate le O scritte con la lettera maiuscola)

L'istruzione CLEAR è usata per riservare uno spazio sufficiente alla memorizzazione delle stringhe. Nelle righe 20, 30, 40 e 50 la melodia viene indicata nelle sue parti caratteristiche. La riga 60 riproduce la combinazione delle stringhe e quindi la melodia completa. Le righe 80 e 90 determinano la seconda e la terza voce.

SOUND L'ultima istruzione di cui parleremo in questo capitolo è l'istruzione SOUND. Quest'istruzione può essere capita bene solo se sappiamo che il nostro microprocessore suono separato è dotato di un certo numero di registri. Collocando dei determinati valori in questi registri il microprocessore produce tutti i tipi di suoni.
Il nostro microprocessore suono è munito di 13 registri che hanno il seguente significato:

numero di registro	*per che cosa si usa*
0,1	la combinazione tra il contenuto dei registri 0 e 1 determina la frequenza del suono per il canale sonoro A
2,3	lo stesso ma ora per il canale B
4,5	lo stesso ma ora per il canale C.
6	determina il timbro che produce un suono tipo 'rumore'. Il valore può variare tra 0 e 31
7	determina se un canale deve produrre un rumore o un suono.
8	determina il volume del canale A (valori tra 0 e 15)
9	come l'8 ma ora per il canale B
10	come 1'8 ma ora per il canale C
11, 12	determinano il timbro e anche la cosiddetta modulazione di un suono.
13	determina il timbro (crescendo e diminuendo l'intensità)

Per poter calcolare la frequenza di un determinato suono usiamo il seguente programma di supporto:

```
10 INPUT "FREQUENZA DESIDERATA"; F
20 A=1789772.5
30 B=INT(A/(16*F))
40 C=INT(B/256)
50 D=B-C
60 PRINT D,C
```

I valori di D e C sono i valori che devono essere collocati nel primo e nel secondo registro per ottenere il suono in questione.

Esempi Supponete che si voglia riprodurre il suono A (frequenza 440). Il nostro programma produce i valori:

```
254    0
```

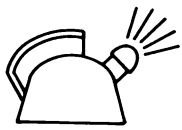
Ciò rimanda alle seguenti istruzioni SOUND:

```
10 SOUND 0,254
20 SOUND 1,0
30 SOUND 8,11
```

Sentiamo ora il suono A (provate anche PLAY 'A'). Il suono può essere interrotto con CTRL/STOP.

Rumore (effetti speciali)

In pratica usiamo l'istruzione SOUND per tutti i generi di effetti sonori speciali, per riprodurre per esempio il suono delle sirene o di un tonfo. Gli esempi seguenti mostrano alcune possibilità.

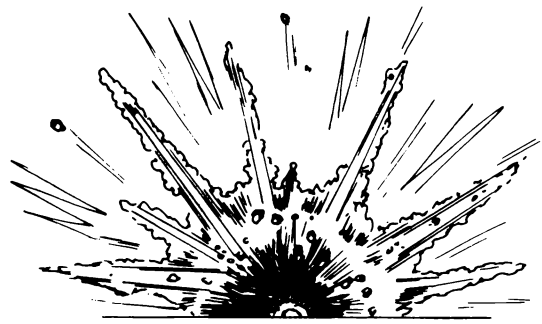


```
10 REM BOLLITORE A FISCHIO
20 FOR K=255 TO 0 STEP - 1
30 PRINT K
40 SOUND 0,K
50 SOUND 1,0
60 SOUND 8,10
70 NEXT K
```

Con l'esempio seguente possiamo provare diversi effetti:

```
10 INPUT K,L,M
20 SOUND 0,250:SOUND 1,0
30 SOUND 6,K:SOUND 7,L:SOUND 13,M
40 FOR J=15 TO 0 STEP -0.05
50 SOUND 8,J
60 NEXT J
```

Per K=10, L=10 e M=10 viene riprodotto un suono che va via via diminuendo, ma per K=20, L=20 e M=20 si ode un'esplosione.

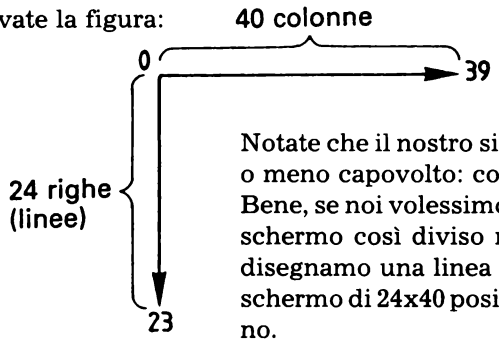


2 RAPPRESENTAZIONI GRAFICHE: SCREEN, PSET, COLOR E PRESET

Introduzione Una rappresentazione grafica non è altro che una parola molto distinta che significa immagine. Costruiamo queste immagini servendoci di punti, linee e curve.
Il nostro computer MSX offre moltissime possibilità di rappresentare su schermo bellissime figure di ogni tipo. Questo è importante perché per molte applicazioni si è proprio legati all'elaborazione di immagini belle.

SCREEN Per capire come vengono costruite le immagini con un computer MSX dobbiamo prima dire qualcosa di più sulla composizione del nostro schermo (in inglese: screen). In tutte le rappresentazioni offerte fino ad ora, il computer ha utilizzato una divisione in 24 righe, ognuna delle quali era composta di 40 caratteri.

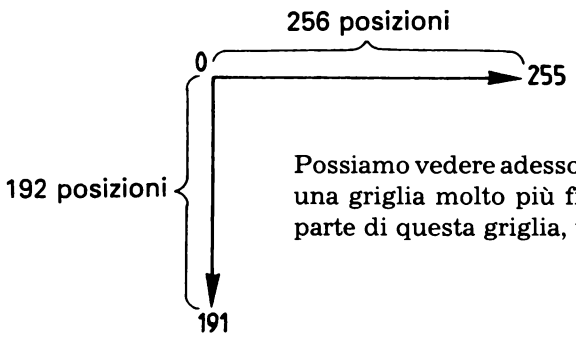
Osservate la figura:



Notate che il nostro sistema cartesiano viene rappresentato più o meno capovolto: così viene usato di solito con i computer.
Bene, se noi volessimo rappresentare un disegno più preciso, lo schermo così diviso non sarebbe sufficiente. Se per esempio disegnamo una linea formata da tante piccole crocette su uno schermo di 24x40 posizioni, apparirà un disegno molto grossolano.
Se invece vogliamo rappresentare delle linee 'molto più sottili' (i depliant parlano di alta risoluzione) dobbiamo prima di tutto predisporre il computer in altro modo, così da ottenere una diversa composizione dello schermo. Dopo l'istruzione:

```
SCREEN 2
```

il computer elabora una divisione dello schermo che si presenta così:



Possiamo vedere adesso che il nostro schermo TV è suddiviso in una griglia molto più fitta. Se adesso osserviamo meglio una parte di questa griglia, vediamo una serie di quadretti.

Da ora in poi chiameremo ogni quadretto 'punto immagine' (o brevemente: punto) e indicheremo le coordinate di ogni punto con un valore X e un valore Y, secondo la seguente convenzione e cioè che l'asse orizzontale è l'asse della X quello verticale è l'asse della Y.

Dove X può variare da 0 a 255 e Y da 0 a 191.
Il punto (0,0) rappresenta dunque il punto in alto a sinistra e il punto (255, 191) il punto in basso a destra.
Se abbiamo predisposto il computer, tramite l'istruzione SCREEN 2, ad una rappresentazione a griglia fitta, siamo in grado di indicare punti e linee con l'aiuto delle funzione che sono a nostra disposizione.

PSET Cominciamo con l'istruzione PSET (*point*set, cioè rappresentazione di un determinato punto), con la quale possiamo visualizzare dei punti sullo schermo.
Nella sua formulazione più semplice l'istruzione si presenta come segue:

```
PSET (coordinata-x, coordinata-y)
```

Cominciamo subito con un esempio:

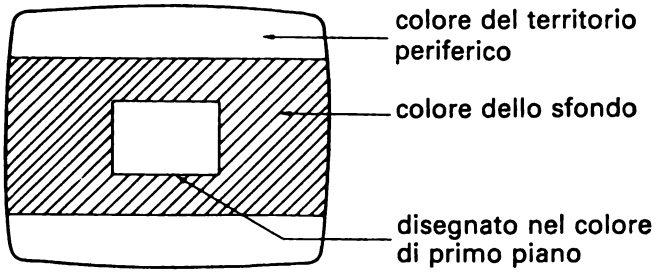
```
10 INPUT "X=";X
20 INPUT "Y=";Y
30 SCREEN 2
40 PSET (X,Y)
50 GOTO 50
```

Le righe 10 e 20 assegnano un valore a X e a Y. La riga 30 contiene l'istruzione indispensabile SCREEN 2. Poi con l'istruzione PSET si disegna il punto. L'ultima istruzione impedisce che il programma finisca. (Dopo la fine del programma si esegue automaticamente SCREEN O).
Per interrompere il programma dobbiamo tenere premuto il tasto CTRL e, nello stesso tempo, premere il tasto STOP (CTRL/STOP).

La tabella sottostante illustra tutte le possibilità di modi SCREEN.

Modo SCREEN	*Descrizione*
0	Per la riproduzione del testo: 40 × 24
1	Per la riproduzione del testo: 32 × 24.
2	Grafica: 256×192 pixels (=punti immagine).
3	Grafica: 64×48 blocchi

COLORE Prima di indicare come si possono usare i colori (in americano: colors), bisogna dire qualcosa a proposito dei termini 'primo piano', 'sfondo', e 'territorio periferico'.
La figura seguente mostra la composizione dello schermo.

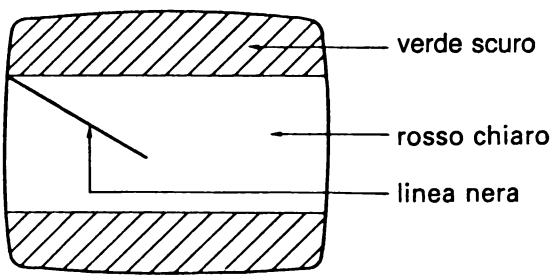


Con il termine primo piano intendiamo segni, figure, simboli e via dicendo, che possiamo rappresentare . Queste cose possono essere rappresentate in un determinato colore, perciò si parla di 'colore di primo piano'.
Il colore su cui raffiguriamo questi segni, figure, simboli e via dicendo, viene chiamato il colore dello sfondo. Infine possiamo notare che si distingue ancora un territorio periferico al quale possiamo anche assegnare un colore.

Un esempio

```
10 SCREEN 2
20 COLOR 1,9,12
30 CLS
40 FOR K=1 TO 100
50 PSET (K,K)
60 NEXT
70 GOTO 70
```

Appare ora l'immagine seguente:



Con le istruzioni delle righe da 40 a 60, vengono rappresentati 100 punti che insieme formano la linea nera.
In questo esempio incontriamo una istruzione COLOR. Questa istruzione ha la seguente formulazione:

`COLOR` colore di primo piano, colore dello sfondo, colore del territorio periferico

Per i colori standard dovete utilizzare la seguente tabella:

cifra/colore	*cifra/colore*	*cifra/colore*	*cifra/colore*
0 trasparente	4 blu scuro	8 rosso	12 verde scuro
1 nero	5 azzurro	9 rosso chiaro	13 magenta
2 verde	6 rosso scuro	10 giallo scuro	14 grigio
3 verde chiaro	7 celeste	11 giallo chiaro	15 bianco

Così nel nostro esempio la linea sarà effettivamente raffigurata con un colore di primo piano nero (1) su uno sfondo rosso chiaro (9) e con un territorio periferico verde scuro (12).
Ancora qualche esempio:

`COLOR 1` : rende nero solo il colore in primo piano (1), in altre parole disegna delle righe nere.

`COLOR ,2` : fa diventare verde il colore dello sfondo. Notate che abbiamo messo una virgola prima del numero.

`COLOR 1,2,11` : rende nero il colore in primo piano (1), verde quello dello sfondo (2) e giallo chiaro quello del territorio periferico (11).

`COLOR ,,11` : cambia soltanto il colore del territorio periferico. Notate che prima della cifra 11 sono state messe due virgole.

Infine, osserviamo ancora che se si vuole cambiare il colore dello sfondo, dobbiamo sempre collocare dopo l'istruzione COLOR, un'istruzione CLS (CLEAR SCREEN: pulisci lo schermo).

Una specificazione di colore dietro un'istruzione grafica

L'istruzione PSET è un'istruzione grafica in quanto, utilizzandola, possiamo costruire delle immagini. Nel seguente capitolo tratteremo ancora delle istruzioni LINE e CIRCLE. Anche queste sono istruzioni grafiche. In una tale istruzione grafica si possono anche includere *direttamente* delle specificazioni di colore e precisamente collocando il numero di colore direttamente dopo l'istruzione. In questo modo viene definito il colore di primo piano.

3 RAPPRESENTAZIONI GRAFICHE: LINE, DRAW, CIRCLE e PAINT

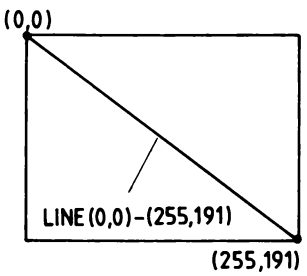
Introduzione In questo capitolo parliamo di tutte le altre possibilitá esistenti per rappresentare delle linee sullo schermo. Le istruzioni LINE e DRAW ci offrono la possibilitá di rappresentare molto facilmente delle linee rette, mentre con CIRCLE possiamo disegnare sullo schermo delle curve (parti di un cerchio o di una ellisse) Tutte queste istruzioni sono organizzate per lo schermo grafico, come abbiamo giá spiegato nel precedente capitolo.

LINE Nella sua formulazione più semplice l'istruzione si presenta così:

```
LINE (X1,Y1)-(X2,Y2)
```

Con (X1,Y1) viene dato il punto iniziale della linea e con (X2,Y2) il punto finale.
Esempio:

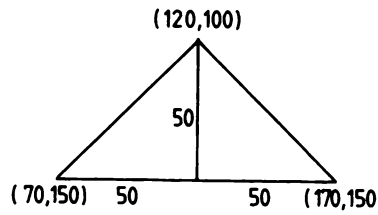
```
10 SCREEN 2
20 LINE (0,0)-(255,191)
30 GOTO 30
```



Come risultato vedremo comparire sullo schermo una linea obliqua.

LINE (0,0)–(255,191)
(255,191)

Nel programma successivo disegneremo un triangolo secondo i seguenti dati:



Programma:

```
10 SCREEN 2
20 LINE (70,150)-(170,150)
30 LINE (170,150)-(120,100)
40 LINE (120,100)-(70,150)
50 GOTO 50
```

Eventualmente possiamo omettere la posizione iniziale. In questo caso viene preso come punto iniziale il punto finale che è stato calcolato in base all'istruzione precedente. Perció con il seguente programma otteniamo lo stesso triangolo:

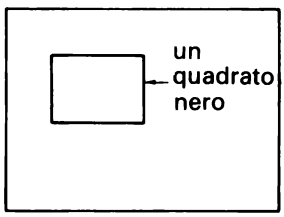
```
10 SCREEN 2
20 LINE (70,150)-(170,150)
30 LINE -(120,100)
40 LINE -(70,150)
50 GOTO 50
```

Inoltre possiamo colorare una linea precisamente seguendo lo stesso metodo usato con PSET, cioè mettendo una virgola e un codice del colore, così:

```
LINE (X1,Y1)-(X2,Y2), codice del colore
```

Con l'istruzione LINE possiamo facilmente disegnare dei semplici blocchetti. Per questa ragione aggiungiamo ,B all'instruzione, così:

```
LINE (X1,Y1)-(X2,Y2), colore ,B
```

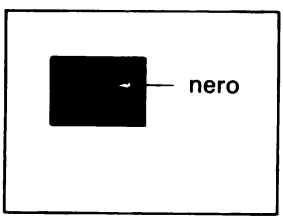


Esempio:

```
10 SCREEN 2
20 LINE (50,50)-100,100), 1,B
30 GOTO 30
```

Come risultato si ottiene: (qui accanto)

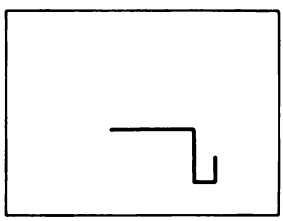
Notate che abbiamo completamente definito il blocchetto indicando nell'istruzione LINE i due vertici che sono obliquamente opposti l'uno all'altro. Se ora al posto di B inseriamo BF, il quadrato viene completamente colorato, osservate l'esempio:




```
10 SCREEN 2
20 LINE (50,50)-(100,100),1,BF
30 GOTO 30
```

si ottiene così: (qui accanto)

DRAW L'istruzione DRAW è un'istruzione molto importante che serve per disegnare linee orizzontali, verticali e oblique. Riportiamo qui sotto un semplice esempio:



```
10 SCREEN 2
20 PSET (127,95)
30 DRAW "R50D50R25U25"
40 GOTO 40
```

Risultato: (qui accanto)

La riga 20 indica il punto di partenza del nostro disegno. L'istruzione successiva DRAW indica, tramite una stringa, ciò che deve essere disegnato, in questo modo:

R50 disegna una linea spostando 'la penna' di 50 punti verso destra (la R sta per 'right' che significa 'destra')
D50 spostati ora verso il basso di 50 punti (la D sta per 'down' che significa 'verso il basso')
R25 spostati poi ancora di 25 punti verso destra.
U25 e poi di 25 punti verso l'alto (la U sta per 'up' che significa verso l'alto)

Una visione generale delle possibilità offerte da DRAW viene data in Panorama sui compiti dell'MSX-BASIC.

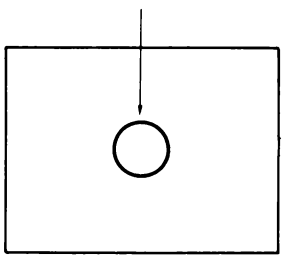
CIRCLE Questa è l'ultima istruzione riguardante la rappresentazione di linee che verrà spiegata. In questo caso l'istruzione riguarda le curve. La formulazione generale è questa:

```
CIRCLE (X,Y),    raggio, colore, angolo iniziale, angolo
                 finale, proporzione tra gli assi
```

e cioè:

(X,Y)	coordinate di un punto mediano
raggio	valore del raggio
colore	codice del colore
angolo iniziale	angolo da dove la curva inizia a essere disegnata (dato in radianti: 0..2π).
angolo finale	angolo fino a dove la curva viene disegnata (dato in radianti: 0..2π)
proporzione	numero che indica di quanto un cerchio debba essere schiacciato, cioè nel caso si voglia disegnare una ellisse. Se si assegna il valora 1.4 si ottiene un cerchio nello SCREEN modo 2 e 3.

cherchio nero con punto mediano (127,95) e raggio R=50



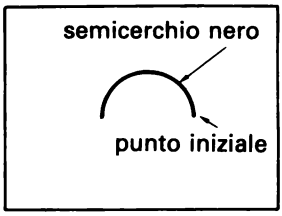
Diamo alcuni esempi:

```
10 SCREEN 2
20 CIRCLE (127,95),50,1,,,1.4
30 GOTO 30
```

Risultato: (qui accanto)

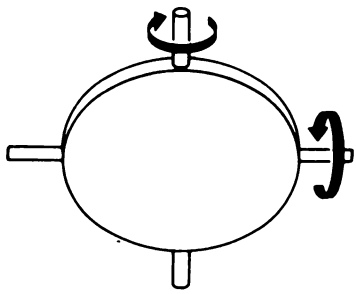
Possiamo vedere l'effetto dell'angolo iniziale e dell'angolo finale nel seguente programma:

```
10 INPUT B
20 INPUT E
30 SCREEN 2
40 CIRCLE (127,95),50,1,B,E,1.4
50 GOTO 50
```



Se si assegna a B il valore 0 e a E il valore 3.1415 si ottiene il seguente semicerchio: (qui accanto)

L'ultimo numero dell'istruzione CIRCLE determina la cosiddetta proporzione tra gli assi. La cosa migliore per capire è osservare la seguente figura:



Se il disco viene girato intorno a uno degli assi, vedremo il disco da un'altra angolazione, cioè vedremo la proporzione tra gli assi.

Questo effetto può essere capito meglio tramite il seguente programma:

```
10 INPUT K
20 SCREEN 2
30 CIRCLE (127,95),50,1,,,K
40 GOTO 40
```

PAINT L'istruzione seguente di cui parleremo in questo capitolo è l'istruzione PAINT. Con questa istruzione possiamo colorare determinate aree soltanto indicando un punto a caso dell'area da colorare.
La formulazione generale è questa:

```
PAINT (X,Y), colore dell'area, colore della linea
```

e cioè:

(X,Y)	le coordinate del punto da indicare
colore dell'area	colore con cui l'area deve essere colorata
colore della linea	colore con cui devono essere indicati i limiti della linea

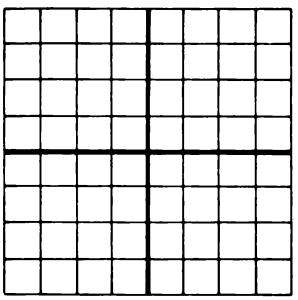
Di solito omettiamo l'ultimo codice in quanto, per lo schermo grafico, il colore del limite dell'area deve essere uguale a quello dell'area. Poiché dobbiamo sempre indicare i limiti dell'area prima che questa venga colorata, il codice del colore rimane in realtà lo stesso.

4 SPRITES

Introduzione

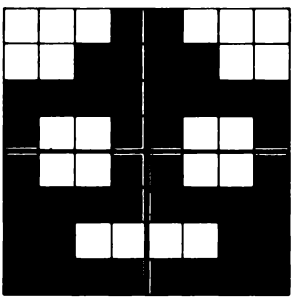
La prima domanda che poniamo in questo capitolo naturalmente è 'cos' è di preciso uno sprite?'. Bene, uno sprite è un disegno piccolissimo che può essere costruito tramite un telaio a quadretti. Questo disegno viene indicato con un numero. Poi tramite una istruzione speciale possiamo eseguire con questo disegnino ogni genere di cosa. Il compito più importante è poter collocare questo piccolo disegno sullo schermo in maniera semplice.
Naturalmente possiamo anche definire tutti i vari tipi di disegno con le istruzioni PRINT e PSET. Per costruire dei simili disegni con le istruzioni PRINT e PSET si ha bisogno di un numero di istruzioni relativamente alto e, tra l'altro una costruzione così veloce del disegno in realtà non è possibile. Lavorare con gli sprite offre perciò dei vantaggi maggiori. In particolare gli sprite forniscono uno strumento efficace per far girare delle figure simili ai packman sullo schermo.

Come si costruisce uno sprite



Adesso mostreremo come possiamo definire un simile disegno, cioè uno sprite. Partiamo da una matrice di formato 8x8 che viene raffigurata qui di seguito.

Poi coloriamo determinati quadretti in nero per ottenere il disegno che desideriamo.
Per ottenere per esempio un fantasmino possiamo colorare i quadretti nella seguente maniera:



```
0001 : 1000   18
0011 : 1100   3C
1111 : 1111   FF
1001 : 1001   99
1001 : 1001   99
1111 : 1111   FF
1100 : 0011   C3
1111 : 1111   FF
```

A sinistra vediamo il fantasmino e nella colonna di numeri a sinistra vediamo come lo stesso disegno possa essere definito in uni e zeri (nero e bianco).
La linea punteggiata è una 'linea ausiliaria'; questa linea divide ogni riga in due file di quattro uni e/o zeri. La prima riga in alto, per esempio, è composta dalle serie 0001 e 1000.
Nella colonna di destra si possono leggere delle notazioni con degli altri codici. Questi codici possono essere ricavati diretta-

mente dalla tabella seguente (N.B. si tratta della cosidetta notazione esadecimale)

serie	*codice*	*serie*	*codice*	*serie*	*codice*	*serie*	*codice*
0000	0	0100	4	1000	8	1100	C
0001	1	0101	5	1001	9	1101	D
0010	2	0110	6	1010	A	1110	E
0011	3	0111	7	1011	B	1111	F

Tramite questi codici definiamo ora uno sprite in un programma adoperando la seguente istruzione:

SPRITE$ (numero)=serie di istruzioni CHR$

Nel nostro caso:

```
SPRITE$(1)=CHR$(&H18)+CHR$(&H3C)+CHR$(&HFF)+
           CHR$(&H99)+CHR$(&H99)+CHR$(&HFF)+
           CHR$(&HC3)+CHR$(&HFF)
```

Notate che qui le serie dei codici 18, 3C, FF, ecc. sono introdotte dalle istruzioni CHR$, in cui ogni codice viene preceduto dai simboli &H.

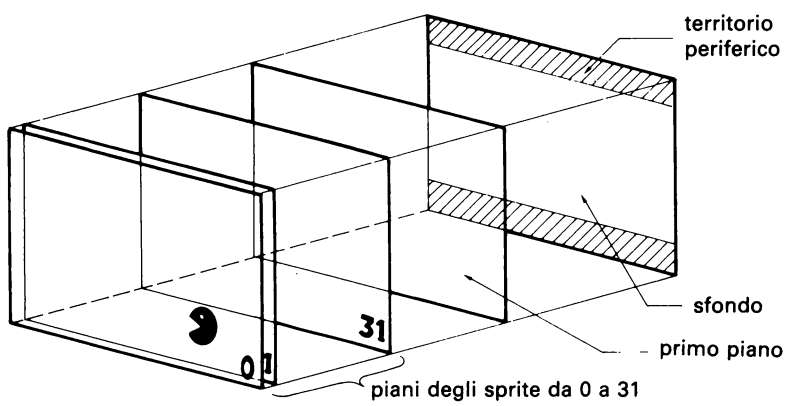
PUT SPRITE

Il nostro sprite (numero 1) è stato completamente realizzato nel modo che abbiamo appena spiegato ed ora possiamo collocare il disegno in uno spazio a caso dello schermo. Per poterlo fare usiamo l'istruzione PUT SPRITE che presenta la seguente formulazione:

PUT SPRITE piano, (x,y), colore, numero di sprite

Quindi:

piano — un numero tra 0 e 31. Per capire meglio guardate il disegno seguente:



Gli sprite si muovono su degli schermi di vetro trasparente che sono situati uno dietro l'altro come si vede nella figura. La posizione di un piano determina quale sprite viene coperto nel caso che due sprite si sovrappongano.

(x,y) le coordinate che indicano la posizione dello sprite che deve essere raffigurato.

colore il numero che indica come deve essere colorato lo sprite.

numero il numero che indica lo sprite in base all'istruzione SPRITE$ (numero).

ON SPRITE GOSUB e SPRITE ON

In molti giochi è importante sapere se in un dato momento, due sprite si sovrappongono o, come si dice comunemente, si scontrano.
Per poterlo constatare l'MSX-BASIC ci offre l'istruzione ON SPRITE GOSUB. Questa istruzione presenta sempre la seguente formulazione:

```
ON SPRITE GOSUB numero di riga
```

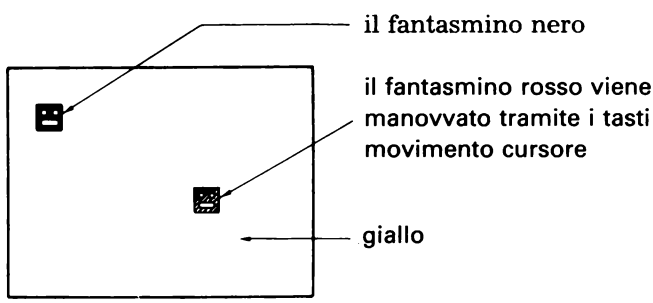
Se il computer constata ora che due sprites si sono sovrapposti, salta alla subroutine il cui numero è indicato nell'istruzione.

A tale scopo è importante attivare l'attenzione del computer. Ciò è possibile grazie all'istruzione SPRITE ON.

A tale scopo è importante attivare l'attenzione del computer. Ció è possibile grazie all'istruzione SPRITE ON.

Un esempio

Il programma seguente ci mostra un esempio. Un fantasmino (quello nero) è fermo, l'altro (quello rosso) invece viene controllato con i tasti movimento cursore. Il funzione STICK è stato giá spiegato nel Panorama dei romandi MSX.
Osservate la figura:



Il programma completo si presenta così:

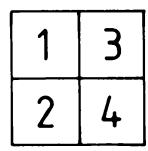
```
10 CLS
20 COLOR,11,11
30 SCREEN 2
```

```
 40 X=100:Y=100
 50 FOR K=1 TO 2
 60 SPRITE$(K)=CHR$(&H18)+CHR$(&H3C)+CHR$(&HFF)+
               CHR$(&H99)+CHR$(&H99)+CHR$(&HFF)+
               CHR$(&HC3)+CHR$(&HFF)
 70 NEXT K
 80 SPRITE ON
 90 PUT SPRITE 0,(40,40),1,1
100 D=STICK(0)
110 IF D=1 THEN Y=Y-1
120 IF D=2 THEN X=X+1:Y=Y-1
130 IF D=3 THEN X=X+1
140 IF D=4 THEN X=X+1:Y=Y+1
150 IF D=5 THEN Y=Y+1
160 IF D=6 THEN X=X-1:Y=Y+1
170 IF D=7 THEN X=X-1
180 IF D=8 THEN X=X-1:Y=Y-1
190 PUT SPRITE 1,(X,Y),6,2
200 ON SPRITE GOSUB 220
210 GOTO 100
220 PLAY "CCC"
230 RETURN
```

Le righe da 10 a 30 si capiscono da sole. La riga 40 determina la posizione iniziale del fantasmino rosso. Subito dopo, dalla riga 50 alla riga 70 vengono definiti i due fantasmini, che tra l'altro hanno precisamente la stessa forma (cioè quella di cui abbiamo già parlato). Dopo di ciò grazie all'istruzione SPRITE ON (riga 80) il computer viene messo in posizione di 'attenzione agli scontri'. La riga 90 colloca il fantasmino nero sullo schermo e precisamente nella posizione (40,40). Con questa istruzione possiamo manovrare il fantasmino rosso.
La riga 200 mostra la 'prova dello scontro'; se avviene uno scontro, il computer salta alla subroutine della riga 220.

Osservazioni Scriviamo infine le seguenti osservazioni sugli sprite:

- possiamo ingrandire lo sprite di un fattore 2 e precisamente trasformando la riga 30 in SCREEN 2,1
- possiamo raffigurare gli sprite anche con una matrice di formato 16x16. Una tale matrice è composta allora di 4 blocchi di matrici di formato 8x8. Questi blocchi vengono numerati in questa maniera:



Il modo più semplice è di definire questo modello in quattro stringhe:

```
A$=CHR$()+ ecc.      (definizione del blocco 1)
B$=CHR$()+ ecc.      (definizione del blocco 2)
```

e lo stesso per C$ e D$

```
SPRITE$(1)=A$+B$+C$+D$
```

Nel caso che si adoperino gli sprite formato 16x16, si usa l'istruzione SCREEN 2,2 o SCREEN 2,3 cioè nel caso che si desideri un ingrandimento con il fattore 2.

- possiamo definire un massimo di 256 modelli sprite usando una matrice di formato 8x8, e un massimo di 64 modelli sprite con una matrice di formato 16x16. Considerate comunque che su ogni piano può essere riprodotto solo uno sprite.
- In una fila possono venire raffigurati orizzontalmente un massimo di 4 modelli sprite.

5 ARCHIVI

Introduzione

Un archivio (in inglese: file) è un nome generico per una raccolta di una certa quantità di dati. Questa raccolta di dati può riguardare per esempio una certa quantità di indirizzi (un archivio di indirizzi), ma anche un programma BASIC o un programma in codice macchina. Qualunque sia il contenuto,il file ha in ogni caso un nome unico con il quale possiamo specificare nelle istruzioni o nei comandi il file in questione. Paragonatelo per un attimo a una libreria. Ogni libro contiene una certa quantità di dati e può essere quindi considerato come un file. Il titolo del libro corrisponde al nome del file. Per che cosa vengono usati i files?

Bene, se disponiamo per esempio di un driskdrive, possiamo sempre memorizzare i nostri programmi come files sul dischetto.

Ma questa non è certamente l'unica possibilità che offre l'MSX per quanto riguarda i files.

Qui di seguito diamo una breve esposizione delle diverse possibilità offerte dal nostro sistema insieme a delle brevi definizioni che dobbiamo includere nelle instruzioni.

CAS Il registratore a cassette.

Il registratore a cassette è usato come un dispositivo di memorizzazione economico per i programmi e i dati.

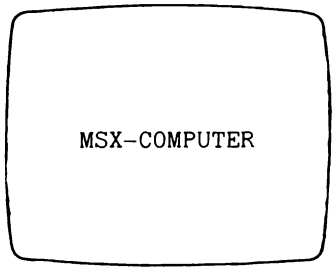
GRP Lo schermo grafico.

Non è molto evidente poterlo considerare questo come dispositivo di memorizzazione dei files. Effettivamente non usiamo questo schermo per esempio insieme al registratore a cassette. Tuttavia è importante sapere che possiamo usare GRP in determinate istruzioni poiché viene offerta la possibilità di riprodurre un testo sullo schermo grafico.

Mostriamo subito un semplice esempio. Più avanti vedremo che cosa significhino le diverse istruzioni specificate in questo programma.

```
10 SCREEN 2
20 OPEN "GRP:TEST" FOR OUTPUT AS #1
30 A$="MSX-COMPUTER"
40 PRESET (50,50)
50 PRINT #1,A$
60 GOTO 60
```

Dopo il comando RUN, vediamo:



In breve, vediamo il testo (cioè: *MSX-COMPUTER*) sullo schermo grafico (SCREEN2)!

CRT Lo schermo alfanumerico.
Come lo schermo grafico, anche lo 'schermo alfanumerico 'può essere considerato come dispositivo di memorizzazione dei files.

LPT La stampante
In particolare la stampante è un'unità periferica che riproduce i files su carta.
Una descrizione dettagliata della stampante viene riportata nell'appendice B.

COM L'interfaccia RS232C
Con questa interfaccia possiamo collegare, per esempio, il nostro computer ad un altro in modo che possano trasmettersi dei messaggi. Invece di 'trasmissione di messaggi', si parla anche di comunicazione, da cui deriva la definizione COM.
Una descrizione dettagliata viene data nell'appendice E.

Da A a F Diskdrive da A a F.
Se nelle istruzioni dei file specifiche assegnamo una lettera da A a F, siamo sicuri che verrà considerato il diskdrive indicato.
Il diskdrive rappresenta il più professionale dispositivo di memorizzazione per i files. E' l'unico dispositivo com i quale possiamo memorizzare i dati secondo una maniera precedentemente indicata.

Il nome di un file Un nome valido di un file può essere composto al massimo di 8 caratteri (ad eccezione del registratore a cassette: massimo 6!). Esempi di nomi sono:

```
TEST, TEST1 e EXP1
```

Accanto alle lettere e ai numeri possono essere aggiunti anche i seguenti simboli:

```
$ & # % @ ( ) _ \ ^ { } ~ e !
```

Dopo il nome valido del file possiamo collocare eventualmente un punto e una cosiddetta estensione (ad eccezione del registratore a cassette). Questo serve per indicare a che *tipo* di dati a si riferisce il file (programmi BASIC, dati generici, e così via).

Esempio:

```
TEST.BAS
```

Questo nome indica che il file TEST si riferisce a un programma BASIC.

Normalmente usiamo le seguenti estensioni per i nomi di files su dischetto.

estensione	*significato*
ASC	file, formato da codici ASCII
ASM	programma assembler
BAK	il cosiddetto 'back up file' (copia da un file)
BAS	programma BASIC
COM	il cosiddetto 'utility programma'
TEX	file alfanumerico
PIC	il cosiddetto 'picture file'

(N.B. i nomi seguenti sono vietati per le estensioni: AUX, CON, LST, PRN e NUL).

A volte vogliamo indicare, con dei determinati comandi, non un solo programma, ma un insieme di programmi. Usiamo allora dei simboli speciali; le cosiddette 'wild cards'.
Quindi:

```
TEST.*
```

vorrà significare; 'tutti i programmi chiamati TEST e con una estensione non specificata'. Se indichiamo:

```
*.BAS
```

intendiamo; 'tutti i programmi con l'estensione BAS'.

Ancora un esempio:

```
TEST*.BAS
```

vuol dire; 'tutti i programmi con estensione BAS e i cui primi 4 caratteri del nome proprio sono composti da TEST'

Oltre al simbolo dell'* usato come 'wild card', anche il punto interrogativo ? può essere usato nella stessa maniera. Perciò l'indicazione del file

```
TEST?
```

si riferisce a tutti i files le cui prime 4 lettere corrispondono a TEST, mentre la quinta non è specificata. Se per esempio disponessimo dei files TEST1, TEST2, TEST3, tutti questi sarebbero compresi nell'indicazione 'TEST?'

Le istruzioni OPEN, PRINT# e CLOSE

Se vogliamo lavorare con i files, dobbiamo prima assegnare un nome e un numero. Questo è possibile grazie all'istruzione OPEN. Dopo di ciò possiamo riempire il file sequenziale grazie all'istruzione PRINT. Infine se lavoriamo con un programma contenente dei files, dobbiamo sempre chiuderlo con l'istruzione CLOSE.
Fino a qui un'esposizione concisa sulle istruzioni necessarie a nominare i files e a fornire dati.
Per un utente agli inizi tutto ciò non sarà completamente chiaro. Illustreremo ora dettagliatamente le istruzioni appena citate con un esempio.
Esempio:

```
10 REM LETTURA DI UN FILE
20 A$="PIPPO"
30 B$="PLUTO"
40 C$="PAPERINO"
50 OPEN "CAS:DATA" FOR OUTPUT AS#1
60 PRINT #1, A$
70 PRINT #1, B$
80 PRINT #1, C$
90 CLOSE #1
```

Le righe da 10 a 40 non comportano dei problemi. Alla riga 50 si trova l'istruzione OPEN. Questa istruzione ha la seguente formulazione:

OPEN 'nome del dispositivo di memorizzazione:nome del file' FOR OUTPUT AS#numero del file

Vediamo che nell'esempio precedente è stato inserito, come nome del dispositivo di memorizzazione, il termine CAS. Ciò significa chiaramente che vogliamo lavorare con un registratore a cassette. Il nome del file è stato abbreviato in DATA. L'ultima parte:

```
FOR OUTPUT AS#1
```

vuo dire: 'trasmetteremo i dati dalla memoria del computer al registratore' e anche 'indichiamo questo file con il numero #1'.Con ciò possiamo vedere che i dati che collochiamo sulla cassetta formano tutti insieme il file 'CAS:DATA' e inoltre che questo file viene sufficientemente indicato dal suo numero (#1).
Al posto del termine OUTPUT avremmo anche potuto usare il termine INPUT. In questo caso il computer capisce che verrano inseriti i dati che provengono dal dispositivo di memorizzazione; la procedura inversa, quindi. Mostreremo un esempio un po' più avanti.
Le righe 60 e 70 mostrano come memorizziamo le stringhe A$, B$ e C$ nel file grazie all'istruzione PRINT#. Notate che subito dopo PRINT si trova il numero del file.
Quando tutti i dati sono stati inseriti, dobbiamo eseguire l'istru-

zione CLOSE. Dopo CLOSE si nota il numero del file. La ragione per l'introduzione dell'istruzione CLOSE è la seguente. Il trasporto effettivo dei dati dalla memoria del computer al dispositivo di memorizzazione avviene attraverso il cosiddetto buffer (memoria di transito). Senza l'istruzione CLOSE esiste la possibilità che il buffer non possa venir completamente letto.

Ancora un esempio:

```
10 REM LETTURA DI UN FILE
20 OPEN "CAS:DATA" FOR INPUT AS#1
30 INPUT #1, A$, B$, C$
40 PRINT A$, B$, C$
50 CLOSE #1
```

Risultato:

```
PIPPO   PLUTO   PAPERINO
```

Vediamo che la costruzione di questo programma è praticamente uguale a quella precedente. Vediamo che dove nel precedente programma si è usato OUTPUT, qui è stato usato invece INPUT. In questo esempio si tratta di una lettura dei dati dal registratore. Al posto delle istruzioni PRINT# vengono ora usate le istruzioni INPUT#; dobbiamo sempre inserire i dati.
L'istruzione PRINT della riga 40 è stata introdotta soltanto per verificare che i dati originali siano stati effettivamente registrati. Infine mostriamo ancora degli esempi sulle istruzioni OPEN senza darvi peró un programma completo.

Esempio 1

```
OPEN "A:PROG.DAT" FOR INPUT AS#3
```

Si tratta chiaramente di un file sul dischetto A dal quale verranno letti i dati. Il numero del file è 3 e il nome è PROG.DAT.

Esempio 2

```
OPEN "GRP:TEST" FOR OUTPUT AS#1
```

Questa istruzione è stata usata nel programma che si trova nell'introduzione a questo capitolo. L'informazione del numero di file 1 viene collocata ora sullo schermo grafico.

Una dettagliata descrizione dell'istruzione OPEN viene data in Panorama sui compiti dell'MSX-BASIC.

Come vengono memorizzati i files sequenziali

Il termine sequenziale significa che le informazioni si susseguono , cioè vengono memorizzate una dopo l'altra. Nel paragrafo che segue parleremo dei simboli 'CR', 'LF' e ',', che vengono utilizzati come segni di divisone. Iniziamo questa descrizione con l'esposizione dell'istruzione PRINT.

Parliamo della virgola (CR e LF)

Se abbiamo eseguito un'istruzione PRINT, il computer avrà collocato il cursore all'inizio di una nuova riga.
L'abbreviazione CR indica che bisogna andare ad una nuova riga, mentre LF si riferisce al trasferimento verso una nuova riga. CR e LF hanno entrambi un loro specifico codice ASCII. Questi simboli vengono anche usati come segni di divisione se per esempio, tramite l'istruzione PRINT#, vengono registrati dei dati su cassetta.
Esempio 1

```
10 OPEN "CAS:ADDR" FOR OUTPUT AS#1
20 A=15:B=23.5:C=10:D=25.2:E=13
30 PRINT #1, A; B; C
40 PRINT #1, D; E
50 CLOSE #1
```

In questo caso i dati su cassetta possono essere riportati su schermo in questa maniera:

15	,	23.5	,	10	CR	LF	25.2	,	13	CR	LF

State attenti quando dovete inserire in un'istruzione PRINT# delle stringhe separate. In questo caso vi consigliamo di inserire sempre anche una virgola.

Esempio 2

```
10 OPEN "CAS:NOMI" FOR OUTPUT AS#1
20 A$="TIZIO":B$="CAIO"
30 PRINT #1, A$;",";B$
40 CLOSE #1
```

Appare perciò:

TIZIO	,	CAIO	CR	LF

Infine scriviamo ancora le seguenti osservazioni:

- Il numero massimo di archivi che possono essere aperti contemporaneamente ammonta a 1, a meno che con il comando MAXFILES non venga indicato un altro numero (per es.MAXFILES=3)

Sul dischetto si possono aprire un massimo di sei files.

- Al posto di PRINT#, si può anche usare PRINT USING#.
- Al posto di INPUT#, si può anche usare LINE INPUT#.

Files ad accesso casuale

Oltre ai files sequenziali l'MSX riconosce anche i files ad accesso casuale. Questa possibilità esiste solo per i dischetti. Nell'appendice D viene data una descrizione dettagliata.

APPENDICI

A

USO DEL REGISTRATORE A CASSETTE

Trattiamo ora brevemente l'uso del registratore a cassette. Usate preferibilmente un registratore dati MSX. Con altri registratori ci possono essere dei problemi al momento della messa a punto del tono e del volume giusti. Soltanto provando e riprovando si ottiene una giusta regolazione!
Iniziamo dal seguente programma dimostrativo:

```
10 REM DEMO CASSETTE
20 END
```

Per poter mantenere questo piccolo programma sulla cassetta, dobbiamo eseguire le seguenti operazioni:

a. collegare il registratore a cassette
b. aver cura che il programma in questione si trovi nella memoria del computer
c. mettere il registratore in posizione 'registrazione' e dare subito dopo il comando CSAVE 'DEMO'
Terminata l'esecuzione appare sullo schermo 'OK'
E' possibile verificare la correttezza della registrazione con il comando CLOAD? 'DEMO'. Dopo CLOAD non dimenticatevi di inserire il punto interrogativo. Questo comando viene dato immediatamente dopo che la cassetta è stata riavvolta e messa in posizione PLAY. Nella memoria del computer si trova ancora il programma originale che ora viene messo a confronto con quello registrato.

Poi per caricare il programma dalla cassetta, usiamo il comando:

```
CLOAD
```

Il primo programma 'visto' dal computer viene caricato. Potete anche assegnare un nome al programma: il comando diventa allora CLOAD 'nome del programma'. Secondo il nostro esempio il comando diventerà:

```
CLOAD"DEMO"
```

Annotiamo qui le seguenti osservazioni:

- Se qualcosa non dovesse funzionare durante il caricamento dalla cassetta, esistono due possibilità:

  – il computer visualizza il messaggio 'DEVICE I/O ERROR' ('errore del dispositivo I/O'.)
  – Il computer non visualizzato alcun messaggio.

  In questo caso riavvolgete il nastro e fate degli esperimenti usando il volume e il tono del vostro registratore per individuare come caricare correttamente. Generalmente potete regolare la manopola del volume su 80% e quella del tono su 'max'. Usate inoltre soltanto cassette al ferro e non quelle al cromo (CrO2).

- Il registratore a cassetta può essere utilizzato anche per memorizzare file sequenziali e programmi.

- Alcune cassette già incise non possono essere copiate. Ciò non dipende dal vostro registratore ma da un dispositivo di protezione presente su tali cassette!

Panorama generale dei comandi

I sequenti commandi riguardano l'uso dei cassette. Una descriptione dettagliata si trova nel Panorama sulle instruzioni dell'MSX-BASIC.

BLOAD	per caricare un programma in codice macchina.
BSAVE	per memorizzare un programma in codice macchina.
CLOAD	per ricevere un programma.
CSAVE	per memorizzare un programma.
INPUT#	per inserire in memoria un file sequenzial.
LINE INPUT#	per inserire un file sequenziale una riga per volta.
LOAD	per ricevere un programma in ASCII.
MERGE	per inserire un programma.
OPEN	specifica come viene usato un file.
PRINT#	per collocare i dati in un file sequenziale.
PRINT# USING	per collocare i dati in un file sequentiale con cui viene indicato anche il formato.
SAVE	per memorizzare un programma in ASCII.

Panorama sulle funzioni

Descrizioni vengono nuoramente ampliate nel Panorama sulle instruzioni dell'MSX-BASIC.

EOF	fine del file
INPUT$#	per inserire i dati alfanumerici
LINE INPUT$#	per inserire i dati alfanumerici una riga alla volta
VARPTR#	per trovare l'indirizzo del blocco di controllo del file

B USO DELLA STAMPANTE

Al computer possono essere collegate sia stampanti MSX che stamper non MSX. Una stampante MSX è una stampante standard per sistemi MSX sviluppata in modo tale da non provocare alcun tipo di problema in nessun caso (per collegamenti, codici di controllo, impostazione dei caratteri). Naturalmente usando una 'stampante non MSX', si può andare incontro a questi tipi di problemi.
Prima di tutto avrete bisogno della stampante per stampare dei listati del programma (comando LLIST). Potete anche usarla per stampare i dati (istruzione LPRINT). Infine potete memorizzare i dati in un file sequenziale (per un esempio si veda la descrizione di MAXFILES nel Panorama sulle istruzioni dell'MSX-BASIC).

Panorama sui comandi

I comandi seguenti riguardano il lavoro con la stampante. Una descrizione più ampia si trova nel Panorama sulle istruzioni dell'MSX-BASIC

CLOSE	per chiudere i files.
LLIST	per stampare un programma.
LFILES	per stampare i nomi dei files che si trovano sul dischetto.
LPRINT	per stampare i dati.
OPEN	specifica come deve essere utilizzato un file (da una stampante non è più caricare, perciò può essere usata soltanto la forma 'OUTPUT').
PRINT#	per stampare i dati come in un file sequenziale.
PRINT# USING	per stampare i dati come in un file sequenziale determinando il formato di stampa.

Panarama sulle funzioni

Verrano nuovamente ampliate nel Panorama sulle istruzioni dell'MSX-BASIC):

VARPTR#	per trovare l'indirizzo del blocco di controllo del file.

C USO DEI CONNETTORI JOYSTICK

Questi connettori vengono indicati sulla macchina con un 'controllo manuale'.
Tramite questi conettori potete connettere un pannello di tasto, una matita luminosa, un 'mouse', o un track ball.

Panorama sui comandi

I comandi seguenti riquardano l'uso del joystick. Una descrizione più ampia si trova nel Panorama sulle istruzioni dell'MSX-BASIC.

ON STRIG GOSUB — controlla se il pulsante d'azione del joystick è stato premuto.
STRIG/ON/OFF/STOP — attiva il controllo del pulsante d'azione deljoystick.

Panorama sulle funzioni

Queste funzioni vengono descritte dettagliatamente in Panorama sui compiti dell'MSX-BASIC.

PAD — indica lo stato del pannello di contatto, della penna ottica, del mouse o 'track ball'.

PDL — indica la posizione del 'controllore del paddle'.

STICK — indica la posizione del joystick.

STRIG — indica se il pulsante d'azione del joystick è premuto.

D USO DEL DISKDRIVE

Se si dispone di un diskdrive, questo può essere utilizzato per memorizzare i dati e i programmi. In molti casi il software viene fornito su un dischetto. Inserito questo dischetto nel diskdrive e acceso il computer, un programma viene generalmente avviato automaticamente. Se ciò si verifica, questo dipende dal fatto che un file chiamato AUTOEXEC.BAS è stato memorizzato sul dischetto. Di questo file se ne parlerà un po'più avanti.
Se non volete che un programma venga automaticamente avviato, accendete prima il computer e dopo il messaggio 'OK' inserite il dischetto nel drive. Con il comando FILES potete vedere tutti i files che si trovano sul dischetto. Anche questo punto verrà descritto più ampiamente.
Nel seguente paragrafo spiegheremo dettagliatamente l'uso del dischetto.
Descriviamo i comandi COPY (per copiare un file), FILES (per poter vedere tutti i files presenti sul dischetto) e KILL (per cancellare i files)..
Successivamente parliamo di FORMAT (per preparare all'uso un dischetto che non è stato ancora mai usato), SAVE (per trascrivere un file sul dischetto) e LOAD (per caricare un file dal dischetto in memoria).
Infine spieghiamo come usare i cosiddetti 'Random Access Files' (file ad accesso casuale). Questa appendice termina con una lista di istruzioni e comandi specifici relativi all'uso dei dischetti.

Indicazioni sui drive e i comandi COPY, FILES e KILL

L'elemento nel quale si inserisce il dischetto viene chiamato comunemente 'drive'. Possiamo collegare il nostro computer anche a più di un drive e a proposito di questo ci si può chiedere naturalmente come si faccia a sapere di quale drive si tratta. La risposta è semplice: indichiamo il drive simbolicamente con una lettera. Se per esempio abbiamo due drive, li possiamo chiamare A e B. Queste lettere vengono anche usate per indicare in quale drive si trova un determinato file.
Perciò:

```
A:TEST.BAS
```

vorrà dire 'il programma TEST.BAS che si trova nel dischetto del drive A'.
Mostreremo ora un esempio di un comando, e precisamente del comando COPY, nel quale appaiono delle indicazioni sia per files che per i drives. Questo è il comando per poter copiare dal file.

Il comando:

```
COPY "A:TEST.BAS" TO "B:"
```

vuol dire: 'copia il programma TEST.BAS, che si trova sul dischetto del drive A, sul dischetto che si trova nel drive B'.
Ancora un esempio:

```
COPY "A:TEST.BAS" TO "B:EXP1.BAS"
```

che significa: 'copia il programma TEST. BAS e memorizzalo con il nome di EXP1.BAS sul dischetto che si trova nel drive B'.

Ancora un esempio e precisamente un esempio che si riferisce al comando che ci offre un panorama di tutti i file che si trovano su un dischetto. Questo comando si chiama FILES.
Perciò:

```
FILES "A:"
```

vuol dire: 'lista tutti i files che si trovano sul dischetto del drive A'.
Possiamo anche indicare un solo nome nel comando FILES per controllare se il file in questione si trova veramente sul dischetto:

```
FILES "A:TEST.BAS"
```

Se questo file è veramente presente, viene riprodotto ancora una volta sullo schermo il nome TEST.BAS. Se invece non è presente, il nostro computer ci manda un avviso:

```
File not found
```

che significa che il file in questione non è stato trovato.

Ancora un esempio e precisamente un esempio che riguarda la cancellazione dei files. Per questo scopo usiamo il comando KILL.
Quindi

```
KILL "A:TEST.BAS"
```

vorrà dire; 'cancella il programma TEST.BAS che si trova sul dischetto del drive A.'
Che cosa vorrà invece dire l'esempio seguente?

```
KILL "A:*.*"
```

...giusto, cancella tutti i programmi che si trovano sul dischetto del drive A.
Tra breve vedremo anche come usiamo i nomi A e B se disponiamo solo di un drive.

AUTOEXEC.BAS Un file con il nome AUTOEXEC.BAS verrà avviato automaticamente dopo l'avviamento del sistema, purchè il relativo disco si trovi nel drive A.

FORMAT Adesso prendiamo un dischetto vuoto ma non lo inseriamo ancora al suo posto. Dobbiamo prima sottoporlo ad un'operazione, che viene chiamata 'formattare'. Ciò significa che deve aver luogo un'operazione preliminare necessaria per poter, subito dopo, inserire i files sul dischetto.
Per questa operazione digita:

```
_FORMAT
```

oppure

```
CALL FORMAT
```

e seguite le istruzione che appaiono sullo schermo. Se il dischetto è stato formatato apparirà sullo schermo:

```
Format complete
Ok
```

Per controllare se abbiamo veramente un dischetto sul quale possiamo memorizzare un programma, battiamo un breve programma dimostrativo:

```
10 PRINT "MSX DISK BASIC"
```

Per poter memorizzare questo breve programma sul dischetto, usiamo il comando SAVE. Diamo perciò il seguente comando:

```
SAVE "A:TEST.BAS"
```

che vuol dire 'registra il programma che si chiama TEST.BAS sul dischetto che si trova nel drive A'. Dopo aver premuto il tasto return il nostro drive emetterà un ronzio... il programma sarà stato veramente memorizzato sul dischetto?
Possiamo andare a controllarlo cancellando tramite il comando NEW il programma che si trova ancora in memoria (provate a fare anche questo!). Adesso carichiamo il programma dal dischetto tramite il comando LOAD:

```
LOAD "A:TEST.BAS"
```

Dopo aver premuto RETURN il drive produrrà di nuovo un ronzio. Se poi diamo il comando LIST potremo vedere chiaramente che anche il nostro programma si trova ora nella memoria del computer.
Naturalmente per controllare se il file TEST.BAS si trovasse veramente sul dischetto, avremmo anche potuto dare il comando FILES di cui abbiamo già parlato, per esempio:

```
FILES A:*.*
```

(fate anche questo!) o in maniera ancora più breve:

```
FILES
```

Disponendo solo di un drive

Nella maggior parte dei casi si dispone sempre di un solo drive, e allora ci si può domandare 'e ora come si fa?'
Bene, in questo caso i nomi A e B non si riferiscono più ai drive ma ai dischetti. Illustriamo meglio con l'esempio seguente:
Supponete che si abbiano due dischetti (A e B quindi) e che si voglia trasferire il programma TEST.BAS dal dischetto A al dischetto B. Supponete anche che il dischetto A si trovi ancora nel drive.
Diamo ora il seguente comando:

```
COPY "A:TEST.BAS" TO "B:"
```

Questo programma verrà prima di tutto caricato nella memoria del computer, e successivamente il computer ci mostrerà:

```
Insert diskette for drive B
and strike a key when ready
```

In questo caso dovremo inserire il dischetto B al suo posto. Se il programma non può essere trasferito tutto in una volta, deve essere fatto per gradi. Il computer indica sempre se dobbiamo inserire il dischetto A o il dischetto B.
Ancora un suggerimento molto pratico: annotate sul dischetto il nome, cioè A o B; ci salva da molti problemi!

Files sequenziali

L'MSX-Disk BASIC offre tra l'altro la possibilità di lavorare con i files sequenziali.
Il principio del file sequenziale è già stato spiegato nell'appendice A, quindi non diremo altro su questo argomento. Ma c'è ancora un punto che merita di essere discusso più da vicino e cioè la funzione APPEND. Grazie a questa funzione possiamo ampliare un file sequenziale, il che non è possibile usando una cassetta.
Se vogliamo memorizzare altri dati in un file già esistente, in altre parole se vogliamo ampliare questo file, scriviamo la seguente instruzione OPEN:

```
OPEN "nome:" FOR APPEND AS#numero
```

Supponete che abbiamo formato un piccolo file con questo breve programma:

```
10 OPEN "A:DAT" FOR OUTPUT AS#1
20 INPUT X$
30 IF X$="FINE" THEN CLOSE #1:END
40 PRINT #1,X$
50 GOTO 20
```

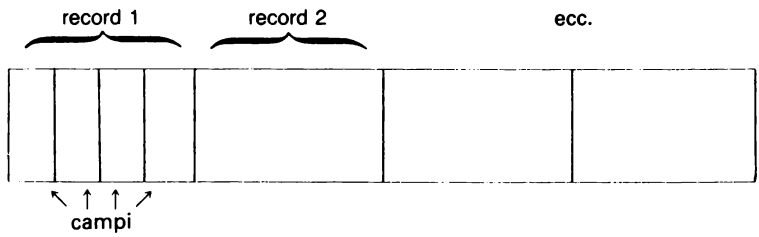
Se abbiamo memorizzato delle stringhe in questa maniera in A:DAT, sul dischetto sarà presente il file DAT.
Il programma seguente ci mostra come possiamo aggiungere dei dati a questo file già esistente:

```
10 OPEN "A:DAT" FOR APPEND AS#1
20 INPUT X$
30 IF X$="FINE" THEN CLOSE #1:END
40 PRINT #1, X$
50 GOTO 20
```

L'unica differenza dal programma precedente sta solo nella parola APPEND.

Files ad accesso casuale

Il cosiddetto 'random access file' è un file con una proprietà particolare, cioè offre la possibilità di poter memorizzare dei dati all'interno di determinate posizioni del file. Per poter capire meglio guardiamo la figura seguente.



In questa figura vediamo una serie di records, e vediamo anche che ogni record è suddiviso nei campi (FIELDS).
Ciò vuol dire che ogni record ha una determinata lunghezza (LEN) in termini di byte e questa lunghezza è formata da campi anche loro di una determinata lunghezza. Per definire un file ad accesso casuale (random access file) dobbiamo prima specificare la lunghezza LEN, per esempio:

```
OPEN "A:DATA" AS#1 LEN=34
```

In questo esempio ogni record ha una lunghezza di 34 byte. Poiché non abbiamo indicato in questa istruzione OPEN l'espressione FOR INPUT, OUTPUT o APPEND, il computer sa automaticamente che si tratta di un file ad accesso casuale.
Inoltre dobbiamo anche indicare come è composto ogni campo e anche il nome simbolico con cui si rimanda a questo campo.
Per esempio

```
FIELD #1, 20 AS A$, 8 AS D$, 4 AS S$, 2 AS I$
```

Vediamo che in questo esempio ogni record è formato di 4 campi chiamati A$, D$, S$ e I$ che comprendono la quantità di byte specificata. La quantità totale dei byte deve essere naturalmente uguale alla quantità che abbiamo espresso in LEN (controllate che ciò sia vero!).

Ora che abbiamo spiegato la struttura di un file, possiamo vedere come vengono sistemati i dati in un file. Perciò usiamo le istruzioni LSET e RSET. LSET si usa sempre per memorizzare delle stringhe, mentre RSET serve per memorizzare dei numeri. L'esempio seguente ci mostra le diverse possibilità:

```
LSET A$ = WA$
RSET D$ = MKD$(WD#)
RSET S$ = MKS$(WS!)
RSET I$ = MKI$(WI%)
```

Qui WA$ rappresenta una stringa, WD# un numero in doppia precisione, WS! un numero in singola precisione e W% un numero intero. Notate che ogni numero deve essere sempre prima trasformato in stringa con la funzione adattata a questo scopo.

Adesso dobbiamo inserire questi valori in un file... ma come? Bene, questo dipende da dove vogliamo collocare questi dati, cioè in quale record. Per collocare i dati si usa questa istruzione:

```
PUT #1, numero del record
```

per esempio

```
PUT #1,3
```

Notate bene che qui tutti i dati che abbiamo assegnato a A$, D$, S$ e I$ tramite LSET e RSET sono stati memorizzati in un file, quindi nella stessa maniera possiamo riassegnare dei nuovi dati a A$, D$, S$ e I$ che possono così essere nuovamente memorizzati in un altro record.

Infine, come ultima cosa, dobbiamo sempre chiudere il file alla fine di un programma, e quindi:

```
CLOSE #1
```

Durante la visualizzazione definiamo il file ad accesso casuale nella stessa maniera in cui è stata già indicato. Se per lo stesso file di un programma dobbiamo eseguire sia delle azioni di input che di output, basta definire il file in questione una volta sola e sempre nella maniera ormai conosciuta.
La rappresentazione si ottiene con l'istruzione GET.
Quindi con:

```
GET #1,3
```

verrà assegnato il contenuto dei campi del record 3 a A$, D$, S$ e a I$. Questi valori li possiamo per esempio stampare con:

```
PRINT A$
PRINT CVD(D$)
PRINT CVS(S$)
PRINT CVI(I$)
```

Notate le istruzioni CVD, CVS e CVI per ottenere nuovamente dei numeri.

Panorama generale dei commandi

I seguenti commandi riguardano l'uso dei dischetti. Una descrizione dettagliata si trova nel Panorama sulle istruzioni dell'MSX-BASIC.

BLOAD	per caricare un programma in codice macchina o la memoria del video.
BSAVE	per memorizzare un programma in codice macchina o la memoria del video.
CLOSE	per chiudere i files.
COPY	per copiare i files o parti dello schermo.
DSKO$	per collocare i dati in un settore specificato.
FIELD	per avere accesso al buffer usato da un file ad accesso casuale.
FILES	dà in visione tutti i files su un dischetto.
LFILES	è uguale al comando precedente, ma ora i files vengono stampati con una stampante.
_FORMAT	per formattare un dischetto.
GET	per inserire un record nel buffer di un file ad accesso casuale.
INPUT#	per inserire in memoria un file sequenziale.
KILL	per cancellare un file.
LINE INPUT#	per inserire un file sequenziale una riga per volta.
LOAD	per ricevere un programma.
LSET, RSET	viene utilizzato dai file ad accesso casuale.
MAXFILES	indica il numero massimo di file che possono essere aperti (massimo 6 file).
MERGE	per inserire un programma.
NAME	per assegnare un nuovo nome di file.
OPEN	specifica come viene usato un file.
PRINT#	per collocare i dati in un file sequenziale.
PRINT# USING	per collocare i dati in un file sequenziale con cui viene indicato anche il formato.
PUT	viene utilizzato con i files ad accesso casuale.
SAVE	per memorizzare un programma.
_SYSTEM	per tornare al sistema MSX-DOS.

Panorama sulle funzioni

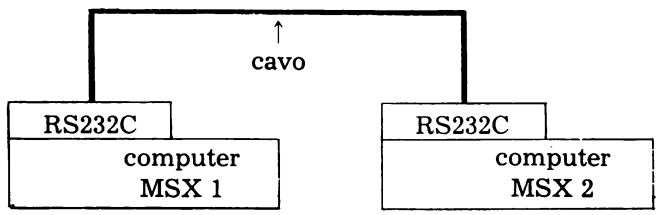
Descrizioni vengono nuovamente ampliate nel Panorama sulle istruzioni dell'MSX-BASIC:

CVI, CVS, CVD	per convertire le stringhe in numeri. Vengono utilizzate con i files ad accesso casuale.
DSKF	determina lo spazio rimasto ancora sul dischetto.
DSKI$	per caricare un settore.
EOF	fine del file.
INPUT$#	per inserire i dati alfanumerici.

LINE INPUT$#	per inserire i dati alfanumerici una riga alla volta.
LOC	indica l'ultima locazione del file specificato.
LOF	indica la lunghezza di un file.
MKI$, MKS$, MKD$	Si veda CVI, CVS e CVD. Si tratta ora della conversione di un numero in una stringa.
VARPTR#	per trovare l'indirizzo del blocco di controllo del file.

E USO DELL'INTERFACCIA RS232C

La comunicazione gioca un ruolo molto importante nel mondo dei computer. Intendiamo cioè dire che possiamo mandare informazioni da un posto all'altro. Pensate un attimo a due computer che sono collegati fra loro. Non è meraviglioso poter trasmettere dei programmi o dei dati in maniera così semplice? Bene, per renderlo possibile il computer MSX ha sviluppato una interfaccia RS232C. Questo dispostitivo serve per trasformare le informazioni che devono essere trasmesse in uno speciale formato adatto a questo scopo.
Quando parliamo di RS232C, parliamo di un'interfaccia sequenziale in quanto gli uni e gli zeri con cui tutti i dati sono stati determinati, vengono trasmessi uno dopo l'altro, cioè in serie (quindi sequenziale).
Per far svolgere questo programma come si deve, si ha bisogno naturalmente di segnali di controllo. Mostreremo qui di seguito come due computer MSX dotati ambedue di una interfaccia RS232C possano comunicare fra di loro. Guardate il disegno:



Per chiarezza: nel nostro esempio un computer MSX comunica con un altro computer MSX, ma in generale l'altro computer potrebbe anche essere un qualsiasi altro dispositivo dotato di una interfaccia RS232C (pensate per esempio a un modem che comunica via telefono con un altro computer).
L'interfaccia è formata di una cartuccia senza cavo. Su questa cartuccia si trova un connettore per slot (il 25 pins Centronics connector).
In commercio si trovano diversi cavi che possono allacciare le cartucce dei due computer (vedere la figura).

Trasmettere un programma

Supponete di battere nel computer 1. il seguente programma:

```
10 PRINT "RS232C"
20 PRINT "MSX"
```

Diamo ora il seguente comando:

```
SAVE "COM:"
```

Vogliamo 'salvare' (SAVE) questo programma tramite l'indicazione COM, che si riferisce alla nostra interfaccia RS232C. Se ora scriviamo sul computer 2:

```
LOAD "COM:"
```

e dopo aver premuto il tasto return, il programma verrà trasmesso in entrambi i computer. Fate attenzione dopo aver dato il comando SAVE a non aspettare troppo prima di battere il comando LOAD, altrimenti appare un messaggio di errore (Device I/O error). La ragione è che battendo il comando SAVE si intende che il computer 1 trasmetterà i dati e controllerà se il computer 2 'è predisposto' (tramite indicazione del comando LOAD) a ricevere i dati. Se questo intervallo dovesse durare troppo a lungo sullo schermo appare il messaggio d'errore.
Infatti questo vale per ogni comunicazione: se il dispositivo ricevente non indica 'predisposto a ricevere i dati', il collegamento viene automaticamente interrotto.

Trasmettere i dati

L'esempio seguente ci mostra come possiamo trasmettere dati sotto forma di un file dal computer 1 al computer 2.
Nel computer 1 battiamo il seguente programma:

```
10 OPEN "COM:"FOR OUTPUT AS#1
20 INPUT A$
30 PRINT #1, A$
40 CLOSE #1
```

Nel computer 2 battiamo questo programma:

```
10 OPEN "COM:" FOR INPUT AS#1
20 INPUT #1, A$
30 PRINT A$
40 CLOSE #1
```

Se ora, dopo l'inizio del programma, inseriamo nel computer 1 la stringa MSX, sullo schermo del computer 2 (se anche nel programma del computer 2 abbiamo dato il comando RUN) apparirà nello stesso tempo la stringa MSX.

Ancora un esempio.
Se diamo ad entrambi i computer il comando

```
CALL COMTERM
```

tutto quello che viene inserito nel computer 1 diventa visibile anche sul computer 2 e viceversa tutto quello che viene inserito nel computer 2 è visibile anche sul computer 1. In questo caso abbiamo fatto dei computer due semplici terminali (una macchina da scrivere che trascrive dati) e in questo modo possiamo scambiare delle semplici notizie.

Collegamento con un dispositivo diverso dal computer MSX (p.es. con un computer di un'altra marca)

Detto in generale questo non è un argomento. Una delle ragioni dipende dal fatto che dobbiamo mettere a punto moltissime cose come:
- la velocità di trasmissione
- il cosidetto controllo di parità
- la lunghezza dei byte
- il numero dei cosiddetti stopbit

e tante altre.
Diamo ancora un avviso ai principianti; comunicare con l'interfaccia RS232C non sarà molto semplice se non disponete di una necessaria conoscenza tecnica.
Ma, anche per i principianti può essere possibile, se possono usufruire di una descrizione chiara di come organizzare la comunicazione!

Panorama dei comandi

I comandi seguenti sono utilizzati quando si lavora con l'interfaccia RS232C. Una descrizione dettagliata si trova nel Panorama sulle istruzioni dell'MSX BASIC.

CALL COMINI	per la messa a punto della interfaccia RS232C (velocità in baud, lunghezza delle 'porzioni'che vengono trasmesse ecc.).
CALL COMON CALL COMOFF CALL COMSTOP CALL COM	istruzioni che vengono utilizate per eseguire automaticamente una subroutine in BASIC nel caso venga ricevuto un simbolo via interfaccia RS232C.
CALL COMTERM	avvia il cosiddetto 'modo di simulazione del terminale'.
CALL COMBREAK	per trasmettere i cosiddetti 'caratteri di interruzione'.
CALL COMDTR	per interrompere temporaneamente interfaccia RS232C.
CALL COMSTAT	usato quando si verifica un errore per individuare cosa è successo.
CLOSE	per chiudere il collegamento con l'interfaccia RS232C.
INPUT#	per ricevere i dati via interfaccia RS232C
LOAD	per ricevere un programma via interfaccia RS232C.
LINE INPUT#	per ricevere i dati una riga alla volta via interfaccia RS232C.
MERGE	per inserire un programma via interfaccia RS232C.
OPEN	per aprire il collegamento con l'interfaccia RS232C.
PRINT#	per trasmettere i dati via interfaccia RS232C.
PRINT# USING	per trasmettere i dati via interfaccia RS232C e con cui inoltre si determina il formato.
SAVE	per trasmettere un programma via interfaccia RS232C.

Panorama delle funzioni

Descrizioni vengono di nuovo ampliate nel Panorama sulle istruzioni dell'MSX BASIC):

EOF	fine del file trasmesso via interfaccia RS232C.
INPUT$#	per ricevere i dati alfanumerici via interfaccia RS232C.
LINE INPUT$#	per ricevere i dati alfanumerici una riga per volta via interfaccia RS232C.
VARPTR#	per trovare l'indirizzo del blocco di controllo del file.

F SEGNI E ESPRESSIONI SPECIALI

Segni di operazione

La tabella sottostante ci mostra un quadro dei simboli che possono essere usati nelle operazioni matematiche. L'ultima cifra indica il valore di priorità: le espressioni vengono eseguite secondo questo valore di priorità (dalla cifra più piccola a quella più grande). Se il valore di priorità è uguale, le espressioni vengono eseguite seguendo l'ordine da sinistra verso destra.

segno	*significato*	*esempio*	*priorità*
+	addizione	3+5	6
–	sottrazione	5–3	6
*	moltiplicazione	5x3	3
/	divisione	5/3	3
^	elevazione a potenza	2^3	1
–	cambiamento di segno	–3	4
MOD	resto divisione intera	12 MOD 3	5

Segni nelle espressioni logiche

I segni seguenti possono apparire nelle espressioni in cui viene controllato se sono soddisfatte o no (per esempio dopo il termine IF)

segno	*significato*	*esempio*
=	è uguale a	IF A=B THEN...
<	minore di	IF A<B THEN...
>	maggiore di	iF A>B THEN...
<> o ><	diverso da	IF A<>B THEN...
<= o =<	minore o uguale a	IF A<=B THEN...
>= o =>	maggiore o uguale a	IF A>=B THEN...

Termini usati per combinare insieme espressioni logiche

Le espressioni come A=B e C<>D possono essere combinate con dei termini specifici.
La tabella seguente mostra i diversi termini insieme alla tabella di verità. In questa tabella viene indicato con 1 l'espressione 'soddisfatta' (che è giusta) e con 0 l'espressione 'non soddisfatta' (che non va bene, che è sbagliata). Facciamo un esempio con il termine AND:
E1 **AND** E2
Il risultato è 1 se E1 e E2 sono uguali ad 1. Con E1 e E2 si intendo-

no espressioni come A=B e C<>D. La tabella indica in questo caso che l'espressione completa è soddisfatta (1) se anche E1 e E2 sono soddisfatte(1)

termine			*tabella verità*
NOT	E1		NOT E1
(negazione)	1		0
	0		1
AND	E1	E2	E1 AND E2
	1	1	1
(prodotto logico)	1	0	0
	0	1	0
	0	0	0
OR	E1	E2	E1 OR E2
(somma logica)	1	1	1
	1	0	1
	0	1	1
	0	0	0
XOR	E1	E2	E1 XOR E2
(OR esclusivo)	1	1	0
	1	0	1
	0	1	1
	0	0	0
EQV	E1	E2	E1 EQV E2
	1	1	1
	1	0	0
	0	1	0
	0	0	1
IMP	E1	E2	E1 IMP E2
implicazione logica	1	1	1
	1	0	0
	0	1	1
	0	0	1

Segni di operazioni per stringhe

Con le stringhe si usa soltanto il segno + per collegarle fra loro. Esempio:
"AB"+"CD" diventa "ABCD"

G

MESSAGGI DI ERRORE

Bad allocation table (60)
Il dischetto non è stato inizializzato(cioè non è stato formattato, o formattato in maniera giusta).

Bad drive name (62)
Viene indicato un dischetto drive che non c'è.

Bad file mode (61)
Si usa PUT, GET e EOF con un file sequenziale, o si cerca di caricare un file ad accesso casuale con LOAD. Infine possiamo trovare questo messaggio di errore quando l'istruzione OPEN viene usata con un file che non è stato definito correttamente.

Bad file name (56)
Il nome dell'archivio (file) non è giusto. Il nome per definire il 'device' in un comando OPEN, SAVE o LOAD non è giusto.

Bad file number (52)
Il numero del file è più grande di quello consentito o viene indicato un file che non è stato ancora 'aperto'.

Can't CONTINUE (17)
Si prova a dare il comando CONT anche se il programma non è stato interrotto da un errore o da un'istruzione STOP. Appare anche se cambiamo il programma e proviamo il CONT subito dopo.

Device I/O error (19)
Il caricamento di un programma o di un archivio non funziona bene. Il registratore a cassette è installato bene? E' stata indicata la porta I/O giusta?

Direct statement in file (57)
In un programma che viene caricato appare un'istruzione senza numero di riga, oppure in un file che viene caricato non cé'un programma.

Disk full (66)
Tutte le capacità del dischetto sono state utilizzate

Disk I/O error (69)
Viene constatato un errore durante un'azione di inserimento. Questo è quello che si chiama 'fatal error' (l'errore fatale) cioè vuol dire che il sistema MSX Disk BASIC deve essere riniziato dal programmatore.

Disk offline (70)
Non c'è alcun dischetto nel drive indicato

Disk write protected (68)
Si è cercato di cancellare qualcosa da un dischetto 'write protected' (cioè protetto per la scrittura)

Division by zero (11)
Dividere per zero non è permesso. Questo errore appare anche se si cerca di dividere per una variabile alla quale in precedenza non era stato assegnato alcun valore.

Field Overflow (50)
La quantità di bytes assegnati nella istruzione FIELD è superiore a 256.

File already exists (65)
Il nuovo nome del file (tramite il comando NAME) è un nome che è già stato usato per un file che si trova sul dischetto.

File already open (54)
Si cerca di aprire un file che è già stato aperto.

File not found (53)
Un LOAD, un comando KILL o un'istruzione OPEN rimanda ad un file che non si trova sul dischetto.

File not OPEN (59)
Si rimanda ad un archivio che non è stato ancora aperto da un'istruzione OPEN

File still open (64)
Il file non è stato chiuso

Illegal direct (12)
Si prova ad eseguire un'istruzione come se fosse un comando, anche se ciò non è possibile

Illegal function call (5)
Viene richiamata una funzione BASIC con un argomento sbagliato. Questo messaggio si incontra:

1 Con un vettore a indice negativo, o con indici che cadono fuori della dimensione di un vettore;
2 con un argomento negativo con LOG o SQR;
3 con una mantissa negativa con esponente reale;
4 con l'uso di una funzione USR senza che venga specificato un indirizzo di partenza;
5 con un argomento sbagliato in MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR, ON...GOTO;
6 quando viene dato un comando grafico mentre lo schermo non è ancora predisposto nel modo grafico.

Input past end (55)
Quando si prova a leggere un dato da un archivio nonostante che tutti i dati siano stati già letti. Può succedere anche quando l'archivio è vuoto, cioè quando non contiene più alcun dato.

Internal error (51)
L'interprete BASIC stesso presenta un errore, per esempio qualcosa che non funziona bene nella memoria ROM.

Line buffer overflow (25)
Il buffer (memoria di transito) per l'inserimento di una riga è completo.

Missing operand (24)
C'è qualcosa di sbagliato in uno dei dati che deve essere usato in un'espressione.

NEXT without FOR (1)
E' stata trovata un'istruzione NEXT senza la relativa istruzione FOR.

NO RESUME (21)
Il programma di gestione degli errori non contiene alcuna istruzione RESUME.

OUT of DATA (4)
Quando si prova a leggere, con un'istruzione READ, dei dati che non sono stati collocati in una lista DATA.

Out of memory (7)
Quando il programma è troppo lungo per lo spazio di memoria disponibile, o contiene troppe variabili, troppe istruzioni GOSUB, o cicli FOR 'nidificati'.

Out of string space (14)
Lo spazio di memoria disponibile per le stringhe è completo. Tramite l'istruzione CLEAR si può ampliare questo spazio di memoria.

Overflow (6)
Un'operazione fornisce un numero troppo grande (il cosiddetto overflow). Può anche essere che un determinato valore cada fuori della portata permessa.

Redimensioned array (10)
Quando si cerca di specificare due volte la stessa variabile con l'istruzione DIM. Può succedere se un'istruzione DIM segue un'istruzione nella quale appare già un vettore di variabili.

Rename across disk (71)
Tramite il comando NAME viene assegnato un nuovo nome al file; le indicazioni del drive sono diverse dal drive nel quale si trova il dischetto con il file in questione.

RETURN without GOSUB (3)
Quando si trova un'istruzione RETURN senza che sia stato effettuato, tramite GOSUB, un salto ad una subroutine.

RESUME without error (22)
Quando si trova un'istruzione RESUME senza la routine di gestione degli errori relativa.

Sequential I/O only (58)
Si è cercato di trattare il file sequenziale come un file ad accesso casuale.

String formula too complex (16)
L'espressione è troppo complicata e deve perciò essere suddivisa in parti più piccole.

String too long (15)
Quando la stringa è composta da più di 255 caratteri.

Subscript out of range (9)
Quando si usano gli indici di vettori che eccedono la dimensione specificata.

Syntax error (2)
Indicazione di errore generale: l'espressione non soddisfa le regole del BASIC.

Too many files (67)
Quando si cerca di creare un nuovo file nonostante siano già disponibili 255 files.

Type mismatch (13)
Quando si cerca di assegnare ad una variabile stringa un valore numerico o viceversa.

Undefined line number (8)
Quando c'è un rinvio ad un numero di riga non esistente.

Undefined user function (18)
Viene richiamata una funzione FN che non è stata precedentemente definita dall'istruzione DEF FN.

Unprintable error (23, 26-49, 60-255)
Per il codice indicato non esiste alcun messaggio di errore standard.

Verify error (20)
Viene rilevata una differenza tra il programma registrato e quello presente in memoria.

H SEQUENZE DI ESCAPE

Con il termine 'Sequenze di Escape' intendiamo un codice speciale con il quale si definiscono determinate operazioni sullo schermo.
Se per esempio vogliamo dare il codice <ESC>B indicato come sequenza di escape, dobbiamo battere:

```
PRINT CHR$(27)+"B"
```

La tabella seguente ci mostra le diverse possibilità.

<ESC>	sposta il cursore una riga verso l'alto
<ESC> B	Sposta il cursore una riga verso il basso
<ESC> C	sposta il cursore di una posizione verso destra
<ESC> D	sposta il cursore di una posizione verso sinistra
<ESC> H	sposta il cursore nell'angolo in alto a sinistra
<ESC> Y	<numero di riga+32> <numero di colonna+32> colloca il cursore alla posizione specificata
<ESC> j	CLS
<ESC> E	CLS
<ESC> K	cancella fino alla fine della riga
<ESC> J	cancella fino alla fine dello schermo
<ESC> l	cancella la riga intera
<ESC> L	inserisce una riga
<ESC> M	rimuove una riga
<ESC> x4	cambia la forma del cursore in un quadratino
<ESC> x5	disattiva il cursore
<ESC> y4	cambia la forma del cursore in una lineetta
<ESC> y5	riattiva il cursore

I CODICI DI CONTROLLO

Un certo numero di tasti possono svolgere una funzione speciale se vengono premuti insieme al tasto CTRL e, se successivamente, tenendo sempre premuto il tasto CTRL, si preme L, lo schermo viene cancellato. Annotiamo questa azione con il nome CTRL/L. Quindi CTRL/R vorrà dire: premiamo R mentre teniamo premuto CTRL. Qui di seguito diamo una visione globale di tutte le possibilità.

CTRL/A	il tasto che segue ora, rimanda ad un simbolo della serie dei simboli alternativi.
CTRL/B	sposta il cursore alla parola precedente.
CTRL/C	ferma il comando AUTO.
CTRL/E	cancella tutti i simboli dal cursore in poi.
CTRL/F	sposta il cursore alla parola seguente.
CTRL/G	BEEP.
CTRL/H	corrisponde al tasto BS.
CTRL/I	equivale a TAB: manda allo stop successivo.
CTRL/J	sposta il cursore all'inizio della riga seguente.
CTRL/K	sposta il cursore all'angolo in alto a sinistra.
CTRL/L	equivale a CLS.
CTRL/M	equivale al tasto RETURN.
CTRL/N	sposta il cursore alla fine della riga (cioè al primo posto libero dopo l'ultimo simbolo della riga).
CTRL/R	equivale al tasto INS
CTRL/U	cancella la riga.
CTRL/\	corrisponde al tasto cursore →
CTRL/]	corrisponde al tasto cursore ←
CTRL/^	corrisponde al tasto cursore ↑
CTRL/–	corrisponde al tasto cursore ↓

J COSTRUZIONE DEI SIMBOLI

In questa appendice viene offerto un panorama generale di tutti i simboli (caratteri) con i rispettivi codici. Questi codici possono essere usati per esempio con le funzioni CHR$. Così si può ottenere il carattere A tramite PRINT "A" ma anche usando PRINT CHR$(65). Nel modo SCREEN 0 non vengono rappresentate le due colonne a destra della matrice con cui ogni simbolo viene definito.
Potete vedere le differenze se confrontate da voi stessi i due esempi riportati qui sotto.

```
10 SCREEN 0
20 PRINT CHR$(210)
```

e

```
10 SCREEN 1
20 PRINT CHR$(210)
```

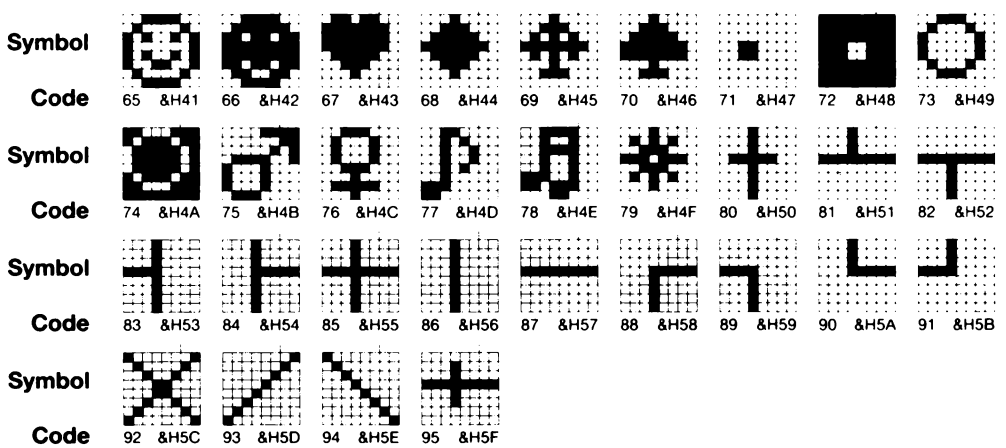
I simboli standard sono raffigurati nelle pagine seguanti.

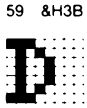
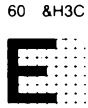
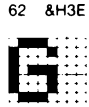
Simboli alternativi

Oltre ai simboli di cui abbiamo già parlato ne esistono degli altri che si ottengono scrivendo CHR$(1) nella istruzione PRINT. Per esempio:

```
10 SCREEN 1
20 PRINT CHR$(65)
30 PRINT CHR$(1) + CHR$(65)
```

La riga 20 genera una A e nella riga 30 si ottiene come risultato il disegno di una faccia.



Symbol									
Code	32 &H20	33 &H21	34 &H22	35 &H23	36 &H24	37 &H25	38 &H26	39 &H27	40 &H28
Symbol									
Code	41 &H29	42 &H2A	43 &H2B	44 &H2C	45 &H2D	46 &H2E	47 &H2F	48 &H30	49 &H31
Symbol									
Code	50 &H32	51 &H33	52 &H34	53 &H35	54 &H36	55 &H37	56 &H38	57 &H39	58 &H3A
Symbol									
Code	59 &H3B	60 &H3C	61 &H3D	62 &H3E	63 &H3F	64 &H40	65 &H41	66 &H42	67 &H43
Symbol									
Code	68 &H44	69 &H45	70 &H46	71 &H47	72 &H48	73 &H49	74 &H4A	75 &H4B	76 &H4C
Symbol									
Code	77 &H4D	78 &H4E	79 &H4F	80 &H50	81 &H51	82 &H52	83 &H53	84 &H54	85 &H55
Symbol									
Code	86 &H56	87 &H57	88 &H58	89 &H59	90 &H5A	91 &H5B	92 &H5C	93 &H5D	94 &H5E
Symbol									
Code	95 &H5F	96 &H60	97 &H61	98 &H62	99 &H63	100 &H64	101 &H65	102 &H66	103 &H67
Symbol									
Code	104 &H68	105 &H69	106 &H6A	107 &H6B	108 &H6C	109 &H6D	110 &H6E	111 &H6F	112 &H70
Symbol									
Code	113 &H71	114 &H72	115 &H73	116 &H74	117 &H75	118 &H76	119 &H77	120 &H78	121 &H79
Symbol									
Code	122 &H7A	123 &H7B	124 &H7C	125 &H7D	126 &H7E	127 &H7F	128 &H80	129 &H81	130 &H82
Symbol									
Code	131 &H83	132 &H84	133 &H85	134 &H86	135 &H87	136 &H88	137 &H89	138 &H8A	139 &H8B
Symbol									
Code	140 &H8C	141 &H8D	142 &H8E	143 &H8F	144 &H90	145 &H91	146 &H92	147 &H93	148 &H94

Symbol									
Code	149 &H95	150 &H96	151 &H97	152 &H98	153 &H99	154 &H9A	155 &H9B	156 &H9C	157 &H9D
Symbol									
Code	158 &H9E	159 &H9F	160 &HA0	161 &HA1	162 &HA2	163 &HA3	164 &HA4	165 &HA5	166 &HA6
Symbol									
Code	167 &HA7	168 &HA8	169 &HA9	170 &HAA	171 &HAB	172 &HAC	173 &HAD	174 &HAE	175 &HAF
Symbol									
Code	176 &HB0	177 &HB1	178 &HB2	179 &HB3	180 &HB4	181 &HB5	182 &HB6	183 &HB7	184 &HB8
Symbol									
Code	185 &HB9	186 &HBA	187 &HBB	188 &HBC	189 &HBD	190 &HBE	191 &HBF	192 &HC0	193 &HC1
Symbol									
Code	194 &HC2	195 &HC3	196 &HC4	197 &HC5	198 &HC6	199 &HC7	200 &HC8	201 &HC9	202 &HCA
Symbol									
Code	203 &HCB	204 &HCC	205 &HCD	206 &HCE	207 &HCF	208 &HD0	209 &HD1	210 &HD2	211 &HD3
Symbol									
Code	212 &HD4	213 &HD5	214 &HD6	215 &HD7	216 &HD8	217 &HD9	218 &HDA	219 &HDB	220 &HDC
Symbol									
Code	221 &HDD	222 &HDE	223 &HDF	224 &HE0	225 &HE1	226 &HE2	227 &HE3	228 &HE4	229 &HE5
Symbol									
Code	230 &HE6	231 &HE7	232 &HE8	233 &HE9	234 &HEA	235 &HEB	236 &HEC	237 &HED	238 &HEE
Symbol									
Code	239 &HEF	240 &HF0	241 &HF1	242 &HF2	243 &HF3	244 &HF4	245 &HF5	246 &HF6	247 &HF7
Symbol									
Code	248 &HF8	249 &HF9	250 &HFA	251 &HFB	252 &HFC	253 &HFD	254 &HFE	255 &HFF	

K NOMI RISERVATI

Le seguenti parole hanno un significato riservato. Le parole contrassegnate da una stelleta indicano degli ampliamenti futuri.

ABS	DEFSTR	KEY	PAINT	STRING$
AND	DELETE	KILL	PDL	SWAP
ASC	DIM	LEFT$	PEEK	TAB(
*ATTR$	DRAW	LEN	PLAY	TAN
ATN	DSKF	LET	POINT	THEN
AUTO	DSKI$	LFILES	POKE	TIME
BASE	DSKO	LINE	POS	TO
BEEP	ELSE	LIST	PRESET	TROFF
BIN$	END	LLIST	PRINT	TRON
BLOAD	EOF	LOAD	PSET	USING
BSAVE	EQV	LOC	PUT	USR
CALL	ERASE	LOCATE	READ	VAL
CDBL	ERL	LOF	REM	VARPTR
CHR$	ERR	LOG	RENUM	VDP
CINT	ERROR	LPOS	RESTORE	VPEEK
CIRCLE	EXP	LPRINT	RESUME	VPOKE
CLEAR	FIELD	LSET	RETURN	WAIT
CLOAD	FILES	*MAX	RIGHT$	WIDTH
CLOSE	FIX	MERGE	RND	XOR
CLS	FN	MID$	RSET	
*CMD	FOR	MKD$	RUN	
COLOR	*FPOS	MKI$	SAVE	
CONT	FRE	MKS$	SCREEN	
COPY	GET	MOD	SET	
COS	GO TO	MOTOR	SGN	
CSAVE	GOSUB	NAME	SIN	
CSNG	GOTO	NEW	SOUND	
CSRLIN	HEX$	NEXT	SPACE$	
CVD	IF	NOT	SPC(	
CVI	IMP	OCT$	SPRITE	
CVS	INKEY$	OFF	SQR	
DATA	INP	ON	STEP	
DEF	INPUT	OPEN	STICK	
DEFDBL	INSTR	OR	STOP	
DEFINT	INT	OUT	STR$	
DEFSNG	*IPL	PAD	STRIG	

PANORAMA SULLE ISTRUZIONI DELL'MSX-BASIC

In questo capitolo vengono trattate brevemente tutte le istruzioni, i comandi, le variabili di sistema e le funzioni BASIC.
Di ogni istruzione e di ogni comando viene data la sintassi completa (cioè viene indicato completamente come devono essere scritti). Inoltre vengono usati due simboli con il seguente significato:

[...] tutto ciò che si trova fra le parentesi quadre è facoltativo, cioè può essere eventualmente omesso;

<...> i dati che si trovano tra queste parentesi uncinate devono essere riempiti dal programmatore stesso.

In un certo numero di espressioni possiamo trovare indicati 3 punti, come per esempio:

CALL<nome>[<stringa>,<stringa>...]

I tre punti alla fine di questa espressione vògliono dire che 'l'espressione può essere continuata nello stesso modo'. In questo caso perciò possono essere introdotte più stringhe, separate da una virgola.

Questo capitolo offre allo stesso tempo la sintassi completa tutte le funzioni BASIC e variabili di sistema.
Inoltre vengono usate due notazioni col seguente significato:

<X> Una variabile numerica o un'espressione a cui dobbiamo assegnare un valore.

<X$> Una variabile stringa o un'espressione a cui dobbiamo assegnare un valore.

Ad ogni comando è menzionata la categoria alle quale esso appartiene: istruzione, comando vero e proprio, variabile di sistema o funzione. Dopo l'esecuzione di un comando il MSX-BASIC stamperà l'indicazione OK e il sistema aspetterà finchè battiate il comando seguente.
Una funzione o variabili di sistema può esser usata solo in combinazione con un'istruzione o con un comando.

ABS(<X>)
Indica il valore assoluto di <X>.
Categoria: funzione
Esempio: `PRINT ABS(-3)`

ASC(<X$>)
Indica il codice ASCII del primo carattere grafico di <X$>.
Categoria: funzione
Esempio: `PRINT ASC("MSX")`

ATN(<X>)
Indica l'arcotangente di <X> in radianti.
Categoria: funzione
Esempio: `PRINT ATN(3)`

AUTO [<numeri di riga>] [,[<intervallo di incremento>]]
Genera automaticamente i numeri di riga durante l'inserimento di un programma. Se non viene specificato alcun <numero di riga>, inizia con il numero di riga 10, altrimenti inizia secondo il <numero di riga> specificato. Se non viene indicato alcun <intervallo di incremento> il numero di riga viene accresciuto di 10, altrimenti viené accresciuto secondo l'<intervallo di incremento> specificato.
Se un numero di riga che era stato già adoperato viene generato automaticamente, appare un asterisco * come avvertimento.
Il comando viene interrotto con CTRL/C, cioè tenendo abbassato il tasto CTRL e contemporaneamente premendo il tasto C.
Categoria: comando
Esempi: `AUTO 100,50`

BASE(<X>)
Comprende il primo indirizzo del cosiddetto Video Display Processor (la tabelle VDP). La tabella seguente offre un quadro delle abbreviazioni usate.
ma = modo alfanumerico
mg = modo grafico
tm = tabella modelli
tms = tabella modelli sprite
tas = tabella attributi sprite
tc = tabella colori
tn = tabella nomi

<X>	*Significato*			*<X>*	*Significato*		
0	tn	in	ma1	11	tc	in	mg1
2	tm	in	ma1	12	tm	in	mg1
5	tn	in	ma2	13	tas	in	mg1
6	tc	in	ma2	14	tms	in	mg1
7	tm	in	ma2	15	tn	in	mg2
8	tas	in	ma2	16	tm	in	mg2
9	tms	in	ma2	17	tas	in	mg2
10	tn	in	mg1	19	tms	in	mg2

Categoria: variabile di sistema

```
Esempio: 10 SCREEN 0
         20 PRINT BASE(2):END
```

BEEP
Genera un suono tipo 'bip' di breve durata.
Lo stesso effeto può essere raggiunto con PRINT CHR$(7).
Categoria: istruzione
```
Esempio: 10 FOR K=1 TO 1000:PRINT K:NEXT
         20 BEEP:END
```

BIN$(<X>)
Trasforma il numero dato in una notazione binaria.
<X> varia tra –32768 e 65535. Se <X> è negativo, si ottiene la cosiddetta forma 'two's complement'
Categoria: funzione
```
Esempio: PRINT BIN$(100)
```

BLOAD "<dev>:[<nome file>]"[,R] [,<spostamento>]
Viene caricato un programma scritto in codice-macchina, che era stato memorizzato con un comando BSAVE.
Se si inserisce CAS al posto di dev (in inglese: device, cioè si intende il dispositivo di registrazione) si indica un registratore a cassette.
A <dev> si possono assegnare anche le lettere da A a F. Queste lettere corrispondono ai diskdrives da 1 a 6.
Il termine <nome archivio> si riferisce al nome con cui è stato indicato (massimo di 6 caratteri per il registratore a cassette e 8 per il diskdrive. Se la lettera R viene indicata, ciò vuol dire che, dopo aver caricato il programma, questo inizia immediatamente a 'girare'.
Se invece viene indicato uno spostamento (un numero intero), il nuovo indirizzo di partenza sarà uguale al vecchio + lo spostamento.
Categoria: comando
```
Esempio: BLOAD "CAS:TEST",R,&H20
```

BLOAD"<dev>:[<nome file>]",S
Come il comando precedente solo che in questo caso si tratta della memoria RAM del video; <dev> qui può essere solo un dischetto.
<dev> può essere soltanto una lettere da A a F.
Categoria: comando
```
Esempio: BLOAD "A:TST",S
```

BSAVE"<dev>:[<nome file>]",<indirizzo iniziale,indirizzo finale>[,<indirizzo iniziale per l'esecuzione del programma>]
In questo caso un programma scritto in codice macchina viene memorizzato su un dispositivo esterno. I termini <dev> e <nome archivio> sono stati spiegati quando abbiamo parlato del comando BLOAD. L'indirizzo iniziale e l'indirizzo finale indicano le aree di memoria che vengono trasferite su di un dispositivo esterno.
Categoria: comando
```
Esempio: BSAVE "CAS:TEST",&HC000,&HE0FF,&HC020
```

BSAVE"<dev>:[<nome file>]",<indirizzo iniziale>,<indirizzo finale>,S
Come il comando precedente solo che in questo caso si tratta della memoria RAM del video; <dev> qui può essere solo un dischetto.

Categoria: comando
Esempio: `BSAVE "A:TST",&H0,&HFFFF,S`

CALL<nome>[(<stringa>,<stringa>...)]
Comando generale per eseguire delle determinate subroutine, memorizzate su cartuccia ROM.
Le stringhe specificate sono delle costanti alfanumeriche, con cui i valori possono essere passati alla subroutine. Al posto del termine CALL, si può anche usare il segno di 'sottolineatura' (_).
Categoria: istruzione
Esempio: `_TALK`

CALL COM([<X>:],GOSUB<Y>)
Questa istruzione può essere usata se l'interfaccia RS232C è presente. <X> indica il numero dell'interfaccia. Il valore di default è 0. <Y> indica il numero di riga. Con questa istruzione si indica che, se viene specificata l'interfaccia RS232C, bisogna saltare alla subroutine che inizia con il numero di riga <Y>. Questa deve essere già stata attivata in precedenza con l'istruzione CALL COMON.
Categoria: istruzione
Esempio:
```
10 OPEN "COM":FOR INPUT AS#1
20 CALL COMON("")
30 CALL COM(,GOSUB 60)
40 PRINT "PROGRAMMA PER RICEVERE I DATI"
50 GOTO 50
60 PRINT "RICEVO ORA";
70 A$=INPUT$(1,#1)
80 PRINT A$
90 RETURN
100 END
```

CALL COMBREAK(["<n>:"],<codice di specificazione>)
Con questo comando possiamo trasmettere i 'break-characters'. via interfaccia RS232C.
Il <codice di notazione>indicato viene usato come segnale per interrompere la comunicazione (un numero tra 3 e 32767). Questo comando è attivo solo se l'RS232C è già stato aperto. Con <n> viene indicata l'interfaccia RS232C in questione. Il valore di default è 0.
Categoria: istruzione
Esempio:
```
10 OPEN "COM:" FOR OUTPUT AS#1
20 CALL COMBREAK(,3)
30 CLOSE
40 END
```

CALL COMDTR (["<n>:"],<0 o non-0>)
Con questo comando si indica che la nostra interfaccia RS232 viene interrotta temporaneamente, perlomeno se l'espressione <,0 o non-0> è uguale a 'non -0'. Questo comando vale solo se l'RS232C è già stato aperto.
Con <n> viene indicata l'interfaccia RS232C in questione. Il valore di default è 0.
Categoria: istruzione
Esempio:
```
10 OPEN "COM:" FOR OUTPUT AS#1
20 CALL COMDTR(,0)
30 CLOSE
40 END
```

CALL COMINI [([<esponente stringa>][,[<Rx velocità in baud>][,<Tx velocità in baud>][,[<limite di tempo>]]])]
Istruzione per inizializzare l'interfaccia RS232C.
Significato dei campi:

<esponente stringa>: vedere sotto

<Rx velocità in baud>: velocità in baud (bit/sec) con cui l'interfaccia RS232C riceve i caratteri.
Default: 1200 baud

<Tx velocità in baud>: velocità in baud (bit/sec) con cui l'interfaccia trasmette i caratteri.
Default: 1200 baud

<limite di tempo>: tempo (in sec) che il computer aspetterà per la conferma di connessione prima di far apparire un messaggio d'errore. Limite di tempo =0 significa che il computer può aspettare per sempre, non ha limiti.
Default: 0

<stringa exp.>: ha questa forma:

"[<knl>:][<bl>[<pa>[<sl>[<x>[<hs>[<ilf>[<slf>[<ss>]]]]]]]]"
Significato:

<knl> numero del canale: numero dell'interfaccia che si sta adoperando. Necessario soltanto quando si possiedono più di una interfaccia.
Default: 0

<bl>: lunghezza bytes: lunghezza (in bit) di una unità byte trasmessa in una volta. Si possono assegnare i valori "5", "7" o "8".
Default: "8"

<pa>: parità
"E" = parità pari (Even)
"0" = parità dispari (Odd)
"I" = ignorare la parità (Ignore)
"N" = nesuna parità (No)
Default: "E"

<sl>: stop di lunghezza: lunghezza (in bit) della pausa tra la trasmissione di due byte
"1" = 1 bit
"2" = 1.5 bit
"3" = 2 bit

Default: "1"

<x>: controllo XON/XOFF
"X" = controllare
"N" = non controllare
Default:"X"

<hs>:conferma di connessione CTS-RTS
"H" = conferma
"N" = nessuna conferma
Default: "H"

<ilf>: inserire un line feed (un cambio di riga) ogni volta che viene ricevuto un carriage return (ritorno di carrello).

"A" = inserito
"N" = non inserire

Default: "N"

<slf>: manda un linefeed dopo il carriage return

"A" = manda il linefeed
"N" = non manda il linefeed

Default: "N"

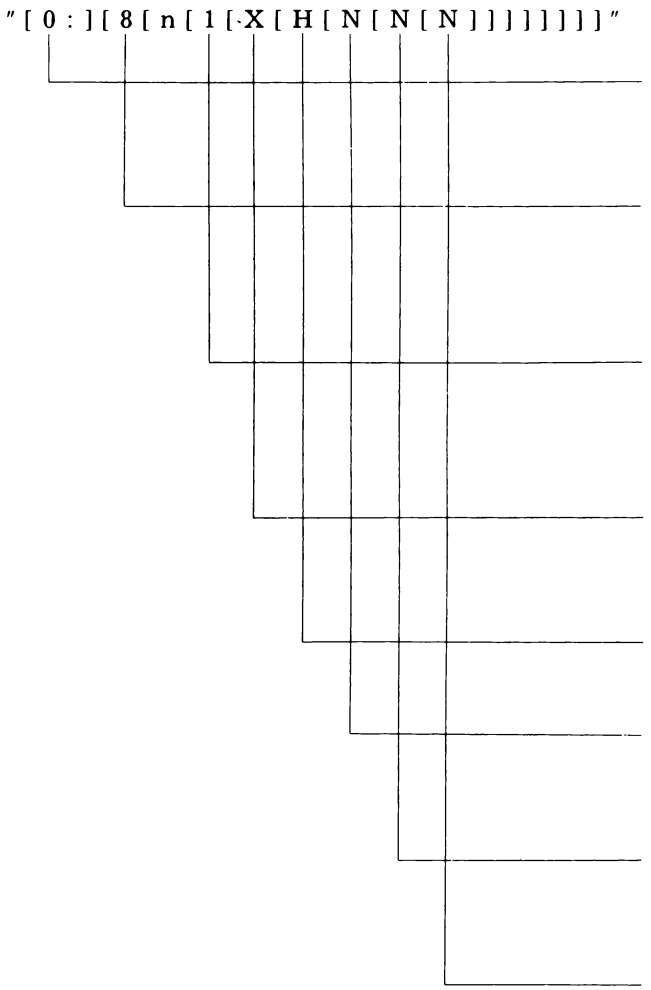
<ss>: controllo shift in/shift out

"S" = controllo
"N" = no controllo

Default: "N"

Categoria: istruzione

Esempio: `CALL COMINI("0:7N2XHAAN",75,1200)`



"numero di canale. Se c'è un solo *canale questa indicazione può essere ommessa. Il valore di "default" è 0.*

Lunghezza dati
"5" – 5 bit
"6" – 6 bit
"7" – 7 bit
"8" – 8 bit

Controllo di parità
"E" – Even (pari)
"O" – Odd (dispari)
"I" – Ignorata
"N" – nessuna

Stop bits (lunghezza)
"1" – 1 bit
"2" – 1.5 bit
"3" – 2 bit

Conferma di connessione CTS-RTS
"H" – conferma
"N" – nessuna conferma

Inserire LF (linefeed) se viene ricevuto CR
"A" – inserire
"N" – non inserire

Inserire LF (linefeed) se viene trasmesso CR
"A" – inserire
"N" – non inserire

Shift in/Shift out control
(non permesso per)
lunghezza dati = 7 bit
"S" – stabilire il controllo
"N" – non stabilire il controllo

CALL COMON("[<n>:]")
CALL COMOFF("[<n>:]")
CALL COMSTOP("[<n>:]")
Con <n> viene indicata l'interfaccia RS232C in questione.
Facendo uso di un certo numero di comandi CALL potete far eseguire ad un programma BASIC una subroutine non appena viene ricevuto un segno dall'interfaccia RS232C. Questi comandi eseguono una funzione analoga a quella di ON <evento> GOSUB <numero di riga>.
Categoria: istruzione
Esempio: `si veda CALL COM`

CALL COMSTAT (["<n>:"],<nome variabile>)
Con <n> viene indicata l'interfaccia RS232C in questione.
Se si verifica un Device I/O error durante la comunicazione via interfaccia RS232C, alla variabile viene assegnato un valore dal quale risulta eventualmente che cosa è stato sbagliato.
La tabella seguente ci mostra tutte le possibilità. In un certo numero di casi sono stati adottati i termini inglesi standard.

Bit n°	*Significato*	
15	Buffer overflow error	0 - nessun buffer overflow 1 - buffer overflow
14	Time out error (TMENBT)	0 - no 1 - sì
13	Framing error	0 - no 1 - sì
12	Over run error	0 - no 1 - sì
11	Parity error	0 - no 1 - ŝi
10	Il tasto 'Control break' è stato premuto	0 - no 1 - sì
9	non viene usato	
8	non viene usato	
7	Clear to send (pronto a trasmettere)	0 - no 1 - sì
6	Timer/counter output-2	0 - timer/counter viene tolto 1 - timer/counter viene messo in funzione
5	non viene usato	
4	non viene usato	
3	Data Set Ready	0 - no 1 - sì

Bit n°	*Significato*	
2	break detect	0 - no 1 - sì
1	Ring Indicator	0 - no 1 - sì
0	Carrier Detect	0 - no 1 - sì

Categoria: istruzione
Esempio:
```
10 OPEN "COM:"FOR INPUT AS#1
20 CALL COMSTAT(,A%)
30 A$="0000000000000000"+BIN$(A%)
40 PRINT RIGHT$(A$,16)
50 CLOSE
60 END
```

CALL COMTERM [("<n>:")]
Avvia il cosiddetto 'modo di simulazione del terminale' dell'interfaccia RS232C. Con <n> si indica l'interfaccia in questione; il valore di default è 0.
Il canale RS232C deve essere chiuso con CLOSE prima che CALL COMTERM possa venire eseguito.

I tasti F6, F7 e F8 hanno dei significati speciali:
F6 attiva e disattiva il cosiddetto 'modo autodefinito'. In questo stato i codici dei caratteri da 0 a 31 vengono stampati con il simbolo ∧ e seguiti dalla lettera che si ottiene sommando il codice 64.
F7 modo duplice medio/intero. Nel 'modo duplice medio' i caratteri battuti appaiono sullo schermo così come sono trasmessi via il canale RS232C.
F8 attiva e disattiva il cosiddetto eco di stampante.
Categoria: comando
Esempio: `CALL COMTERM`

CALL FORMAT
Per formattare un dischetto non ancora usato.
Attenzione: un dischetto già usato viene completamente cancellato con il comando FORMAT!
Categoria: comando
Esempio: `CALL FORMAT`

CALL SYSTEM
Per abbandonare il comando a livello dell'MSX-BASIC e passare il sistema sotto il controllo dell'MSX-DOS Operating System. Ma l'MSX-DOS deve essere già stato avviato.
Categoria: comando
Esempio: `CALL SYSTEM`

CDBL(<X>)
Trasforma <X> in un numero in doppia precisione.
Categoria: funzione
Esempio: `PRINT CDBL(7/6)`

CHR$(<codice ASCII>)
Indica il carattere grafico che appartiene al <codice ASCII>.
Categoria: funzione
Esempio: `PRINT CHR$(65)`

CINT(<X>)
Arrotonda il valore di <X> a numero intero. Il valore deve variare tra –32768 e 32767.
Categoria: funzione
Esempio: `PRINT CINT(7/6)`

CIRCLE [STEP] (<x,y>),<r>[,[<colore>][,[<angolo iniziale>]
[,[<angolo finale>][,<schiacciamento>]]]]
Genera una ellisse. Con <x,y> viene dato il punto centrale, con <r> si indica il raggio. Eventualmente si può anche disegnare solo una parte indicando un angolo iniziale e un angolo finale (dati in radianti). Se si omette STEP vengono introdotte le coordinate normali. Usando STEP il sistema delle coordinate si sposta verso il punto in cui si trova il cursore. <schiacciamento> è un numero che indica il rapporto tra il raggio orizzontale e quello verticale.
Categoria: istruzione
Esempio:
```
10 SCREEN 2
20 CIRCLE (127,95),50,,,1.4
30 GOTO 30
```

CLEAR [<spazio di memoria stringhe>[,<massimo indirizzo>]]
Tutte le variabili vengono rese uguali a 0 e inoltre viene riservata un'area di memoria per la memorizzazione dei caratteri (stringhe, lettere).
Il 'massimo indirizzo' determina il più grande indirizzo che possa essere usato con il BASIC. Infine, con CLEAR, tutti gli archivi, eventualmente ancora aperti, vengono chiusi. Pensate che senza CLEAR, il computer potrebbe riservare solo 200 posizioni per la memorizzazione di lettere, ecc.
Notiamo tra l'altro che vengono cancellate nello stesso tempo tutte le funzioni precedentemente programmate che erano state specificate grazie a DEF FN così come le variabili standard definite da DEF IN ecc.
Vengono perse anche tutte le tabelle che erano state definite con una istruzione DIM.
Categoria: istruzione
Esempio:
```
10 A=10:B$="TEST"
20 PRINT A,B$
30 CLEAR
40 PRINT A,B$:END
```

CLOAD ["<nome file>"]
Si tratta di un comando per trasferire un programma, a cui si è dato eventualmente un nome (massimo 6 lettere o cifre), da una cassetta. Così, per esempio, il vostro programma si chiamerà 'PROG4' se è stato caricato dalla casetta con il nome 'PROG4'. Se qualcosa non dovesse funzionare bene si può interrompere questa azione premendo CTRL/STOP, riavvolgendo poi il nastro e infine iniziando nuovamente a caricare il programma.
Se viene omesso il nome dell'archivio viene caricato il primo programma che si trovi su cassetta.
Categoria: comando
Esempio: `CLOAD"TEST"`

CLOAD? ["<nome file>"]
E' un comando che rende possibile confrontare il programma che è stato caricato con il programma presente in memoria. Se tutto va bene, appare 'Ok'. Se invece viene riscontrato un errore, appare 'Verify error'.
Categoria: comando
Esempio:
```
CLOAD?"TEST"
```

CLOSE [#][<numero file>][,[#]<numero file>...]
Chiude gli archivi indicati dai numeri. Se non è stato specificato alcun numero, chiude tutti gli archivi.
Categoria: istruzione
Esempio:
```
10 MAXFILES=1
20 OPEN "CAS:TEST" FOR OUTPUT AS#1
30 A$="MSX"
40 PRINT#1,A$
50 CLOSE#1
60 END
```

CLS
Pulisce lo schermo.
Categoria: istruzione
Esempio:
```
10 CLS
20 END
```

COLOR[<primo piano>][,<sfondo>][,<territorio pereferico>]
Con questa istruzione si può determinare il colore delle aree indicate. La tabella seguente mostra i colori standard.

Numero del colore	*Colore*	*Numero del colore*	*Colore*
0	trasparente	8	rosso
1	nero	9	rosso chiaro
2	verde	10	giallo scuro
3	verde chiaro	11	giallo chiaro
4	blu scuro	12	verde scuro
5	azzurro	13	magenta
6	rosso scuro	14	grigio
7	celeste	15	bianco

CONT
Con questo comando viene ripresa l'esecuzione del programma dal punto in cui era stata interrotta con CRTL/STOP o con le instruzioni STOP o END.
Categoria: comando
Esempio:
```
CONT
```

COPY "[<dev>:]<nome del file>" TO "[<dev>:]<nome del file>"

Con questa istruzione si può copiare un file.
<dev> può memorizzare soltanto su un diskdrive (da A a F).
Categoria: istruzione
Esempio:
```
COPY "A:TEST.ASC" TO "B:"
```

COS (<X>)
Definisce il coseno di X.
Categoria: funzione
Esempio: `PRINT COS(3.1415/6)`

CSAVE"<nome file>"[<velocità in baud>]
E' un comando che serve per memorizzare un programma su cassetta (vedere anche CLOAD). La velocità in baud può essere:
1 1200 baud
2 2400 baud

Se questa indicazione viene omessa, il computer assume la velocità di 1200 baud.
La velocità in baud può anche essere introdotta con SCREEN.
Categoria: comando
Esempio: `CSAVE "TEST"`

CSNG(<X>)
Trasforma il valore di <X> in un numero in singola precisione.
Categoria: funzione
Esempio: `PRINT CSNG(9/7)`

CSRLIN
Indica la coordinata y della posizione in cui si trova il cursore.
Categoria: funzione
Esempio:
```
10 SCREEN 0:LOCATE 10,20:PRINT CSRLIN
20 END
```

CVI (<2byte stringa>)
CVS (<4byte stringa>)
CVD (<8byte stringa>)
Queste funzioni decodificano le stringhe specifiche in valori interi (CVI), in singola precisione (CVS) e in doppia precisione (CVD). Le stringhe devono essere codificate secondo le funzioni inverse (MKI$, MKS$ e MKD$) o lette da un file ad accesso casuale. Possiamo usare queste funzioni per ritrovare dei numeri che sono stati memorizzati in un file ad accesso casuale. In un comando FIELD possono essere indicate solo delle variabili stringa; con queste funzioni possiamo visualizzare dei numeri, da un record, che sono stati codificati sotto forma di stringa.
Categoria: funzione
Esempio: `PRINT CVD(D$);CVS(S$);CVI(I$)`

DATA <n1>[,<n2>...]
Con questa istruzione si possono raccogliere in un programma una lista di valori fissi che possono essere poi rivisti uno per uno grazie all'istruzione READ. La lista può essere composta sia da valori numerici che da stringhe (tra virgolette (")) e devono essere separati dalle virgole. READ ripresenta i valori in ordine sequenziale. Si può ricominciare la lettura dei dati elencati in DATA dall'inizio, eseguendo prima di tutto l'istruzione RESTORE.
Categoria: istruzione
Esempio:
```
10 READ A,B,C:PRINT A,B,C
20 DATA 5,6,7
30 END
```

DEF FN <nome della funzione>[(<lista argomenti>)]=<definizione di funzione>
Con questa funzione possiamo definire le funzioni da noi stessi. Il nome con il quale la funzione deve essere richiamata ha il <nome di funzione> preceduto da FN. Con le funzioni stringa il nome termina con il simbolo $. Eventualmente si possono assegnare alla funzione certi argomenti che vengono usati nella <definizione di funzione>. I nomi degli argomenti servono solo per definire la funzione e non influiscono assolutamente sulle eventuali variabili con lo stesso nome che si trovano nel programma principale. Gli argomenti devono essere separati tra loro da una virgola.
Nella <definizione di funzione> possono comparire anche dei nomi di variabili che non sono stati indicati come argomenti nell'<elenco argomenti>, ma a cui è stato già assegnato un valore nel programma principale per richiamare la funzione. La <definizione di funzione> può avere al massimo la lunghezza di una riga. Esempio:
Categoria: istruzione
Esempio:
```
10 DEF FNAB(X,Y)=X^2+Y*Z
20 Z=5
30 I=2:J=3
40 T=FNAB(I,J)
50 PRINT T
60 END
```

DEF <tipo><indicazione lettera>
Questa istruzione controlla che tutte le variabili che iniziano con le <lettere> date, appartengano al <tipo> specificato. I <tipi> possibili sono:
INT intero
SGN singola precisione
DBL doppia precisione
STR stringa

I caratteri di dichiarazioni di tipo (perciò %, #, $) hanno la precedenza rispetto all'istruzione DEF.
Esempi:
```
10 DEFDBL A-E
```
Tutte le variabili che inizino con A, B, C, D o E sono variabili in doppia precisione.
```
10 DEFSTR A
```
Tutte le variabili che inizino con la lettere A sono variabili stringa.

Categoria: istruzione
Esempio:
```
10 DEFINT I
20 I=3/2:PRINT I
30 END
```

DEF USR [<numero>]=<indirizzo di memoria>
Con questa istruzione si specifica l'indirizzo di partenza di una subroutine in linguaggio machina. Il <numero> deve essere tra 0 e 9 e viene assegnato come nome alla subroutine in linguaggio macchina. Se si omette il <numero> il computer assumerà il valore 0. In base a questo <numero> e grazie alla funzione USR si possono richiamare le subroutine in linguaggio macchina.
Categoria: istruzione
Esempio:
```
10 DEF USR0=1000
20 X=USR0(9*2)
30 END
```

DELETE [<numero di riga>][-<numero di riga>]
Cancella tutte le righe specificate. Dopo l'esecuzione di questo comando si ritorna sempre in stato comandi Basic.
Categoria: comando
Esempi: `DELETE 10`

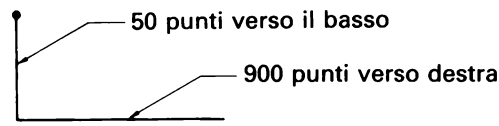
DIM <nome vettore>(<indice massimo>)[,<nome vettore>...]
Crea degli spazi in memoria per specifici vettori e fa iniziare gli elementi del vettore da 0. Se ci si riferisce ad un vettore che non sia stato indicato in precedenza da un'istruzione DIM, gli viene assegnato 10 come indice massimo. L'indice minimo è sempre 0. Con ERASE possiamo cancellare un vettore.
Categoria: istruzione
Esempio:

```
10 DIM A(20)
20 FOR K=1 TO 20:A(K)=K:NEXT K
30 FOR K=1 TO 20:PRINT A(K):NEXT K
40 END
```

DRAW <stringa>
Tramite DRAW si può rappresentare qualsiasi tipo di linea retta con dei codici molto semplici. I codici formano sempre una stringa.

Esempio:

```
10 SCREEN 2
20 DRAW "D50R100"
30 GOTO 30
```



Si ottengono due linee rette che si congiungono. La prima si dirige di 50 punti verso il basso ('down 50' o D50) e la seconda si dirige di 100 punti verso destra (R100). Il risultato è il seguente:

Ad ogni linea possiamo assegnare un colore tramite la lettera C e il codice del colore, per esempio:

```
DRAW "C8D50C10R100"
```

La tabella seguente ci mostra le varie possibilità:

Codice *Significato*

S Indica la scala. Il codice S viene seguito da un numero. La scala corrisponde quindi al numero /4.

A Dopo A vengono i valori 0, 1, 2 o 3. Con questo codice il sistema coordinate viene girato di 90° ogni volta. Se A non viene specificato, il computer assume A0.

C Indica il colore. Vedere l'esempio dato sopra.

M Per disegnare una linea obliqua. Dopo M vengono due numeri separati da una virgola, per esempio:

M30,50

In questo caso verrá tracciata una linea che parte dall'ultima posizione e arriva al punto che si ottiene sommando la coordinata x=30 e la coordinata y=50. I valori di x e di y possono essere anche negativi.

U Significa 'up' cioè 'verso l'alto' per esempio:

U=30.

indica che una linea deve essere tracciata verso l'alto per 30 punti a partire dall'ultima posizione.

D Significa 'down'. Si usa per tracciare una linea verso il basso.

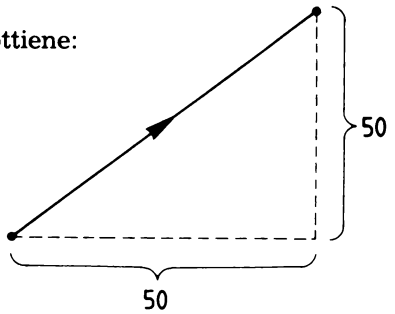
R Significa 'right'. Si usa per tracciare una linea verso destra.

L Significa 'left'. Si usa per tracciare una linea verso sinistra.

E Si usa per tracciare una linea verso destra in alto con un angolo di 45°.
Esempio:

```
E50
```

come risultato si ottiene:



F Si usa per tracciare una linea verso destra in basso con un angolo di 45° (vedere anche E).

G Si usa per tracciare una linea verso sinistra in alto con un angolo di 45° (vedere anche E).

H Si usa per tracciare una linea verso sinistra in basso con un angolo di 45° (vedere anche E).

Notate che le linee vengono sempre tracciate a partire dall'ultima posizione raggiunta. Possiamo ottenere una posizione fissa iniziale con BMx,y, dove x e y indicano le coordinate iniziali.
Esempio: `DRAW "BM50,50D30"`
qui viene tracciata una linea che dal punto (50,50) scende verso il basso di 30 punti.
Si può anche mantenere l'ultimo punto raggiunto come punto di partenza per una nuova linea, ciò è possibile collocando N prima del codice. Si può indicare la stringa anche tramite una variabile stringa.
Esempio:

```
....
50 B$="BM50,50D30"
60 DRAW B$
```

In un'istruzione DRAW si può riprodurre anche una parte della stringa, tramite una variabile. Collochiamo allora una X nell'istruzione DRAW.
Esempio:

```
....
50 B$="BM50,50D30"
60 DRAW "R100XB$"
```

Infine si possono indicare anche i numeri che appaiono nella stringa DRAW tramite una variabile. La variabile appare tra i simboli = e;
Esempio:

```
....
50 K=30
60 DRAW "B10,10R=K;"
```

Categoria: istruzione
Esempio:
```
10 SCREEN 2
20 DRAW "BM80,80H20"
30 GOTO 30
```

DSKF (<numero del drive>)
Questa funzione indica quanto spazio è avanzato sul dischetto. I drive del dischetto vengono numerati in questa maniera:

0= default drive
1=drive A
2=drive B
3=drive C
4=drive D
5=drive E
6=drive F

Categoria: funzione
Esempio:
```
10 PRINT DSKF(1)
20 END
```

DSKI$(<numero del drive>,<numero del settore>)
Questa funzione carica, se chiamata, il settore specifico nella parte di memoria che viene assegnata dalle locazioni di memoria &HOF351 e &HOF352.

I numeri da 0 a 6 indicano i diskdrives da A a F

Categoria: funzione
Esempio:
```
10 SCREEN 0:WIDTH 38:COLOR 15,4,4:DEFINT I
20 PRINT "DISKCOPY SECTOR TO SECTOR"
30 PRINT:PRINT
40 PRINT "inserire il dischetto di origine nel drive A:"
50 PRINT
60 PRINT "inserire il dischetto vuoto nel drive B:"
70 PRINT
80 PRINT "batti un tasto"
90 A$=INPUT$(1)
100 PRINT:PRINT
110 PRINT "TRACK:"
120 PRINT
130 PRINT "SECTOR:"
140 FOR I=1 TO 719
150 A$=DSKI$(1,I)
160 LOCATE 8,10:PRINT INT(I/9)
170 LOCATE 8,12:PRINT IMOD9
180 DSKO$ 2,I
190 NEXT I
200 END
```

DSKO$<numero del drive>,<numero del settore logico>
Scrive il contenuto della memoria (indicato tramite il contenuto degli indirizzi di memoria &HOF351 e &HOF352) nel settore specificato.
I numeri da 0 a 6 indicano i diskdrives da A a F.

Categoria: istruzione
Esempio: si veda DSKI$

END
Finisce un programma. Questa istruzione può apparire dappertutto nel programma. Il termine END alla fine del programma è facoltativo, cioè non è obbligatorio. Tutti i files ancora aperti vengono chiusi.
Categoria: istruzione
Esempio:
```
10 INPUT A
20 IF A<5 THEN END
30 END
```

EOF(<numero di file>)
Accerta se è stata raggiunta la fine di un archivio (file) durante la lettura dei dati.
Categoria: funzione
Esempio:
```
IF EOF(2) THEN CLOSE #2
```

ERASE <nome del vettore>[,<nome del vettore>...]
Con ERASE possiamo cancellare un vettore; in questo modo si libera nuovamente uno spazio in memoria (vedere anche DIM).
Categoria: istruzione
Esempio:
```
10 DIM K(100),X$(50)
...
500 ERASE K,X$
```

ERR e ERL
Se in un programma appare un errore, la variabile ERR contiene il numero dell'errore apparso (vedere appendice messaggi d'errore) e ERL contiene il numero della riga in cui l'errore è apparso. ERR e ERL possono essere usati nella parte del programma che riguarda i messaggi d'errore (vedere anche ON ERROR GOTO) e soprattutto con costruzioni IF...THEN...ecc.
Per esempio: IF ERR=11 AND ERL=160 THEN...ecc
Categoria: funzione
Esempio:
```
10 ON ERROR GOTO 50
20 A=25:PRINT A
30 B=A/0
40 END
50 PRINT ERL
60 RESUME NEXT
```

ERROR <numero di errore>
Battete sullo schermo il codice di errore appartenente al <numero di errore> specificato. Il <numero di errore> deve essere più grande di 0 e più piccolo di 255. Esempio: battete ERROR 11. Sullo schermo apparirà /0 ERROR.
Non vengono usati dal BASIC tutti i numeri di errore tra 0 e 255. I numeri che non vengono adoperati, possono esere usati per formulare i propri messaggi di errore.
L'esempio che segue mostra che non sempre viene stampato il messaggio di errore standard, ma che possiamo anche far stampare un messaggio scritto da noi.
Categoria: funzione

Esempio:
```
10 ON ERROR GOTO 400
20 INPUT "X:-",A
30 IF A>100 THEN ERROR 210



400 IF ERR=210 THEN PRINT "MASSIMO 100!"
410 RESUME 20
```

EXP(<X>)
Calcola la potenza 'e' elevata alla <X>.
Categoria: funzione
Esempio: `PRINT EXP(1)`

FIELD[#]<numero di file>,<ampiezza del campo> AS <variabile stringa>
Con questo comando possiamo avere accesso al buffer (memoria di transito) di un file ad accesso casuale.
In un programma BASIC possiamo ora leggere e stampare il buffer usando le variabili stringa che sono nominate nella parte AS.
Categoria: istruzione
Esempio: `FIELD #1,20 AS A$,10 AS B$`
In questo comando le prime 20 posizioni del buffer del file 1 vengono assegnate a A$, e le successive 10 posizioni a B$. Se ora eseguiamo un comando GET possiamo chiedere il record letto in A$ e B$. Possiamo scrivere un nuovo record assegnando nei comandi LSET e RSET dei nuovi valori a A$ e B$, e successivamente eseguendo il comando PUT. FIELD non può leggere o scrivere da sè sul dischetto.
La somma delle entità dei campi in un comando FIELD non può essere più grande della lunghezza del record del file (indicato nel comando OPEN).

FILES ["<nome file>"]
Dà in visione i files che si trovano sul dischetto.
Categoria: comando
Esempi: `FILES`

FIX(<X>)
Indica la parte intera di <X>.
Categoria: funzione
Esempio: `PRINT FIX(1.7)`

FOR...NEXT
La sintassi completa è:

FOR <variabile>=<n> TO <m>[STEP<k>]
istruzione 1
istruzione 2
...
...
NEXT [<variabile>][,<variabile>...]

dove n, m e k sono espressioni numeriche.
Tutte le istruzioni tra FOR e NEXT vengono eseguite ripetute. E ciò tante volte finché la

(=<variabile>) contatore non superi il valore di m. Il valore iniziale della variabile contatore è n e, ogni volta che tutte le istruzioni sono state eseguite, viene aumentata di k. Se k non viene specificato la variabile contatore viene incrementata sempre di 1.
Le istruzioni FOR...NEXT possono contenere a loro volta altre istruzioni FOR...NEXT.
Categoria: istruzione
Esempio:
```
10 FOR K=1 TO 10:PRINT K:NEXT K
20 END
```

FRE(0) o FRE(<" ">)
Se l'argomento è 0, FRE indica la grandezza in byte dello spazio di memoria non ancora usato. Se l'argomento è una variabile stringa, FRE indica il numero di byte liberi in memoria, che sono riservati alle variabili stringa. Questo ultimo spazio di memoria può essere adattato da CLEAR.
Categoria: funzione
Esempio:
```
PRINT FRE(0)
```

GET[#]<numero di file>[,<numero di record>]
Con GET possiamo leggere un record dal buffer di un file ad accesso casuale. Il contenuto del record si può usare successivamente in un programma, se è stato dato in precedenza un comando FIELD, in quanto vengono usate quelle variabili stringa che erano state indicate in quel comando FIELD.
In un file ad accesso causale ogni record ha un numero.
Dopo aver letto un record, il suo numero viene memorizzato all'interno e con il successivo comando GET viene automaticamente letto il record successivo. Se però nel comando GET viene indicato un numero di record, si rimanda alla lettura di quel record.
Il numero dell'ultimo record si può trovare con la funzione LOC (vedere LOC).
Categoria: istruzione
Esempio:
```
10 OPEN "VBD.DAT"AS#1
20 FIELD #1,2 AS A$,10 AS B$
30 FOR K%=1 TO 10
40 GET #1,K%
50 PRINT CVI(A$);B$
60 NEXT
70 CLOSE #1
80 END
```

GOSUB <numero di riga> RETURN [<numero di riga>]
Con questa istruzione si può saltare ad una subroutine. Il <numero di riga> corrisponde al numero della prima riga della subroutine. In una subroutine deve apparire da qualche parte un RETURN che permette così di ritornare alla prima istruzione che si trova immediatamente dopo l'istruzione GOSUB. Le subroutine possono a loro volta rimandare ad altre subroutine.
Categoria: istruzione
Esempio:
```
10 GOSUB 40
20 GOSUB 40
30 END
40 PRINT "SUBROUTINE"
50 RETURN
```

GOTO <numero di riga>
L'esecuzione del programma viene continuata dal <numero di riga> indicato.
Categoria: istruzione
Esempio:
```
10 GOTO 30
20 PRINT "A"
30 PRINT "13"
40 END
```

HEX$(<X>)
Indica una stringa che rappresenta il valore esadecimale di <X>. <X> viene prima arrotondato a numero intero. <X> varia tra – 32768 e 65535.
Categoria: funzione
Esempio: `PRINT HEX$(63)`

IF...THEN
La sintassi completa si presenta così:

IF <espressione booleana>[<operatore booleano><espressione booleana>...]THEN <numero di riga dell'istruzione(i) >[ELSE <numero di riga dell'istruzione(i)>]

N.B. Con 'espressione booleana'si intende una 'condizione logica' che può essere soddisfatta o no. Nella 'espressione booleana' possono apparire i seguenti segni di relazione (operatori):

<	: minore di	<=	: minore o uguale a
>	: maggiore di	>=	: maggiore o uguale a
=	: uguale a	<>	: diverso da

Se si collocano più istruzioni dopo THEN o ELSE, queste devono venir separate dal segno del due punti (:). L'espressione completa deve occupare solo una riga di programma. Nelle espressioni possono essere usati questi termini OR, AND, NOT, XOR, EQV o IMP (vedere appendice F).
Se l'intera espressione tra IF e THEN è soddisfatta, vengono eseguite le istruzioni dopo THEN. Se dopo THEN non c'è alcuna istruzione ma un numero di riga, si procede allora alla riga indicata.
Se l'intera espressione tra IF e THEN non è soddisfatta, vengono eseguite le istruzioni dopo ELSE o si salta alla riga indicata. Se non c'è ELSE allora si procede alla riga che segue direttamente l'istruzione IF...THEN.
Le istruzioni dopo THEN e ELSE possono contenere ancora un'istruzione IF...THEN, purché tutta l'espressione si mantenga su una riga, perciò l'istruzione:

```
IF X>Y THEN PRINT "MAGGIORE" ELSE IF Y>X THEN PRINT "MINORE" ELSE PRINT "UGUALE"
```

è possibile. In un'istruzione del genere, ELSE viene riferito all'IF più vicino, a cui però non è stato trovato un ELSE che gli appartenga.
Categoria: istruzione
Esempio:
```
10 INPUT A
20 IF A<3 THEN PRINT "A<3"
30 END
```

INKEY$
Questa funzione viene usata con la formulazione <variabile stringa>=INKEY$. Questa funzione controlla se è stato premuto un tasto sulla tastiera. Se è così, viene assegnato alla <variabile stringa> il carattere grafico corrispondente. Se nessun tasto è stato premuto, la <variabile stringa> sarà formata da una stringa vuota (la cosidetta stringa nulla).
Categoria: funzione
Esempio:
```
10 PRINT "BATTI LA LETTERA H"
20 A$=INKEY$
30 IF A$< >"H" THEN 20
40 PRINT A$
50 END
```

INP (<X>)
Fornisce un byte che viene letto inserito attraverso la porta di input X.
Categoria: funzione
Esempio:
```
10 A=INP(&HA8)
20 A$="00000000"+BIN$(A):PRINT RIGHT$(A$,8)
```

INPUT ["<testo>";] <variabile 1> [,<variabile 2>...]
Inserisce in memoria i valori che devono essere battuti tramite tastiera dal programmatore. Sullo schermo appare sempre un punto interrogativo (?).
Se il <testo> viene specificato, questo appare sullo schermo, prima del punto interrogativo.
I valori inseriti vengono assegnati a delle <variabili>. Devono essere inseriti, separati da una virgola, così tanti valori quante sono le variabili che appaiono nell'istruzione INPUT. Possono essere inserite anche variabili stringa, ma non vi devono apparire delle virgole. Se si riscontra un errore nei valori inseriti appare il messaggio di errore ?Redo e l'istruzione INPUT viene nuovamente eseguita.
Categoria: istruzione
Esempio:
```
10 INPUT A:PRINT A:END
```

INPUT #<numero di file>,<variabile 1>[,<variabile 2>...]
Questa istruzione può essere paragonata all'istruzione INPUT solo che ora si tratta di valori di un file. Il <numero del file> indica il numero che è stato assegnato, tramite OPEN, al file in questione. I valori del file devono stare nello stesso ordine delle variabili nell'istruzione INPUT. Durante l'inserimento in memoria di una stringa, la fine della stringa viene definita da una virgola, da RETURN o da line feed. Se all'inizio della stringa si trova il segno delle virgolette ("), le seconde virgolette sono considerate come il segno di fine della stringa.
Categoria: istruzione
Esempio:
```
10 OPEN "CAS:DATA" FOR INPUT AS#1
20 IF EOF(1) THEN GOTO 40
30 INPUT#1,X$:PRINT X$:GOTO 20
40 CLOSE#1:END
```

INPUT$(<X>)/INPUT$(<X>,[#]<Y>)
Questa funzione viene usata con la formulazione <variabile stringa>=INPUT$(<X>). La funzione inserisce una stringa, tramite tastiera, dei caratteri grafici di <X>. Al posto della tastiera può anche essere indicato il file <Y>. Vengono accettati tutti i caratteri grafici tranne CTRL/C.

Categoria: funzione
Esempio: `10 A$=INPUT$(4):PRINT A$:END`

INSTR([I,]<X$>,<Y$>)
Controlla se <Y$> appare in <X$> e indica poi la posizione in <X$>. Se <Y$> non appare in <X$>, INSTR dà il valore 0. INSTR inizia a cercare dall'inizio di <X$>, a meno che con [I] non sia stata indicata un'altra posizione di partenza. Se <Y$> è una stringa nulla, INSTR indica il valore 1.
Categoria: funzione
Esempio: `10 A$="ABCDEFG":PRINT INSTR(A$,"BCD"):END`

INT(<X>)
Questa funzione indica il numero intero più grande che deve essere, comunque, o minore o uguale a <X>.
Categoria: funzione
Esempio: `PRINT INT(3.7)`

INTERVAL ON
INTERVAL OFF
INTERVAL STOP
Serve per indicare se un'interruzione, data da un orologio incorporato, viene inserita, interrotta o sospesa. L'intervallo deve essere indicato nell'istruzione ON INTERVAL GOSUB.
Dopo l'istruzione INTERVAL ON il computer, finito il tempo di intervallo, salterà alla subroutine che viene indicata nell'istruzione ON INTERVAL GOSUB.
Categoria: istruzione
Esempio:
```
10 ON INTERVAL=300 GOSUB 40
20 INTERVAL ON
30 GOTO 30
40 K=K+6:PRINT K;"SEC"
50 RETURN
```

KEY <n>,"<stringa comando>"
Per l'assegnazione di una <stringa comando> ad un tasto funzione. <n> è il numero del tasto funzione (da 1 a 10). La <stringa comando> deve trovarsi tra virgolette (") e puó essere composta al massimo di 15 caratteri. Un RETURN puó essere inserito anche nella <stringa comando> aggiungendo +CHR$(13).
Categoria: istruzione
Esempio: `KEY 1,"RUN"+CHR$(13)`
Se ora viene usato il tasto funzione 1 il programma viene direttamente eseguito, poiché nel comando RUN si trova giá un RETURN.

KEY LIST
Dà in visione le stringhe comando dei dieci tasti funzione
Categoria: istruzione
Esempio: `10 KEY LIST:END`

KEY ON
KEY OFF
Controllano se le prestazioni dei tasti funzione vengono riprodotte o no su schermo.

Categoria: istruzione
Esempio: `KEY OFF`

KEY(<n>)ON
KEY(<n>)OFF
KEY(<n>)STOP
Per controllare se uno dei tasti funzione è stato premuto. Premendo ON si inserisce questa condizione e con OFF si interrompe.
Usando STOP il computer non salterá subito alla subroutine indicata da ON KEY GOSUB, ma soltanto dopo che KEY(n) ON è stata data.
Categoria: istruzione:
Esempio:
```
10 KEY(1) ON
20 ON KEY GOSUB 50
30 GOTO 10
40 END
50 PRINT "KEY1":KEY(1)OFF:RETURN
```

KILL "<nome file>"
Per cancellare un file sul dischetto.
Categoria: istruzione
Esempio: `KILL "A:X1.DAT"`

LEFT$(<X$>,<I>)
Indica una stringa composta dai primi <I> caratteri grafici <X> di <X$>. <I> varia tra 0 e 255.
Categoria: funzione:
Esempio: `PRINT LEFT$("MSX",2)`

LEN (<X$>)
Indica il numero dei caratteri grafici di cui è composta <X$>. Vengono contati anche gli spazi.
Categoria: funzione
Esempio: `PRINT LEN("MSX")`

[LET]<variabile>=<valore>
Questa istruzione assegna un valore ad una variabile. Il termine LET può essere eventualmente omesso.
Categoria: istruzione
Esempio:
```
10 LET A=14:PRINT A
20 LET A=A+3:PRINT A:END
```

LFILES ["<nome file>"]
Dà in visione tutti i files che si trovano sul dischetto. Con LFILES questo elenco viene stampato con la stampante.
Categoria: comando
Esempi: `LFILES`

LINE[[STEP](<X1>,<Y1>)± [STEP](<X2>,<Y2>)[,<Z>][,E[F]]
Disegna una linea tra (x1,y1) e (x2,y2). Con STEP possiamo indicare che il sistema di coordinate viene spostato al punto in cui si trova il cursore. Se vi vuole indicare una diagonale e si chiude questa istruzione con B, viene disegnato un quadrato che ha per diagonale quella in questione. Con BF il quadrato viene colorato.
Nei modi SCREEN 2 e 3 la coordinata X varia da 0 a 255.
Nei modi SCREEN 2 e 3 la Y varia da 0 a 191. Nei modi SCREEN 2 e 3. Z deve trovarsi tra 0 e 15.
Categoria: istruzione
Esempio:
```
10 SCREEN 2
20 LINE (0,0)-(255,191)
30 GOTO 30
```

LINE INPUT ["<testo>";]<variabile stringa>
Inserisce in memoria una stringa che viene battuta sulla tastiera. La stringa può essere formata di tutti i caratteri possibili e di una lunghezza a caso fino ad un massimo di 255 caratteri.
Categoria: istruzione
Esempio
```
10 LINE INPUT A$: PRINT A$
```

LINE INPUT #<numero di file>,<variabile stringa>
Questa istruzione può essere paragonata all'istruzione LINE INPUT, ma in questo caso, la stringa viene letta da un file. Il <numero del file> indica il numero che è stato assegnato al file in questione tramite OPEN. Una stringa viene letta totalmente fino a RETURN.
Categoria: istruzione
Esempio:
```
10 OPEN "CAS:DAT" FOR INPUT AS#1
20 IF EOF(1) THEN 50
30 LINE INPUT#1,A$:PRINT A$
40 GOTO 20
50 CLOSE#1:END
```

LIST [<numero di riga>[-[<numero di riga>]]]
Questo comando riproduce su schermo le righe specificate.
Se non è indicato alcun numero, viene presentato nuovamente tutto il programma.
Categoria: comando
Esempio:
```
LIST
```

LLIST [<numero di riga>>[-[<numero di riga>]]]
Ha la stessa funzione di LIST solo che ora tutte le righe vengono stampate con una stampante.
Categoria: comando
Esempio:
```
LLIST
```

LOAD"<dev>:<nome del programma>"[,R]
Per caricare un programma nella memoria del computer.
Per <dev> si possono considerare:
CAS, A, B, C, D, E, F, COM<numero>.
Se si usa un registratore a cassette il programma deve essere stato registrato sulla cassetta con un comando SAVE. Se aggiungete ',R' al comando, il programma viene avviato immediatamente dopo essere stato caricato.

Categoria: comando
Esempio: `LOAD"CAS:DEMO"`

LOC (<numero di file>)
Questa funzione dà un risultato dipendente dal modo con il quale il file è stato aperto. Se riguarda un file ad accesso casuale, la funzione fornisce il numero dell'ultimo record creato, inserito o rappresentato: vedere GET. Con gli altri tipi di file la funzione fornisce il numero di record letti o scritti da quando il file è stato aperto.
Categoria: funzione
Esempio:
```
10 OPEN "CAS:DAT" FOR OUTPUT AS#1
20 INPUT A$:PRINT#1,A$:PRINT LOC(1)
30 IF A$<>"END" GOTO 20
40 CLOSE#1:END
```

LOCATE [<X>][,[<Y>][,[<cursore in funzione/non in funzione>]]]
Sposta il cursore verso la posizione indicate. Il cursore può essere attivato o memo, e ciò viene indicato rispettivamente con un 1 o uno 0.
E' valido solo con i modi alfanumerici 1 e 2. Il campo delle variazioni di X e di Y dipende da un'istruzione WIDTH, già data o non data.
Categoria: istruzione
Esempio: `10 SCREEN 0:CLS:LOCATE 10,10:PRINT "*"`

LOF (<numero di file>)
Questa funzione indica la lunghezza (in byte) del file specifico.
Il file che viene specificato deve essere giá stato aperto in precedenza.
Categoria: funzione
Esempio:
```
10 OPEN "A:DAT" FOR INPUT AS#1
20 PRINT LOF(1):CLOSE#1:END
```

LOG(<X>)
Indica il logaritmo naturale di <X>.<X> deve essere maggiore di 0.
Categoria: funzione
Esempio: `PRINT LOG(1.5)`

LPOS(<X>)
Qui X può essere un nummero a caso. LPOS indica la posizione della testina della stampante nel buffer di stampa (buffer=memoria intermedia)
Categoria: funzione
Esempio
```
10 LPRINT:PRINT LPOS(0)
20 LPRINT "MSX":PRINT LPOS(0)
```

LPRINT [[USING<formato di stampa>;]<espressione>]
Uguale a PRINT USING, solo che può essere controllato il formato stampa.
Categoria: istruzione
Esempio: `LPRINT "MSX"`

LSET <variabile stringa>=<espressione stringa>
RSET <variabile stringa>=<espressione stringa>
Con questo comando possiamo inserire i dati nel buffer di un file ad accesso casuale. LSET

giustifica a sinistra la stringa all'interno del campo. Con RSET la stringa viene completamente spostata a destra.
Categoria: istruzione
Esempio:

```
10 MAXFILES=1
20 OPEN "A:TEST" AS#1
30 FIELD #1,2ASN1$,4ASN2$,8ASN3$,20ASN4$
40 INPUT "A%";A%
50 INPUT "B!";B!
60 INPUT "C#";C#
70 INPUT "D$";D$
80 RSET N1$=MKI$(A%):RSET N2$=MKS$(B!):RSET N3$=MKD$(C#):
   LSET N4$=D$
90 PUT #1,1
100 A%=0:B!=0:C#=0:D$=""
110 PRINT A%;B!;C#;D$
120 GET #1,1
130 A%=CVI(N1$):B!=CVS(N2$):C#=CVD(N3$):DS=N4$
140 PRINT A%;B!;C#;D$
150 CLOSE #1
160 END
```

MAXFILES=<quantità>
Indica il numero massimo di files che possono essere aperti contemporaneamente.
In un programma possono essere aperti 15 files contemporaneamente, e 6 sul dischetto.
Categoria: istruzione
Esempio:

```
10 MAXFILES=2
20 OPEN "CAS:DEMO" FOR INPUT AS#1
30 OPEN "LPT:"FOR OUTPUT AS#2
40 INPUT#1,A$
50 PRINT#2,A$
60 CLOSE
```

MERGE "<dev>:[<nome file>]"
Aggiunge un programma, che è stato memorizzato su memoria esterna, ai programmi presenti in memoria. Il programma deve essere stato memorizzato in codice ASCII.
Le righe del programma vengono sistemate secondo l'ordine crescente dei numeri di riga. Se due righe hanno lo stesso numero di riga, la riga che si trova in memoria viene sostituita dalla riga del programma nuovamente letto. Il nuovo programma rimane in memoria.
Categoria: comando
Esempio:

```
MERGE "CAS:PROG1"
```

MID$(<X$>,<I>[<,J>])
Indica una stringa che è parte di <X$>. La stringa inizia dalla posizione <I> e <J> indica il numero dei caratteri grafici da considerare. Se <J> non viene specificato vengono considerati tutti i caratteri grafici a partire da <I>.
Categoria: funzione
Esempio:

```
PRINT MID$("BASIC",2,3)
```

MID$ (<stringa>,<primo carattere>[,<un certo numero di caratteri>])=<stringa 2>
Tramite questo comando possiamo modificare la <stringa 1>. Per tale ragione indichiamo

quanti caratteri della stringa 1 devono essere modificati e a partire da dove, sostituendo per esempio la stringa 2 al posto di una parte della stringa 1.
Categoria: istruzione
Esempio:
```
10 A$="PHIPLIE"
20 MID$(A$,4,4)="LIPE"
```
Come risultato sarà assegnata a A$ la stringa 'PHILIPE'.

MKI$(<espressione intera>)
MKS$(<espressione reale>)
MKD$(<espressione reale in doppia precisione>)
Queste funzioni forniscono una stringa che comprende la versione codificata del numero dato. La stringa può essere decodificata tramite le funzioni inverse CVI, CVS e CVD (andare a vedere).
Le stringhe che otteniamo applicando queste funzioni possono essere scritte in un record di un file ad accesso casuale.
Categoria: funzione
Esempio:
```
10 RSET D$=MKD$(WD$)
20 RSET S$=MKS$(WS!)
30 RSET I$=MKI$(WI$)
```

MOTOR ON
MOTOR OFF
Per controllare a distanza un registratore. Con MOTOR ON è come se si premesse il tasto 'play' del registratore, con MOTOR OFF questo dispositivo viene di nuovo interrotto.
I termini ON e OFF possono eventualmente essere omessi. In questo caso si passa alternativamente da uno stato all'altro.
Categoria: istruzione
Esempio:
```
10 MOTOR ON
20 CLOAD "DEMO"
```

NAME <nome di archivio vecchio>AS<nome di archivio nuovo>
Per rinominare i files su un dischetto.
Categoria: istruzione
Per esempio: `NAME "VBD" AS "EXP1"`

NEW
Il programma memorizzato viene cancellato.
Tutte le variabili vengono cancellate e tutti i files aperti vengono chiusi.
Categoria: comando
Esempio: `NEW`

OCT$(<X>)
Indica una stringa che rappresenta il valore ottale di <X>.<X> viena prima arrotondato a numero intero.
Categoria: funzione
Esempio: `PRINT OCT$(636)`

ON ERROR GOTO <numero di riga>
Grazie a questa istruzione, se si incontra un errore nel programma, questo non appare sullo schermo come al solito, ma causa la continuazione del programma al numero di riga indi-

cato. In questo modo si ha la possibilità di scrivere una subroutine per poter gestire i propri errori. Il <numero di riga> si riferisce alla prima riga della subroutine di gestione degli errori. Nella subroutine si possono usare le variabili ERR e ERL.
Con RESUME si ritorna al programma principale. L'istruzione può essere di nuovo annullata con ON ERROR GOTO 0. E' bene raccomandare l'esecuzione dell'istruzione ON ERROR GOTO 0 in una subroutine di messaggi di errore per poter correggere gli errori imprevisti.
Categoria: istruzione
Esempio:
```
10 ON ERROR GOTO 50
20 INPUT "TESTO";A$
30 IF LEN(A$)>5 THEN ERROR 250
40 END
50 IF ERR=250 THEN PRINT "TESTO TROPPO LUNGO":RESUME 20
60 ON ERROR GOTO 0
70 END
```

ON <espressione intera> GOTO <numero di riga>[,<numero di riga>...]
Per spostare l'esecuzione di un programma ad un numero di riga che viene determinato dal valore della <espressione intera>. Se per esempio questa <espressione intera> fornisce il valore 3, il programma sarà continuato a partire dal terzo numero di riga che è stato specificato dopo il GOTO.
Il valore della <espressione intera> non può essere negativo, o uguale a 0 o >255. Se questo valore è 0 o maggiore della quantità dei numeri di riga indicati, il computer prosegue il programma con la prima istruzione eseguibile che segue.
Categoria: istruzione
Esempio:
```
10 INPUT N
20 ON N GOSUB 30,40
30 PRINT "1":GOTO 50
40 PRINT "2":GOTO 50
50 END
```

ON <espressione intera> GOSUB <numero di riga>[,<numero di riga>...]
Questa istruzione è uguale all'istruzione ON GOTO, ma ora i numeri di riga indicati devono riferirsi alle righe di una subroutine.
Categoria: istruzione
Esempio:
```
10 INPUT N
20 ON N GOSUB 40,50
30 GOTO 60
40 PRINT "1":RETURN
50 PRINT "2":RETURN
60 END
```

ON INTERVAL=<tempo> GOSUB <numero di riga>
Indica che dopo una determinata pausa introdotta da INTERVAL, il programma può saltare alla subroutine specificata.
Categoria: istruzione
Esempio: si veda INTERVAL ON/OFF/STOP

ON KEY GOSUB <numero di riga>[,<numero di riga>...]
Indica a quale subroutine si deve saltare se viene premuto uno dei tasti da F1 a F10.

Categoria: istruzione
Esempio: si veda KEY(<n>) ON/OFF/STOP

ON SPRITE GOSUB <numero di riga>[,<numero di riga>...]
Indica a quale subroutine si deve saltare se due sprite si sovrappongono.
Categoria: istruzione
Esempio: si veda SPRITE

ON STOP GOSUB<numero di riga>[,<numero di riga>...]
Indica a quale subroutine si deve saltare se il programma viene interrotto con CTRL/STOP
Categoria: istruzione
Esempio: si veda STOP ON/OFF/STOP

ON STRIG GOSUB <numero di riga>[,<numero di riga>...]
Indica a quale subroutine si deve saltare. Con i numeri di riga 1, 2, 3, 4 e 5 si controlla:

numero	*pulsante d'azione o barra spaziatrice*
1	barra spaziatrice
2	joystick 1, pulsante d'azione 1
3	joystick 2, pulsante d'azione 1
4	joystick 1, pulsante d'azione 2
5	joystick 2, pulsante d'azione 2

Categoria: istruzione
Esempio: si veda STRIG (<X>) ON/OFF/STOP

OPEN "<dev>:[<nome file>]" FOR <elaborazione> AS [#] <numero>
Con OPEN viene aperto un file. Al posto del <dev> possono essere inserite le seguenti parole:

CAS:	registratore a cassette
CTR:	schermo alfanumerico
GRP:	schermo grafico
LPT:	stampante
COM[<Z>]:	comunicazione interfaccia RS232
Da A a F:	diskdrive da 1 a 6.

Come <nome del file> può essere preso un nome composto al massimo da 8 caratteri, e un estensione di 3 caratteri (ad eccezione del registratore a cassetta, dove il nome deve essere composto al massimo da 6 caratteri esclusa l'estensione).
Se al posto di <elaborazione> inseriamo OUTPUT si tratta di un file sequenziale nel quale vengono inseriti dei dati. Se invece inseriamo INPUT si tratta di un file sequenziale dal quale vengono letti dei dati.
Se è omesso FOR<elaborazione>, si tratta di un file di accesso a caso.
Accanto a OUTPUT e a INPUT può essere dato anche APPEND. In questo caso si tratta di un ampliamento di un file sequenziale. La tabella seguente dà una lista di opzioni.

<dev>	*OUTPUT*	*INPUT*	*APPEND*	*Tralasciare per FOR <elaborazione>*
CRT	*			
GRP	*			
LPT	*			
CAS	*	*		
A – F	*	*	*	*
COM	*	*		*

Con l'espressione AS[#]<numero> viene assegnato un numero al file, che poi viene utilizzato nelle istruzioni come INPUT#, PRINT#. Infine aggiungiamo ancora che con l'istruzione OPEN si suppone che il <dev> specificato sia anche realmente collegato. Altrimenti si verifica un messaggio di errore.
Categoria: istruzione
Esempi: si veda CLOSE, GET,INPUT# e LINE INPUT#.

OUT <numero di porta,espressione>
Il dato indicato dall'<espressione> viene mandato alla porta indicata dal <numero di porta>.
Tutti e due i dati specificati devono trovarsi tra 0 e 255.
Categoria: istruzione
Esempio: `10 OUT &HA8,INP(&HA8)`

PAD(<X>)
Con questa importante funzione possiamo controllare come è lo stato di un pannello di contatto.

La seguente tabella mostra quale dispositivo viene indicato.

X	*Viene utilizzato per*
Da 0 a 3	Pannello di contatto collegato al connettore joystick 1
Da 4 a 7	Pannello di contatto collegato al connettore joystick 2

Per il pannello di contatto è valida la tabella seguente:

valore X	*significato del valore di funzione*
0 o 4	definisce se il panello è stato toccato; 0 (non toccato) o –1(toccato)
1 o 5	coordinata X del punto toccato: PAD(0) oppure PAD(4) deve essere –1
2 o 6	coordinata Y del punto toccato: PAD(0) oppure PAD(4) deve essere –1
3 o 7	definisce se l'interruttore è stato premuto (O=no e –1=si)

Categoria: funzione
Esempio:
```
10 SCREEN 2
20 AA=0
30 IF PAD(0)=0 THEN 20
40 X=PAD(1):Y=PAD(2)
50 IF AA=0 THEN PSET(X,Y) ELSE LINE -(X,Y)
60 AA=1
70 GOTO 30
```

PAINT [STEP](<X,Y>)[,[<colore area>][,<colore limiti>]]
Indica come deve essere colorata una determinata area. Con STEP viene spostato il sistema delle coordinate. Con X e Y indichiamo un punto. L'area in cui si trova il punto viene colorata. Se le linee che limitano questa area sono in qualche punto interrotte, viene colorato tutto lo schermo.
Nei modo SCREEN 2 il <colore area> deve essere uguale al <colore limiti>. In questo caso si può non specificare il <colore limiti>. Nei modo SCREEN 3 invece possono esserci delle differenze tra queste definizioni di colore.
Nei modo SCREEN 2 e 3 i numeri dei colori della tavolozza devono trovarsi tra 0 e 15.
Categoria: istruzione
Esempio:
```
10 SCREEN 3:COLOR 15,4,4
20 CIRCLE (80,80),40,8
30 PAINT (80,80),2,8
40 GOTO 40
```

PDL(<X>)
Indica la posizione del 'paddle'. <X> può variare da 1 a 12. Il risultato di questa funzione variano da 0 a 255.
Per <X> = 1, 3, 5, 7, 9, o 11 il computer capisce che il joystick è stato collegato tramite il connettore 1. Per <X> = 2, 4, 6, 8, 10 o 12 viene considerato il collegamento al connettore 2.
Categoria: funzione
Esempio:
```
10 PRINT PDL(1):GOTO 10
```

PEEK(<X>)
Indica il contenuto dell'indirizzo di memoria di X. Come numero decimale X deve variare tra –32768 e 65536. Se X è negativo viene adoperato il cosiddetto complemento 2: PEEK(-1) =PEEK(65536 – 1).
Categoria: funzione
Esempio:
```
PRINT PEEK(65535)
```

PLAY [<primo canale>][,[<secondo canale>][,<terzo canale>]]
E' un'istruzione che permette di riprodurre il suono secondo determinate indicazioni. Con PLAY 'CDE' possiamo sentire per esempio le note C, D e E. Per dare delle indicazioni sulle note (C, D, E, F, G, A, B e anche per C+ corrispondente a CIS, cioè DO diesis, ecc.) possono essere scritte delle lettere e dei numeri che hanno un significato speciale:

codice	*significato*
On	Indica l'ottava in questione, per esempio PLAY"05CDE": vuol dire suona C,D e E in ottava 5. Per n vale: $1 \leq n \leq 8$. Se non vengono date ulteriori informazioni il computer assume O4.
Ln	Indica la durata di una nota così come è specificato qui

durata	*codice*
4 quarti	L1
2 quarti	L2
1 quarto	L4
1/2 quarto	L8
1/4 di quarto	L16
1/8 di quarto	L32
1/16 di quarto	L64

Se non vengono date ulteriori indicazioni il computer assume L4.

Nn — n varia tra 0 e 96: indica una nota con il numero n

Da A a G — Indica la nota. Se per esempio si indica A5, ciò corrisponde a O5A, in breve alla A della quinta ottava

Rn — Indica la pausa a secondo:

durata	*codice*
4 quarti	R1
2 quarti	R2
1 quarto	R4
1/2 quarto	R8
1/4 di quarto	R16
1/8 di quarto	R32
1/16 di quarto	R64

Vn — Indica il volume: n=15 corrisponde al 'volume massimo'. Se non vengono date ulteriori indicazioni il computer assume V8.

• — allunga la durata di una nota di un fattore 1,5.

Sn — Indica il timbro ($0 \leq n \leq 15$). Se non vengono date ulteriori indicazioni, il computer assume S1.

Tn — Indica il tempo. Il valore di n determina la quantità dei quarti di nota in un minuto; n può variare tra 32 e 255. Il valore standard è 120.

Mn — Indica la gamma di variazioni del timbro. Se non vengono date indicazioni, il computer assume M255 ($1 \leq M \leq 65535$).

Categoria: istruzione

Esempio:
```
PLAY "CDECDEEFGC"
```

PLAY(<X>)

Dà informazioni sui canali sonori che vengono usati: 0=tutti i canali, 1=canale 1, 2=canale 2, 3=canale 3.

Se il canale sonoro specificato è attivo, PLAY fornirà il valore –1. Altrimenti PLAY assume il valore 0.

Categoria: funzione

Esempio:
```
10 PRINT PLAY(0)
20 PLAY "CDEFG"
30 PRINT PLAY(0):END
```

POINT(<X>,<Y>)

Definisce il numero di colore del punto indicato dalle coordinate X e Y. Il valore delle variazioni della X e della Y dipende dal modo grafico specificato (si veda SCREEN).

Categoria: funzione

Esempio:
```
PRINT POINT(50,50)
```

POKE<indirizzo,dato>
Colloca i dati in registri di memoria specificati.
<indirizzo> può variare tra –32768 e 65535 incluso. <dato> si trova nell'area tra 0 e 255.
Categoria: istruzione
Esempio: `POKE 5000,100`

POS(0)
Indica la posizione del cursore sulla riga. La posizione più a sinistra è la posizione 0. 0 non è un vero e proprio argomento e non ha quindi altri significati.
Categoria: funzione
Esempio:
```
10 SCREEN 0:LOCATE 10,20:PRINT POS(0)
```

PRESET [STEP](<x,y>[,<colore>]
Rappresenta o cancella un punto (x,y) dello schermo grafico. Se viene indicato un colore, l'effetto di PRESET è uguale a quello di PSET. Senza quest'indicazione, un punto già indicato viene cancellato. Con STEP si può eventualmente spostare il sistema delle coordinate.
Categoria: istruzione
Esempio:
```
10 SCREEN 2
20 FOR K=1 TO 100:PRESET(K,K),1:NEXT
30 FOR K=1 TO 50:PRESET(K,K):NEXT
40 GOTO 40
```

PRINT [<espressione>[, o;][<espressione><, o;>...]]
Riproduce sullo schermo il valore delle variabili, delle espressioni numeriche o delle stringhe. Le stringhe devono essere scritte tra virgolette (").
La posizione dei dati sullo schermo viene determinata dai segni di separazione. Se questo segno di separazione è un punto e virgola (;) o uno spazio, i dati vengono sistemati uno dopo l'altro. Il BASIC divide ogni riga in zone di 14 posizione. Se il segno di separazione tra due dati è una virgola (,), il secondo dato viene collocato nella zona seguente.
Se l'istruzione PRINT viene chiusa da una virgola o da un punto e virgola, i dati della istruzione PRINT successiva vengono sistemati sulla stessa riga di prima, altrimenti in quella successiva. Il termine PRINT può essere indicato anche da un punto interrogativo, per esempio: `10 ? "RISULTATO"; A`
Categoria: istruzione
Esempio:
```
10 A=2:PRINT"MSX-";A
```

PRINT USING"<codice di notazione>";<espressione>[;<espressione>...]
Questa istruzione è un ampliamento della istruzione PRINT, con cui, grazie al <codice di notazione> possiamo indicare da noi stessi in quale formato vogliamo che siano riprodotti i dati. Le <espressioni> devono essere divise da un punto e virgola (;).
Per la rappresentazione delle stringhe si può scegliere fra i tre seguenti <codici di notazione>:

!	specifica che deve essere riprodotto soltanto il primo carattere della stringa.
\n spazi	specifica che devono essere riprodotti i primi 2+n caratteri di una stringa.
&	La stringa viene riprodotta completamente.

Per la rappresentazione di valori numerici si può scegliere fra i seguenti caratteri codici di notazione:

Indica il numero di posizioni prima e dopo la virgola. I numeri troppo grandi vengono arrotondati. I numeri troppo piccoli vengono resi possibili tramite degli zeri e degli spazi. Se il numero è troppo grande per il <codice di notazione> specificato, il simbolo % precede il numero.

```
Esempio: PRINT USING "##.##";1.2345
         1.23
         PRINT USING "##.##";99.996
         %100.00
```

+ Un segno più all'inizio o alla fine di un <codice di notazione> rappresenta un segno di più o di meno prima o dopo il numero.

– Un segno di meno alla fine del <codice di notazione> assicura che i numeri negativi rappresentati siano seguiti da un segno di meno.

****** Un doppio asterisco all'inizio del <codice di notazione> indica che se eventualmente ci dovessero essere degli spazi prima del numero, questi sono occupati da degli asterischi. Inoltre questi due asterischi valgono per due posizioni extra.

$$ Un doppio segno di dollaro $ all'inizio di un <codice di notazione> assicura che venga rappresentato un segno di dollaro $ prima del numero. La notazione scientifica (vedere più avanti) non può essere usata insieme a $$.

****$** Combina gli effetti dei due simboli appena descritti.

, Una virgola alla sinistra del punto decimale assicura che una virgola venga rappresentata ogni tre posizioni.

```
Esempio: PRINT USING "####.##";1234.5
         1,234.50
```

Una virgola alla fine di un <codice di notazione> viene rappresentata normalmente.

```
Esempio: PRINT USING "####.##,";1234.5
         1234.50,
```

^^^^ Se vengono collocate quattro freccette (simboli esponenziali) alla fine del <codice di notazione>, il numero viene rappresentato sotto forma di notazione scientifica.

```
Esempi: PRINT USING "##.##^^^^";123.45
        1.23E+0.1
```

testo Il testo prima o dopo il <codice di notazione> viene rappresentato normalmente purché sia separato dal <codice di notazione> da uno spazio.

```
Esempio: PRINT USING "##.## DOLLARI";12.34
         12.34 DOLLARI
```

Categoria: istruzione
Esempio: si veda sopra

PRINT# e PRINT# USING
La sintassi completa si presenta così:

```
PRINT #<numero del file>,[USING"<codice di notazione>";]<espressione>
[;<espressione>...]
```

Con queste istruzioni i dati vengono scritti su un file.
Il <numero del file> è il numero che è stato assegnato al file, tramite OPEN. Per il <codice di notazione>valgono le stesse possibilità che sono state scritte per l'istruzione PRINT USING. Le <espressioni> vengono scritte sul file una dopo l'altra e come segno di separazione si deve usare solo il punto e virgola (;). Se si usano le virgole, vengono scritti sul file anche gli spazi che vengono aggiunti per la rappresentazione su schermo.
Anche le stringhe vengono collocate sul file una dopo l'altra, quindi non sono più riconoscibili come stringhe separate. Supponete per esempio che A$=TIZIO e B$=CAIO, quindi si scrive l'istruzione:

```
PRINT #1,A$;B$
```

Come risultato si ottiene sul file TIZIOCAIO. Questo è possibile solo quando una stringa viene riletta.
Possiamo prevenire questo problema aggiungendo noi stessi dei segni di separazione. Per esempio:

```
PRINT #1,A$;",";B$
```

dà come risultato TIZIO,CAIO
Un'altra possibilità è scrivere le stringhe sul file con due istruzioni:

```
PRINT #1,A$
PRINT #1,B$
```

Se l'istruzione PRINT non viene chiusa da un punto e virgola, viene aggiunto automaticamente un RETURN alla fine della stringa. Il RETURN funge ora da segno di separazione.
Se la stringa stessa contiene una virgola, nella rilettura, la stringa viene considerata composta di due stringhe. Per esempio A$=TIZIO,CAIO e l'istruzione

```
PRINT #1,A$
```

fornisce nel file TIZIO,CAIO e, dopo aver letto:

```
INPUT #1,A$,B$
```

fornirà come risultato che A$=TIZIO e B$=CAIO. Questo problema può essere evitato collocando la stringa tra virgolette (") nel file (vedere anche l'istruzione INPUT). L'istruzione diventa ora:

```
PRINT #1,CHR$(34);A$;CHR$(34).
```

34 è il codice ASCII per le virgolette(")
Categoria: istruzione
Esempio: si veda sopra

PSET [STEP](<X,Y> [,<Z>]
Questa istruzione permette di collocare un punto sullo schermo grafico. Le coordinate di quel punto corrispondono a X,Y. I valori che la X e la Y possono assumere dipendono dal modo grafico scelto (si veda SCREEN). Con Z viene indicato il numero di colore. Nei modi SCREEN 2 e 3, Z varia tra 0 e 15. Se non si danno delle specificazioni, Z è 15.

Categoria: istruzione
Esempio:
```
10 SCREEN 2
20 FOR K=1 TO 100:PSET(K,K):NEXT
30 GOTO 30
```

PUT [#]<numero di file>[,<numero di record>]
Con questo comando possiamo trascrivere nel file il record che si trova nel buffer del file indicato. Il numero con il quale il record viene trascritto è incrementato di 1 rispetto al numero dell'ultimo record letto o scritto (se non viene specificato alcun numero di record) o è uguale al numero dato.
Vedere anche FIELD, GET, LSET, RSET e OPEN.

Categoria: Istruzione
Esempio:
```
10 OPEN "EXPL.DAT"AS#1
20 FIELD #1,2 AS A$,10 AS B$
30 FOR K%=1 TO 10
40 INPUT N%,S$
50 LSET A$=MKI$(N%)
60 RSET B$=S$
70 PUT #1,K%
80 NEXT
90 CLOSE #1:END
```

PUT SPRITE<numero area>[,[STEP](x,y)][,<colore>][,<numero di sprite>]
Colloca lo sprite nella posizione (x,y) con colore e numero specificato. <numero del piano> è il numero di priorità dello sprite. 0 è la priorità più alta. Se 2 sprite si trovano nella stessa posizione sullo schermo, lo sprite con priorità superiore è visibile.
Con STEP si può eventualmente spostare il sistema di coordinate. Per uno sprite 8x8 si può assumere, come numero di sprite, un numero tra 0 e 255. Per uno sprite 16x16 si può assumere un numero tra 0 e 63.
Nei modi SCREEN 2 e 3 possono essere rappresentati (su una riga) 4 sprites uno accanto all'altro.
Categoria: istruzione
Esempio:
```
10 CLS:COLOR,11,11:SCREEN 2
20 A$="":FOR K=1 TO 8:A$=A$+CHR$(16):NEXT
30 SPRITE$(1)=A$:PUT SPRITE 0,(40,40),1,1
40 GOTO 40
```

READ <variabile>[,<variabile>...]
READ viene usato sempre in combinazione con DATA e assegna i valori specificati da DATA alle <variabili> indicate.
Categoria: istruzione
Esempio: si veda DATA

REM <commenti>
Con questa istruzione è possibile aggiungere dei commenti in un programma. Tutte le informazioni dopo REM vengono ignorate dal BASIC. REM può essere abbreviato con questo simbolo '.
Categoria: istruzione
Esempio:
```
10 REM BEETHOVEN
20 PLAY "GR8GR8GR8L2D+"
```

RENUM [<nuovo numero di riga>][,[<vecchio numero di riga>][,[<intervallo di incremento>]]]
Serve per rinumerare le righe. Il <nuovo numero di riga> è il numero d'inizio della prima riga. Se non viene specificato si inizia dal numero 10.
Il <vecchio numero di riga> indica da quale riga deve iniziare la nuova numerazione. Se non viene indicato, si inizia allora dalla prima riga.
L'<intervallo di incremento> è il numero col quale si indica di quanto deve essere incrementato il numero di riga. 10 è il numero standard.
Categoria: comando
Esempio:
```
RENUM 50,10,20
```

RESTORE [<numero di riga>]
Con RESTORE possono venire rilette dall'inizio, con READ, le liste di valori che sono state specificate da DATA. Vedere inoltre DATA. Eventualmente specificando il <numero di riga> si può indicare una riga per leggerne i dati contenuti.
Categoria: istruzione
Esempio:
```
10 READ A,B,C:PRINT A,B,C
20 RESTORE
30 READ D,E:PRINT D,E
40 DATA 5,6,7
```

RIGHT$(<X$>,<I>)
Indica una stringa composta dagli ultimi caratteri grafici <I> di <X$>.
Categoria: funzione
Esempio:
```
PRINT RIGHT$("MSX",2)
```

RND(<X>)
Dà un numero a caso tra 0 e 1. Il numero iniziale determina quale serie di numeri casuali viene generata.

<X>=negativo	si comincia la serie dei numeri casuali dall'inizio;
<X>=positivo	indica il numero casuale successivo della serie;
<X>=0	indica ancora una volta l'ultimo numero casuale.

Con RND(–TIME) ogni volta otteniamo realmente altri numeri (paragonatelo al RANDOMIZE di altre versioni BASIC).
Categoria: funzione
Esempio:
```
10 FOR K=1 TO 100:PRINT RND(1):NEXT
```

RSET
si veda LSET

RUN[<X>]
RUN"[<dev>:]<nome del programma>"[,R]
Con RUN possiamo avviare un programma. RUN X indica che il programma presente in memoria deve essere eseguito dalla riga X. Se specifichiamo <dev> indichiamo allora che il programma si trova su un dispositivo esterno. In <dev> possiamo inserire: CAS, le lettere da A a F e COM[<n>].
Inoltre RUN provoca la chiusura di tutti i files ancora aperti, a meno che la R non venga specificata.
Categoria: comando
Esempio: `RUN`

SAVE"[<dev>:]<nome del programma>"[,A]
Con questo comando si sposta un programma dalla memoria ad un dispositivo specificato (si veda LOAD). In <dev> si può collocare una delle seguenti definizioni: CAS, le lettere da A a F, e COM[<n>].
Il nome del programma è, allo stesso tempo anche il nome che dobbiamo usare con LOAD e MERGE per richiamare il programma. Con ,A indichiamo che il programma deve essere memorizzato in formato ASCII.
Categoria: comando
Esempio: `SAVE "CAS:DATA"`

SCREEN[<X>[,<Y>[,<Z>[,<XX>[,<YY>]]]]]
Con l'istruzione SCREEN viene indicato come deve essere usato lo schermo. Quando il computer viene acceso, viene assunto automaticamente SCREEN0,0,1,1,0,0.
Le lettere indicate hanno il seguente significato:

<X>	*modo: alfanumerico o grafico*
0	modo alfanumerico 1 con WIDTH 40: 40 col. × 24 righe
1	modo alfanumerico 2 con WIDTH 32: 32 col. × 24 righe
2	modo grafico 1: 256 × 192 pixels
3	modo grafico 2: 64 × 48 blocchi

Il numero dei colori diversi sullo schermo dipende dallo SCREEN modo, in questo modo:

<X>	*Colori*
0	2
1	2
2	16
3	16

Le seguenti istruzioni possono essere usate soltanto con un modo grafico: CIRCLE, DRAW, LINE, PAINT, PSET, PRESET, ON SPRITE GOSUB, SPRITE ON/OFF/STOP, POINT e PUT SPRITE.

<Y>	*Formato degli sprites*
0	8 × 8, senza ingrandimento
1	8 × 8, con ingrandimento (fattore2)
2	16 × 16, senza ingrandimento
3	16 × 16, con ingrandimento (fattore 2)

<Z>	Con o senza suono, nel premere i tasti
0	Non dà alcun segnale bip nel premere i tasti
1	Dà un segnale bip nel premere i tasti

<XX>	Velocità di input/output da una cassetta
1	1200 baud
2	2400 baud

<YY>	Tipo di stampante
0	Stampante MSX
1	Altre

Categoria: istruzione
Esempio: si vedano LINE e PSET

SGN(<X>)
Per <X> = positivo SGN(<X>) diventa = 1
Per <X> = 0 SGN)(<X>) diventa = 0
Per <X> = negativo SGN(<X>)diventa = –1
Categoria: funzione
Esempio: `PRINT SGN(31)`

SIN(<X>)
Indica il seno di <X>. <X> deve essere indicato in radianti.
Categoria: funzione
Esempio: `PRINT SIN(3.1415/12)`

SOUND <nome registro,numero>
E' un'istruzione per generare determinati suoni. Le seguenti convenzioni hanno come valore:

registro	*numero di portata*	*significato*
0	0-255	frequenza del canale A
1	0-15	frequenza del canale A
2	0-255	frequenza del canale B
3	0-15	frequenza del canale B
4	0-255	frequenza del canale C
5	0-15	frequenza del canale C
6	0-31	frequenza rumore
7	0-63	scelta del canale: suono o rumore
8	0-15	volume canale A
9	0-15	volume canale B
10	0-15	volume canale C
11	0-255	frequenza variazione modello
12	0-255	frequenza variazione modello
13	0-14	scelta modello

Categoria: istruzione
Esempio: `10 FOR K=1 TO 10:SOUND K,0:NEXT K`

SPACE$(<X>)
Indica una stringa composta di <X> spazi. <X> viene arrotondata a numero intero e deve variare tra 0 e 255.
Categoria: funzione
Esempio: `10 A$=SPACE$(20):PRINT A$;"A"`

SPC(<X>)
Riproduce gli spazi <X> sullo schermo. SPC può essere usato solo nell'istruzione PRINT o LPRINT. <X> varia tra 0 e 255.
Categoria: funzione
Esempio: `PRINT "MSX";SPC(3);"2"`

SPRITE ON
SPRITE OFF
SPRITE STOP
Con ON, OFF, STOP viene indicato se il computer è predisposto in posizione di controllo per gli 'scontri' tra gli sprites. Vedere ON SPRITE GOSUB. Con SPRITE STOP viene indicato che la situazione di interruzione deve essere continuata. In quel caso si salterà alla subroutine indicata da ON SPRITE GOSUB, solo dopo aver nuovamente incontrato SPRITE ON.
Categoria: istruzione
Esempio:

```
 10 DATA 60,66,165,129,165,153,66,60
 20 DATA 60,126,219,255,255,219,102,60
 30 A$=""
 40 FOR I=1 TO 8
 50 READ A:A$=A$+CHR$(A)
 60 NEXT
 70 B$=""
 80 FOR I=1 TO 8
 90 READ A:B$=B$+CHR$(A)
100 NEXT
110 SCREEN 2,1:COLOR 15,4,1
120 ON SPRITE GOSUB 210
130 SPRITE$(0)=A$:SPRITE$(1)=B$
140 SPRITE ON
150 A=INT(RND(1)*256):B=INT(RND(1)*256)
160 FOR I=0 TO 191
170 PUT SPRITE 0,(A,I),1
180 PUT SPRITE 1,(B,191-I),15
190 NEXT:GOTO 140
200 SPRITE OFF
210 PLAY "L4CDEFEDCREFGAGFER"
220 PUT SPRITE 0,(0,208)
230 PUT SPRITE 1,(0,208)
240 I=191:RETURN
```

SPRITE$
Si tratta di un sistema di variabili che viene usato sempre con la seguente formulazione: SPRITE$(<numero>)=(<espressione stringa>). Il <numero> indica il numero dello sprite.
Il <numero> che dipende dalla definizione del formato sprite, come specificato per esempio dall'istruzione SCREEN, può essere al massimo 63 o 255. Tramite l'<espressione stringa> indichiamo la forma dello sprite. Normalmente questo viene elaborato in base ad un certo numero di funzioni CHR$, per esempio:

```
SPRITE$(1)=   CHR$(&H18)+CHR$(&H3C)+CHR$(&HFF)+CHR$(&H99)
              +CHR$(&H99)+CHR$(&HFF)+CHR$(&HC3)+CHRS(&HFF)
```

Per uno sprite 8x8 abbiamo bisogno di ben 8 funzioni CHR$

SPRITE$ può essere usato nei diversi modi grafici (si veda anche PUT SPRITE).
Categoria: variabile di sistema
Esempio: si veda SPRITE ON/OFF/STOP

SQR(<X>)
Dà la radice quadrata di X (X>0).
Categoria: funzione
Esempio:
```
PRINT SQR(4)
```

STICK(<X>)
Indica la posizione del joystick (1 o 2). Se diamo ad X il valore 0, ci rivolgiamo ai tasti cursore. I valori corrispondono alle seguenti direzioni.
Categoria: funzione
Esempio:
```
PRINT STICK(0)
```

STOP
Interrompe l'esecuzione di un programma. Possiamo nuovamente continuare l'esecuzione del programma con il comando CONT.

STOP può essere collocato in qualsiasi punto del programma.
Categoria: istruzione
Esempio:
```
10 INPUT A
20 PRINT A:IF A=0 THEN STOP
30 GOTO 10
```

STOP ON
STOP OFF
STOP STOP
Regola il modo di un'interruzione causata da CTRL/STOP (vedere ON STOP GOSUB). Con STOP STOP si continua la situazione di interruzione, ciò vuol dire che si salta alla subroutine indicata da ON STOP GOSUB solo se si trova di nuovo uno STOP ON.
Categoria: istruzione
Esempio:
```
10 ON STOP GOSUB 50
20 STOP ON
30 INPUT A$
40 IF A$="END" THEN STOP OFF:END ELSE GOTO 30
50 PRINT "BATTI END (+RETURN)":RETURN
```

STRIG(<X>)
Indica se viene premuto il pulsante d'azione del joystick (-1=si e 0=no).

<X>	*significato*
0	tastiera: barra spaziatrice
1 o 3	joystick 1
2 o 4	joystick 2

Categoria: funzione
Esempio:
```
PRINT STRIG(0)
```

STRIG (<x>) ON
STRIG (<x>) OFF
STRIG (<x>) STOP
Regola il modo di un'interruzione causata da un joystick o da una barra spaziatrice (vedere ON STRIG GOSUB). Con STRIG STOP viene mantenuta la situazione di interruzione, ciò vuol dire che si salta alla subroutine indicata da ON STRIG GOSUB solo se si trova di nuovo l'istruzione STRIG ON.
Categoria: istruzione
Esempio:
```
10 CLS:ON STRIG GOSUB 40
20 STRIG(0) ON
30 GOTO 30
40 LOCATE 5,5:PRINT "BARRA SPAZIATRICE PREMUTA"
50 FOR I=1 TO 300:NEXT:LOCATE 5,5:PRINT SPC(25)
60 RETURN
```

STR$(<X>)
Indica una stringa che rappresenta il valore decimale di <X>.
Categoria: funzione
Esempio:
```
10 A$=STR$(10):PRINT A$:END
```

STRING$(<I>,<J>) o STRING$(<I>,<X$>)
Indica una stringa composta da caratteri <I> che sono uguali a quelli dati in <J> secondo il codice ASCII o corrispondenti al primo carattere di <X$>.
Categoria: funzione
Esempio:
```
PRINT STRING$(4,65)
```

SWAP <variabile>,<variabile>
Scambia i valori di due variabili. Per esempio:
Queste variabili devono essere naturalmente dello stesso tipo.
Categoria: istruzione
Esempio:
```
10 A=3:B=5:SWAP A,B:PRINT A,B
```

TAB(<X>)
Colloca il cursore nella posizione <X> della riga. Se il cursore si trova già nella posizione <X> non succede nulla.
<X> varia tra 0 e 255. 0 è la posizione più a sinistra. TAB può essere usato solo nell'istruzione PRINT o LPRINT.
Categoria: funzione
Esempio:
```
PRINT TAB(12);"*"
```

TAN(<X>)
Indica la tangente di <X>. <X> deve essere indicato in radianti.
Categoria: funzione
Esempio: `PRINT TAN(3,1415/8)`

TIME
La variabile di sistema TIME viene aumentata del valore 'uno' 50 volte al secondo. Possiamo usare questa variabile, in particolare, per ottenere realmente, con una funzione RND, dei numeri casuali (vedere RND).
Categoria: variabile di sistema
Esempio: `PRINT INT (RND(-TIME)*6+1)`

TROFF
TRON
Tramite il comando TRON si predispone il computer a produrre, durante l'esecuzione del programma, anche il numero della riga di cui il computer si occupa in quel momento. Questo comando semplifica il ritrovamento degli errori.
Questa condizione viene interrotta con TROFF.
Categoria: comando
Esempio:

```
10 FOR I=1 TO 3
20 PRINT I
30 NEXT
40 END
TRON
Ok
RUN
[10][20]1
[30][20]2
[30][20]3
[30][40]
Ok
TROFF
Ok
```

USR[<n>](<X>)
Si indica un programma in linguaggio macchina. <n> è il numero che è stato assegnato al programma in linguaggio macchina tramite DEF USR. Se <n> viene omessa, viene considerato USR0.
<X> è un argomento che viene consegnato al programma in linguaggio macchina.
<X> viene passato nel modo seguente:

1. Valori alfanumerici: l'indirizzo &HF663 contiene il valore 3. Gli indirizzi &HF7F8 e &HF7F9 indicano dove si trova questo valore tramite un indirizzamento indiretto. Il valore è memorizzato in tre bytes: il byte 1 contiene la lunghezza, e i bytes 2 e 3 contengono l'indirizzo di memoria del valore alfanumerico.
2. Valori interi: l'indirizzo &HF663 contiene il valore 2. Gli indirizzi da &HF7F6 a &HF7F9 contengono il valore intero (in singola precisione).
3. Valori in singola precisione: l'indirizzo &HF663 contiene il valore 4.
   Gli indirizzi da &HF7F6 a &HF7F9 contengono il valore in singola precisione.
4. Doppia precisione: l'indirizzo &HF663 contiene il valore 8. Gli indirizzi da &HF7F6 a &HF7FD contengono il valore in doppia precisione.

I valori che vengono riportati all'MSX-BASIC dalla subroutine, devono essere memorizzati nelle suddette locazioni dalla subroutine in liguaggio macchina. Con CLEAR possiamo riservare lo spazio necessario.
Categoria: funzione
Esempio:
```
10 CLEAR 200,&HEFFF
20 AB=&HF000
30 FOR I=AB TO AB+9
40 READ A$:A=VAL("&H"+A$)
50 POKE I,A
60 NEXT I
70 DEFUSR=&HF000
80 INPUT "INSERISCI UN NUMERO INTERO"
90 PRINT "NUMERO INTERO=";A%
100 R=USR(A%)
110 PRINT "IL RISULTATO E'UN NUMERO INTERO PIU'1";R
120 END
130 DATA 23,23,4E,23,46,03,70,2B,71,C9
```

VAL(<X$>)
Indica il valore numerico di <X$>. Se il primo carattere di <X$> non è +, -, & o una cifra, VAL assume il valore 0.
Categoria: funzione
Esempio:
```
PRINT VAL("10")
```

VARPTR(<variabile>)
VARPTR(#<X>)
Dà l'indirizzo del primo byte del valore che appartiene a <variabile>, o del primo byte del blocco di controllo del file.
Prima che la funzione possa essere richiamata, bisogna aver precedentemente assegnato un valore alla <variabile> e alla<X>.
Categoria: funzione
Esempio:
```
10 A=10:B=VARPTR(A)
20 IF B<0 THEN B=B+65536
30 C$="0000"+HEX$(B)
40 PRINT RIGHT$(C$,4):END
```

VDP(<X>)
Contiene il contenuto dei registri VDP. X si trova tra 0 e 8.
Il registro 8 può solo essere letto e mai scritto.
Categoria: variabile di sistema
Esempio:
```
10 FOR I=0 TO 8
20 A=VDP(I):B$="00000000"+BIN$(A)
30 PRINT RIGHT$(B$,8)
40 NEXT
```

VPEEK(<X>)
Dà il contenuto dell'indirizzo <X> della memoria del video. X si trovatra 0 e 16383.

Categoria: funzione
Esempio:
```
10 A=VPEEK(0)
20 A$="00"+HEX$(A)
30 PRINT RIGHT$(A$,2)
```

VPOKE <indirizzo, dato>
Uguale a POKE, solo che ora l'<indirizzo> si riferisce ad un indirizzo di una parte della memoria RAM del video.
L'indirizzo può variare tra 0 e 16383. Il dato può variare tra 0 e 255.
Categoria: istruzione
Esempio:
```
10 VPOKE(0),(VPEEK(0))
```

WAIT <numero di porta, espressione 1>[,<espressione 2>]
Si riferisce alla lettura dei dati tramite la porta I/0 con il numero di porta specificato.
I valori letti vengono combinati con l'<espressione 1> tramite la condizione esclusiva OR. Il risultato viene combinato tramite un'operazione AND all'<espressione 2>. Se il risultato che viene fornito è 0, la lettura viene proseguita, altrimenti il computer continua con l'istruzione seguente del programma. Se l'<espressione 2> viene omessa, viene considerato il valore 0.
Categoria: istruzione
Esempio:
```
10 WAIT &HA8,240
```

WIDTH (<numero di colonne>)
Stabilisce il numero di colonne sullo schermo alfanumerico.
Nel modo SCREEN 0 viene definito lo schermo da 1 a 40 colonne. Nel modo SCREEN 1 viene definito lo schermo da 1 a 32 colonne (si veda SCREEN).
Osserviamo ancora. Se il computer viene acceso, l'immagine viene definita con una larghezza di 37 colonne, secondo il modo SCREEN 0. Se si passa al modo SCREEN 1, l'immagine viene suddivisa in 29 colonne.
Categoria: istruzione
Esempio:
```
SCREEN 0:WIDTH 40
```

INDICE ANALITICO

A
ABS O2, I11
AND A17
APPEND A8
archivio U21
array I30
ASC$ I34, O2
ATN I11, O2
AUTO I4, O2
AUTOEXEC.BAS A7

B
BASE O2
BEEP U1, O3
BIN$ O3
BLOAD A2, A11, O3
BSAVE A2, A11, O3

C
CALL O4
CALL COM A15, O4
CALL COMBREAK A15, O4
CALL COMDTR A15, O5
CALL COMINI A15, O5
CALL COMOFF A15, O7
CALL COMON A15, O7
CALL COMSTAT A15, O7
CALL COMSTOP A15, O7
CALL COMTERM A15, O8
CALL FORMAT O9
CALL SYSTEM A11, O8
campi A9
canale sonoro U2
CAS U21
CDBL O8
CHR$ I34, O9
ciclo I27
CINT O9
CIRCLE O9, U14
CLEAR I34, O9
CLOAD A1, A2, O9
CLOAD? O10
CLOSE U24, A2, A3, A11, A15, O10
CLS I22, O10
codici di controllo A23
COLOR U10, O10
COM U23
comunicazione U22
CONT O10
COPY U19, A5, A11, O10,
COS I11, O11
CRT U22
CSAVE A2, O11
CSNG O11
CSRLIN O11
CTRL A23
CVD A11, O11
CVI A11, O11
CVS A11, O11

D
DATA I13, O11
DEF O12
DEF FN I36, O12
DEF USR O12
DELETE I4, O13
DIM I30, O13
diskdrive A5
divisione I1
doppia precisione I17
DRAW U13, O13
drive A5
DSKF A11, O15
DSKI$ A11, O15
DSKO$ O15, A11

E
END I36, O16
EOF A2, A11, A15, O16
EQV A17
ERASE O16
ERL O16
ERR O16
ERROR O16
espressioni logiche A16
estensioni U23
etichetta I7
EXP I11, O17

F
FIELD A11, O17
file U21
FILES A6, A11, O17
files ad accesso casuale A9
FIX O17
FOR...NEXT I26, O17
FORMAT A7, A11
formattare A7
FRE O18
funzioni standard I9
funzioni stringa I33

G
GET A11, O18
GOSUB O18
GOSUB...RETURN I35
GOTO I24, O19
GRP U21

H
HEX$ O19

I
IF...THEN I24, O19
IF...THEN...ELSE I26 6
IMP A17
INKEY$ I34, O20
INP O20
INPUT I12, O20
INPUT# A2, A11, A15, O20
INPUT$ O20
INPUT$# A2, A11, A15
INSTR O21, I34
instruzioni di controllo I24
INT I11, O21
interfaccia A13
interfaccia RS232C A13
interfaccia sequenziale A13
INTERVALL OFF O21
INTERVAL ON O21
INTERVAL STOP O21
istruzioni I3

J
joystick A4

K
KEY O21
KEY LIST O21
KEY OFF O21
KEY ON O27
KILL A6, A11, O22

L
LEFT$ I33, O22
LEN I33, O22
LET O22
LFILES A3, A11, O22
LINE U12, O29
LINE INPUT O23
LINE INPUT# A2, A11, A15, O23
LINE INPUT$# A2, A11, A15
LIST I4, O23
LLIST A3, O23
LOAD A2, A11, A15, O23
LOC A12, O24
LOCATE I22, O24
LOF A12, O24
LOG I11, O24
LPOS O24
LPRINT A3, O24
LPT U22
LSET A11, O22

M
MAXFILES U26, A11, O25
MERGE A11, A15, O25, A2

messaggi di errore A18
MID$ I34, O25
MKD$ A12, O25
MKI$ A12, O25
MKS$ A12, O25
modem A13
MOTOR OFF O25
MOTOR ON O25
multiplicazione I1

N
NAME A11, O26
NEW I4, O26
nomi riservati A27
NOT A17
numeri I16
numeri binari I18
numeri esadecimali I19
numeri interi I18
numeri octali I19

O
OCT$ O26
ON ERROR GOTO O26
ON...GOSUB O27
ON...GOTO O27, I29
ON INTERVAL...GOSUB O27
ON SPRITE GOSUB U18, O28
ON STOP GOSUB O28
ON STRIG GOSUB A4, 028
OPEN U24, A2, A3, A11, A15, O28
OR A17
OUT O29

P
PAD A4, O29
PAINT U15, O30
parentesi I2
PDL A4, O30
PEEK O30
PLAY U1, O30
POINT O32
POKE O32
POS O32
PRESET O32
PRINT I1, I21, O32
PRINT USING I23, O38
PRINT# U24, A2, A3, A11, A15, O38
PRINT# USING A2, A3, A11, A15, O38
programma I2
PSET U9, O35
PUT A11, O35
PUT SPRITE U17, O35

R
random access files A9
rappresentazioni grafiche U8
READ I13, O36
records A9
registratore al cassette A1
regole di priorita I2
REM I23, O36
RENUM I4, O36
RESTORE I15, O36
RETURN I1
RIGHT$ I33, O36
RND O36
RSET A11, O22, O37
RUN I4, O37

S
SAVE A2, A11, A15, O37
SCREEN U8, O37
segni di operazione A16
segni di relazione I26
segni di separazione I11, I21, U26
sequenzi di escape A22
SET VIDEO O47
SGN I11, O38
simboli alternativi A24
simboli standard A25, A26
SIN I11, O38
singola precisione I17
SOUND U6, O38
SPACE$ I34, O39
SPEC O39
SPRITE OFF O39
SPRITE ON U18, O39
SPRITE STOP O39
SPRITE$ U17, O40
sprites U16
SQR I11, O40
stampante A3
stampare A3
STEP I27
STICK A4, O40
STOP O40
STOP OFF O40
STOP ON O40
STOP STOP O40
STR$ I34, O41
STRIG A4, O41
STRIG OFF A4, O41
STRIG ON A4, O41
STRIG STOP A4, O41
STRING$ O41
stringhe I32
subroutine I35
SWAP O41

T
TAB I22, O41
TAN I11, O42
tasti speciali I6
testo I2
TIME O42
trasmettere i dati A14
trasmettere un programma A13
TROFF O42
TRON O42

U
USR O42

V
VAL I34, O43
variabili I9
variabili stringa I32
variabili vettore I30
VARPTR O43
VARPTR# A2, A3, A12, A15
VDP O43
VPEEK O43
VPOKE O44

W
WAIT O44
WIDTH I22, O44
wild cards U23

X
XOR A17

Che cosa significa standard MSX? Perché lo standard MSX è destinato a recitare un ruolo di primo piano nel mercato dei computer? Che cosa si può fare con il linguaggio MSX-Basic? In questo libro molto facile e comprensibile vi sono la risposta a queste e ad altre domande sui computer. Esso è stato escritto considerando i lettori dei novizi nel mondo dell' informatica. In questo libro scoprirete tutti i segreti dell' affascinante mondo della programmazione.

PHILIPS

8 622 541 00004