NSB8

Interprete BASIC per famiglia 80

MANUALE D'USO

grifo®
ITALIAN TECHNOLOGY

Via dell' Artigiano 8/6
40016 San Giorgio di Piano
(Bologna) ITALY
Tel. (051) 89 20 52 (4 lin. r.a.)
FAX (051) 89 36 61

NSB8 Edizione 3.0 *Rel. 5 Gennaio 1993*

NSB8

Interprete BASIC per famiglia 80

MANUALE D'USO

NSB8 è un potente interprete BASIC sviluppato per la famigli di CPU Z80 e compatibili. E' provvisto di un completo set di Floating Point da 8 cifre significative con la trattazione anche delle funzioni trigonometriche e trascendentali.
Tramite questo linguaggio l'utente può affrontare applicativi anche di notevole complessità. L'ambiente di sviluppo è estremamente amichevole, e l'impiego di tempo per la generazione degli applicativo, è estremamente ridotto.
La produttività del codice e l'immediatezza con cui è possibile intervenire sull'hardware, rendono questo BASIC un impagabile strumento di lavoro ad ogni livello.

grifo®
ITALIAN TECHNOLOGY

Via dell' Artigiano 8/6
40016 San Giorgio di Piano
(Bologna) ITALY
Tel. (051) 89 20 52 (4 lin. r.a.)
FAX (051) 89 36 61

NSB8 Edizione 3.0 *Rel. 5 Gennaio 1993*

INDICE GENERALE

COMANDI:

ALOAD 1
APPEND 2
ASCSAVE 3
AUTO 4
BYE 5
CAT 6
CONT 8
DEL 9
LIST 10
LOAD 11
MEMSET 13
PSIZE 14
REN 15
RUN 17
SAVE 18
SCR 20

ISTRUZIONI:

CHAIN 21
CLOSE 22
DATA 23
DEF 24
DIM 25
ERRSET 27
EXIT 28
FILL 29
FNEND 31
FOR 32
GOSUB 33
GOTO 34
IF 35
INPUT 36
INPUT1 38
LET 39
LINE 40
NEXT 42
ON 43
OPEN 44
OUT 46
PRINT 47
READ# 49
READ 51

segue ISTRUZIONI:

REM **52**
RESTORE **53**
RETURN **54**
RETURN **55**
STOP **56**
WRITE **57**

ARGOMENTI:

CONTROL-C **59**
DISPOSITIVI DI I/O MULTIPLI **60**
LE FUNZIONI **61**
FUNZIONI INTERNE **61**
ARGOMENTI **61**
FUNZIONI MATEMATICHE **62**
FUNZIONI PER OPERAZIONI SU STRINGA **63**
FUNZIONI PER PARTICLARI INPUT **63**
FUNZIONI PER LA MANIPOLAZIONE DI FILE SU DISCO **64**
FUNZIONI DI CARATTERE GENERALE **64**
FUNZIONI-UTENTE **65**
NOME DELLE FUNZIONI-UTENTE **65**
FUNZIONI-UTENTE COMPOSTE DA UNA SOLA LINEA **66**
PASSAGGIO PARAMETRI ALLE FUNZIONI-UTENTE **66**
FUNZIONI-UTENTE COMPOSTE DA PIU' LINEE **67**
NOTE FINALI **68**
I NUMERI **69**
COSTANTI **69**
PRECISIONE **69**
FRAZIONI **69**
PRECISIONE 8 CIFRE **70**
UN NUMERO MOLTO GRANDE **70**
UN NUMERO MOLTO PICCOLO **70**
INTERVALLO **70**
VARIABILI **71**
OPERATORI **71**
OPERATORI ARITMETICI **71**
OPERATORI RELAZIONALI **72**
ESPRESSIONI **72**
ESEMPI DI ESPRESSIONI NUMERICHE LEGALI **72**
ESEMPI DI ESPRESSIONI NUMERICHE ILLEGALI **72**
OPERATORI BOOLEANI **73**
ORDINE DI VALUTAZIONE DEGLI OPERATORI **73**
I VETTORI **74**
INDICI **74**
VETTORI A PIU' DIMENSIONI: MATRICI **74**
DIMENSIONAMENTO DI DEFAULT **75**
I VETTORI NON POSSONO ESSERE RIDIMENSIONATI **76**
RIFERIMENTI AD ARRAY NELLE ESPRESSIONI NUMERICHE **76**

segue ARGOMENTI:

LE STRINGHE 77
COSTANTI DI TIPO STRINGA 77
LA STRINGA NULLA 77
VARIABILI DI TIPO STRINGA 77
DIMENSIONAMENTO VARIABILI DI TIPO STRINGA 78
SOTTOSTRINGHE 78
APERTURA E CHIUSURA DI UNA SOTTO-STRINGA 79
CONCATENAZIONE DI STRINGHE 79
FUNZIONI SU STRINGHE 79
ESPRESSIONI DI TIPO STRINGA 79
COMPARAZIONE TRA STRINGHE 80
ASSEGNAMENTI A STRINGHE E A SOTTO-STRINGHE 81
MASSIMA LUNGHEZZA DI STRINGHE E LUNGHEZZA CORRENTE 82
SETTAGGIO CATARRERI IN BASIC 82
FORMATO DI STAMPA 83
STAMPA DI UN NUMERO IN FORMATO NORMALE ED ESPONENZIALE 83
COSA RAPPRESENTA IL FORMATO DI UN NUMERO ? 83
RAPPRESENTAZIONE A DESTRA 84
NUMERO DI DECIMALI 84
FORMATO DI DEFAULT E FORMATO CORRENTE 85
ALTRI FORMATI CARATTERI 86
FLUSSO DI ESECUZIONE E DI CONTROLLO 87
IL LOOP FOR-NEXT 88
CORPO DEL CICLO 88
LA VARIABILE DI CONTROLLO ED IL VALORE LIMITE 88
VALORE DEL PASSO (OPZIONALE) 88
ANNIDAMENTI DI CICLI FOR-NEXT 89
INSERIMENTO OPZIONALE DELLA VARIABILE DI CONTROLLO IN NEXT ... 90
USO DI EXIT 90
USCITA DA LOOP INNESTATI 91
LE PROCEDURE 92
I FILE DATI 93
NOME DEI FILES 93
SPAZIO OCCUPATO DAI FILE (LUNGHEZZA) 94
TIPO DEI FILES 94
APERTURA DEI FILES 94
CHIUSURA DEI FILE 94
TIPI DI DATI NEI FILES 95
ACCESSO DEI DATI 95
ACCESSO SEQUENZIALE 95
AGGIUNTA DI NUOVI DATI IN UN FILE SEQUENZIALE 97
ACCESSO SEQUENZIALE AI BYTE 97
ACCESSO RANDOM DEI DATI 98
PROCEDURE IN LINGUAGGIO MACCHINA 100
CONCATENAZIONE (Sequenza automatica del programma) 101
COMUNICAZIONE TRA PROGRAMMI 101
GLI ERRORI E LE SOLUZIONI 102

segue ARGOMENTI:

LA LINEA DI EDITOR 103
INTRODUZIONE ALL'EDITOR 103
IL COMANDO EDIT 104
SPECIFICHE E FUNZIONI DEL LINE EDITOR 105
COPIA DEL RESTO OLD LINE ALLA FINE DELLA NEW LINE (Control-G) 105
COPIA DI UN CARATTERE DALLA OLD LINE (Control-A) 106
PRENDERE UN CARATTERE (Control-Q) 106
CANCELLAZIONE DI UN CARATTERE DALLA OLD LINE (Control-Z) 106
COPIA SU UNO SPECIFICO CARATTERE (Control-D) 106
ATTIVARE O DISATTIVARE IL MODO DI INSERZIONE (Control-Y) 107
CANCELLARE E REINSERIRE UNA NUOVA LINEA (Control-N) 107
COMPATIBILITA' CON ALTRI BASIC 108
MANIPOLAZIONE DELLE STRINGHE 108
VETTORI DI TIPO STRINGA 108
DICHIARAZIONI DELLE STRINGHE 108
TRASLAZIONE DEGLI INPUT 109
ARITMETICA IN BCD DEL BASIC NSB8 109
LA VALUTAZIONE IF ... THEN 109
INDIRIZZI SPECIALI DI NSB8 110
BASIC PERSONALIZZATO 111
PERSONALIZZAZIONE DEL BASIC NSB8 111
1. MEMORY SIZE 111
2. FISSARE UNA VARIABILE COME ORIGINE DEL BASIC 112
3. LUNGHEZZA DI LINEA 112
4. IMPAGINAZIONE VIDEO 112
5. CARATTERE DI "BACKSPACE" 113
6. INIBIZIONE DEL CONTROL-C 113
7. IL BOOTSTRAP PROM NON STANDARD 113
8. BASIC RISTRETTO 114
TABELLA DI RIFERIMENTO DEGLI INDIRIZZI DI NSB8 115
NOTE DI IMPLEMENTAZIONE 116
FORMATI DELLA MEMORIA-DATI DEI DISCHETTI 116
DIMENSIONI DEL BUFFER DEL FILE - "TEMPO DI VITA" DEI BUFFERS 117
TABELLA DEI DISPOSITIVI DI STAMPA 117
TABELLA DI GESTIONE DEI FILE 117
PROGRAMMI BASIC PRE-ESEGUITI 117
FORMA INTERNA DI UN PROGRAMMA 118
USO DELLA RAM DURANTE L'ESECUZIONE DEL PROGRAMMA 118
MESSAGGI DI ERRORE DEL BASIC NSB8 119

APPENDICI:

APPENDICE A: ESECUZIONE PROGRAMMI DA GDOS 123
APPENDICE B: GUIDA RAPIDA AL BASIC NSB8 125
APPENDICE C: INDICE ANALITICO 133

ALOAD

Comando:

ALOAD <nomefile>

Azione:

La codifica ASCII del programma contenuta nel file specificato, è caricato nella memoria RAM, diventa cioè il programma corrente.

Esempi:

ALOAD TEST1
ALOAD PROG2

Note:

Come per il comando LOAD, tale operazione, comporta l'utilizzo implicito, del comando SCR, in modo, che l'area di memoria, adibita a ricevere il contenuto del file, risulti vuota all'atto del trasferimento.

Messaggi di errore:

TOO LARGE OR NO PROGRAM ERROR
Vedi comando **LOAD**

HARD DISK ERROR
Vedi comando **LOAD**

FILE ERROR
Vedi comando **LOAD**

ARG ERROR
Vedi comando **LOAD**

TYPE ERROR
Vedi comando **LOAD**

Vedi anche:

Comando **ASCSAVE**
Comando **LOAD**
Comando **SRC**

APPEND

Comando:

APPEND <nome file>

Azione:

Unisce un programma BASIC al programma corrente, contenuto nel file specificato.
Perchè abbia successo l'operazione di APPEND il più basso numero linea del programma contenuto nel file deve essere maggiore del più alto numero linea presente nel programma corrente.

Esempi:

APPEND MYPROG
APPEND TESTER,2

Note:

Se non esiste un programma corrente, il comando APPEND, agisce esattamente come il LOAD.
L'operazione di APPEND, se ha successo, azzera tutte le variabili nell'area destinata al programma e ai dati.

Messaggi di errore:

LINE NUMBER ERROR
Il primo numero linea del programma da unire è minore o uguale all'ultimo numero linea del programma corrente.

TOO LARGE OR NO PROGRAM ERROR
Nel file specificato non è presente un programma BASIC valido, o il programma risultante dall'operazione di APPEND è troppo grande per inserirlo nella memoria disponibile. Da notare, che nelle situazioni d'errore specificate, il programma corrente, rimane inalterato.
Si prega di fare riferimento ai comandi LOAD e SAVE per avere dettagli sugli errori che possono verificarsi durante l'operazione APPEND, e specificatamente:

HARD DISK ERROR
FILE ERROR
ARG ERROR
TYPE ERROR

ASCSAVE

Comando:

ASCSAVE <nome file>

Azione:

Il programma corrente, è salvato permanentemente, in un file ASCII, sul dischetto.

Esempi:

ASCSAVE PROG
ASCSAVE TEST,2

Note:

Il comando ASCSAVE, è una speciale forma di SAVE, ed è utilizzato per salvare il programma sotto forma di ASCII, in un file su disco.
La codifica ASCII del programma, può essere utile, per esempio, se si intende stampare il programma con una stampante, o si intende modificare il programma, con altri EDITOR.

Messaggi di errore:

ARG ERROR
Vedi comando **SAVE**

TYPE ERROR
Vedi comando **SAVE**

Vedi anche:

Comando **SAVE**
Comando **ALOAD**

AUTO

Comando:

AUTO
AUTO <numero linea>,<numero linea>

Azione:

Si entra nel modo di numerazione automatica linee, nel quale, NSB8 genera automaticamente nuovi numeri linea per successive linee del programma. Lo specifico numero linea, sarà il primo numero linea usato in auto-mode. Ogni nuovo numero linea, sarà incrementato automaticamente rispetto al precedente, di un particolare valore, espresso dall'indice.
Il numero che esprime l'indice, deve essere un numero intero compreso tra 1 e 65535.
Da ricordare, che non è ammesso inserire l'indice, senza inserire prima il numero di linea iniziale, inoltre se uno o entrambi i parametri, vengono omessi, NSB8 come default assume il valore 10.

Esempi:

AUTO
AUTO 400
AUTO 1000,100

Note:

Nel modo di numerazione automatica, un nuovo numero linea, sarà scritto all'inizio di ogni linea. Tale modo di numerazione, resterà attivo finchè non si presenterà, una di queste situazioni:

A) scrittura di un carriage return immediatamente dopo il numero linea.
B) scrittura di una linea senza il corrispondente numero (per effettuare tale operazione, cancellare il numero linea, non si deve fare altro che utilizzare le risorse dell'edit di NSB8.)
C) generazione di un numero di linea, superiore a 65535.

Da ricordare che in caso venga generato un numero di linea già esistente, il vecchio viene rimpiazzato dal nuovo, perciò si perde l' intera linea precedente.

Messaggi di errore:

OUT OF BOUNDS ERROR
Il numero di linea iniziale o il valore dell'indice o entrambi, sono maggiori di 65535 o minori di 0.

ARG ERROR
Il numero di linea iniziale o il valore dell'indice o entrambi, sono negativi, o non sono interi.

Vedi anche:

Comando **REN**

BYE

Comando:

BYE

Note:

Il comando BYE, non influenza l'area dati/programma di NSB8 in nessun caso e quindi il programma corrente e ogni dato associato rimane intatto.
È possibile rientrare in BASIC e riprendere il lavoro con l'ultimo programma corrente sapendo che la memoria contenente il BASIC e la sua area dati/programma non è stata modificata nel frattempo.
Per rientrare in BASIC e riprendere il lavoro abbandonato, basta digitare RIT.

Messaggi di errore:

Nessuno

CAT

Comando:

CAT
CAT <numero drive>
CAT <espressione del dispositivo di uscita>
CAT <espressione del dispositivo di uscita>,<numero drive>

Azione:

Una lista dei file presenti nel dischetto inserito nel drive specificato è stampata sul dispositivo di output dichiarato nel comando. L'espressione che rappresenta il dispositivo di output deve essere formata dal simbolo # seguito da un numero compreso tra 0 e 7. Il numero che rappresenta il drive, invece, deve essere compreso tra il valore 1 e 4. Se l'espressione che rappresenta il dispositivo di uscita e'omessa, la stampa della lista è inviata come default al terminale (dispositivo #0).
Se il numero di drive non è specificato, si associa al comando (come default) il drive #1

Esempi:

CAT	{la directory del drive #1 è mandata alla consolle}
CAT 3	{la directory del drive #3 è mandata alla consolle}
CAT #1	{la directory del drive #1 è mandata al dispositivo #1}
CAT #2,3	{la directory del drive #3 è mandata al dispositivo #2}

Note:

La lista prodotta è identica a quella che si ottiene con il comando **DIR** del sistema operativo, inoltre come nel listato del programma, anche la directory può essere impaginata, per effettuare però tale operazione, si consiglia di consultare il manuale del sistema operativo.
Infine si raccomanda all'utente di accertarsi che i dispositivi di uscita e i drive che intende usare, siano effettivamente associati ai parametri immessi nel comando, in caso contrario altrimenti si possono provocare errori.

Messaggi di errore:

HARD DISK ERROR
Questo errore può scaturire, quando si presentano le seguenti circostanze:

1) Il drive specificato, non è installato nel sistema.
2) Il drive specificato non è attivato.
3) Il dischetto non è inserito nel drive, correttamente (la porta del drive è aperta, ecc.).
4) Non è presente alcun file sul dischetto inserito nel drive specificato.
5) La directory sul dischetto è rovinata.

FILE ERROR
Il numero che specifica il drive, è maggiore di 4.

Vedi anche:

Manuale del sistema operativo

CONT

Comando:

CONT

Azione:

CONT, fa partire l'esecuzione del programma, dalla linea in cui ci si era bloccati per effetto di uno STOP o di un control-C.

Programma di esempio:

```
10  PRINT "QUESTA LINEA E` SCRITTA DOPO IL RUN"
20  STOP
30  PRINT "QUESTA LINEA E` SCRITTA DOPO IL CONT"
```

Note:

CONT non può essere usato nel caso in cui la precedente esecuzione si è fermata per effetto di un errore o dell'istruzione END, oppure se si è modificato il testo del programma durante l'interruzione. E' possibile utilizzare istruzioni dirette durante l'interruzione causata da uno STOP o da un control-C, in modo per esempio da cambiare direttamente i valori delle variabili utilizzate nel programma.In caso si prema control-C durante l'esecuzione di una istruzione di input, la successiva esecuzione prodotta con il comando CONT, ripeterà tutta la linea.

Messaggi di errore:

CONTINUE ERROR
Questo errore avviene quando si presenta uno di questi motivi:

1) Il programma si è arrestato, perchè è stata eseguita l'istruzione END.
2) L'esecuzione si è arrestata causa un errore nel programma.
3) Il programma è stato cambiato durante l'interruzione.
4) Il programma corrente, non è mai entrato in esecuzione.

Vedi anche:

Capitolo **CONTROL-C**
Istruzione **STOP**

DEL

Comando:

DEL <numero linea>,<numero linea>

Azione:

Tutte le linee di programma comprese nell'intervallo sono cancellate dal programma corrente.
Il secondo numero linea specificato nell'intervallo, deve essere maggiore del primo.

Esempi:

DEL 10,20
DEL 1000,1075

Note:

DEL è usato, per cancellare blocchi di programma in una sola volta.
Se si desidera invece, cancellare una sola linea, l'operazione più semplice è quella di digitare il numero linea corrispondente e premere <CR>. Tutte le variabili risultano azzerate come utilizzando il comando DEL (o qualsiasi altro che modifica il programma corrente. Ricordare che una volta utilizzato tale comando le linee cancellate sono perse e quindi se si rivogliono nel programma, bisogna reinserirle.

Messaggi di errore:

ARG ERROR
Il secondo numero linea specificato nell'intervallo, è minore del primo.

LINE NUMBER ERROR
Una o entrambi le linee specificate nell'intervallo non esiste nel programma corrente.

OUT OF BOUNDS ERROR
Uno o entrambi i numeri linea specificati nell'intervallo sono minori di 0 o maggiori di 65535.

LIST

Comando:

LIST
LIST <intervallo numero linea>
LIST <espressione numero dispositivo>
LIST <espressione numero dispositivo>,<intervallo numero linea>

Azione:

Stampa il testo del programma corrente. L'opzionale espressione dispositivo è formata dalla scrittura del crosshatch (#) con un numero da 0 a 7, corrispondente ad un dispositivo di output. Se nessun dispositivo è fornito come default è assunto il dispositivo #0 (consolle terminal). Se si è definito, nell'istruzione l'intervallo di numeri di linea, le linee, stampate, saranno solo quelle che hanno un numero di linea compreso nell'intervallo suddetto. L'intervalo puo'essere definito così:

<numero linea>	Solo la linea con il numero linea specificato è visualizzata.
<numero linea>,	Vengono visualizzate tutte le linee del programma, iniziando dal numero linea specificato.
<numero linea>,<numero linea>	Vengono visualizzate tutte le linee comprese nell'intervallo specificato.

Se nessun intervallo è specificato, l'intero programma è visualizzato.

Esempi:

LIST
LIST 1000
LIST 30,
LIST 100,200
LIST #1
LIST #,30,700

Note:

Nell'intervallo, il secondo numero linea (se dato), deve essere maggiore o uguale al primo.
Per la convenienza dell'utente con un monitor CRT il programma listato può essere automaticamente impaginato.

Messaggi di errore:

LINE NUMBER ERROR
Una o entrambi le linee specificate nell'intervallo non esiste nel programma corrente.

OUT OF BOUNDS ERROR
Una o entrambi le linee specificate nell'intervallo non rientrano nel range 0-65535.

LOAD

Comando:

LOAD <nome file>

Azione:

Il programma BASIC contenuto nel file specificato, è caricato nell'area dati e di programma, e diventa il programma corrente.

Esempi:

```
LOAD PROG3      {carica dal drive #1}
LOAD TEST8,2    {carica dal drive #2}
```

Note:

Il file specificato deve essere di "tipo 2".
L'operazione di load, comporta l'uso del comando SCR, in quanto prima di caricare un programma, l'area adibita a riceverlo deve risultare vuota.

Messaggi di errore :

TOO LARGE OR NO PROGRAM ERROR
Il programma nel file specificato è troppo grande per essere caricato nella memoria, o il file non contiene un programma BASIC valido. In entrambi i casi comunque occorre una pulizia della memoria (SCRatch) .
Vedere il comando **MEMSET** e il capitolo **BASIC PERSONALIZZATO** per avere informazioni di come si procede per aumentare le dimensioni dell'area programma.

HARD DISK ERROR
Riferimento al comando **CAT.**
Dipende dallo stato durante l'operazione di caricamento, comunque una pulizia della memoria avviene, ogni qualvolta si presenta un errore.
Se si verifica un errore durante il caricamento, l'area di memoria, non viene modificata.

FILE ERROR
Vedi comando **SAVE**

ARG ERROR
Vedi comando **SAVE**

TYPE ERROR
Vedi comando **SAVE**

Vedi anche:

Comando **SAVE**
Comando **ASCSAVE**
Comando **ALOAD**
Comando **SCR**

MEMSET

Comando:

MEMSET <indirizzo memoria>

Azione:

Il limite superiore della regione di memoria dati/programma utilizzata da NSB8, è cambiato nell'indirizzo specificato, il quale deve essere un numero intero compreso tra 0 e 65535.

Esempi:

MEMSET 24575	{la successiva cella di memoria è 5FFFH}
MEMSET 32767	{la successiva cella è 7FFFH}
MEMSET 40959	{la successiva cella è 9FFFH}

Note:

Da notare che l'indirizzo specificato nel comando MEMSET deve essere espresso come numero decimale (base 10).
L'indirizzamento alla memoria del microcomputer, comunque è dato in esadecimale (base 16).
Se l'indirizzo di memoria desiderato è conosciuto in esadecimale, bisogna ricordarsi che per utilizzarlo come parametro nel comando, è necessario effettuare una conversione da esadecimale a decimale.
Tutte le variabili nell'area dei dati e del programma, sono "pulite" dopo un MEMSET, ma nessun programma corrente, rimane intatto.
MEMSET inoltre, modifica la copia del BASIC in RAM in modo che se alcune copie di esso sono fatte, esse assumeranno in caso di esecuzione la configurazione della nuova memoria.

Messaggi di errore:

ARG ERROR
L'indirizzo di memoria specificato come "limite alto", non contiene memoria utilizzabile.

OUT OF BOUNDS ERROR
Questo errore si può verificare nei seguenti casi:

1) L'indirizzo è maggiore di 65535.
2) Nel caso ci sia un programma corrente, il limite specificato, è troppo piccolo per contenere il suddetto programma.
3) Se non esiste un programma corrente, il limite specificato, implica l'eliminazione totale dell'area dati/programma.

Vedi anche:

Capitolo **BASIC PERSONALIZZATO**

PSIZE

Comando:

PSIZE

Azione:

Stampa sul terminale la dimensione in blocchi del programma corrente.

Esempio:

PSIZE

Note:

Il comando PSIZE è utile per sapere quanto spazio del dischetto serve per memorizzare il programma corrente, in modo da aiutare l'utente nella creazione di nuovi file programmi, e nell'uso del comando SAVE, nel caso si lavori con il sistema operativo **GDOS**.
Il numero approssimativo di byte che rappresenta appunto la dimensione del programma, si ottiene moltiplicando il valore ottenuto con PSIZE con il numero 256.

Messaggi di errore:

Nessuno.

Vedi anche:

Comando **SAVE**
Apeendice A: **ESECUZIONE DI PROGRAMMI DA GDOS**

REN

Comando:

REN
REN <numero linea>
REN <numero linea>,<indice>

Azione:

L'intero programma corrente è rinumerato.
Alla prima linea del programma viene associat il numero linea specificato nel comando (10 se non è specificato nessun numero linea).
Se viene dato il numero linea, si può associare a questo un indice, perciò tutte le linee seguenti, avranno come numero linea la somma del numero linea precedente più l'indice (come default l'indice è 10).
L'indice deve essere un valore compreso tra 1 a 32767.

Esempi:

```
REN
REN 1000
REN 1000,100
```

Dopo il comando REN 100 il programma A viene cambiato nel relativo programma B:

PROGRAMMA A

```
1     REM LEGGE E SCRIVE DATI
2     REM NELLA LINEA 1000
3     READ Z
10    IF Z<0 THEN 2000
70    PRINT Z \ GOTO 3
1000  DATA 1,2,3,-1
3000  REM LA LINEA 2000 NON È ANCORA STATA SCRITTA.
```

PROGRAMMA B

```
100   REM LEGGE E SCRIVE DATI
110   REM NELLA LINEA 1000
120   READ Z
130   IF Z<0 THEN 2000
140   PRINT Z \ GOTO 120
150   DATA 1,2,3,-1
160   REM LA LINEA 2000 NON È ANCORA STATA SCRITTA
```

Note:

Rinumerare un programma può essere utile ai fini dell'inserzione di nuove linee, infatti utilizzando tale comando si può fare in modo di avere tra un numero linea ed il successivo, un valore costante, rendendo così facile ogni qualsiasi intervento.
Come detto precedentemente, non e possibile inserire l'indice senza inserire prima il numero di linea, ma è possibile fare il contrario, infatti come default, l'indice viene posto a 10.
Da notare che i riferimenti alle linee di programma nelle istruzioni tipo GOTO, GOSUB, RESTORE, cambiano dopo un REN, in modo da riflettere la nuova struttura assunta dai numeri linea.
Inoltre in caso nel programma originario le istruzioni suddette non specificano il numero linea, dopo una operazione di REN, esse rimangono inalterate, cioè restano senza riferimento.

Messaggi di errore:

Se si presenta uno dei seguenti errori, la rinumerazione non è eseguita.

OUT OF BOUNDS ERROR
Questo errore si presenta nei seguenti casi:

1) Il numero linea, specificato nel comando è maggiore di 65535;
2) L'indice è maggiore di 32767, o minore di 1;
3) La combinazione del numero linea di partenza con il valore dell'indice, potrebbe dare nel programma, un valore che superi 65535.

ARG ERROR
Tale errore compare quando il numero linea o il valore dell'indice, non è un numero intero positivo, oppure quando i due valori non sono separati dalla virgola.

Vedi anche:

Comando **AUTO**

RUN

Comando:

RUN <numero linea>

Azione:

RUN è il comando d'esecuzione del programma corrente. In caso si specifica, il numero linea, l'esecuzione parte da quest'ultimo, altrimenti tale operazione, parte dalla prima linea del programma.

Esempi:

RUN
RUN 100

Note:

Prima dell'esecuzione del programa, tutte le variabili, vengono "pulite", cio` sta a significare che tutte le variabili numeriche sono resettate a 0; mentre quelle di tipo stringa ed i vettori, sono distrutte e vengono inizializzate a 0 in caso vengano create durante l'esecuzione del programma corrente. Da notare che qualsiasi variabile settata nel modo diretto prima di RUN, sarà "pulita", per effetto del comando stesso.

Messaggi di errore:

NO PROGRAM ERROR
Il comando RUN è stato dato, prima del caricamento del relativo programma (il programma non è presente in memoria).

LINE NUMBER ERROR
L'opzionale numero linea specificato nel comando RUN non esiste nel programma corrente.

ARG ERROR
L'argomento opzionale, non è un numero linea valido.

Vedi anche:

Comando **CONT**
Istruzione **CHAIN**

SAVE

Comando:

SAVE <nome file>

Azione:

Il programma corrente è salvato permanentemente in un file BASIC sul dischetto.

Esempi:

```
SAVE PROG      {PROG è salvato sul dischetto nel drive #1}
SAVE TEST7,2   {TEST7 è salvato sul dischetto collocato nel drive #2}
```

Note:

Le dimensioni del file, devono essere sufficienti per contenere il programma, in caso contrario, l'operazione di SAVE non ha successo e si verificano degli errori.
È possibile salvare nel file nessun programma (tale operazione si ottiene dando il comando SCR, immediatamente prima di SAVE).
Così facendo si può cancellare qualsiasi programma presente nel file, ottenendo così un file vuoto.
Il comando SAVE non cambia lo spazio dedicato ai dati del programma, perciò e possibile utilizzare il comando CONT dopo un SAVE, oppure dopo una interruzione del programa causata da un control-C o da una istruzione di STOP.

Messaggi di errore:

OUT OF BOUNDS ERROR
Il programma corrente è troppo grande per poter essere salvato, nel file specificato.

FILE ERROR
Il nome file specificato non è accettabile, questo può voler dire che è troppo lungo, che contiene caratteri illegali, o che il numero drive specificato è illegale.
Questo tipo di errore inoltre, può avvenire, quando il dischetto nel drive specificato, è protetto in scrittura.

ARG ERROR
Il file specificato non esiste.

TYPE ERROR
Il file specificato, non è un file programma di tipo BASIC (tipo 2).

HARD DISK ERROR
Vedi comando **CAT**

Vedi anche:

Comando **ASCSAVE**
Comando **LOAD**
Comando **ALOAD**
Manuale del sistema operativo

SCR

Comando:

SCR

Azione:

SCR cancella il programma corrente e ogni variabile dallo spazio di lavoro dell'utente.

Esempio:

SCR

Note:

SCR è il comando che cancella lo spazio di lavoro, in modo da predisporlo a ricevere un nuovo programma.
Solo il programma corrente viene cancellato, perciò eventuali copie presenti nel dischetto non vengono alterate.
Da ricordare che una volta utilizzato tale comando, l'unica maniera di riavere il programma è quella di riscriverlo, percio'si consiglia, prima di utilizzare SCR, di fare una copia sul dischetto del programma, in modo da salvarlo.

Messaggi di errore:

Nessuno.

Vedi anche:

Comando **SAVE**
Comando **ASCSAV**
Comando **ALOAD**
Comando **LOAD**
Comando **DEL**

CHAIN

Istruzione:

CHAIN <nome file-programma>

Azione:

Il programma BASIC, contenuto nel file specificato, è caricato in memoria, e sostituisce il programma corrente, successivamente, viene eseguito, partendo dal primo numero linea.
Il nome del file, deve essere una stringa, rispettante le specifiche trattate nel capitolo **I FILE DATI**.

Esempi:

```
10   CHAIN “PROG,2”
100  CHAIN P$+D$
200  CHAIN “PROG”+N$(X,X)+”,2"
```

Note:

CHAIN, permette di caricare ed eseguire programmi, senza la supervisione dell’utente.
Dopo tale operazione, ogni precedente programma o ogni precedente dato, non esiste più.
Tutti i file aperti, nel programma chiamante, vengono automaticamente chiusi.
La comunicazione tra programmi, può avvenire tramite l’utilizzo di files dati comuni, o tramite l’uso di EXAM e FILL.
Essendo una istruzione diretta, CHAIN può sostituire perfettamente la sequenza di operazioni (LOAD e RUN), che si effettua per il caricamento e l’esecuzione di un programma.
Da ricordare, che il nome del file, deve venir chiuso, tra virgolette, per esempio: CHAIN “PROG”.

Messaggi di errore:

Vedi comando **LOAD**

Vedi anche:

Capitolo **CONCATENAZIONE**
Comando **LOAD**
Comando **RUN**

CLOSE

Istruzione:

CLOSE #<espressione numero file>

Azione:

Chiude il file specificato, in modo da dissociare il numero. Inoltre, salva nel file, gli eventuali valori che pur facendo parte del file, erano stati spostati momentaneamente in RAM.

Esempi:

CLOSE #1
CLOSE #A*2
CLOSE #B7(3)

Note:

Lo svuotamento del buffer che avviene tramite l'istruzione CLOSE, avviene anche quando si presentano le seguenti circostanze:

A) Il puntatore del file, è cambiato per indirizzare una locazione in un altro file block.
B) E' stata eseguita una istruzione END o una istruzione CHAIN
C) Una istruzione di STOP è stata eseguita o una interruzione (CONTROL-C) si è presentata.
D) L'esecuzione del programma si ferma, perchè c'è un errore nel programma.

Comunque, solo le istruzioni CLOSE, END o CHAIN, dissociano il file, dal corrispondente numero file. Durante una interruzione causata da uno STOP, da CONTROL-C o da un errore nel prgramma, ogni file aperto rimane tale, ed è quindi accessibile nel modo diretto.

Messaggi di errore:

FILE ERROR
L'espressione del numero file, non genera un intero tra 0 e 7 o il dischetto è protetto in scrittura.

Vedi anche:

Istruzione **OPEN**
Capitolo **I FILE DATI**

DATA

Istruzione:

DATA <lista di costanti>

Azione:

Le stringhe o le costanti numeriche, incluse nella lista, sono salvate come dati, ai quali si accede in ordine da programma (essi stessi fanno parte del programma).
Nella lista, ogni costante, è separata dalla successiva, da una virgola.

Esempi:

```
1000 DATA “STRING DATA”, “NUMBER IS NEXT”,2
20   DATA 15
115  DATA 2, 7, 25, “HI”, 0
```

Note:

L’istruzione DATA, immaganizza informazioni, direttamente all’interno del programma.
Tali informazioni sono accessibili mediante l’utilizzo dell’istruzione READ.
L’istruzione DATA, non è eseguibile e perciò è ignorata dal BASIC (per questo motivo può essere messa in qualsiasi parte del programma.)

Messaggi di errore:

SYNTAX ERROR
Una costante non valida è stata inserita, per esempio si è inserito una stringa senza le virgolette.
Tale errore avviene quando si esegue una istruzione READ che vuole appunto accedere a quella determinata costante.

Vedi anche:

Istruzione **READ**
Istruzione **RESTORE**

DEF

Istruzione:

DEF <nome funzione>(<lista parametri>)=<espressione>
DEF <nome funzione>(<lista parametri>)

Azione:

La prima dichiarazione definisce una funzione-utente in linea singola di tipo numerico o di tipo stringa. Tale tipo di funzione, restituisce il valore generato dall'espressione a destra del segno di uguale. Il tipo del valore generato deve essere uguale al tipo del nome funzione (numerico, di tipo stringa). La seconda dichiarazione, definisce invece una funzione-utente composta da più linee di programma. In questo caso, il valore è determinato dall'espressione nell'istruzione RETURN, presente nel corpo della definizione. Il tipo dell'espressione, deve essere uguale a quello del nome funzione, come accade per le funzioni-utente in linea singola.
Il nome delle funzioni-utente deve essere formato da due lettere FN seguite da una stringa o da un nome di variabile numerica (per esempio: FNA$, FNQ7, ecc.).

Esempi:

Linea Singola

```
70   DEF FNH(X,Y)=SQRT((X^2)+(Y^2)) \ REM IPOTENUSA
45   DEF FNU$(L$)=CHR$(ASC(L$-32)
```

Linea multipla

```
110  DEF FNQ(A,B,C)
599  DEF FNA7$(A$,A,M)
```

Note:

L'addizione del prefisso FN, permette la distinzione tra nomi funzioni e nomi variabili.
Se durante l'esecuzione di un programma, si incontra l'istruzione DEF, l'esecuzione, riprende, dalla prima istruzione dopo la definizione.
Le dichiarazioni delle funzioni, possono essere scritte, in ogni parte del programma, ma debbono essere inserite, comunque, prima che inizi l'esecuzione.

Messaggi di errore:

FUNCTION DEF ERROR
Una apparente funzione in linea singola, è stata definita impropriamente, oppure è stato fatto un tentativo di definizione di una funzione, senza la dichiarazione di una funzione a più linee.

Vedi anche:

Capitolo **LE FUNZIONI** (in particolare il paragrafo **Funzioni-Utente**)
Istruzione **FNEND**

DIM

Istruzione:

DIM <lista della dichiarazione di vettori o stringhe>

Azione:

Riserva dell'area di memoria per dimensionare le stringhe e gli array dichiarati.

Esempi:

```
10   DIM A$(30),Q(100),Z(5,2)
60   DIM X7(X,Y),X8(X,X,X)
70   DIM C$(100*3)
```

Note:

Un'istruzione DIM, inizializza automaticamente, le variabili dichiarate in essa. Tale istruzione, dopo che è stata eseguita, fa sì che la lunghezza di ogni stringa dichiarata in essa, corrisponda ad una relativa area di memoria, in grado di memorizzare un numero di caratteri, pari alla lunghezza della stringa dichiarata. Tutti gli elementi dei vettori, dichiarati con DIM, sono inizializzati a 0.
Nella dichiarazione di stringhe, l'espressione numerica, chiusa tra parentesi, specifica il numero massimo di caratteri che la variabile stringa può contenere. La stessa cosa avviene nella dichiarazione dei vettori, dove abbiamo appunto vari valori che specificano il massimo indice di ogni dimensione. Negli esempi fatti precedentemente, si può notare che Q è un vettore con un indice massimo di 100, Z è un vettore bidimensionale avente 5 righe e 2 colonne, infine X8 è un array tridimensionale avente come dimensioni i relativi contenuti della variabile X.
Se si utilizzano variabili stringa o array senza averli prima dimensionati si ha un dimensionamento automatico da parte del BASIC stesso, che pone ogni indice a 10.
Da ricordare che in uno stesso programma, non si può dimensionare una variabile o un array più volte. In caso non sia rispettata questa specifica si provoca un errore di dimensionamento: **DIMENSION ERROR**.

Messaggi di errore:

MEMORY FULL ERROR
Non esiste abbastanza area di memoria per la dichiarazione.
Vedi il Capitolo **NOTE DI IMPLEMENTAZIONE** per dettagli sulle locazioni di memoria.

Vedi anche:

Capitolo **LE STRINGHE**
Capitolo **I VETTORI**

END

Istruzione:

END

Azione:

Fine dell'esecuzione programma.

Esempio:

```
10    REM PRINT "2+2=",P
20    END
```

Note:

Tale istruzione è simile a quella di STOP, ma non permette la riesecuzione con CONT e non invia il messaggio sul terminale. END, fa terminare l'esecuzione del programma ed effettua un ritorno al modo diretto.In caso durante l'esecuzione, si giunga all'ultima istruzione del programma, senza avere incontrato un END, essa termina, come se ci fosse un END.

Messaggi di errore:

Nessuno.

Vedi anche:

Istruzione **STOP**

ERRSET

Istruzione:

ERRSET
ERRSET <numero linea>, <variabile numerica>, <variabile numerica>

Azione:

Questa istruzione, provoca un salto al numero linea specificato, in caso di errore, in modo da poter gestire tramite una routine, quest'ultimo.
Il numero linea dove è stato provocato l'errore, viene assegnato alla prima variabile.
Alla seconda variabile invece, viene assegnato il codice del corrispondente errore.
L'esecuzione di una istruzione di ERRSET, senza numero linea e senza nomi variabili, provoca la disabilitazione della ricerca di errori da parte del BASIC.

Esempi:

```
10  ERRSET 1000, L, E
20  ERRSET 570,E(0),E(1)
30  ERRSET
```

Note:

Tale istruzione è molto utile, in quanto rende possibile la gestione degli errori direttamente da programma. Dopo una interruzione, un'altra istruzione ERRSET, deve essere eseguita per fare riprendere il modo di ricerca.
Quando tale ricerca degli errori è disabilitata, un eventuale errore, comporta l'immediata terminazione del programma, e l'invio sul minitor, di un messaggio di errore.
Tale istruzione, non può essere usata nel modo diretto.
Pur rimanendo abilitata, la ricerca degli errori, dopo uno STOP, essa non si attiva fino a chè non si riprende l'esecuzione del programma interrotto. Non tutti gli errori sono gestiti da ERRSET.
Da ricordare, che è possibile trattare l'interruzione del control-C, come un errore, e che la procedure, le funzioni ed i loop FOR-NEXT, rimangono "integre" dopo la gestione dell'errore, è quindi possibile riprendere l'esecuzione del programma, senza perdere alcun dato.

Messaggi di errore:

Vedi istruzione **GOTO**

Vedi anche:

Capitolo **GLI ERRORI E LE SOLUZIONI**
Capitolo **BASIC PERSONALIZZATO**
Capitolo **CONTROL-C**
Capitolo **MESSAGGI DI ERRORE**

EXIT

Istruzione:

EXIT

Azione:

Termina l'esecuzione del loop FOR-NEXT, in cui è inserita, e trasferisce il controllo al numero linea specificato.

Esempo:

```
20    EXIT 95
```

Note:

Tale istruzione è una particlare forma dell'istruzione GOTO, e come quest'ultima, trasferisce, l'esecuzione del prgramma, da un punto ad un'altro. La sola differenza, è che EXIT, deve essere utilizzata esclusivamente, per "saltare", da un punto interno al loop FOR-NEXT, ad uno esterno. Da ricordare, che EXIT, fa terminare solo il loop in cui essa stessa è inserita (vedi il capitolo **IL LOOP FOR-NEXT**, per maggiori informazioni sull'innesto di loop).

Messaggi di errore:

CONTROL STACK ERROR
L'istruzione EXIT, non é stata usata, durante l'esecuzione di un loop FOR-NEXT.

LINE NUMBER ERROR
Vedi istruzione **GOTO**

OUT OF BOUNDS ERROR
Vedi istruzione **GOTO**

Vedi anche:

Capitolo **IL LOOP FOR-NEXT**
Istruzione **NEXT**
Istruzione **FOR**
Istruzione **GOTO**

FILL

Istruzione:

FILL <indirizzo memoria>, <valore del byte>

Azione:

Il byte specificato, è posto nella cella di memoria puntata dall'indirizzo dichiarato.
Il valore che esprime il byte, deve essere generato da una espressione numerica e deve essere un numero intero compreso tra 0 e 255. L'indirizzo deve essere una espressione numerica, in grado di generare, un valore compreso tra 0 e 65535.

Esempi:

```
FILL M+S,0
FILL (2*16^3)+(13+12^2)+(1*16^1)+(3*16^0),16
FILL FNC("2D13"),16
FILL 65535,B
FILL 100,31
```

Note:

Tale istruzione, permette all'utente, di cambiare uno specifico byte in memoria RAM. Con tale operazione è possibile quindi realizzare le seguenti applicazioni:

1) Personalizzare il BASIC
2) Caricare routine in linguaggio macchina
3) Inserire parametri direttamente in memoria
4) Manipolare la memoria video-display, nelle applicazioni grafiche

Da notare, che sia l'indirizzo che il valore del byte, debbono essere espressi in decimale, è compito poi del BASIC, convertirli in binario durante l'esecuzione dell'istruzione FILL.
Quest tipo di BASIC, non accetta l'espressione dei numeri in esadecimale, quindi in caso si voglia utilizzare per forza questa rappresentazione, bisogna utilizzare una funzione di conversione esadecimale-decimale (vedi il capitolo **LE FUNZIONI**).
Il BASIC, nel caso in cui i valori esprimenti il byte e l'indirizzo non risultano essere interi, effettua un troncamento, in modo che la parte frazionale, viene eliminata.
Se dopo l'operazione di "troncamento", il valore del byte e` maggiore di 255, soltanto il resto della divisione tra il byte e 256, viene utilizzato.
Per quanto riguarda l'indirizzo comunque, tale procedimento non è valido.

Attenzione:

FILL è una istruzione molto potente, in quanto rende accessibile al programmatore tutta la memoria, e quindi, serve da parte di quest'ultimo, una certa cautela, in quanto potrebbe provocare errori disastrosi, cambiando per esempio della memoria codice del BASIC ecc.

Messaggi di errore:

OUT OF BOUNDS ERROR
Si può verificare nei seguenti casi:

1) Il valore esprimente il byte o l'indirizzo, sono numeri minori di 0
2) L'indirizzo è maggiore di 65535

Vedi anche:

Capitolo **BASIC PERSONALIZZATO**
Capitolo **LE FUNZIONI**
Capitolo **PROCEDURE IN LINGUAGGIO MACCHINA**

FNEND

Istruzione:

FNEND

Azione:

Questa istruzione, dichiara la fine della definizione di una funzione-utente composta da più linee.

Esempio:

```
10  DEF FNF(X) \ REM CALCOLO DEL FATTORIALE
15  X=INT(ABS(X)) \ REM ELIMINAZIONE DEGLI ARGOMENTI NON ESATTI
20  IF X=0 OR X=1 THEN RETURN 1 ELSE RETURN FNF(X-1)*X
30  FNEND
```

Note:

Tale istruzione non deve essere confusa, con l'istruzione RETURN, usata per dichiarare la fine di una funzione-utente a più linee.
L'istruzione FNEND non può apparire sulla stessa linea della corrispondente istruzione DEF.

Messaggi di errore:

CONTROL STACK ERROR
Questa istruzione non è stata creata per essere eseguita.
Tale errore, indica che è stata eseguita, appunto, una istruzione FNEND.

FUNCTION DEF ERROR
Le istruzioni FNEND e DEF, sono presenti sulla stessa linea, o esiste una istruzione FNEND senza la corrispondente istruzione DEF.

Vedi anche:

Capitolo **LE FUNZIONI** (in particolale il paragrafo **Funzioni-Utente**)
Istruzione **DEF**
Istruzione **RETURN**

FOR

Istruzione:

FOR <variabile di controllo>=<valore iniziale> TO <valore limite>
FOR <variabile di controllo>=<valore iniziale> TO <valore limite>
STEP <valore passo>

Azione:

Inizio di un loop di FOR-NEXT.

Esempi:

```
15   FOR J=1 TO 10 \ REM CAUSA 10 ITERAZIONI
25   FOR Q(7)=3 TO 1 \ REM NON FA NESSUN LOOP
40   FOR A=B*7 TO SQRT(X)
50   FOR X=.1 TO 1.3 STEP .1
90   FOR J=3 TO 1 STEP -1
70   FOR I=10+J TO 100+J STEP D(X)
```

Note:

Per una completa descrizione di tale loop vedi il capitolo **IL LOOP FOR-NEXT**.
I valori che esprimono il numero iniziale, il limite ed il passo devono essere generati, da una espressione numerica.
Il corpo del loop, non viene eseguito se il valore iniziale è maggiore del limite ed il passo è negativo, inoltre se il valore iniziale è minore del limite ed il passo è negativo.

Messaggi di errore:

MISSING NEXT ERROR
Il BASIC non ha trovato, l'istruzione NEXT, associata al relativo FOR.

Vedi anche:

Capitolo **IL LOOP FOR-NEXT**
Istruzione **NEXT**
Istruzione **EXIT**

GOSUB

Istruzione:

GOSUB <numero linea>

Azione:

La locazione dell'istruzione successiva al GOSUB, è salvata (viene ricordata) dal BASIC, e l'esecuzione salta, alla linea specificata.
Il GOSUB, è utilizzato, per eseguire una sequenza di istruzioni, chiamata procedura.
L'esecuzione viene riportata alla locazione "salvata", se viene eseguita una istruzione di RETURN.

Esempio:

```
20  PRINT "PRONTO A CHIAMARE LA SUBROUTINE"
30  GOSUB 1000
40  PRINT "SONO RITORNATO AL MAIN"
50  END
```

Questo esempio, immagina che ci sia una subroutine alla linea 1000, che scriva sul terminale, "SONO NELLA SUBROUTINE".
L'esecuzione quindi genera il seguente risultato:

```
PRONTO A CHIAMARE LA SUBROUTINE
SONO NELLA SUBROUTINE
SONO RITORNATO AL MAIN
```

Note:

È possibile, chiamare una subroutine, all'interno di un'altra, è possibile cioè, effettuare una operazione di innesto tra subroutine (ricordare che tale operazione, necessita memoria per salvare le locazioni di ritorno, quindi bisogna stare attenti, per evitare lo "sfondamento" di quest'ultima).

Messaggi di errore:

LINE NUMBER ERROR
Vedi istruzione **GOTO**

OUT OF BOUNDS ERROR
Vedi istruzione **GOTO**

Vedi anche:

Istruzione **RETURN**
Istruzione **GOTO**
Capitolo **LE PROCEDURE**

GOTO

Istruzione:

GOTO <numero linea>

Azione:

L'istruzione GOTO, causa una salto al numero linea specificato, perciò l'esecuzione sequenziale del programma, riprende da quella locazione.

Esempi:

```
10  PRINT "PRIMA STAMPA"
20  GOTO 40
30  PRINT "MESSAGGIO CHE NON SARA` MAI STAMPATO"
35  PRINT "TERZA STAMPA"
37  END
40  PRINT "SECONDA STAMPA"
50  GOTO 35
```

Note:

Non devono esserci spazi tra GO e TO in quanto, GOTO è una singola parola BASIC.
Il numero linea deve essere un numero intero e costante (non può essere una varianile o una espressione).

Messaggi di errore:

LINE NUMBER ERROR
Il numero linea specificato, non esiste nel programma.

OUT OF BOUNDS ERROR
Il numero linea specificato, è maggiore di 65535 (tale errore compare immediatamente dopo l'inserimento della linea errata e non durante l'esecuzione).

Vedi anche:

Capitolo **FLUSSO DI ESECUZIONE E DI CONTROLLO**
Istruzione **EXIT**
Istruzione **ON**

IF

Istruzione:

IF <espressione logica> THEN <istruzione>
IF <espressione logica> THEN <istruzione> ELSE <istruzione>

Azione:

Quando l'espressione logica è vera, si esegue l'istruzione dopo il THEN, se invece risulta essere falsa, viene eseguita l'istruzione dopo l'ELSE.
In caso si ometta l'ELSE e la condizione risulta essere falsa, viene eseguita, l'istruzione successiva all'IF (in ordine sequenziale).
L'inserimento di un numero linea, dopo il THEN o l'ELSE, comporta un salto alla locazione specificata (equivale a GOTO <numero linea>).

Esempi:

```
10   IF X=5 THEN 1000
100  IF A$="MARIO" THEN PRINT "E' BELLO" ELSE PRINT "E' BRUTTO"
75   IF Q(7)<>3 AND W THEN GOSUB 110 ELSE LET X=15
230  IF A$="E'" THEN IF B$="QUI" THEN PRINT "SI' ? ",
999  IF Z THEN END
```

Note:

Solo una delle due parti THEN o ELSE è eseguita, quando si interpreta una istruzione IF.
Nei due blocchi di esecuzione è possibile inserire altri IF, in modo da creare un "innesto" di IF, logicamente tale catena, finirà quando si raggiungerà il "fine linea".

Messaggi di errore:

L'istruzione IF di per se stessa, non causa errori.
La generazione di errori dipende quindi dalla scrittura errata della espressione logica o dalle istruzioni presenti dopo le "parole chiavi" THEN e ELSE.
Per tale motivo, è opportuno consultare, in caso di errorre, il capitolo che descrive la relativa istruzione che ha generato l'errore.

Vedi anche:

Capitolo **I NUMERI** (in particolare il paragrafo **Operatori**)
Istruzione **GOTO**

INPUT

Istruzione:

INPUT <lista di variabili>
INPUT <costante di tipo stringa>, <lista variabili>
INPUT #<espressione dispositivo>, <lista variabili>
INPUT #<espr. dispositivo>, <costante stringa>, <list. variab.>

Azione:

Si richiede e si attende l'immissione di una stringa o di un dato numerico, utilizzando il dispositivo dichiarato. Se non è immessa l'espressione indicante il dispositivo, si assume come default il dispositivo #0 (consolle-terminale).
L'espressione suddetta, deve generare un numero compreso tra 0 e 7.
I dati immessi dall'utente, sono assegnati alle rispettive variabili specificate nella <lista variabili>, presente nell'istruzione. Se nessuna costante di tipo stringa è fornita, come prompt (carattere segnalante che si è in una condizione di attesa, si sta aspettando l'immissione del dato), viene visualizzato il punto interrogativo (?).
Da ricordare, che il dato viene accettato, solo dopo che si è premuto il <CR> (carriage return).

Esempi:

```
10   INPUT A,B,Q$
70   INPUT "YOUR NAME: ",N$
35   INPUT #3,X,Y
30   INPUT #X,"COMMAND: ",C$(5,9)
19   INPUT "",X \ REM NESSUN PROMPT E` DATO.
```

Note:

Questa istruzione, non puo essere data nel modo diretto.
Dopo l'esecuzione di un INPUT, si rimane in una condizione di attesa, fino a che non viene premuto un <CR>.
Una stringa, deve essere inserita senza le virgolette (""), se non si rispetta questa specifica, la stringa inserita, risultera` essere composta anche dalle virgolette.
In caso un INPUT richieda l'inserimento, di più dati oltre a digitarli separatamente:

<dato> <CR>, <dato> <CR>, ecc.

è possibile effettuare, un inserimento simultaneo, tale operazione si ottiene separando ogni dato con una virgola, in questo modo:

<dato>, <dato>, <dato>,..... <CR>

Da notare, che tale metodo, non è utilizzabile qualora si debbano inserire più stringhe consecutivamente su una stessa linea, in questo caso, infatti, bisogna far uso dell'istruzione INPUT1 (vedi istruzione **INPUT1**).
Per chiarire meglio il funzionamento di questa istruzione sono riportati in seguito alcuni inserimenti effettuati, in risposta all'istruzione di INPUT presente nel paragrafo **Esempi** prima trattato.
L'esecuzione del primo esempio, comporta la comparsa sul terminale, del punto interrogativo (?) che indica l'ingresso in una situazione di attesa. Inserendo poi 2, 3, ciao <CR> si otterranno delle specifiche assegnazioni alle variabili presenti nell'istruzione, e più specificatamente, A conterrà 2, B conterrà 3 e Q$ infine, la stringa "ciao".
Se si preme <CR> senza inserire alcun dato, si possono verificare due condizioni:

1) Compare un messaggio di errore (**INPUT ERROR**). Si ha tale condizione quando si è digitato il <CR> senza avere inserito niente prima, mentre il sistema aspettava un dato numerico.
2) Viene accettato il "no input". In altre parole, si e` inserita una stringa vuota mentre il sistema aspettava l'inserimento di una stringa.

Concludendo quindi, dipende tutto dal tipo di variabile presente nella lista al momento di quella determinata assegnazione: se tale variabile è di tipo stringa, (esistendo la stringa vuota), l'inserimento del solo <CR> è accettato, altrimenti si crea una situazione di errore.
Da ricordare, che l'editor di linea, permette la modificazione del dato inserito, naturalmente prima dell'invio del <CR>.
Quando si inseriscono meno dati, rispetto a quelli che si attende il BASIC, compare sullo schermo il doppio punto interrogativo (??) indicante appunto, che si sta aspettando l'invio di altri dati. Tale operazione, naturalmente, si ripete fino a che tutte le variabili presenti nella lista, non hanno ricevuto, la rispettiva assegnazione.
Da ricordare che l'istruzione INPUT, non equivale alla funzione interna INP.

Messaggi di errore:

LENGTH ERROR
La linea dei dati in ingresso è troppo lunga.

INPUT ERROR —PLEASE RETYPE
È stato inserito un valore non-numerico, mentre si attendeva un valore numerico.
Viene dato all'utente la possibilità di riinserire il dato.
In caso ci sono più dati, bisogna riinserirli tutti, nella stessa sequenza.

Vedi anche:

Capitolo **I NUMERI**
Capitolo **CONTROL-C**
Capitolo **LE FUNZIONI** (in particolare la funzione **INP**)
Istruzione **INPUT1**

INPUT1

Istruzione:

INPUT1 <lista di variabili>
INPUT1 <costante di tipo stringa>, <lista variabili>
INPUT1 #<espressione dispositivo>, <lista variabili>
INPUT1 #<espr. disp.>, <costante stringa>, <lista variabili>

Azione:

Il funzionamento è lo stesso dell'istruzione INPUT, eccetto che, quando l'utente preme il <CR> per comunicare la fine della linea di input, il <CR> non viene visualizzato sul terminale, e quindi ingressi o risultati si avranno sulla stessa linea.

Esempi:

```
50    INPUT1 Z,W,B7,A(3)
25    INPUT1 #D(Q), "GUESS? ",G
```

Note:

Vedi istruzione **INPUT**.

Messaggi di errore:

Vedi istruzione **INPUT**.

LET

Istruzione:

LET <variabile numerica>=<espressione numerica>
LET <variabile stringa o sottostringa>=<stringa>
<variabile numerica>=<espressione numerica>
<variabile stringa o sottostringa>=<stringa>

Azione:

Il valore dell'espressione è assegnata alla relativa variabile.
L'istruzione LET è opzionale e perciò può essere omessa.

Esempi:

```
10   X=X+1
50   LET A(X)=6
35   LET Q=SQRT(X)+Y
20   B$="CIAO"
61   M$(2,11)=FNN$("415-549-0858")
150  LET Z$=STR$(Q)+Z$(1,2)+"BOX"
```

Note:

Il BASIC, permette un solo assegnamento, per ogni istruzione LET.
Se si intende comunque disporre su una stessa linea più assegnazioni, bisogna interporre tra una e l'altra il seguente simbolo "\", come mostrato nell'esempio:
10 A=0\B=0\C=0
Da ricordare che a sinistra dell'uguale, deve esserci solo un nome di una variabile singola, in quanto è impossibile effettuare una sola assegnazione per "riempire" tutti gli elementi di un array; ciascuno di essi, deve avere una assegnazione individuale.

Messaggi di errore:

TYPE ERROR
Il tipo di espressione è diverso dal tipo della variabile (per esempio è sbagliato assegnare un valore di tipo stringa a una variabile numerica o un numero ad una variabile stringa).

Vedi anche:

Capitolo **I NUMERI**
Capitolo **LE STRINGHE**
Capitolo **I VETTORI**

LINE

Istruzione:

LINE <espressione numerica>
LINE #<espressione dispositivo>, <espressione numerica>

Azione:

a lunghezza della linea, per il dispositivo di I/O specificato, viene cambiata ed assume il valore della espressione numerica, da notare che tale valore deve essere un numero intero compreso tra 10 e 132. In caso non venga specificato il dispositivo, come default, viene assunto #0 (consolle terminale).

Esempi:

```
100   LINE 132
70    LINE L(X)+40
250   LINE #3,B
900   LINE #D(Q), 64
```

Note:

Fissare la lunghezza della linea di I/O è una necessità, in quanto NSB8, deve avere una traccia della posizione corrente di scrittura sul terminale o sul monitor, per fare operare la funzione di TAB correttamente.
L'uso della istruzione LINE, inoltre, rende possibile all'utente, la comunicazione con con particolari terminali aventi particolari specifiche. Per esempio, molti video-display, prevedono una scrittura per linea, di 32 o 64 caratteri, mentre altri terminali hanno 80 caratteri per linea. Non dimentichiamo, inoltre, le stampanti che variano da 40 a 132 caratteri per linea.
Se una linea di informazione in output è maggiore della lunghezza linea settata per il terminale, si ha una "spaccatura" della linea suddetta, in modo che i caratteri che non stanno nella prima riga, vengano passati alla seguente (un CARRIGE RETURN è generato automaticamente da NSB8, per fare avanzare il resto della riga alla successiva linea).
Se viene effettuato un tentativo di ricezione che assume più caratteri di quelli presenti su una linea, un messaggio di errore del tipo "**LENGHT ERROR**" viene inviato sul monitor.
LINE deve essere usata come una istruzione diretta.
La lunghezza della linea settata con l'istruzione LINE, rimane attiva fino a che non si termina una sessione di lavoro con il BASIC. Per esempio, una lunghezza linea di 132, rimane settata, fino a che il programma in cui si è fatta tale operazione, finisce. NSB8, inizializza la lunghezza della linea del dispositivo #0 (consolle e terminale), a 80 caratteri, mentre pone a 80 gli altri 7 possibili dispositivi di I/O. Tali inizializzazioni, comunque, possono essere variate, utilizzando alcune procedure, descritte nel capitolo **BASIC PERSONALIZZATO**.

Messaggi di errore:

OUT OF BOUNDS ERROR
Il numero del dispositivo, o la lunghezza linea specificata nell'istruzione di LINE, è fuori dal rispettivo RANGE.

Vedi anche:

Capitolo **LE FUNZIONI** (in particolare la funzione **TAB**)
Istruzione **INPUT**
Istruzione **INPUT1**
Capitolo **BASIC PERSONALIZZATO**

NEXT

Istruzione:

NEXT
NEXT <variabile numerica>

Azione:

Termina l'esecuzione del loop FOR.
Per maggiori informazioni, leggere il capitolo **IL LOOP FOR-NEXT**.
Se la variabile numerica, è associata a tale istruzione, il BASIC effettua un controllo per verificare se tale variabile, corrisponde alla variabile di controllo nell'istruzione FOR.

Esempi:

```
NEXT
NEXT Q
NEXT A(1)
```

Note:

L'omissione della variabile in NEXT, risulta utile per la velocizzazione dell'esecuzione del prgramma. Dopo la fine del loop, tale variabile, contiene un valore che risulta essere uguale a quello del limite, incrementato di uno. Il seguente esempio, mostra meglio, tale concetto:

```
10   FOR K=1 TO 5 STEP 2
20   NEXT K
30   PRINT K
```

L'esecuzione produce:

```
7
```

Da ricordare, che NEXT, non può essere usato, in una istruzione IF come corpo di THEN o ELSE.

Messaggi di errore:

CONTROLL STACK ERROR
Si è eseguita una istruzione NEXT, senza aver aperto precedentemente il loop relativo oppure, la variabile specificata in NEXT non è uguale a quella dichiarata in FOR.
Usualmente ciò accade quando si effettua un innesto tra loop non corretto.

Vedi anche:

Istruzione **FOR**
Capitolo **IL LOOP FOR-NEXT**

ON

Istruzione:

ON <espressione numerica> GOTO <lista di numeri linea>

Azione:

A seconda della espressione numerica, si effettua un salto a un relativo numero linea specificato nella lista. L'esecuzione quindi, riprenderà da tale locazione.

Esempi:

```
10   ON C GOTO 100, 200, 300, 400
105  ON X-10 GOTO 10, 20, 30, 40, 50, 60, 70
```

Note:

L'espressione numerica, deve generare un numero, maggiore o uguale ad uno.
Su una stessa linea di questa istruzione, possono essere inseriti più numeri linea, fino a quando non si raggiunge la fine della medesima.
L'esecuzione passa alla linea specificata dal primo numero linea, se l'espressione genera 1, alla seconda se viene generato 2 ecc.
Per esempio, se nell'istruzione 10 prima scritta, si ottiene C=3, l'esecuzione passerà alla linea 300.
Per quanto detto precedentemente, una istruzione ON...GOTO con N numeri linea nella sua lista, potrà generare valori interi, compresi tra 1 e N.

Messaggi di errore:

SYNTAX ERROR
Può avvenire quando l'espressione genere un numero minore di 1 o maggiore del numero di N.

TYPE ERROR
L'espressione specificata, non è una espressione numerica.

LINE NUMBER ERROR
Vedi istruzione **GOTO**

OUT OF BOUNDS ERROR
Vedi istruzione **GOTO**

Vedi anche:

Istruzione **GOTO**

OPEN

Istruzione:

OPEN #<espressione numero file>,<nome file>
OPEN #<espr. numero file>,<nome file>,<spazio variabile>
OPEN #<espr. numero file>,%<espressione del tipo>,<nome file>
OPEN #<espr. numero file>,%<espressione del tipo>,<nome file>,<spazio variabile>

Azione:

Al nome file specificato, è assegnato uno specifico numero file. Finchè il file non è chiuso ogni riferimento al file deve avvenire utilizzando il numero associatogli. L'espressione del numero associato, deve generare un valore intero, compreso tra 0 e 7. Se il tipo dell'espressione è omesso, come default, viene assunto il tipo 3 (file dati). L'operazione di apertura, ha successo, se e sol se, il file è del tipo specificato. L'espressione indicante il tipo, deve generare un numero intero compreso tra 0 e 127. Il nome file deve essere una espressione di tipo stringa rispettante le specifiche trattate nel capitolo **I FILE DATI**. Se viene utilizzata la variabile per lo spazio, lo spazio espresso in blocchi, del file aperto, sarà assegnato alla variabile numerica specificata.

Esempi:

```
OPEN #1,"DATA"
OPEN #7%4,"CUSTLIST"+D$
OPEN #F%T,F$,S
```

Note:

Un numero associato ad un file, non può essere utilizzato per rappresentare un'altro file, prima che si sia chiuso (CLOSE), il precedente.
I seguenti comandi, chiudono tutti i file aperti: RUN, END, SCR, LOAD, CHAIN

Messaggi di errore:

TYPE ERROR
Il file non è del tipo specificato nell'istruzione OPEN (tipo 3, se nessun tipo è specificato).

FILE ERROR
Può essere causato da queste tre condizioni:

1) Il numero del file, già esiste ed è associato ad un altro file
2) La sintassi del nome file non è corretta
3) Il file specificato, non esiste sul dischetto nel drive

OUT OF BOUNDS ERROR
Il numero file, o il valore che esprime il tipo, è esterno al corrispondente RANGE.

Vedi anche:

Capitolo **I FILE DATI**
Istruzione **CLOSE**

OUT

Istruzione:

OUT <numero del port>, <valore del byte>

Azione:

Il byte specificato, è inviato al dichiarato port di output del 8080 o Z80.
Entrambi i parametri, devono essere generati da espressioni numeriche, e devono essere numeri interi compresi tra 0 e 255.

Esempi:

OUT 2,65
OUT P,B
OUT P7+1,ASC("0")

Note:

Sia il numero indicante il port, che quello indicante il byte, devono essere espressi in decimale.
In molti casi, è utile sapere se un determinato port di output è pronto a ricevere un dato, tale condizione, si può testare, grazie alla funzione interna INP.
Un programma quindi, prima di spedire un dato ad un port, deve prima verificare che il suddetto port sia pronto per la ricezione, e se non lo è, deve entrare in una condizione di attesa.
Per concludere, il programmatore ha la completa responsabilità del dato da inviare, in poche parole, deve sempre accertarsi che il port sia pronto, in quanto OUT, non effettua tale operazione implicitamente.

Messaggi di errore:

OUT OF BOUNDS ERROR
Uno dei due parametri, è esterno all'intervallo 0-255.

Vedi anche:

Capitolo **LE FUNZIONI** (in paricolare la funzione **INP**)
Istruzione **FILL**

PRINT

Istruzione:

PRINT
PRINT <lista di stringhe e/o di espressioni numeriche>
PRINT #<espressione dispositivo>
PRINT #<esp. disp.>, <lista di espressioni numeriche/stringhe>

Azione:

I dati indicati nella lista, sono inviati al dispositivo di output specificato. Dopo la stampa dei dati, il cursore si posiziona all'inizio della successiva riga. Se non esiste una lista di output, viene stampata solo una linea vuota. Se non è specificato il dispositivo di OUTPUT, la stampa viene inviata per default al dispositivo #0 (terminale).
L'espressione che identifica il dispositivo deve fornire un valore compreso tra 0 e 7.
L'indicazione del formato di stampa, può essere messa nella lista di output (per ulteriori informazioni, vedi il capitolo **FORMATO DI STAMPA**). Tutti gli elementi, nella lista, devono essere separati da una virgola. Gli elementi di una stessa lista, saranno stampati su una stessa linea.
Le informazioni che non potranno essere messe in una stessa linea, saranno stampate nella successiva linea. Se si mette una virgola dopo la lista di output, il cursore non viene spostato alla successiva linea, così che una eventuale successiva scrittura pone i caratteri sulla stessa linea .

Esempi:

```
PRINT
PRINT "LA RISPOSTA E`: ",
PRINT A,B,C,A7
PRINT #D
PRINT #Q,A,B,"HELLO",C(3),Q$
```

Il seguente programma mostra l'istruzione di PRINT in esecuzione:

```
10    A=3
20    B=4
30    PRINT "A EQUALS",A,
40    PRINT "   B EQUALS",B
50    END
```

Il risultato dell'esecuzione è:

```
A EQUALS 3    B EQUALS 4
```

Note:

Il punto esclamativo (!) può essere usato come abbreviazione per la parola PRINT, perciò PRINT "STRING" può essere scritto anche come !"STRING".
Tale opportunità è molto utile quando si lavora in modo diretto.
Nell'istruzione PRINT è possibile utilizzare, per una corretta "spaziatura" tra una stampa e l'altra, la funzione TAB, di cui però una opportuna spiegazione, viene data nel capitolo **LE FUNZIONI**.

Messaggi di errore:

Nessuno.

Vedi anche:

Capitolo **FORMATO DI STAMPA**
Capitolo **DISPOSITIVI DI I/O MULTIPLI**
Istruzione **LINE**

READ#

Istruzione:

READ #<espressione numero file>, <lista variabili>
READ #<espr. numero file> %<indirizzo random>, <lista variabili>

Azione:

Ad ogni variabile nella lista, viene assegnato un rispettivo valore (in modo sequenziale) presente nel file specificato. La lettura, comincia ad un specifico punto, se il file è di tipo random.
L'indirizzo che specifica appunto, da quale posizione, si vuole cominciare la lettura, è dichiarato come segue:

%<espressione>

dove espressione deve generare un numero intero, compreso tra 0 e l'indirizzo dell'ultimo byte all'interno del file. Il numero file, deve essere generato da una espressione numerica, deve essere un valore intero ed infine deve essere compreso tra 0 e 7.
Le variabili nella lista, possono avere il prefisso "&", in questo caso, il BASIC, assegna loro, i corrispondenti valori decimali di ciascun byte letto nel file.

Esempi:

```
READ #2, A,B,C
READ #3,Q,%B7,A$
READ #F%L,&X,&Y,&Z
READ #0%FNL(I)+3,R8,Z$,R9
```

Note:

Il BASIC, genera un puntatore, per ogni file aperto. All'atto dell'apertura di un file, il puntatore è settato all'inizio del file stesso, ossia al primo byte. Ogni volta che viene fatta una operazione di lettura, il puntatore, viene incrementato, in modo da indirizzare al primo byte del successivo valore nel file. L'utilizzo dell'espressione di indirizzamento random, comporta il settaggio del puntatore ad una specifica locazione.

Messaggi di errore:

TYPE ERROR
È stato assegnato ad una variabile, un valore illegale (il tipo del dato ed il tipo della variabile, non coincidono). Tale errore può venire provocato, inoltre, se si effettua un tentativo di lettura dell'ENDMARK (fine file), o se il puntatore risulta essere settato in modo che venga indirizzata una parte intermedia di una stringa o di un valore numerico all'interno del file.

OUT OF BOUNDS ERROR
Avviene quando almen una di queste condizioni si verifica:

1) L'indirizzo di accesso random è minore di 0 o è maggiore del (spazio occupato dal file in blocchi)*256-1.
2) Il numero file è minore di 0 o maggiore di 7

Vedi anche:

Capitolo **I FILE DATI**
Istruzione **WRITE#**

READ

Istruzione:

READ <lista variabili>

Azione:

Ad ogni variabile della lista viene assegnato un relativo dato presente in una istruzione DATA.

Esempio:

```
10   READ A,B
20   READ C(3),Q$
30   PRINT A,B, C(3), Q$
40   READ X
50   PRINT X
60   DATA 1,2,3," HI",4
```

Effettuando un RUN del programma si ottiene:

```
1 2 3 HI
4
```

Note:

La variabile ed il corrispondente elemento devono essere dello stesso tipo (interi, stringa ecc.).
Il BASIC, utilizza uno speciale puntatore interno, per sapere la posizione del corrente elemento in una istruzione DATA. Tale puntatore, all'atto dell'esecuzione, viene inizializzato e punta (in caso ci siano istruzioni DATA), al primo elemento della prima istruzione di DATA. In caso non ci siano istruzioni del genere, esso punta invece, al "END OF DATA". Dopo una lettura (READ), il puntatore punta al successivo elemento della lista, se si è giunti a fine lista, allora viene ricercata la successiva istruzione DATA, e si sposta il puntatore al primo elemento della nuova istruzione. Tale processo, si ripete fino a chè non esistono più DATA, in tal caso il puntatore, viene settato a "END OF DATA". Raggiunta questa condizione, una successiva lettura (READ), causa un errore nel programma. A differenza dell'istruzione RESTORE, i dati letti con READ, sono prelevati una sola volta.

Messaggi di errore:

READ ERROR
L'elemento, non è dello stesso tipo della variabile a cui deve essere assegnato o si è giunti in una condizione di "END OF DATA".

Vedi anche:

Istruzione **DATA**
Istruzione **RESTORE**

REM

Istruzione:

REM <linea opzionale di testo>

Azione:

Nessuna.
Questa istruzione è ignorata da NSB8.

Esempi:

```
10   REM QUESTA ISTRUZIONE, SERVE PER INSERIRE
20   REM COMMENTI NEL PROGRAMMA.
30   REM PER ESEMPIO —
35   REM
40   N=G-W\REM —
```

Note:

Come si può notare dall'esempio, tale istruzione può comparire nella stessa linea di un'altra, ma deve essere necessariamente l'ultima, in quanto qualsiasi cosa compaia dopo REM, è ignorata dal BASIC. Come per le altre istruzioni, i caratteri ":", ";", "[", e "]" sono trasformati rispettivamente in "\", ",", "(", e ")".

Messaggi di errore:

Nessuno.

RESTORE

Istruzione:

RESTORE
RESTORE <numero linea>

Azione:

Il puntatore al successivo elemento, è spostato, al primo elemento della prima istruzione READ, presente nel testo del programma. In caso viene fornito il numero linea, il puntatore viene spostato, in modo che punti al primo elemento della linea specificata.

Esempio:

```
10   READ A \ PRINT A
20   RESTORE
30   READ A \ PRINT A
40   RESTORE 70
50   READ A \ PRINT A
60   DATA 1,2,3,4
70   DATA 5,6,7,8
```

L'esecuzione dà come risultato:

```
1
1
5
```

Note:

RESTORE, permette di leggere i dati nelle istruzioni DATA, più volte.
In questo modo è possibile riutilizzare i dati (nell'esempio, linee 10, 20, 30) o effettuare letture in modo non sequenziale (linee 40, 50).
Il comando RUN causa una automatica operazione di RESTORE.

Messaggi di errore:

Vedi istruzione **GOTO**

Vedi anche:

Istruzione **READ**
Istruzione **DATA**

RETURN

Istruzione:

RETURN

Azione:

È l'istruzione di chiusura della subroutine e comporta il ripristino della esecuzione alla locazione successiva alla chiamata (GOSUB).

Esempio:

```
1099 RETURN
```

Note:

Ci sono due versioni dell'istruzione di RETURN, la prima è utilizzata per chiamare le sottoprocedure, mentre la seconda, serve per chiamare le funzioni (tale istruzione sarà trattata nel prossimo capitolo).

Messaggi di errore:

CONTROL STACK ERROR
È stata eseguita una istruzione RETURN, mentre nessun GOSUB era in attivo.

Vedi anche:

Istruzione **GOSUB**
Capitolo **LE PROCEDURE**

RETURN

Istruzione:

RETURN <espressione numerica o di tipo stringa>

Azione:

Viene fatta terminare la valutazione della funzione-utente a linea multipla.
Il valore della funzione, diventa il valore dell'espressione nell'istruzione RETURN.

Esempi:

```
10  RETURN F$+",2"
20  RETURN A
30  RETURN X+3
40  RETURN "COSTANTE"
```

Note:

Da non confondere questa forma dell'istruzione RETURN, con quella utilizzata per le procedure.
Una eventuale scrittura impropria, provoca un **SYNTAX ERROR**.
I valori restituiti da una funzione multi-linea, devono essere dello stesso tipo del nome funzione, quindi, una funzione di tipo stringa, non può restituire un valore numerico e viceversa.

Messaggi di errore:

SYNTAX ERROR
Il valore di ritorno della funzione, non è dello stesso tipo del corrispondente nome funzione.

Vedi anche:

Capitolo **LE FUNZIONI** (in particolare il paragrafo **Funzioni-Utente**)
Istruzione **FNEND**
Istruzione **DEF**

STOP

Istruzione:

STOP

Azione:

Questa istruzione, causa lo stop dell'esecuzione.
Un messaggio indicante il punto in cui si è fermata l'esecuzione, ed inviato al terminale.

Esempio:

```
20  STOP
```

Note:

Questa istruzione, viene di norma generalmente usata, durante lo svolgimento del programma.
In questo modo è possibile creare dei break-point software che riportano il computer ad operare in modo diretto, in questa condizione, naturalmente, è possibile visualizzare o cambiare il contenuto delle variabili (PRINT e LET). Per riprendere l'esecuzione dallo stesso punto, si deve utilizzare il comando CONT (da ricordare, che le variazioni sulle variabli fatte nel modo diretto, rimangono valide, perciò esse conterranno i nuovi valori).
Se si intende utilizzare CONT, è necessario non modificare il testo del programma durante l'interruzione (inserimento o eliminazione di linee di programma), in quanto potrebbero verificarsi degli errori.

Messaggi di errore:

Nessuno.

Vedi anche:

Istruzione **END**
Comando **CONT**
Capitolo **CONTROL-C**

WRITE

Istruzione:

WRITE #<numero file>, <lista espressioni>
WRITE #<numero file> %<indirizzo random>, <lista espressioni>

Azione:

Ogni valore della lista, è inserito nel file specificato. Se la lista è composta da più valori, l'inserimento di quest'ultimi, avviene in modo sequenziale cioè uno dopo l'altro. Effettuato l'inserimento di tutti i dati, il BASIC, inserisce un ENDMARK, alla fine del file, ossia dopo l'ultimo dato inserito. Da notare, che dopo ogni scrittura, il puntatore, indirizza all'ENDMARK, perciò bisogna fare attenzione, in caso si voglia effettuare una lettura (l'ENDMARK non si può leggere, ogni tentativo di effettuare tale operazione, provoca un errore).
E' possibile non inserire l'ENDMARK, dopo ogni scrittura, opportunità molto utile se si deve gestire un file random, per fare ciò bisogna inserire la parola riservata **NONENDMARK** come ultimo elemento nell'istruzione WRITE. Il numero file e l'indirizzo random, devono essere generati da espressioni numeriche, inoltre il primo, deve essere un valore compreso tra 0 e 7, mentre il secondo deve essere compreso tra 0 e l'indirizzo dell'ultimo byte del file.
Ogni espressione numerica della lista, può avere come prefisso l'ampersand (&), questo simbolo, segnala al BASIC, che la scrittura nel file, deve avvenire byte per byte.

Esempi:

```
90  WRITE #1,A,B,C$
75  WRITE #F, "HI THERE",Q,X7(B),NONENDMARK
85  WRITE #0%P,R$
33  WRITE #X, &B1,&B2,&1
20  WRITE #3%Z(M), &E, NONENDMARK
49  WRITE #2%(R-1)*S,X$,Y$,Z$
```

Note:

Ogni volta che si utilizza il simbolo "&" per effettuare la scrittura di un solo byte, viene anche inserito l'ENDMARK, perecio` per esempio, WRITE #1,&B, effettua la scrittura di due byte, il primoo rappresenta B, mentre il secondo, l'ENDMARK.
Quando si desidera scrivere solo un byte, quindi, nell'istruzione WRITE, deve comparire anche la parola riservata NONENDMARK.

Messaggi di errore:

FILE ERROR
Il dischetto contenente il file specificato, è protetto in scrittura

OUT OF BOUNDS ERROR
Accade, quando almeno una di queste condizioni si verifica:

1) L'indirizzo di accesso random, è minore di 0 o è maggiore dell'indirizzo dell'ultimo byte presente nel file.
2) Il numero file non è compreso tra 0 e 7

Vedi anche:

Capitolo **I FILE DATI**
Istruzione **READ#**
Istruzione **OPEN**
Istruzione **CLOSE**
Capitolo **NOTE DI IMPLEMENTAZIONE**

CONTROL-C

Se si desidera fermare l'esecuzione del programma, si deve premere il tasto "control" ed il tasto "C" contemporaneamente.
Effettuata tale operazione sullo schermo, a conferma dello stop dato, compare la seguente scritta:

STOP IN LINE XXXXX

dove XXXXX rappresenta il numero linea al quale l'esecuzione si è fermata.
In caso durante la stampa del listato del programma, si preme control-C, la linea attuale visualizzata non viene interrotta, ma viene completata e solo allora sul terminale compare il messaggio STOP.
Quando si usa, control-C, si ritorna al modo diretto di NSB8, perciò si è liberi di esaminare, il programma e le variabili.
Per rimandare in esecuzione il programma dal punto in cui si è interrotto quando è stato premuto control-c, si deve usare il comando CONT (naturalmente non si può usare CONT, se durante l'interruzione si è modificato il programma).
Da ricordare che è possibile disabilitare il control-C, tale operazione però è descritta, in un'altra sezione (**BASIC PERSONALIZZATO**)

Vedi anche:

Comando **CONT**
Istruzione **STOP**
Capitolo **BASIC PERSONALIZZATO**

DISPOSITIVI DI I/O MULTIPLI

Un sistema può includere, molti dispositivi di uscita e di ingresso, come ad esempio il terminale, la stampante, il display grafico ecc.. Il BASIC, può gestire, fino a 8 dispositivi ognuno dei quali è individuato da un particolare valore compreso tra 0 e 7. Il dispositivo #0, corrisponde alla console. Di solito, si accede a tali dispositivi con le istruzioni PRINT, INPUT, INPUT1, LINE.
Le precedenti istruzioni, hanno una sintassi in cui compare sempre il cross-hatch (#), e di seguito una espressione numerica che indica appunto l'unità logica di uno specifico dispositivo.
L'espressione che indica il dispositivo di I/O da utilizzare, è di norma la prima cosa che compare dopo la parola dell'istruzione:

```
PRINT #1, "TEST"
PRINT #Q,X,B,7
PRINT #D+3, "CRAZY",Q
PRINT #D7(X)

INPUT #B, L3
INPUT #7, "COMMAND: ",C$

LINE #1,132
LINE #D, L
```

Come esempio finale, assumendo la consolle come dispositivo #0, la stampante come dispositivo #1 e il CRT come dispositivo #2, viene riportato un piccolo programma, in grado di stampare messaggi diversi, su ognuno dei tre dispositivi specificati:

```
10  REM DIMOSTRAZIONE DI I/O
20  PRINT "QUESTO MESSAGGIO VA ALLA CONSOLLE."
30  PRINT #0,"ANCHE QUESTO"
40  PRINT #1,"QUESTO VA ALLA STAMPANTE"
50  PRINT #2,"QUESTO VA AL CRT"
```

Da ricordare che l'espressione del dispositivo caratterizzata da (#) nelle istruzioni PRINT e INPUT, non va confusa, con la specificazione formato iniziante con (%).

Vedi anche:

Istruzione **PRINT**
Istruzione **INPUT**
Istruzione **INPUT1**
Manuale del sistema operativo

LE FUNZIONI

FUNZIONI INTERNE

Il BASIC, mette a disposizione dell'utente, alcune funzioni interne, molto utili per semplificare il programma che si deve creare.
Tali funzioni comprendono il calcolo del coseno, del seno, della lunghezza di una stringa, ecc.
Uilizzarle in modo diretto, o in una linea di programma, risulta essere molto semplice, come mostrano i seguenti esempi:

```
PRINT COS(0)      viene stampato il COS di 0 (MODO DIRETTO)
10 B=SIN(9)       viene assegnato il seno di 9 (DA PROGRAMMA)
```

ARGOMENTI

Il valore chiuso tra parentesi, in una chiamata di funzione, è denominat ARGOMENTO.
Le funzioni, utilizzeranno il valore (s) dell'argomento (s) specificato, per generare il valore della funzione.
SQRT(4), per esempio, utilizza il numero 4 per generare la corrispondente radice quadrata (2).
Le funzioni, possono avere, anche più di una istruzione, in tal caso, tali parametri, vengono separati da delle virgole, in modo da formare una "LISTA ARGOMENTI", senza parentesi.
Un argomento, può essere anche formato da un'espressione:

```
COS(2*7)=COS(14).
```

e se A=14:

```
COS(A)=COS(14)
```

Le funzioni, possono essere utilizzate in espressioni, e come argomenti per esempio:

```
A=2*SQRT(100)
COS(SQRT(100)/10-1)=COS(0)
```

Da ricordare, che i parametri, devono essere forniti alle funzioni in maniera inequivocabile (nello stesso ordine e dello stesso tipo rispetto alla dichiarazione).
Se tale specifica, non è rispettata, il BASIC produce un **SYNTAX ERROR**; per esempio SQRT("HI"), l'argomento è una stringa, mentre deve essere un valore numerico.
Le seguenti pagine, descrivono le funzioni interne del BASIC.
Da notare, che tali descrizioni, includono, l'ordine ed il tipo di parametri, il nome della funzione e cosa produce.

FUNZIONI MATEMATICHE

ABS(<espressione numerica>)
Fornisce il valore assoluto, dell'espressione numerica:
ABS(3)=3, ABS(-3)=3, ABS(0)=0

SGN(<espressione numerica>)
Fornisce il segno dell'espressione numerica (1,0,-1):
SGN(10)=1, SGN(0)=0, SGN(-3.2)=-1

INT(<espressione numerica>)
Fornisce, il più grande valore intero, minore di o uguale al valore dell'argomento:
INT(3)=3, INT(3.9)=3, INT(-3.5)=-4

LOG(<espressione numerica>)
Fornisce una approssimazione del logaritmo naturale del valore dell'espressione numerica.
Se LOG è chiamato con un valore dell'argomento, minore o uguale a zero, viene generato un errore:
LOG(1)=0, LOG(7)=1.9459101, LOG(.1)=-2.3025851

EXP(<espressione numerica>)
Fornisce l'esponenziale in base E, del valore dell'espressione:
EXP(0)=1, EXP(2)=7.3890562, EXP(-2.3025851)=.1, EXP(1)=2.7182817

SQRT(<espressione numerica>)
Fornisce la radice quadrata dell'espressione numerica.
Se l'argomento è negativo, si provoca, un errore.
Es: SQRT(0)=0, SQRT(10)=3.1622776, SQRT(.3)=.54772256

SIN(<espressione numerica>)
Questa funzione, calcola il seno trigonometrico, del valore dell'espressione numerica.
L'espressione, deve specificare, un angolo, in radianti.
Es.: SIN(0)=0, SIN(3.1415926/2)=1

COS(<espressione numerica>)
Calcola, il coseno trigonometrico del valore dell'espressione, che deve essere specificato in radianti.
Es.: COS(0)=1, COS(3.1415926/2)=0

ATN(<espressione numerica>)
Calcola una approssimazione dell'arcotangente.
Il valore di ritorno, è espresso in radianti.
Es.: ATN(5)=1.3734007, ATN(1.7)=1.0390722

FUNZIONI PER OPERAZIONI SU STRINGA

LEN(<nme stringa>)
Fornisce, la lunghezza corrente, della stringa memorizzata nella variabile, avente il nome specificato.
Es.: se A$=”GATTO” allora LEN(A$)=5, se A$=”” allora LEN(A$)=0

CHR$(<espressione numerica>)
Fornisce il carattere corrispondente, secondo la tabella ASCHII, al valore dell’espressione numerica.
L’argoment, deve essere un intero compreso tra 0 e 255.
Es.: CHR$(65)=”A”, CHR$(97)=”a”, CHR$(32)=” “ (spazio)

ASC(<costante stringa, variabile stringa, o riferimento sottostringa>)
Fornisce il valore corrispondente, utilizzando la tabella ASCHII, al primo carattere contenuto nell’argomento.
L’argomento non deve essere la stringa nulla.
Es.: ASC(“B”)=66, ASC(“CLUNK”)=67 (da notare, che ASC, e` l’inverso della funzione CHR$).

VAL(<espressione di tipo stringa>)
Converte il valore della espressione di tipo stringa, in un numero.
Se l’espressione, non fornisce una costante numerica legale, viene generato un errore.
Gli spazi, sono ignorati.
Es.: VAL(“123”)=123, VAL(“000000”)=0, VAL(“ABCDE”)=ERRORE, VAL(“”)=ERRORE, VAL(“ “)=ERRORE, VAL(“123ABC”)=123, VAL(“ABC123”)=ERRORE

STR$(<espressione numerica>)
Questa è la funzione inversa di VAL, converte, in stringa, il valore numerico specificato.
Il frmato della stringa, dipende da quello di default specificato nell’istruzione PRINT (vedi istruzione **PRINT** e il capitolo **FORMATO DI STAMPA**).

FUNZIONI PER PARTICLARI INPUT

INCHAR$(<espressione numerica>)
Questa funzione, fa entrare il programma in una situazione di attesa, da cui esce appena viene inviato un carattere, dal dispositivo di input specificato dal numero dell’espressione numerica.
Tale funzione quindi, fornisce il carattere del tasto premuto, compresi i caratteri di controllo e quelli speciali (control-C, viene fornito solo se è disabilitato la “interruzione-programma”, vedi il capitolo **CONTROL-C**).
Es.:

```
10   T$=INCHAR$(0) \ REM RICHIESTA DEL CARATTERE ....
20   PRINT T$, \ REM VISUALIZZAZIONE DELLO STESSO
```

INP(<espressione numerica>)
Questa funzione, sostituisce l’istruzione di IN in ambiente Z80 e 8080, infatti legge il valore presente sul PORT, specificato, dall’espressione numerica. Il valore numerico che fornisce la funzione, corrisponde al valore contenuto dall’accomulatore (nell’intervallo da 0 a 255) dopo l’istruzione IN.
Da notare, che tale funzione, non aspetta come INCHAR$, INPUT ecc. che venga inserito il dato, infatti legge il valore momentaneo, del PORT, e perciò, può capitare, che il valore letto non corrisponde al valore desiderato dall’utente.

FUNZIONI PER LA MANIPOLAZIONE DI FILE SU DISCO

TYP(<espressione numerica>)
Questa funzione, fornisce l'indicazione sul tipo (numerico=2, stringa=1, fine del file=0) di dato del successivo elemento presente nel file aperto nel disco.
Il numero del file aperto, è specificato dal valore dell'argomento (vedi il capitolo **I FILE DATI**).

FILE(<espressione di tipo stringa>)
Fornisce un numero, che rappresenta il tipo del file, specificato dall'espressione, la quale deve generare un nome di file legale, in concordanza alle specifiche trattate nel capitolo **I FILE DATI**.
Se l'argomento non è un nome file corretto o non corrisponde al nome file presente sul dischetto, viene restituito il valore -1.
Per esempio, se si assume come nome file "ABC", presente sul drive 2, allora avremo che FILE("ABC,2") restituisce il valore 2.
FILE("DOS") restituisce uno 0 se il dischetto nel drive 1 e` un disco di sistema.

FUNZIONI DI CARATTERE GENERALE

RND(<espressione numerica>)
Questa funzione, fornisce un valore pseudo-casuale tra 0 e 1.
Il numero generato, dipende esclusivamente dal precedente numero creato.
Il primo numero della sequenza, viene denominato "SEME", o valore iniziale.
Se il valore dell'argomento è negativo, il BASIC, seleziona un SEME a caso (basandosi sullo stato del disco sistema), e genera il valore della funzione basandosi su tale valore.
Se l'argomento è 0, il precedente valore generato, è usato per generare un altro valore casuale nella sequenza.
Se l'argomento è ridotto ad un valore compreso tra 0 e 1, esso è usato come nuovo valore iniziale, la sequenza riparte ed il primo valore generato dal nuovo valore iniziale, corrisponde al parametro che fornisce la funzione stessa.
Il seguente esempio, setta un valore iniziale casuale e stampa il relativo valore random:

```
10  J=RND(-1)
20  FOR J=1 TO 10
30  PRINT RND(0)
40  NEXT
```

EXAM(<espressione numerica>)
Fornisce il contenuto della locazione di memoria specificata dal valore dell'espressione numerica.
L'argomento deve generare un intero compreso tra 0 e 65535.
Il valore restituito è invece un numero intero compreso tra 0 e 255.

FREE(<espressione numerica>)
Fornisce un valore (numero di bytes) rappresentante la memoria ancora disponibile per l'utente.
Il valore dell'argomento, di solito è posto, dai programmatori, a 0.

TAB(<espressione numerica>)
Questa funzione, può essere utilizzata esclusivamente, in una istruzione PRINT.
Tale funzione permette di posizionare il cursore, in varie parti della riga, in modo che la stampa risulti il più chiara possibile.
Il cursore avanza alla posizione carattere specificata nell'argomento di TAB, ed in concordanza a ciò, il BASIC, stampa l'appropriato numero di spazi.
La prima posizione carattere è 0, le altre si determinano, in modo sequenziale, incrementando sempre di 1 la posizione precedente.
Se il cursore o il dispositivo di stampa, supera la posizione specificata, successivamente esso, non potrà essere spostato.

CALL(<espressione numerica>)
CALL(<espressione numerica>, <espressione numerica>)
Tale funzione, permette di utilizzare subroutine in linguaggio macchina, in programmi BASIC.
Il valore restituito, è un intero compreso tra 0 e 65535, e rappresenta il valore contenuto nel registro accoppiato HL quando la procedura in linguaggio macchina, restituisce il controllo al BASIC.
Il primo argomento di tale funzione è un valore numerico, compreso tra 0 e 65535 e rappresenta il valore decimale dell'indirizzo di memoria, da cui inizia la procedura in linguaggio macchina.
Il secondo argomento, che è opzionale, ma deve essere compreso sempre tra 0 e 65535, potrà essere passato alla routine come parametro, e verrà salvato nel registro accoppiato DE.
Per maggiori informazioni su CALL e sulle procedure in linguaggio macchina, leggere il capitolo **PROCEDURE IN LINGUAGGIO MACCHINA**.

FUNZIONI-UTENTE

E' possibile, in BASIC, creare delle procedure, direttamente nel programma, in modo da risparmiare tempo e memoria. Tali procedure, potranno essere poste, in qualsiasi parte del programma e restituiranno sia valori di tipo stringa che di tipo numerico.
Naturalmente, come per le funzioni interne, è possibile passare ad esse, parametri che a loro volta, potranno essere numerici o di tipo stringa.

NOME DELLE FUNZIONI-UTENTE

Un nome funzione, deve essere composto da due lettere FN, seguite da una stringa o da un nome variabile numerica, per esempio:

FNX, FNQ7, FNA$, FNZ3$

Il tipo del nome variabile della funzione, determina il tipo del valore che la funzione restituisce, per esempio FNX è una funzione che restituirà un valore numerico, mentre FNA$, restituirà un valore di tipo stringa.
Da notare, che i nomi delle funzioni, sono ben separati e distinti dai nomi delle variabili, in particolare quindi, il valore restituito per esempio da FNA$, non influenzerà il valore assegnato a A$ e viceversa.

FUNZIONI-UTENTE COMPOSTE DA UNA SOLA LINEA

Una funzione, può essere definita da una sola linea, come accade nel seguente esempio:

```
10    DEF FNR(V,P)=INT((V*10^P)+.5)/(10^P)
```

La funzione FNR, avente due parametri (V,P), restituirà il valore V, troncato di tante cifre, quanto indicato da P, per esempio FNR(3.1415,2) genera 3.14

PASSAGGIO PARAMETRI ALLE FUNZIONI-UTENTE

Una istruzione DEF, deve includere una lista di stringhe e/o di nomi variabili numeriche, chiamati PARAMETRI della funzione.
Questa lista, deve essere chiusa tra parentesi tonde, e deve essere successiva al nome funzione come in questo esempio:

```
50    DEF FNW(X$,Y,Z)=LEN(X$)+Y+Z
```

La chiamata ad una funzione-utente, deve includere a sua volta, la lista di parametri da passare alla stessa, e come nella dichiarazione, tale lista deve essere racchiusa in parentesi tonde e deve essere successiva al nome della funzione.
Quando una funzione è chiamata, i valori delle espressioni, nella lista espressioni presente nella chiamata stessa, vengono assegnati ai relativi parametri della funzione, in modo che al termine di tale operazione, un determinato parametro della funzione contenga il corrispondente valore dichiarato nella chiamata.
Da ricordare che tra parametri e valori, deve esserci anche una corrispondenza di tipo, in poche parole, il valore ed il parametro che lo deve contenere, devono essere dello stesso tipo.
Se tale specifica non viene rispettata, viene generato o un **SYNTAX ERROR** o un **ARGUMENT MISMATCH ERROR**.
I parametri che possono errere inviati alle funzioni-utente, sono di due tipi, di seguito descritti.

PARAMETRI NUMERICI:

In una chiamata, prima che ad ogni variabile numerica nella lista di parametri venga assegnato il rispettivo valore della lista delle espressioni, viene effettuata una operazione di salvataggio dei precedenti valori contenuti nei parametri.
Completata la funzione, ai parametri, rivengno assegnati i valori salvati; questo significa che è possibile utilizzare un parametro come variabile, in più parti del programma come mostrato dal seguente esempio:

```
10    DEF FNX(B)=B*3
20    B=2 \ PRINT B
30    PRINT FNX(3)
40    PRINT B
```

Mandando in esecuzione si ha:

```
2     9     2
```

PARAMETRI DI TIPO STRINGA:

A differenza dei parametri numerici, i parametri di tipo stringa, non vengono salvati all'atto di una chiamata, perciò essi conterranno l'ultimo valore assegnato.
Da ricordare che se il parametro di tipo stringa non è stato dimensionato prima che avvenga la chiamata alla funzione che lo utilizza, il BASIC effettua come per l'operazione di assegnazione, un dimensionamento automatico di 10.
Per capire meglio le differenze tra parametri numerici e parametri di tipo stringa, mandare in esecuzione il seguente programma:

```
10   DEF FNQ(X,X$)=ASC(X$)+X
20   X=7 \ X$="FIRST"
30   PRINT X$,X
40   PRINT FNQ(1,"NEXT")
50   PRINT X$,X
```

Si noterà che X manterrà il valore iniziale, mentre X$ conterrà il valore assegnatogli nella funzione.

FUNZIONI-UTENTE COMPOSTE DA PIU' LINEE

Questo secondo tipo, permette di ricavare il valore della funzione, utilizzando più istruzioni BASIC. Per questo motivo, esse risultano essere molto simili alle subroutine, ma permettono un passaggio di informazioni più rapido, tramite i parametri, e la generazione di un solo risultato.
La dichiarazione di tali funzioni, è così strutturata:

<numero linea> DEF FN<nome funzione>(<lista parametri>)

Le istruzioni per calcolare il valore della funzione, devono essere successive alla suddetta.
Quando il valore è stato calcolato, una speciale versione dell'istruzione RETURN, causa la fine dell'esecuzione della funzione, e presenta il valore calcolato, come valore della funzione.
Infine per dichiarare la fine fisica, della dichiarazione della funzione viene utilizzata l'istruzione FNEND.
Esempio:

```
10   DEF FNM(X,M)
20   IF M<=0 OR M<>INT(M) THEN 40
30   RETURN ABS(X)-(INT(ABS(X)/M)*M)
40   PRINT "ERROR IN MODULO" \ RETURN -1
50   FNEND
```

In generale, le funzioni dichiarate su più linee, si rendono necessarie, quando l'algoritmo per calcolare il relativo valore, risulta essere troppo lungo per essere inserito in una sola linea.

NOTE FINALI

Da ricordare che non si possono dichiarare funzioni all'interno di altre, se ciò avviene si provoca un **FUNCTION DEF ERROR**.
Tutte le funzioni-utente, debbono avere almeno un parametro nella dichiarazione.
Non è possibile passare come parametri, interi array, ed effettuare chiamate di funzioni-utente nel modo diretto, in caso contrario, si provoca un **ILLEGAL DIRECT ERROR**.

Vedi anche:

Istruzione **DEF**
Istruzione **RETURN**
Istruzione **FNEND**

I NUMERI

Questa sezione descrive i numeri e come devono essere usati per essere considerati validi per il BASIC NSB8. Per informazioni più specifiche sulle non-standard versioni del BASIC, bisogna leggere il capitolo **BASIC PERSONALIZZATO**.

COSTANTI

I seguenti numeri rappresentano delle COSTANTI NUMERICHE:

0	347	-33.333	.00176	1.003
.1	-8	123.4567	-0.3	0.2

Tali costanti numeriche possono essere anche scritte in notazione scientifica (molto utile se bisogna rappresentare numeri molto grandi) in questo modo:

0E+00	3.47E+02	-3.3333E+01	1.76E-03	1.003E+00
1E-01	-8E+00	1.234567E+02	-3E-01	2E-01

I numeri nella notazione scientifica, vengono scissi in una parte MANTISSA e in una ESPONENZIALE. Queste due parti, sono separate dalla lettera E, perciò 1.76E-03 si legge: 1.76 per 10 elevato alla -3.

PRECISIONE

I numeri nella versione standard del BASIC NSB8, hanno una precisione di 8 cifre. Altre precisioni sono trovabili leggendo il capitolo **BASIC PERSONALIZZATO**.
NSB8 utilizza per la rappresentazione dei numeri, il codice BCD (vedi il capitolo **COMPATIBILITA' CON ALTRI BASIC**).
Tutte le operazioni aritmetiche, sono arrotondate a 8 cifre; per esempio, la somma tra 0.12345678 e 0.011111111 pur dando come risultato 0.134567891 (9 cifre) viene arrotondata a 8 cifre (0.13456789).

FRAZIONI

Che cosa è la rappresentazione decimale, di 2/3 ?
Una stringa minore di 6 cifre dopo il punto decimale è la sola risposta corretta.
Il BASIC, come altri linguaggi, rappresenta le frazioni con una certa precisione definita dalle sue caratteristiche.
Per esempio 2\3 viene espresso come 0.66666667.
Notare che viene troncata la rappresentazione dopo 8 cifre decimali.
Per quanto riguarda invece frazioni del tipo 1\2, la rappresentazione in BASIC risulta semplice, in questo caso il risultato sarebbe 0.5.

PRECISIONE 8 CIFRE

Prima di essere manipolati dalla macchina, alcuni numeri devono essere arrotondati in quanto il BASIC lavora con 8 cifre complessive, per es. 1234.56789 (9 cifre), viene arrotondato dal BASIC in 1234.5679 (notare che è stata effettuata una approssimazione sulla penultima cifra decimale, in modo che pur dovendo troncare la rappresentazione del suddetto numero, esso rimanga il più possibile "vicino" al valore reale.
In caso si debba operare con numeri aventi dollaro e centesimi nella loro rappresentazione maggiori del grado di precisione sudetto ($999,999.99), si deve utilizzare una versione del BASIC particolare (precisione maggiore).

UN NUMERO MOLTO GRANDE

Il numero 987654321 in BASIC viene arrotondato in 987654320 mentre la sua rappresentazione scientifica risulta 9.8765432E+08.
Come si può notare la precisione a 8 cifre, è rispettata in questa conversione e la cifra meno significativa, è tralasciata, in quanto non può essere rappresentata; da notare che la rappresentazione scientifica assicura il "ricordo" di uno 0 al posto della cifra meno significativa, questo per non alterare il valore posizionale delle rimanenti cifre.

UN NUMERO MOLTO PICCOLO

Il numero 0.00000000123 non è arrotondato da NSB8, il seguente numero invece sì: 0.00000000123456789.
Per dare una risposta al perchè della differenza suddetta, bisogna pensare alla rappresentazione scientifica dei due numeri.
Il primo risulta essere 1.23E-09. La mantissa che è la sola parte che viene arrotondata, è composta da 3 cifre e quindi è in concordanza con la precisione del BASIC (massimo 8 cifre).
Il secondo numero diviene 1.2345679E-09.
Infine se si aggiunge 1 ad entrambi i numeri, essi saranno arrotondati e diventeranno esattamente 1.
Controlla la versione del formato esponenziale per capire meglio.
I due numeri risultano quindi questi: 1.00000000123E+00 e 1.0000000012345679E+00.
Entrambe le mantisse superano le 8 cifre di lunghezza.
Arrotondando tali numeri quindi si ottiene soltanto il numero 1 per ognuno.

INTERVALLO

Un numero, può essere positivo negativo o uguale a 0.
I numeri positivi e negativi secondo la precisione di questo BASIC, devono essere compresi nell'intervallo tra 1E-64 e 9.9999999E+62.

In caso si faccia operare il BASIC con una costante numerica troppo grande, esso provoca l'invio sul monitor di un messaggio di errore (**SYNTAX ERROR**).
Se in caso contrario, lo si fa operare con un numero molto piccolo, esso lo arrotonda a 0.

VARIABILI

In BASIC, come in molti altri linguaggi, una variabile è considerata come un locazione di memoria, a cui si può assegnare un certo valore numerico.
In caso venga assegnato un nuovo valore alla variabile in esame, il precedente è perso per sempre, quindi la gestione delle variabili richiede molta attenzione.
Tutte le variabili numeriche vengono inizializzate con il valore 0, a meno che non si voglia esplicitamente cambiare tale velore con l'istruzione LET.
Ad una variabile, bisogna associare un NOME, il quale deve identificare il relativo contenuto durante la scrittura del programma, questo naturalmente per motivi di chiarezza.
Sono ritenuti validi in questo BASIC, nomi di variabili numeriche di questo tipo: A B7 C3 Z Q N8 P0.
Come si può notare un nome per essere valido deve cominciare con una lettera maiuscola, la quale può essere opzionalmente seguita da un numero compreso tra 0 e 9.
Questi tipi di variabili, vengono chiamate "variabili semplici", in quanto possono contenere solo un valore alla volta.

OPERATORI

Operatori sono usati nel BASIC, come nella tradizionale aritmetica, ossia per effettuare calcoli tra due operandi o per modificare un numero in certe occasioni.
Esistono tre calassi di operatori: aritmetici, relazionali, booleani.

OPERATORI ARITMETICI

Gli operatori appartenenti a questa classe sono i seguenti:

OPERATORE	FUNZIONE	ESEMPIO
^	esponenziale	9^2=81
*	moltiplicatore	5*1.5=7.5
/	divisore	3/2=1.5
-	sottrattore	3.2-2=1.2
+	sommatore	7.9+2.1=10
-	negatore	-3, -27

OPERATORI RELAZIONALI

Gli operatori relazionali, sono usati per la comparazione di numeri. Il risultato numerico che ne deriva, può assumere il valore 0 (falso) o il valore 1 (vero). Di solito tali operatori compaiono nei costrutti del tipo IF...THEN (vedi comando **IF**). Per esempio, ad un certo punto del programma, si potrebbe desiderare di assegnare il valore 10 alla variabile T se il valore di X è maggiore di 10; la comparazione (X>10) verrebbe quindi usata come: IF X>10 THEN T=10
La seguente tabella mostra gli operatori relazionali validi in questo BASIC:

OPERATORE	RELAZIONE	ESEMPI
>	maggiore di	(6>1)=1 (vero) (2>3)=0 (falso)
<	minore di	(0<0)=0 (falso) (1<3)=1 (vero)
<=	minore o uguale a	(5<=5)=1 (3<=5)=1 (6<=5)=0
>=	maggiore o uguale a	(8>=7)=1 (7>=7)=1 (6>=7)=0
=	uguale a	(9=9)=1 (9=7)=0
<>	diverso da	(4<>5)=1 (2<>2)=0

ESPRESSIONI

Una espressione numerica è una valida combinazione di: costanti numeriche, nomi variabili numeriche, operatori, chiamate alle funzioni e nomi di array. (il capitolo **LE FUNZIONI** e il capitolo **I VETTORI** per ulteriori chiarimenti riguardanti le chiamate alle funzioni ed i nomi degli elementi degli array). La costante 3.14, o un nome variabile tipo A, è una espressione a sè stante.
In contrasto, un lungo costrutto tipo:

```
(NOT(3+(SQRT(X*Y)/M3-47)/8)^3
```

è soltanto una espressione numerica.

ESEMPI DI ESPRESSIONI NUMERICHE LEGALI

```
3.14
43+A
((X+2)^(Q-R))*SQRT(Z)
```

ESEMPI DI ESPRESSIONI NUMERICHE ILLEGALI

```
438,000.33      (MOTIVO: le costanti non possono contenere virgole)
7**Y            (MOTIVO: due operatori in una riga non sono legali)
((3*ABS(A))+4   (MOTIVO: inserimento parentesi improprio)
```

OPERATORI BOOLEANI

Gli operatori booleani (AND, OR e NOT), possono essere usati per combinare o modificare, espressioni relazionali in complesse valutazioni logiche. Tutti i valori numerici, possono essere l'oggetto di una operazione booleana: tutti i valori tranne 0 sono trattati come "veri"(1), mentre lo 0 e` trattato come "falso" (0). La seguente tabella fornisce informazioni su tali operatori, ricordare che <A1> e <A2> rappresentano due eventuali operandi.

OPERATORE	AZIONE	ESEMPI
<A1> AND <A2>	Se entrambe <A1> e < A2> sono vere (non 0), l'operazione AND è "VERA" (1), altrimenti è "FALSA" (0).	(3>5 AND 2<3)=0 (3>2 AND 0<=0)=1 (2=3 AND 0>-1)=0
<A1> OR <A2>	Se uno degli argomenti è "VERO" l'operazione di OR è "VERA", se invece entrambi gli argomenti sono "FALSI", l'OR è "FALSO".	(3>5 OR 2<3)=1 (3>2 OR 0<=0)=1 (2=3 OR 0<-1)=0
NOT <A1>	Nega il valore booleano dell'argomento. Se <A1> è "VERO", l'operazione NOT da come risultato "FALSO", altrimenti è "VERA".	

ORDINE DI VALUTAZIONE DEGLI OPERATORI

L'operazione 7+3*2 dà come risultato 20 o 13? Dipende da quale operando è valutato per primo. Per ovviare a tale problema, questo BASIC come molti altri linguaggi, presenta una lista di precedenza. Quindi operatori che hanno una precedenza maggiore, vengono valutati per primi, in caso si debbano valutarne due aventi lo stesso "grado" di precedenza, si procede da sinistra a destra. Gli operatori nelle parentesi vengono valutati prima di quelli fuori. La tabella seguente, mostra una lista che esprime il grado di precedenza degli operandi, perciò più si scende nella lettura, e più il relativo operatore ha un grado minore di precedenza; da ricordare che operatori su una stessa linea, hanno uno stesso grado.

NOT	(-, nega un numero)
^	(esponenziale)
*,/	(moltiplicatore, divisore)
+,-	(sommatore e sottrattore)
=,<,>,<>,<=,>=	(relazionali)
AND	
OR	

Esempi:

7+3*2=13
(7+3)*2=20
3*8/2=12
-5+4=-1

I VETTORI

INDICI

Un vettore, è una collezione ordinata di variabili numeriche.
L'intero vettore, è identificato da un NOME, mentre le variabili che lo compongono chiamate ELEMENTI, vengono identificate dal NOME del vettore e da un INDICE. L'indice, non è altro che un valore che identifica univocamente, la posizione dell'elemento nel vettore.
Per alcune applicazioni, l'indice, può partire da 0 o da 1 o da qualsiasi altro valore, per esempio, un vettore di 50 elementi, avente un indice massimo di 50, si trova ad avere 51 elementi se la numerazione parte da 0. Per rappresentare un elemto di un vettore, in una espressione, si deve fare seguire, il nome del vettore, dall'indice dell'elemento in esame, chiuso tra parentesi, per es. A(0), A(8) ecc. L'indice nella rappresentazione tra parentesi, si può presentare anche sotto forma di espressione numerica, l'importante è, che tale espressione, non dia come risultato un numero negativo, o un indice maggiore, dell'indice massimo del vettore.
In caso non venga rispettata una di queste specifiche, il BASIC, risponde con un **OUT OF BOUNDS ERROR**; inoltre se l'indice è un numero non intero, su di esso viene fatta una operazione di troncamento, cioè viene presa solo la parte intera del numero (3.6 verrebbe considerato 3).
Da notare, che variabili semplici e vettori, aventi lo stesso nome, possono coesistere in uno stesso programma, infatti sono consiserate come entità distinte. E' possibile anche, utilizzare come indice delle variabili semplici, per esempio se la variabile T contiene il valore 4, la scrittura A(T) e A(4), risulta del tutto identica.

VETTORI A PIU' DIMENSIONI: MATRICI

Il BASIC permette la dichiarazione di vettori che hanno più di un indice nella loro rappresentazione. Ogni indice esprime una dimensione, perciò un array (vettore) con N indici, è chiamato vettore a N dimensioni.
Da ricordare, che ogni indice va separato dal successivo con una virgola, e che ognuno di essi può essere il frutto di una espressione numerica.
Per accedere al terzo elemento situato nella quinta riga del vettore M a due dimensioni, per esempio, bisogna scrivere M(5,3).
Assumendo M come numero massimo di riga del vettore X, e Y come indice massimo della colonna, le seguenti istruzioni mostreranno al lettore, come si può leggere e listare il contenuto del vettore trattato:

```
10  FOR I=0 TO X
20  FOR J=0 TO Y
25  REM Stampa il successivo elemento w/no <CR>
30  PRINT TAB(I*15),M(I,J),
35  REM Ogni numero di colonna
36  REM è distanziato di 15 spazi
40  NEXT
45  REM Premi un <CR> prima di partire con la prossima riga.
50  PRINT
60  NEXT
```

Lo spazio per gli array, è riservato, dal programmatore, tramite l'istruzione DIM la quale appunto specifica la dimensione degli array e il massimo indice per ogni dimensione.

```
10 DIM X(1000), Y(2,3), Z(10,10,10)
```

Come si può constatare dall'esempio precedente, X è un array costituito da 1001 elementi e indicizzato da 0 a 1000; Y è un array bidimensionale avente come massimo 2 righe e tre colonne; Z infine è un vettore miltidimensionale avente rispettivamente le dimensioni 10,10,10.
Ogni dimensionamento array, include l'"elemento zero", cosi` che per esempio, l'array Z contiene 11 elementi al posto di 10. Quando sono specificate più dimensioni, l'indice massimo di ognuna, va separato dalle altre mediante una virgola. La virgola inoltre, viene usata per separare, in una stessa linea di dimensionamento, la dichiarazione di più array.
L'indice massimo per ogni dimensione, può essere dato anche, mediante una espressione numerica come viene nel seguente esempio:

```
DIM X(Q*Q*Q), Y(Q/5,3), Z(Q,10,SQRT(Q*Q))
```

N.B. Q=10

Durante la dichiarazione degli array, si deve stare molto attenti, in quanto, occupare molta memoria, è facile, quando si utilizzano vettoro multidimensionali, si consideri per esempio, che il vettore F(10,10,10,10) occupa uno spazio pari aquello occupato da 14,641 variabili.

(Questo corrisponde a 11*11*11*11, non a 10*10*10*10 — ricorda che in ogni dimensione esiste l' elemento 0).
Il precedente discorso, è stato fatto, per far conoscere all'utente, una eventuale comparsa dell'errore **MEMORY FULL ERROR**, che indica appunto che non esiste più memoria disponibile per gli array dichiarati.

DIMENSIONAMENTO DI DEFAULT

Gli array multidimensionali, necessitano di una dichiarazione.
Non avviene così per quanto riguarda i vettori ad una sola dimensione, questi infatti possono essere usati senza dimensionamento, in quanto vengono automaticamente dimensionati (una volta utilizzati) dal BASIC.
Tali vettori vengono inizializzati con un indice massimo di 10 perciò se l' utente vuole un dimensionamento diverso deve utilizzare per forza l'istruzione DIM.
Tutti gli elementi di un vettore, una volta dichiarato, vengono inizializzati con il valore 0.
È da ricordare infine che se si utilizzano vettori multidimensionali non dichiarati, si provoca l'errore **OUT OF BOUNDS ERROR**.

I VETTORI NON POSSONO ESSERE RIDIMENSIONATI

Nessun tipo di vettore, può essere ridimensionato in una successiva DIM nel programma.
In caso questa specifica non sia rispettata, si provoca un errore di dimensionamento (**DIMENSION ERROR**).

RIFERIMENTI AD ARRAY NELLE ESPRESSIONI NUMERICHE

Gli array possono essere utilizzati tranquillamente nelle espressioni, a patto che la loro sintassi di rappresentazione sia corretta.
In seguito vengono riportati alcuni esempi, utili per capire il loro funzionamento:

```
10   X=SQRT(Q(3,5)+ABS(B))
60   PRINT M(F(A,B),L(A,B))
90   N(A)=N(A+1)/2
```

Vedi anche:

Capitolo **I NUMERI**
Istruzione **DIM**

LE STRINGHE

Una stringa è una sequenza di lettere e/o di altri caratteri.
Per esempio sono stringhe valide:

HELLO NG;34* ABC123
THE DATE IS 7/7/78

COSTANTI DI TIPO STRINGA

Le stringhe racchiuse negli apici, vengono chiamate costanti di tipo stringa.
Da ricordare, che gli apici non fanno parte della stringa, ma servono soltanto per definire questo particolare tipo.
Esempi di "costanti stringhe" valide sono:

"HELLO" "NG;34*" "ABC123"
"THE DATE IS 7/7/78"

LA STRINGA NULLA

La stringa rappresentata da due apici consecutivi senza niente in mezzo, è chiamata "stringa nulla".

VARIABILI DI TIPO STRINGA

Proprio come i numeri, che possono essere memorizzati in variabili numeriche, anche le stringhe possono essere memorizzate nelle variabili relative.
La sintassi di un nome di una variabile stringa, è molto simile a quelli che identificano variabili numeriche, ha in più solo, il simbolo $ alla fine del nome.
Il $ appunto, serve per identificare che quel nome rappresenta una variabile di tipo stringa.
Un nome variabile di tipo stringa, in definitiva, può esssere composto da lettere maiuscole (A-Z) e da numeri (0-9), seguiti dal $.
Esempi di nomi validi sono:

A$ Q7$ Z3$ R$

DIMENSIONAMENTO VARIABILI DI TIPO STRINGA

Prima di utilizzare una variabile di tipo stringa, bisogna dimensionarla.
Per dimensionare tale variabile, nell'istruzione DIM devono comparire il nome della variabile e la lunghezza massima in caratteri delle stringhe che verranno salvate in tale variabile.
(Vedere l'istruzione **DIM** per capire come bisogna fare il dimensionamento.)
Nel caso si usi una variabile stringa senza averla dimensionata, il BASIC automaticamente, la dimensiona con una lunghezza massima di 10 caratteri.

Una volta create le variabili stringa non possono essere più ridimensionate nel corso del programma.
Ogni variabile di tipo stringa, può contenere ogni stringa, che abbia una lunghezza minore o uguale a quella dichiarata con il DIM.
La "lunghezza corrente" di una variabile di questo tipo, corrisponde alla lunghezza, in caratteri, del valore che essa contiene.
Per esempio se inseriamo nella stringa A$ dimensionata a 26 caratteri, tre valori:

AB, CAT,STRINGA NULLA

avremo rispettivamente le seguenti lunghezze correnti:

2,3,0

Immediatamente dopo essere stata dimensionata, una variabile di tipo stringa, viene riempita con tutti blanks, perciò per esempio, se la variabile A$ è stata dimensionata a 26, significa che essa contiene una stringa di 26 blank.

SOTTOSTRINGHE

Il programmatore può accedere ad una parte della stringa (insieme di caratteri consecutivi in una stringa). Tale parte viene chiamata SOTTOSTRINGA in quanto rappresenta una parte della stringa.
La sotto-stringa di una stringa, viene rappresentata, dal nome della stringa seguito dall'intervallo della sotto-stringa che si desidera estrarre, racchiuso in parentesi.
Per esempio ipotizzando che la variabile A$ contiene la stringa "abcde" si rappresenta la relativa sotto-stringa "cd" scrivendo A$(3,4) (specifica appunto che i caratteri da estrarre sono quelli compresi nell'intervallo 3-4).
Entrambi i numeri che esprimono l'intervallo della sotto-stringa, possono essere rappresentati mediante espressioni numeriche.
Il valore risultante dall'espressione deve essere maggiore o uguale a 1 e minore uguale al numero di caratteri presenti nella stringa.
Comunque ogni valore numerico nell'intervallo non intero, viene trasformato in intero, in quanto il BASIC ignora la parte frazionale, per esempio 5.6 è accettato come 5, 1.23 come 1 ecc..
Se A=3 e B=4 scrivere A$(A,B) equivale a scrivere A$(3,4), o "CD".
In caso B è maggiore di 5 o A è minore di 1, viene causato un errore e più specificatamente un **OUT OF BOUNDS ERROR**.
Questo errore, può verificarsi inoltre, se il valore della prima espressione è maggiore del valore della seconda.
Per esempio A$(4,2) è illegale.

APERTURA E CHIUSURA DI UNA SOTTO-STRINGA

È possibile estrarre una sotto-stringa da una stringa dando come parametro un solo valore che esprime la posizione del primo carattere della sotto-stringa da estrarre.
Più semplicemente, vengono estratti tutti i caratteri a cominciare da quello identificato dal parametro fornito, che naturalmente deve essere maggiore di 1 e minore della lunghezza della stringa originale.
Per esempio, A(3) contiene “cde”.
L’utilizzo di questo modo per estrarre le sotto-stringhe, può essere utile in certe occasioni in quanto elimina la necessità di conoscere la lunghezza della stringa originale.

CONCATENAZIONE DI STRINGHE

Una sola operazione può essere effettuata sulle stringhe, ed è chiamata CONCATENAZIONE e si rappresenta con l’operatore **+**.
Per esempio

“CAR”+”LOAD” dopo tale operazione risulta essere una stringa cosi` formata: "CARLOAD"

L’operazione di concatenazione si può applicare anche alle sotto-stringhe come nel seguente caso:

A$(2,3)+A$(2) risultato=”bcbcde”

N.B. (ricorda che in A$ si era salvata la stringa “abcde”.)

Si può effettuare una concatenazione anche di molte variabili tipo:

A$(1,1)+A$(3,3)+A$(3,3)+A$(5)+A$(4)+” MEANS YIELD”

che fornisce come risultato “accede MEANS YIELD”.

FUNZIONI SU STRINGHE

Il BASIC, include alcune funzioni interne che forniscono valori di tipo stringa, inoltre dà la possibilità all’utente di dichiarare funzioni che utilizzano stringhe (fare riferimento al capitolo **LE FUNZIONI** per informazioni più dettagliate.)

ESPRESSIONI DI TIPO STRINGA

Una espressione di tipo stringa è un insieme di variabili stringa, sotto-stringhe, funzioni stringa ecc. La concatenazione di due valori di tipo stringa, è inoltre considerata come una espressione di tipo stringa.
Esempi di tali espressioni sono:

```
A$
F$+”,2"
A$(1,X)+CHR$(97)+A$+”GO FOR BROKE”+FNS$(25)
```

COMPARAZIONE TRA STRINGHE

I valori di tipo stringa possono essere comparati fra loro, utilizzando i relativi operatori cioè:

=, >, <, >=, <=, <>.

Il BASIC effettua la comparazione, rispettando le seguenti specifiche:

1) Due stringhe sono valide se esse hanno lo stesso numero di caratteri e se hanno in corrispondenza di una posizione uguale, caratteri uguali.
2) Le stringhe sono comparate carattere per carattere da sinistra verso destra, fino a chè non si riscontra una differenza o una delle stringhe finisce.
3) Se esiste una differenza, e il valore ASCII del primo carattere differente della prima stringa è minore del corrispondente carattere della seconda stringa, allora la prima stringa è "minore della" seconda.
Se invece il carattere nella prima stringa è maggiore del carattere corrispondente nella seconda, allora la prima stringa è maggiore della seconda.
4) Se una delle due stringhe termina prima che si sia trovata una differenza, la stringa più corta è considerata minore della seconda.
5) Come conseguenza del punto 4 la stringa nulla è sempre considerata minore di qualsiasi altra stringa.

Quando si usano stringhe composte soltanto da caratteri alfabetici, la comparazione viene eseguita, tenendo conto dell'ordine alfabetico, cioè una lettera viene considerata minore di un altra se, nell'ordine alfabetico, essa si presenta prima dell'altra, naturalmente viene considerata maggiore se si presenta dopo.
In concordanza a quello detto precedentemente quindi, **"bird" e` minore di "three"** e **"zero" e` maggiore di "aardvark"**.
Le comparazioni di stringhe e la comparazione di parole con caratteri alfabetici, in BASIC sono rese possibili grazie all'utilizzo del codice ASCII.
Per capire meglio il discorso fatto precedentemenete, sono riportati alcuni esempi (ricordarsi delle 5 regole per la comparazione e utilizzare la tabella dei codici ASCII):

"Z"	>	"COCOA"	"120"	<	"75"
"123"	<	"124"	"AB "	>	"AB"
"123"	<	"ABC"	"AB1"	>	"AB01"
"ABC"	<	"abc"	","	>	"!"

N.B.
Gli operatori logici, non si possono utilizzare per relazionare più comparazioni fra stringhe nell'istruzione IF.

ASSEGNAMENTI A STRINGHE E A SOTTO-STRINGHE

Ogni espressione stringa legale, può essere assegnata, a una variabile di tipo stringa o ad una parte di essa (utilizzando le sotto-stringhe) come avviene in questi esempi:

```
A$=”CAT”
Q7$(1,3)=”DOG”
```

(Notare che nell’ultimo esempio, i primi tre caratteri della stringa Q7$ vengono rimpiazzati da “DOG” mentre i rimanenti rimangono inalterati.)
In caso si voglia assegnare una stringa troppo lunga rispetto al dimensionamento della variabile che la deve ricevere , il BASIC compie un troncamento, da sinistra verso destra, dei caratteri, fino a che tale stringa ha una dimensione pari a quella della variabile; il precedente discorso vale anche per le sotto-stringhe.
Seguire attentamente l’esempio per capire meglio tale tipo di assegnazione:

```
10   REM Dimostrazione del troncamento stringa in una assegnazione.
100  DIM L$(13)
110  L$=”ABCDEFGHIJKLMNOPQRSTUVWXYZ”
120  PRINT L$
130  L$(2,3)=”12345"
140  PRINT L$
```

Risultato:

```
ABCDEFGHIJKLM
A12DEFGHIJKLM
```

Come si può notare, la prima riga risulta essere il risultato dell’esecuzione della linea 120 che stampa la assegnazione fatta nella linea 110. Nella variabile stringa L$ vengono inseriti solo 13 caratteri, in quanto è stata dimensionata con un valore massiomo di 13, i restanti caratteri della stringa, quindi vengono troncati. Stesso discorso vale per il secondo risultato frutto della assegnazione effettuata alla riga 130. Quando si effettua una assegnazione a un intervallo di sotto-stringa, e il valore da assegnare è più piccolo in lunghezza, dell’intervallo di sotto-stringa, si ha una non modifica dei caratteri rimanenti. Il seguente esempio, chiarirà meglio le idee:

```
10   REM ASSEGNAZIONE A SOTTO-STRINGHE.
20   DIM L$(13)
30   L$=”ABCDEFGHIJKLM”
40   PRINT L$
50   L$(5,9)=”12345"
60   PRINT L$
70   L$(5,9)=”abc”
80   PRINT L$
```

Risultato:

```
ABCDEFGHIJKLM
ABCD12345JKLM
ABCDabc45JKLM
```

Nell'assegnazione della linea 50, "12345" viene inserito esattamente nella sotto-stringa L$(5,9). Nella linea 70, "abc" e` piu` corto di due caratteri, questo fa si` che i caratteri modificati, risultano essere solo i primi tre. E' possibile utilizzare inoltre, per l'assegnazione di un valore, la forma del "apertura-chiusura" della sotto-stringa, come in questo esempio:

L$(5)=L$(5,LEN(L$))

(LEN(L$) è una funzione interna che rappresenta la lunghezza corrente della variabile L$)
Qualsiasi assegnazione di una stringa nulla a una sott-stringa, causa un non cambiamento nella stringa.

MASSIMA LUNGHEZZA DI STRINGHE E LUNGHEZZA CORRENTE

La massima lunghezza di una variabile di tipo stringa corrisponde al massimo numero di caratteri che possono essere salvati in essa per esempio M$ dimensionata a 50, può contenere al massimo 50 caratteri. La lunghezza corrente invece, rappresenta il numero di caratteri presenti nella variabile, al momento del suo utilizzo (la lunghezza corrente, si ottiene utilizzando la funzione LEN), per esempio se M$ contiene "CAT" la lunghezza corrente risulta essere 3. Fino a che M$="CAT", non esiste una posizione carattere successiva a 3 e quindi una scrittura del tipo M$(3,5) è sbagliata.
Al contrario la precedente scrittura è valida (M$(3,5)), se ad M$ viene assegnata la stringa "STICK".
Da notare che un discorso equivalente vale anche per la dimensione della variabile, perciò M$(40,60) non è una scrittura valida (ricordare che M$ era stata dimensionata a 50).

SETTAGGIO CARATTERI IN BASIC

Il BASIC oltre ai caratteri normali, (numeri, lettere, caratteri di punteggiatura) ne include anche di invisibili (control) e di indefiniti. Ogni carattere in BASIC, viene espresso da un byte, e il numero 256 esprime tutti i valori che il BASIC può gestire. I primi 128 di essi (0-127) corrispondono ai 128 caratteri del codice standard ASCII. I rimanenti 128 caratteri (128-255) sono generalmente indefiniti sopra molti terminali, ma il BASIC li mette a disposizione dell'utente. La funzione interna **CHR$** può essere usata per rappresentare tutti quei caratteri che non possono essere scritti o stampati.
Nel seguente esempio essa è utilizzata per rappresentare gli apici:

```
10   A$="HI THERE"
20   PRINT A$
30   A$=CHR$(34)+A$+CHR$(34)
40   PRINT A$
```

Risultato:

HI THERE
"HI THERE"

Vedi anche:

Capitolo **LE FUNZIONI**
Capitolo **I NUMERI**
Istruzione **DIM**
Tabella **ASCII**

FORMATO DI STAMPA

STAMPA DI NUMERI IN FORMATO NORMALE ED ESPONENZIALE

Normalmente il BASIC può stampare un numero, in forma normale o in notazione scientifica. Prima della stampa del numero, il BASIC, stampa uno spazio, come mostrano i seguenti esempi:

```
 3.1234
  .5678
 8.123
 5.654
```

Il BASIC, in caso il valore da stampare sia troppo grande o troppo piccolo, utilizza automaticamente la rappresentazione scientifica.I seguenti numeri, sono un esempio di tale specie di notazione:

```
 1.23456E+14
 2E-09
 5.234567E+13
```

Come si puo'notare, ogni numero espresso in notazione scientifica è formato, da uno spazio, dal simbolo del meno "-" se il numero è negativo, dalla prima cifra della mantissa, dal punto decimale, dalle rimanenti cifre della mantissa, dall'esponenziale E, da un "+" o da un "-" (tali simboli indicano il segno dell'esponenziale), ed infine da due cifre che indicano il valore a cui E deve essere elevato. Da ricordare, che effettuando una dichiarazione esplicita che inizia sempre con "%", nella lista di output di un PRINT, si puo` scegliere il formato del numero da stampare (intero, in notazione scientifica ecc.).

COSA RAPPRESENTA IL FORMATO DI UN NUMERO ?

La formattazione di un numero, esprime il numero di spazi che devono essere riservati per la sua rappresentazione, in una lista di output, una formattazione di 6 perciò, non può andare bene se si deve rappresentare il numero 1234.56 (7 cifre con il punto decimale). In caso appunto che il formato sia sbagliato, viene provocato un errore e più precisamente, un **FORMAT ERROR**.
Il formato I (intero), può essere utilizzato per stampare solo numeri interi. La sua dichiarazione è formata dal simbolo "%", da un numero e dalla lettera maiuscola "I" (es. %3I).
Il numero dato, specifica, il numero di posizione colonna, sulla linea di stampa, riservato al numero. L'esempio fatto precedentemente quindi, indica che qualsiasi numero stampato deve essere un intero, e deve essere messo nella terza posizione rispetto all'inizio della linea stampata.
I numeri interi rappresentabili quindi sono da -99 a 999.
Da ricordare, che "-" viene considerato un carattere e quindi occupa una posizione, di cui bisogna tenere conto nella dichiarazione del formato.
Quando si stampa un numero avente una determinato formatto il BASIC omette lo spazio all'inizio (cosa che invece non faceva in stampa in formato libero), perciò può accadere che due stampe successive vengano poste una di seguito all'altra, rendendo così l'interpretazione, assai difficile; per evitare ciò, è consigliabile quindi, utilizzare la forma esplicita di spaziatura (inserimento di spazi direttamente nella lista di output), utilizzando la funzione TAB, o specificando un campo di formato contenente almeno uno spazio dopo il numero.

RAPPRESENTAZIONE A DESTRA

Tutti i numeri aventi una determinato formato, nella stampa, risultano essere collocati, a cominciare dalla destra rispetto al loro campo di formato, i seguenti esempi, chiarificano meglio questo concetto:

```
  349
 1234
  7.3
 8.42
2118.37
 1.61
```

L'istruzione:

```
PRINT "HERE IS A GAP:"",%10I,2
```

produce:

```
HERE IS A GAP:         2
```

Come si può notare, nell'ultimo esempio, il numero è stampato nella posizione numero 10, in quanto il campi di formatto è di 10.

NUMERO DI DECIMALI

Se si debbono stampare numeri, aventi parte decimale, o numeri in rappresentazione scientifica, è possibile troncare la rappresentazione di tali numeri ad una certa cifra.
Se per esempio si ha un formato di questo tipo: %7F2 i numeri stampati avranno solo 2 cifre decimali:

```
302.63
 51.00
987.12
1234.56
```

Nota che se il numero è un intero, gli zero, sono immessi, per mantenere, la posizione decimale.
La cancellazione di tali zero, sarà trattata in seguito.
Se un numero che deve essere stampato, ha un numero di posizioni decimali, maggiore di quanto consente la dichiarazione di formato, il valore stampato, è un numero arrotondato, avente l'indicato numero di cifre.

I possibili formati, sono (“n” e “m”, sono due costanti numeriche):

MODELLO	NOME	EFFETTO
nFm	F-format	Ogni successivo valore numerico nella lista di un PRINT sarà stampato con un campo di **n** caratteri e con **m** cifre a destra del punto decimale.
nI	I-format	Ogni successivo valore numerico nella lista di un PRINT, sarà stampato con un campo di **n** caratteri e senza parte decimale (intero). Se si tenta la stampa di un numero non intero, con questo formato, si causa un FORMAT ERROR.
nEm	E-format	Ogni successivo valore numerico nella lista di un PRINT sarà rappresentato in notazione scientifica con un campo di **n** caratteri, e con **m** cifre a destra del punto decimale della mantissa.

Per tornare alla rappresentazione in formato libero, basta digitare il simbolo “%” (percentuale) seguito da un CR.
Tutti i valori numerici, in una lista di output di una istruzione PRINT, vengono stampati nel formato specificato, finchè non appare una successiva dichiarazione, o fino alla fine della lista stessa.
Da ricordare che una dichiarazione di formato è valida solo per la linea in cui si trova, in particolare, se si hanno per esempio due lineee di programma come queste:

```
10  PRINT %3I,A,B,C
20  PRINT D
```

il contenuto della variabile D, viene stampato con un formato libero a differenza di A, B, C che hanno un formato ben specificato (3I).

FORMATO DI DEFAULT E FORMATO CORRENTE

BASIC tiene conto, di due specificazioni di formato: il FORMATO CORRENTE e il FORMATO DI DEFAULT.
Ogni valore numerico, in una lista di output di una istruzione PRINT, è stampato, usando il formato corrente.
All’inizio di ogni istruzione PRINT, il valore del formato corrente, è del tutto uguale a quello di default.
Di conseguenza, il formato corrente, è cambiato, ogni volta che si presenta una dichiarazione di formato, nella lista di output.
Il formato di default, è settato inizialmente a “formato libero” e può essere cambiato, utilizzando il cross-hatch (#) come descritto precedentemente.

ALTRI FORMATI CARATTERI

Molti "caratteri di formato", possono essere utilizzati, per modificare una specificazione di formato. Tali caratteri, devono comunque comparire dopo il "%" e prima della stessa specificazione.
I caratteri menzionati sono:

Z Gli zero dopo il punto decimale sono eliminati, e vengono inseriti al loro posto degli spazi.

Il formato specificato, dopo questo carattere, diverrà il formato di default.
Inoltre la conversione da numero a stringa, è fatta utilizzando il formato di default.
(Vedi il capitolo **LE FUNZIONI**, in paricolare la funzione interna **STR$**).
Notare, che i simboli %# "costringono" il formato libero ad essere il formato di default.
Questo è utilizzabile, nel caso in cui si voglia tornare al formato libero dopo aver scelto un altro formato come default.

C La virgola sarà messa alla sinistra del punto decimale e indicherà ogni sequenza di tre cifre es. 1,234,567.
(Notare che l'opzione C, non è valida con la specificazione del formato E).

$ Il segno del dollaro, sarà posto a sinistra del valore quando esso sarà stampato.

Attenzione, quando si usano C o $ in una specificazione di formato, devi essere sicuro, che il campo, sia specificato in modo che abbia abbastanza posizioni per caratteri, per il numero stesso, per il $ e per le virgole inserite implicitamente dalla macchina.
Per esempio:

```
PRINT %C9F2, D
```

potrebbe generare una stampa tipo $3,478.92 (con D=3478.92) o un **FORMAT ERROR** (con D=107843).
Nel secondo caso infatti, occorrerebbe per rappresentare il numero $107,843.00 un formato in grado di riservare 11 posizioni.

Esempi:

FORMATO	VALORE	RISULTATO
%8F2	19.355	19.36
%$6F2	45.12	$45.12
%C9I	1000000	1,000,000
%C8I	1000000	**FORMAT ERROR**
%10E3	472	+4.720E+02
%Z10E3	472	+4.72E+02
%$C11F2	201758.88	$201,758.88

FLUSSO DI ESECUZIONE E DI CONTROLLO

Il BASIC, effettua l'operazione relativa ad una istruzione, dopo averla eseguita e quindi interpretata.
L'esecuzione avviene in modo sequenziale, si procede quindi dall'istruzione avente numero linea minore a quella avente numero linea maggiore.
In caso siano presenti, su una stessa linea più istruzioni, si procede ad eseguirle da sinistra verso destra.
L'ordine sequenziale di esecuzione, può essere comunque alterato, di norma utilizzando le istruzioni GOTO,IF...THEN, FOR, NEXT, EXIT, GOSUB, RETURN, ON...GOTO, si ottiene tale effetto.
Una volta eseguita una di queste istruzioni, che effettuano "salti", l'esecuzione riprende in modo sequenziale, fino all'incontro di una nuova "istruzione di salto".
Le istruzioni specificate precedentemente, pur effettuando tutte una alterazione del flusso di esecuzione, effettuano operazioni diverse.
Di norma, il GOTO viene utilizzato per effettuare ripetizioni di una parte del programma, l'IF...THEN, per eseguire una determinata parte del programma, in corrispondenza di una certa condizione, il FOR...NEXT per eseguire delle istruzioni un determinato numero di volte, l'ON...GOTO per eseguire differenti procedure, a seconda della valutazione della condizione ed infine il GOSUB...RETURN per ripetere procedure evitando così la trascrizione di stesse parti del programma, più volte.

IL LOOP FOR-NEXT

CORPO DEL CICLO

Un costrutto FOR-NEXT, è così formato:

Esempio #1

```
10    FOR I=1 TO 10
      {CORPO}
99    NEXT
```

Come si può notare, le istruzioni contenute in {CORPO}, vengono ripetute 10 volte.

LA VARIABILE DI CONTROLLO ED IL VALORE LIMITE

Nella linea 10, la variabile I, viene denominata “variabile di controllo del ciclo”.
Il BASIC, utilizzando I come contatore, riesce a determinare la fine del loop. I viene inizializzata a 1 (dipende comunque dalla sintassi nel FOR), ed incrementata di 1, ogni volta che si esegue il {CORPO}. Ad ogni passaggio, il BASIC, confronta il valore contenuto in I con 10, naturalmente per sapere se si è giunti all’ultima ripetizione, se quindi I=10 si esce dal FOR, altrimenti si riesegue il {CORPO}.

VALORE DEL PASSO (OPZIONALE)

Nell’esempio fatto precedentemente, il passo era 1, in quanto ad ogni ciclo, la variabile I, veniva incrementata di 1. E’ possibile mediante la parola STEP, definire il valore di incremento (ricordare che in caso tale valore è negativo, non si ha più un incremento della variabile, ma bensì un decremento), in modo da avere un “passo” superiore a 1.

Esempio #2

```
10    FOR J=1 TO 10 STEP 2
      {CORPO}
99    NEXT
```

Esempio #3

```
10    FOR K(3)=5 TO 1 STEP -1
      {CORPO}
99    NEXT
```

Come si può notare, nell’esempio #2, il corpo, viene ripetuto 5 volte in quanto il passo è 2; la variabile J, assumerà quindi i valori 1,3,5,7,9. Nell’esempio #3 invece, il passo è -1 e perciò si avranno 5 decrementi della variabile K(3).
Se il parametro STEP non è inserito, come default, viene posto il passo uguale a 1.

Da notare che se il valore del passo è positivo, il valore iniziale, deve essere minore o uguale al valore limite, mentre se il passo è negativo, il valore iniziale, deve essere maggiore o uguale al limite.
In caso queste specifiche non vengono rispettate, il {CORPO} non verrà mai eseguito, come accade nel seguente esempio:

Esempio #4

```
10   FOR Q=5 TO 1
20   PRINT “QUESTA LINEA NON VERRA` MAI ESEGUITA”
30   NEXT
40   PRINT “DOPO IL LOOP”
```

Mandando in esecuzione il programma precedente, si ottiene:

```
DOPO IL LOOP
```

Come si può notare, Q è maggiore di 1 e quindi il flusso viene passato alla linea 40.
I parametri nell’istruzione FOR, si possono presentare anche in un formato non intero, come accade nel seguente esempio:

Esempio #5

```
10   FOR I=.1 TO 10.5 STEP .1
     {CORPO}
20   NEXT
100  REM IL CORPO VIENE RIPETUTO 105 VOLTE
```

La variabile I, può essere usata nel {CORPO}, ma richiede molta attenzione, in quanto si deve tenere conto che essa è pur sempre la variabile di controllo. Non è possibile cambiare il valore iniziale o il limite, o il passo, durante l’iterazione essi rimangono stabili e permanenti per tutto il loop.

ANNIDAMENTI DI CICLI FOR-NEXT

E’ possibile inserire dei FOR, all’interno di altri, come mostrato, nel seguente esempio:

Esempio #6

```
10   FOR I=0 TO 100
20        FOR J=1 TO 99
30             FOR K=0 TO 100
40             PRINT I,J,K
50             NEXT
60        NEXT
70   NEXT
```

Come si può notare, si è ottenuta, una intersezione di FOR, infatti, il FOR identificato dalla variabile J, risulta essere il corpo del FOR identificato da I ecc. Si può anche notare, che ad ogni variazione di I, corrispondono 9999 scritture delle variabili I,J,K. Per evitare errori, bisogna ricordare, di chiudere tutti i FOR, con i rispettivi NEXT, in quanto molto spesso, si perde la visione del flusso.

INSERIMENTO OPZIONALE DELLA VARIABILE DI CONTROLLO IN NEXT

La variabile di controllo, può essere messa opzionalmente nell'istruzione NEXT.

Esempio #7

```
10    FOR I=1 TO 10
20          FOR =1 TO 10
30          PRINT I,J
40          NEXT I
50    NEXT J
```

Inserire la variabile di contollo in NEXT, può essere utile ai fini della chiarificazione del programma. Da notare, che se tale parametro viene inserito, il BASIC effettua un controllo per verificare se la variabile in NEXT, coincide con quella in FOR, in caso non lo sia, viene inviato sullo schermo un messaggio di errore.

USO DI EXIT

Un ciclo di FOR, può terminare, senza avere completato, tutti i cicli stabiliti nella sua dichiarazione; tale operazione avviene appunto, se viene eseguita una istruzione di **EXIT**. Questa istruzione, riporta il controllo, ad una istruzione precedente o successiva al loop. L'istruzione EXIT, risulta essere quindi, una specie di GOTO, e può servire, per trasferire il controllo ad una determinata linea o per impedire alcuni cicli di un loop di un FOR. Da ricordare che tale istruzione, può essere usata solo per il ciclo FOR, se non si rispetta tale specifica, possono sorgere problemi, e quindi possono presentarsi messaggi di errore.

Esempio #8

```
10    REM SI RICERCA NEL VETTORE A, IL PRIMO ELEMENTO NON ZERO.
20    REM L'INDICE DI TALE ELEMENTO SARÀ N.
30    REM SE TUTTI GLI ELEMENTI SONO 0, N POTRÀ ANCHE ESSERE 0
40    REM SI ESCE QUANDO SI TROVA UN ELEMENTO DIVERSO DA 0.
80    FOR N=1 TO 10
90    IF A(N)<>0 THEN EXIT 130
110   NEXT
120   N=0 \ DA QUESTO PUNTO A CONTIENE TUTTI ZERO
130   REM DA QUESTO PUNTO, N CONTIENE INDICE CORRETTO O ZERO
```

USCITA DA LOOP INNESTATI

Per esempio, se si é giunti all'esecuzione della linea 70 del loop interno di due loop innestati (tipo esempio #6), e si desidera andare alla linea 600, esterna al loop più esterno, bisogna procedere come nell'esempio seguente (inserire un EXIT in ogni loop):

Esempio #9

```
70 EXIT 71
71 EXIT 600
```

Vedi anche:

Istruzione **FOR**
Istruzione **NEXT**
Istruzione **EXIT**

LE PROCEDURE

Nella scrittura di un programma, può capitare, di dover ripetere parti di programma più volte.
Per esempio, in caso si debba ideare un programma che richieda all'utente, la risposta (SI, NO) a certi quesiti, si può notare, che le seguenti linee (o simili), compaiono più volte:

```
20   INPUT "PER CORTESIA, RISPONDERE SI O NO: ",A$
30   IF A$="" THEN 20
40   A$=A$(1,1)
50   IF   A$="Y" THEN 70
60   IF   A$<>"N" THEN 20
```

Come si può ben capire, ripetere parti di programma, risulta essere non funzionale, in quanto si spreca tempo e memoria. La soluzione ottimale, a questo problema, corrisponde all'utilizzo delle procedure, che contengono appunto le parti di programma da ripetere. Tali procedure, possono venire richiamate, in qualsiasi parte del programma, mediante una sola istruzione.
La chiamata ad una prcedura, comporta un salto alla prima linea della medesima, l'esecuzione del rispettivo corpo, e il ritorno alla istruzione seguente alla chiamata.
In BASIC, GOSUB <numero linea>, rappresenta, la chiamata della procedura iniziante dal numero linea specificato. Il BASIC, riconosce, la fine di una procedura, solo quando esegue l'istruzione RETURN. Appena viene eseguita tale istruzione quindi, il BASIC effettua un salto all'istruzione successiva la chiamata (da ricordare, che il numero linea di tale istruzione, viene salvato, al momento della chiamata stessa). Trasformare, il testo di programma, prima esposto come esempio, in una procedura, risulta esere molto semplice, in quanto basta sostituire l'ultimo REM con RETURN, e modificare i numeri linea, così:

```
1020 INPUT "PER CORTESIA, RISPONDERE SI O NO: ",A$
1030 IF A$="" THEN 1020
1040 A$=A$(1,1)
1050 IF A$="Y" THEN 1070
1060 IF A$<>"N" THEN 1020
1070 RETURN
```

La routine, a questo punto, può essere chiamata in ogni parte del programma, come mostra il seguente esempio:

```
40   PRINT "IL NUMERO E` ESATTO ?"
50   GOSUB 1020
60   PRINT "IL NUMERO E` 10"
70   GOSUB 1020
```

I FILE DATI

I dati, sono registrati sul dischetto, in files. Un file, è una sezione di memoria sul dischetto, che viene riservata per il "salvataggio" dei dati. Per accedere ad un file, bisogna fornire il NOME FILE, e altri attributi come: LA LUNGHEZZA (o spazio), il TIPO, e INFORMAZIONI SULLA DENSITA'.
Si possono "listare" tutte le informazioni riguardanti tutti i file presenti su un dischetto, utilizzando il comando CAT.
Tale lista, compare nel seguente formato:

NOME LOC SPAZIO TIPO DENSITA' TDI

Per esempio:

PROG1 73 20 2 D

L'esempio precedente, mostra che sul dischetto, è presente un file di nome "PROG1", avente inizio dal settore 73, occupante uno spazio di 20 blocchi (256-byte), e di tipo 2.
Il "D" presente alla fine di ogni lista, fornisce informazioni sulla densita`, in quest caso, la densita` usata e` del formato double-density (doppia-densita`). Se un file è registrato in single-density (singola densità), invece di "D" compare nella posizione suddetta la lettera "S".
Il TDI (Type Depended Information), nell'esempio precedente, non è mostrato, in quanto è raramente usato.
Tutte queste informazioni sui files, sono registrate sul dischetto, in un particolare spazio e precisamente, nei primi 4 settori; tale spazio, viene chiamato **DIRECTORY**.

NOME DEI FILES

Il nome di un file, può essere formato, da un massimo di 8 caratteri (per caratteri si intende: lettere maiuscole e minuscole, i numeri da 0 a 9 e i simboli di punteggiatura).
Ogni carattere, può essere usato in qualsiasi posizione nel nome, tranne che lo spazio e la virgola.
Il nome di un file, deve essere univoco all'interno del dischetto, perciò per esempio, solo un file nello stesso dischetto, può avere il nome FILE1.
Da ricordare che esiste la possibilità di utilizzare sia lettere maiuscole che lettere minuscole, perciò su uno stesso disco possono essere presenti per esempio "file1" e "FILE1".
Un numero indicante il drive su cui si desidera salvare il file, può essere associato al nome file.
Tale suffisso deve comparire nel formato seguente:

<NOME FILE>,<SUFFISSO>

Dove suffisso è un numero che rappresenta il drive.
Per esempio, se il file "PROG" e` sul dischetto nel drive #2, la via corretta per scrivere il suo nome e`: PROG,2. Il file POP nel drive #3 sarà denominato POP,3.
Se nessun suffisso è fornito, come default viene assunto il drive #1, perciò i file SINONIMO e SINONIMO,2 rappresentano non lo stesso file, ma bensì, due file ben distinti, in quanto il primo è presente sul drive #1, mentre il secondo è registrato nel drive #2.
Riepilogando, un nome file deve essere univoco e composto da un massimo di 8 caratteri, ai quali può essere aggiunto un suffisso riguardante il drive in cui è presente lo specifico file.

SPAZIO OCCUPATO DAI FILE (LUNGHEZZA)

La lunghezza dei file, è specificata in blocchi. Ogni blocco, può contenere al massimo, 256 bytes di informazioni. Nella directory che si invoca con il comando CAT, lo spazio occupato da ogni file è espresso in blocchi. In un sistema a doppia densità, due blocchi sono inseriti nel mesimo settore, mentre in singola densità, ogni blocco rappresenta un settore.
Ogni file occupa una sezione contigua di memoria e può occupare un numero indefinito di blocchi, che dipende naturalmente dalla capacità del disco.

TIPO DEI FILES

Ad ogni file è associato un tipo che può essere usato per classificare il rispettivo file in accordo all'uso che di esso si vuole fare. Per esempio, il tipo 2 viene utilizzato per file contenenti programmi scritti in BASIC, il tipo 3 per file che contengono dati utilizzati dai programmi BASIC, il tipo 1 per file contenenti programmi eseguibili in linguaggio macchina.
Sono disponibili comunque, 128 tipi da 0 a 127, questi tipi, sono utilizzabili dall'utente a sua discrezione, per esempio l'utente potrebbe decidere che tutti i file di tipo 7 devono contenere programmi di gestione.

APERTURA DEI FILES

Prima di poter usufruire dei dati contenuti nel file, bisogna associare il nome file con un <numero>, usando l'istruzione OPEN. Da quel punto in avanti bisogna utilizzare, ogni volta che si vuole fare riferimento a quel determinato file, il relativo <numero>.
Per esempio supponendo di avere aperto come file #2 il file ACCT, ogni accesso nel vostro programma a tale file, deve essere effettuato usando il numero di riferimento (#2) e nn il nome del file (ACCT).

CHIUSURA DEI FILE

Quando si finisce di utilizzare un file, l'istruzione CLOSE disassocia un tal numero al relativo file e permette quindi l'apertura di un nuovo file avente lo stesso numero di associazione del file precedente. La chiusura di un file, comporta il salvataggio nel file stesso, di tutti quei dati che pur facendo parte del file, erano stati posti temporaneamente nella memoria RAM.
In caso il vostro programma richieda l'inserimento manuale di più dischetti nel drive, è necessario prima di effettuare tali operazioni, chiudere tutti i file, questo naturalmente per evitare scritture o letture errate.

TIPI DI DATI NEI FILES

Tre tipi di dati possono essere salvati nei file in BASIC e precisamente: NUMERI, STRINGHE e BYTES separati. Il tipo di ogni elemento,occupa un certo spazio sul file, quando è salvato.
I tipi numerici, richiedono sempre un numero fissato di spazio.Questo spazio è sufficiente a contenere un valore numerico. Le stringhe al contrario, necessitano di uno spazio variabile, in concordanza al contenuto corrente della stringa stessa.
I valori di tipo byte, richiedono solamente un byte dello spazio memoria del disco. Ogni elemento di questo tipo, contiene un valore intero binario da 0 a 255.
Il BASIC, scrive le stringhe e i numeri nei file, utilizzando un determinato formato in modo da rendere efficiente e veloce, la scrittura e la lettura dei dati nel file.
I bytes a differenza delle stringhe e dei numeri, non sono riconoscibili in modo così immediato, perciò il programmatore deve sempre a conoscenza della fine del byte, durante la lettura e la scrittura.
Se tale conoscenza non può essere eseguita, la gestione del file risulta pressochè impossibile.

ACCESSO DEI DATI

Le due istruzioni che permettono l'accesso ad un file sono READ# (legge) e WRITE# (scrive).
La prima istruzione legge il dato dal file e lo assegna ad una variabile definita dal programmatore, la seconda invece effettua l'operazione contraria e più precisamente, scrive un dato nel file (vedi l'istruzione **READ#** e l'**WRITE#** per maggiori informazioni). Tali istruzioni, possono essere utilizzate per accedere ai dati di un file, in modo sequenziale o in modo casuale (random).

ACCESSO SEQUENZIALE

Questo tipo di accesso, permette l'accesso ai dati del file, in modo sequenziale, ossia uno dopo l'altro.
Il BASIC, automaticamente, pone uno speciale "marchio" di fine file dopo l'ultimo valore del file sequenziale.
Questa operazione è molto importante in quanto evita errori che potrebbero accadere nel caso in cui si effettuassero letture o scritture in locazioni non appartenenti al file.
Un controllo per verificare se si è giunti a fine file, può essere fatto, utilizzando la funzione interna TYP la quale, necessita del nome file come argomento, e restituisce un valore, indicante il tipo del successivo elemento che si può leggere nel file, più precisamente:

TIPO	VALORE SUCCESSIVO
0	FINE-FILE
1	STRINGA
2	NUMERO

Per esempio, se il valre restituito da TYP(1) è 0, non si possono più effettuare letture, in quanto si è giunti alla fine del file #1.
La funzione precedente è molto utile perchè fornisce anche il tipo del dato successivo da leggere, in questo modo si può evitare di generare un **TYPE ERROR**, che indica appunto che è stato effettuato un tentativo di lettura di un valore numerico in una variabile di tipo stringa e viceversa.

Il seguente esempio, stampa sul video, il contenuto di un file sequenziale tale file, contiene sia stringhe che numeri:

```
10   REM STAMPA IL CONTENUTO DI UN FILE SEQUENZIALE
25   REM ASSUME STRINGHE, COMPOSTE AL MASSIMO DA 500 CARATTERI
30   DIM S$(500),F$(10)
70   INPUT “NOME DEL TIPO DEL FILE DA LEGGERE: “,F$
80   OPEN #1,F$
90   IF TYP(1)=0 THEN 240
100  REM CONTROLLO SE SI È GIUNTI A FINE FILE
110  IF TYP(1)=2 THEN 190
120  REM SI CONTROLLA SE IL SUCCESSIVO ELEMENTO È UN NUMERO
130  REM LETTURA E SCRITTURA DELLA STRINGA
150  READ #1,S$
160  PRINT S$
170  REM STAMPA DI UN NUOVO DATO
180  GOTO 90
190  REM LETTURA E STAMPA DEL NUMERO
200  READ #1,N
210  PRINT N
220  REM INSERIMENTO DI PIÙ DATI
230  GOTO 90
240  REM FINE DEI DATI
250  PRINT “** FINE DEL FILE **”
260  CLOSE #1
270  END
```

Il seguente esempio, scrive i numeri da 1 a 10 nel file “DAT7”, poi successivamente le rilegge e li stampa sul monitor:

```
10   REM SCRIVE 10 NUMERI NEL FILE E POI LI RILEGGE E LI STAMPA
40   OPEN #1, “DAT7”
50   FOR I=1 TO 10
60   WRITE #1,I
70   NEXT
80   CLOSE #1
90   REM ORA LI LEGGE E LI SCRIVE
100  OPEN #1,”DAT7"
110  IF TYP(1)=0 THEN 170
120  REM SI CONTROLLA SE SI È GIUNTI A FINE FILE
130  READ #1,I
140  PRINT I
150  REM LETTURA E SCRITTURA DEL VALORE SUCCESSIVO
160  GOTO 110
170  REM USCITA
180  PRINT “** FINE DEL FILE **”
190  CLOSE #1
200  END
```

AGGIUNTA DI NUOVI DATI IN UN FILE SEQUENZIALE

Per inserire nuovi dati nel file sequenziale, bisogna prima di iniziare la scrittura, leggere il file e raggiungere il fine file, in poche parole la scrittura è possibile solo dopo che si è giunti al fine file. Per esempio il programma seguente, inserisce nuovi numeri e precisamente i numeri da 10 a 20, nel file DAT7:

```
10   REM AGGIUNGE I NUMERI DA 11 A 20 NEL FILE DAT7
20   OPEN #1,”DAT7"
30   REM ORA LEGGE IL FINE FILE
40   IF TYP(1)=0 THEN 70
50   READ #1,N
60   GOTO 40
70   REM ORA AGGIUNGE I NUMERI
80   FOR I=11 TO 20
90   WRITE #1,I
100  NEXT
110  REM USCITA
120  PRINT”DONE”
130  CLOSE #1
140  END
```

ACCESSO SEQUENZIALE AI BYTE

È possibile, effettuare accessi ai file, byte per byte usando semplicemente l’ampersand (&) come prefisso nelle variabili alle quali si vuole assegnare il valore letto o come prefisso nelle espressioni che devono essere inserite:

```
10   REM LEGGE UN BYTE E LO SCRIVE
30   READ #1, &X
40   REM ASSEGNAZIONE DEL BYTE ALLA VARIABILE X
50   WRITE #1,&65
60   REM IL VALORE 65 VIENE INSERITO NEL FILE #1
```

Soltanto espressioni e variabili numeriche possono avere il prefisso “&”. I valori di tipo byte, sono interi e sono compresi nel range 0-255. Bisogna quindi stare molto attenti nelle scritture, e verificare che il dato che si vuole inserire, sia compreso nel range sopra descritto. Da notare, che l’ENDMARK è sempre inserito automaticamente dopo ogni scrittura.
In caso che non si abbia necessità di conoscere quando si è giunti alla fine del file, è possibile mediante una opzione nell’istruzione WRITE, evitare la scrittura dell’ENDMARK.

ACCESSO RANDOM DEI DATI

Il BASIC, mediante un apposito puntatore, è a conoscenza, in ogni istante, della posizione in cui si deve effettuare la lettura o la scrittura del valore successivo. Questo puntatore, specifica il numero di bytes presenti dall'inizio del file, fino alla psizione di lettura\scrittura corrente.
Tale numero è chiamato **RANDOM FILE ADDRESS** (indirizzo del file random).
Quando un file è aperto il puntatore corrispondente, è settato a 0, in modo che il primo dato accessibile è il primo elemento presente nel file. E' possibile cambiare il valore del puntatore, in modo da accedere a qualsiasi elemento presente nel file. Questo metodo è chiamato accesso random ed è molto pratico, in quanto per leggere un valore, non è necessario leggere tutti quelli che lo precedono, come avviene per i file sequenziali. Per accedere ad una determinata locazione, bisogna quindi aggiungere alle istruzioni READ e WRITE, una particolare espressione numerica preceduta dal segno di percentuale "%" (esempio:%R*5). Tale espressione, deve generare, un intero, compreso tra 0 e SIZE*256-1, dove SIZE, rappresenta lo spazio occupato dal file in blocchi. In caso il valore risultante dall'espressione è esterno all'intervallo sopra dichiarato, viene inviato all'utente un particolare errore. Per effettuare una corretta gestione di tale file, è necessario determinare in modo univoco, l'indirizzo di ogni dato all'interno del file, a questo scopo la via più semplice, è quella di definire tutti i dati nel file, devono essere dello stesso tipo. Per esempio, un file, deve essere composto da tutti numeri o da tutte stringhe aventi la medesima lunghezza. In alternativa, un file di tale tipo, potrebbe contenere, 100 records di 62 bytes ognuno e ogni record, potrebbe essere composto da 4 numeri in una riga, più una stringa avente una lunghezza di 40. Conoscendo l'esatta lunghezza di ogni elemento o di ogni record, è possibile trattare il file random, come un grande vettore, in modo che il calcolo dell'indirizzo random del termine "X" si può effettuare tramite l'espressione (X-1)*R, dove R è lo spazio del record o dell'elemento in bytes. L'aggiunta di un segno di percentuale "%" davanti a tale espressione, trasforma l'espressione suddetta, in una espressione di indirizzo random. Per illustrare meglio il discorso fatto precedentemente, avendo un file di stringhe ognuna delle quali occupante 42 bytes, è possibile calcolare facilmente l'indirizzo della prima e della cinquantesima stringa, utilizzando l'espressione prima menzionata, ossia:

(1-1)*42=0
(50-1)*42=2058

Utilizzando tale accesso, è necessario non scrivere l'ENDMARK dopo ogni scrittura, in quanto si rischia di "perdere" dati del successivo record. Il seguente programma, effettua un accesso ad ogni elemento di un file random composto da 1000 stringhe, ognuna delle quali, è lunga, 250 caratteri:

```
10   REM ACCESSO RANDOM ALLE STRINGHE
20   OPEN #1,"RANDSTR"
30   DIM R$(250)
40   R=250+2
50   REM R È LO SPAZIO DI OGNI ELEMENTO
60   INPUT "QUALE STRINGA (1-1000, 0 PER USCIRE)? ",I
70   IF I=0 THEN 130
80   IF I<1 OR I>1000 THEN 60
90   READ #1 %(I-1)*R,R$
100  PRINT "STRINGA #",I,": ",R$ / PRINT
120  GOTO 60
130  PRINT "QUIT"
140  CLOSE #1
150  END
```

I valori di tipo byte, possono essere gestiti in modo random, usando le stesse tecniche sopra descritte, ricordandosi però, di utilizzare l'ampersand che specifica appunto l'accesso a bytes.

Vedi anche:

Istruzione **OPEN**
Istruzione **CLOSE**
Istruzione **READ#**
Istruzione **WRITE#**
Capitolo **LE FUNZIONI**

PROCEDURE IN LINGUAGGIO MACCHINA

Questo tipo di BASIC, fornisce la possibilità di utilizzare procedure in linguaggio macchina.
Tali procedure, debbono essere poste, esternamente all'area di memoria, riservata al DOS, al BASIC e all'area per i dati ed il programma in BASIC. Di solito, si pongono tali routine in "memoria alta" utilizzando per esempio il comando MEMSET. Tali routine, sono accessibili tramite una funzione interna del BASIC, denominata CALL; questa funzione, richiede almeno un parametro, ossia l'indirizzo di partenza della routine che si intende chiamare.
Un secondo argomento opzionale, si può inoltre fornire, sotto forma di espressione numerica, naturalmente servirà, per un utilizzo interno della routine stessa (di solito il parametro viene caricato nel registro accoppiato DE). Da notare che ogni valore non intero, prima del trasferimento, viene troncato, viene eliminata cioè, la parte frazionaria.
Gli argomenti negativi, non vengono accettati, inoltre è possibile l'utilizzo, nella procedura, di tutti i registri, in quanto il BASIC effettua una particolare operazione di salvataggio, di tutti i dati utili a se stesso. Una routine in linguaggio macchina, termina, quando viene eseguita l'istruzione RET, essa infatti, riporta il controllo di esecuzione al BASIC, il quale, fa riprendere l'esecuzione dall'istruzione successiva alla CALL. Da ricordare, che è necessario riservare una particolare area di memoria se la routine fa uso dello STACK. Sia l'area riservata allo STACK che lo STACK POINTER, utilizzati dal BASIC, non devono essere modificati dalla routine in linguaggio macchina.
Il numero restituito dalla procedura, non è altro che la rappresentazione decimale, del contenuto del registro accoppiato HL. In definitiva, è possibile quindi, fornire un valore alla routine (mettendolo in DE), e riceverne un'altro (in HL).

Sintassi per le chiamate:

CALL (<espressione indirizzo>)
CALL (<espressione indirizzo>) , <espressione dell'argomento>

Esempio:

```
10    Q=CALL(60000,A)
```

Dove A èla variabile che contiene il parametro da trasferire in DE, mentre Q, è la variabile a cui verrà assegnato il risultato della routine, ossia Q, conterrà il contenuto di HL.
Altri esempi:

```
200   PRINT CALL(A(3)),A$
570   X=CALL(R+1024,G)
400   Q(CALL(43025,Y))=M
25    DEF FNM(G,D)=CALL(50000,G*256+D)
1030  F=CALL(S,ASC(S$))
```

Vedi anche:

Istruzione **FILL**
Istruzione **PRINT**
Capitolo **DISPOSITIVI DI I/O MULTIPLI**
Capitolo **LE FUNZIONI**

CONCATENAZIONE (Sequenza automatica del programma)

Tramite l'utilizzo dell'istruzione CHAIN, un programma, è in grado di caricare o mandare in esecuzione un altro algoritmo. Tale procedimento, in teoria, potrebbe durare un tempo indefinito, in pratica naturalmente non è così, si pensi per esempio al cambio dei dischi nel drive ecc.
In definitiva, la "concatenazione", puo` essere utile nei seguenti casi:

1) Si desidera, usare più programmi, in modo da crearsi un efficiente "sistema".

2) Il programma creato è troppo grande, per inserirlo in memoria, è necessario, quindi scomporlo, in modo che ogni modulo possa usufruire dei restanti.

COMUNICAZIONE TRA PROGRAMMI

Dopo una istruzione CHAIN, tutte le variabili, vengono "pulite", in modo che quelle utilizzate da piu` moduli, devono essere "risalvate", all'inizio, di ogni modulo.
Per la comunicazione tra programmi, esistono due metodologie.
La prima consiste nel trasferire tra i due programmi, un file; in questo caso, ogni modulo, prima di operare, deve effettuare un esame sullo stato corrente del file. E' possibile inoltre da un programma, settare alcune variabili, in modo che possano venire lette, da altri moduli.
La seconda metodologia invece, consiste nel salvare i dati necessari, in una parte di memoria RAM, esterna all'area dati/programma.
Da ricordare, che per utilizzare la memoria RAM, esistono molti metodi, ma i più importanti, utilizzano la funzione EXAM, e l'istruzione FILL.
Da notare, che se il file specificato, nell'istruzione CHAIN, non esiste, non è del tipo 2 o non è un programma BASIC valido, l'intera operazione, non ha successo.

Vedi anche:

Istruzione **CHAIN**
Istruzione **READ#**
Istruzione **WRITE#**
Istruzione **FILL**
Istruzione **ERRSET**
Capitolo **I FILE DATI**
Capitolo **LE FUNZIONI**

GLI ERRORI E LE SOLUZIONI

Ogni volta che si provoca un errore, durante l'esecuzione di un programma, si ha un arresto dell'esecuzione, ed un invio sul terminale di un messaggio di errore, questo naturalmente, per avvertire il programmatore, che il programma deve essere corretto. Per effettuare le correzioni, direttamente sotto il controllo del programma, il BASIC, ha implementato l'istruzione ERRSET.
Tale istruzione, è utile, in quanto permette una gestione degli errori, ossia essa viene utilizzata per trasferire il controllo, in caso di errore, ad una routine implementata dal programmatore, direttamente nel programma.
Il programmatore deve inoltre specificare, due nomi variabili nella sintassi delle istruzioni, come mostrato, dall'esempio seguente:

```
10    ERRSET 1000,L,E
```

Quando viene provocato un errore, il numero linea dell'istruzione, che ha provocato l'arresto dell'esecuzione, viene assegnato alla prima variabile, ed il codice del corrispondente errore, è assegnato alla seconda. Esaminando queste variabili, è possibile quindi, da parte del programma, conoscere non solo che cosa ha causato l'errore, ma anche l'esatta posizione (in paricolare consulate il capitolo **MESSAGGI DI ERRORE**).
Quando viene provocato un errore, tutte le procedure, le funzioni e i loops FOR-NEXT, rimangono attive, perciò è possibile eseguire un GOTO per tornare al punto dove è stato provocato l'errore, o all'istruzione successiva e riprendere l'esecuzione del programma, dopo la gestione dell'errore.
La ricerca dell'errore da parte del BASIC, può essere esplicitamente disabilitata, basta semplicemente inserire l'istruzione ERRSET, senza nessun argomento.
Fino a chè l'interruzione, tramite il control-C è disabilitata, un errore, verrà generato, ogni volta che verrà premuto il tasto control-C mentre il programma è in esecuzione, nel modo di ricerca-errori.
Se non si vuole trattare il control-C, come un errore, allora bisogna disabilitare lo stesso.

Vedi anche:

Istruzione **ERRSET**
Capitolo **MESSAGGI DI ERRORE**

LA LINEA DI EDITOR

INTRODUZIONE ALL'EDITOR

Tutti quelli che hanno usato tale BASIC per un certo periodo di tempo sono gia'consci delle funzioni di "cancellazione carattere" rappresentate dall'underline, RUB/DEL, e dal tasto di backspace, così bene come la funzione di "cancellazione" del tasto col simbolo @.
Queste sono due delle caratteristiche del piu'completo LINE EDITOR,che ti permette di modificare,velocemente ed efficentemente,le linee di informazione che tu inserisci.
In gran parte, la gente usa il line editor per cambiare o correggere il testo di un programma,una linea alla volta. Tuttavia, l'editor può anche essere usato in comandi e responsi per statement di tipo INPUT o INPUTl. Perchè l'aspetto di svolgimento del programma dell'editor sia il più lontano dalla media dei più importanti BASIC user,questo proposito sarà qui messo in rilievo.

Le funzioni di cancellazione-carattere e cancellazione linea dell'editor, permettono una correzione istantanea degli errori di battitura appena vengono commessi nella scrittura di una linea.
L'editor permette anche la correzione e la modificazione di una linea di programma che è già presente nel sistema. Per esempio, dopo lo SCRatching dell'area di programma/dati, stampate la seguente frase di PRINT in BASIC:

```
10  PRINT "TOTAL RECEIPTS TO DATE: ",Tl
```

Così subito dopo che tu hai premuto il RETURN, questa linea diventa parte del tuo programma corrente, si evidenzierà indicando che tu hai commesso un errore; infatti la variabile che deve essere scritta deve attualmente essere T2 e non Tl. In BASIC senza alcuna facilitazione da parte dell'editor, tu dovrai sforzarti a riscrivere l'intera linea per potere così correggere un carattere errato. Comunque, il BASIC "ricorda" sempre l'ultima linea che tu hai sritto. Questo per convenzione sarà chiamata la **OLD LINE**. Così tutte le volte che tu batti il RETURN per terminare una linea di input per il BASIC,questa linea immediatamente diventerà la OLD LINE (c'è una eccezzione a questa regola,che sarà discussa in un momento). Utilizzando le più alte funzioni del line editor, tu puoi convertire la old line in una **NUOVA LINEA** corretta che riproducerà il suo predecessore nel programma. Per ora, per provare a te stesso che il BASIC realmente "ricorda" la old line, premi **control-G**. Si può notare che la linea che tu hai appena battuto riappare.
Il cursore o print-head sul tuo terminale risiederà subito alla fine della linea stessa.
Premendo control-G prima di scrivere qualsiasi altra cosa, tu dici al line editor di prendere la old line dall'inizio alla fine, e di trattare essa come una nuova linea di input, copiando la linea sul terminale come esso sa fare.

In effetti, usando un carattere di controllo, tu hai riscritto la vecchia linea. Se tu ora premi RETURN, la nuova linea sarà ripetuta come linea 10 ma solo fino a che la nuova linea è identica alla vecchia, non risulterà nessun netto miglioramento: Tl sarà cambiato ancora in T2.
Comunque supponiamo che tu prema il tasto di UNDERLINE. Ora, l'ultimo carattere della nuova linea (la l che deve essere 2) è cancellato, e tu puoi scrivere il carattere corretto.
Se tu premi RETURN a questo punto, la linea corretta sarà ripetuta come la sua precedente errata.
Per correggere la ragionevolmente lunga linea 10, tutto quello che è richiesto è di premere 4 tasti: control-G, UNDERLINE, il tasto 2 e RETURN.

N.B.
Questo procedimento, durande la scrittura di una procedura, è più veloce e meno tedioso che riscrivere la linea per intero, sebbene per questo esempio di introduzione,probabilmente hai speso più tempo a stare attento a leggere le direttive e a osservare i risultati, che se tu avessi riscritto la linea dall'inizio. La pratica con l'editor ti permetterà di aumentare la tua velocità tremendamente.
Persino dopo una sola ora di esperienza con l'editor, tu noterai un gratificante miglioramento nella tua efficenza quando scrivi e modifichi un programma BASIC.

Ora, prendiamo un altro esempio. Realizzare, subito dopo avere battuto il tasto RETURN per terminare la nuova linea, essa diventa la vecchia linea, e tu puoi ora usare l'editor su di essa. Scrivi:

```
20                              e non premere RETURN!
```

Ora batti control-G; devi vedere la seguente frase sul terminale.

```
20      PRINT "TOTAL RECEIPS TO DATE: ", T2
```

Se ora premi RETURN,una nuova linea 20 sarà aggiunta al tu programma corrente. Il suo contenuto sarà identico al contenuto della linea 10. Quello che devi fare è creare una linea completamente nuova, combinando nuove informazioni stampate con parti della vecchia linea. Quando tu hai scritto la linea numero 20, tu stavi scrivendo sopra ai primi due caratteri della vecchia linea.
Premendo control-G, il line editor capisce che deve copiare solo la rimanente parte della vecchia linea sulla nuova linea. I primi 2 caratteri della vecchia linea saranno scartati in favore della nuova informazione. Supponiamo che non ci fosse un terzo carattere nella vecchia linea che era composta da uno o due caratteri. In questo caso, per la funzione control-G, non ci sarà niente da copiare sulla nuova linea. In questo caso, come in altri dove l'editor non può svolgere i tuoi desideri, suona il campanello sul tuo terminale.

IL COMANDO EDIT

Tutto quello che è stato mostrato fino ad ora, è solo come le più recenti linee stampate possono essere modificate o utilizzate per creare una nuova linea. Cosa devi fare se , dopo avere scritto la linea 20 nell'esempio precedente, vuoi tornare indietro e modificare nuovamente la linea 10?
A questo punto la old line sarà la linea 20 e non la 10; quindi l'editor vorrà ancora lavorare con la linea 20. Per ovviare a questo problema,tu puoi costringere il BASIC a trattare la linea 10 come old line, usando il comando **EDIT** come qui mostrato:

```
EDIT 10
```

Questo costringe il line editor a sostituire la old line naturale (l'ultima battuta) con la linea di prgramma che tu specificherai. In questo esempio, la linea 10 diventerà la old line.

N.B.
Scrivendo altri comandi vicini a EDIT, la linea di controllo diventerà la old line.
Il comando EDIT, comunque,è l'eccezione alla regola menzionata precedentemente. Quando premi RETURN dopo avere scritto il comando EDIT, la linea di controllo è scartata, e la linea di programma invece diventa la old line.

Notare che non c'è un responso evidente al comando EDIT, in quanto il cursore semplicemente si sposta all'inizio della linea successiva. Comunque premendo control-G, potrai vedere che la linea 10 diventerà veramente la old line, finchè essa viene immediatamente scritta sul terminale.
Usando il comando EDIT, puoi costringere ogni linea di programma a diventare la old line, e quindi puoi modificare ogni parte del tuo programma, o creare linee totalmente nuove prendendo informazioni da una old line "costretta", e combinando essa, sotto un nuovo numero di linea, con informazioni appena inserite. In seguito si discuteranno tutte le funzioni speciali del line editor, cosìbene come la teoria delle operazioni dell'editor.

SPECIFICHE E FUNZIONI DEL LINE EDITOR

Supponiamo che tu abbia appena premuto RETURN per immettere la linea 10 nel tuo programma; in questo modo, a linea 10 è ora la old line.
Il BASIC sta aspettando che tu inserisca una nuova linea o utilizzi i comandi dell'editor.
In questo stato, la old line è immagazzinata nella memoria del BASIC, e due "puntatori" sono mantenuti: uno alla posizione del carattere corrente nella old line (chiamato **OL pointer**) e l'altro alla posizione del carattere corrente nella nuova linea appena stampata (chiamato **NL pointer**). Prima che tu inizi a scrivere una nuova linea, entrambi i puntatori sono portati all'inizio delle loro rispettive linee. La maggior parte delle funzioni dell'editor sono spiegate più completamente facendo riferimento a questa coppia di puntatori.
Stampando un carattere normale (non un carattere di comandi dell'editor) in assenza di ogni altra funzione di edit, porterà al risultato che i due puntatori avanzeranno entrambi di una posizione. Il carattere stampato, è sommato alla nuova linea, e il puntatore della old line ora punta il carattere successivo nella linea stessa. Nella sequenza precedente, per esempio quando tu stampi 20 per iniziare la nuova linea di programma, l'NL pointer passa dalla linea 10 alla 20, mentre l'OL pointer salta dopo la 10 nella old line ,e punta allo spazio vicino al numero di linea.

PRIMA:

(old line) 10 PRINT etc....
^ **OL pointer**

(new line)
^ **NL pointer** il prossimo carattere inserito va qui

DOPO:

(old line) 10 PRINT etc....
^ **OL pointer**
(new line) 20
^ **NL pointer** il prossimo carattere inserito va qui

Queste sono le funzioni dell'edit, seguenti il CONTROL-CHARACTER che le invoca:

COPIA DEL RESTO OLD LINE ALLA FINE DELLA NEW LINE (Control-G)

Copia tutti i caratteri dalla posizione di OL pointer alla fine della old line sulla nuova linea, partendo dal carattere alla posizione di NL pointer. Se l'OL pointer punta già dopo la fine della old line, nessun carattere sarà copiato, e il suonerà il campanello di allarme.

COPIA DI UN CARATTERE DALLA OLD LINE (Control-A)

Il carattere puntato nella old line dal puntatore OL, è copiato nella nuova linea alla posizione puntata dal puntatorte NL. Come risultato, entrambi i puntatori avanzeranno di una posizione. Se non ci sono caratteri da copiare, suonerà il campanello. Ripetendo l'uso del comando CONTROL-A ,si otterrà lo stesso risultato del comando CONTROL-G.

PRENDERE UN CARATTERE (Control-Q)

Questa funzione cancella l'ultimo carattere della nuova linea, e decrementa sia il puntatore lOL che NL di una unità. Se questi stanno già puntando all'inizio delle rispettive linee, suonerà il campanello. Un UNDERLINE viene visualizzato sul terminale per denotare la cancellazione di un singolo carattere. Premendo i tasti di UNDERLINE, DEL/RUB o BACKSPACE (control-H) si avrà lo stesso risultato del control-Q.

CANCELLAZIONE DI UN CARATTERE DALLA OLD LINE (Control-Z)

Questo comando avanza OL pointer di una posizione, senza copiare nulla sulla nuova linea o avanzare NL pointer. Questo effettivamente "cancella" un carattere dalla vecchia linea in modo che non possa essere ricopiato sulla nuova linea. Un segno di percentuale (%) viene visualizzto sul terminale per indicare l'azione di questo comando. Se OL pointer è già alla fine della old line , allora il comando è rifiutato e il campanello suona per confermare questa situazione di anomalia.

COPIA SU UNO SPECIFICO CARATTERE (Control-D)

Un secondo carattere, denominato, "**carattere di ricerca**", deve essere inserito, prima dell'esecuzione di tale comando. Come risultato, si ha che il contenuto della vecchia linea, a cominciare dal carattere occupante la posizione indirizzata dal puntatore OL, sarà ricopiata sulla vecchia linea, formando così una nuova linea, a partire, dal primo carattere incontrato, che corrisponde al "carattere di ricerca"; da ricordare, che il carattere non è incluso.
Se il carattere di ricerca specificato, non esiste nella vecchia linea, non viene copiato nessun carattere nella nuova linea, e un suono viene prodotto per sottolineare, questa condizione di anomalia.
Per esempio, provare a scrivere la linea:

```
10  PRINT "QUESTA E` UNA LINEA DI TEST"
```

Successivamente inserire control-D e poi la lettera maiuscola S.

Da notare, che sul terminale non compare nè la lettera S, nè nessun carattere identificante l'operazione di control-D, ma viene visualizzato, subito il risultato, ossia:

```
10  PRINT "QUE
```

Come si può notare, la copia di caratteri è iniziata dalla posizione, occupata dalla prima "S" (per copiare tutto il resto della linea, naturalmente, utilizzare control-G).

ATTIVARE O DISATTIVARE IL MODO DI INSERZIONE (Control-Y)

Questo comando, permette di inserire nuovi caratteri, all'interno di una linea, ossia, se il modo di inserzione è in OFF, tutti i caratteri inseriti, verranno sovrascritti ai vecchi caratteri presenti nella linea, se al contrario, tale modo è in ON, tutti i caratteri aggiunti, faranno parte integrante, con i caratteri già esistenti, della linea (nessun carattere viene perduto).
Quando il modo inserzione, viene attivato, sul terminale, appare il simbolo "<" mentre quando viene disattivato, appare ">".
Se viene premuto per esempio, control-G, durante il modo di inserimento, sarà ancora copiato, il resto della vecchia linea sopra la nuova linea, e sarà fatto avanzare il puntatore OL fino alla fine della vecchia linea.
Per maggiori ciarimenti, è riportato di seguito un esempio:

Digitare la seguente linea:

```
10    PRINT "TEST LINE"
```

Ora, usare il comando control-D, per posizionarsi dopo le virgolette (per effettuare tale operazione, premere control-D, T), il terminale, dovrebbe ora presentarsi così:

```
10    PRINT "
```

Ora premere control-Y, il terminale si presenta così:

```
10    PRINT "<
```

Ora digitare HERE IS A e si avrà:

```
10    PRINT "<HERE IS A
```

Premendo di nuovo control-Y si avrà:

```
10    PRINT"<HERE IS A >TEST LINE"
```

A questo punto, premendo RETURN, la nuova linea 10, rimpiazzerà la precedente.

CANCELLARE E REINSERIRE UNA NUOVA LINEA (Control-N)

Questo comando, permette la cancellazione parziale o totale della nuova linea, e permette l'inserimento, di un'altra nuova linea.
La nuova linea cancellata, diventa la vecchia linea, per una successiva scrittura.
Il carattere **@** viene visualizzato all'inizio della successiva linea, quando questo comando è impartito. Questo simbolo può essere inserito direttamente, ottenendo così lo stesso risultato che si ha con control-N.
Dopo che il comando è stato eseguito, entrambi i puntatori della nuova e della vecchia linea, sono resettati all'inizio, delle loro rispettive linee.

COMPATIBILITA' CON ALTRI BASIC

Questa sezione, fornirà alcune informazioni, utili, in caso che si intenda convertire programmi in altre versioni di BASIC, in NSB8.

MANIPOLAZIONE DELLE STRINGHE

Le operazioni e le funzioni usate per accedere alle stringhe e alle sottostringhe spesso, differiscono nelle varie versioni di BASIC esistente.
Il capitolo **LE STRINGHE**, tratta i dettagli, sulla implementazione, in NSB8, di tali argomenti, comunque, l'accesso a sotto-stringhe, è possibile, atraverso l'utilizzo del nome della stringa. Molti BASIC, utilizzano la convenzione "MID-LEFT-RIGHT", che tratta appunto, l'accesso alle sottostringhe, utilizzando le funzioni interne: MID$, LEFT$ e RIGHT$.
I programmi, che utilizzano, questo metodo di accesso, dovranno essere modificati, per riflettere le convenzioni sulle stringhe, utilizzate da NSB8.
I seguenti esempi, spiegano ulteriormente, il concetto sopra trattato:

Altri BASIC		**NSB8**
LEFT$(X$,L)	corrisponde a	X$(1,L)
RIGHT$(X$,R)	corrisponde a	X$(LEN(X$)-R+1)
MID$(X$,L,N)	corrisponde a	X$(L,L+N-1)

VETTORI DI TIPO STRINGA

Molte versioni di BASIC, implementano tali vettori, con la sintassi, utilizzata da NSB8, per utilizzare le soot-stringhe. Perciò in questo BASIC, tali vettori, vengono creati, suddividendo, una variabile di tipo stringa, in sottostringhe, aventi lunghezza fissa.
Per esempio un array di N stringhe, ognuna delle quali ha una lunghezza massima di L, sarà dimensionata così:

```
10  DIM A$(N*L)
```

E l'elemento stringa di indice J (dove J può variare da 0 a N-1), sara accessibile facendo:

```
A$(J*L+1,(J+1)*L)
```

DICHIARAZIONI DELLE STRINGHE

In NSB8, tutte le stringhe più lunghe di 10 caratteri ,devono essere esplecitamente dichiarate in una istruzione DIM. Le stringhe, nella dichiarazione, possono essere dimensionate con una lunghezza a piacere, naturalmente, tale lunghezza non deve superare il limite della memoria del computer.
Da notare che molti tipi di BASIC, non richiedono la dichiarazione delle stringhe, prima del loro uso, ma impongono però limiti molto bassi, alla lunghezza massima delle stringhe.

TRASLAZIONE DEGLI INPUT

Certi caratteri, quando sono inseriti in un programma in NSB8, vengono automaticamente sostituiti da altri, questo per facilitare la conversione in NSB8, di programmi scritti in altre versioni di BASIC. Questa conversione, naturalmente, non è fatta su testi identificanti stringhe.
La seguente tabella, rappresenta tutti i caratteri che vengono convertiti:

Altri BASIC		NSB8
[	diventa	(
]	diventa	)
:	diventa	\
;	diventa	,

Cosìuna linea come la seguente:

```
10    PRINT A$[3,4]; : LET A$[3,4]=”HI”
```

Diventa:

```
10    PRINT A$(3,4), \ LET A$(3,4)=”HI”
```

ARITMETICA IN BCD DEL BASIC NSB8

NSB8, utilizza, per il calcolo delle operazioni aritmetiche, il codice BCD (binari coded decimal). Senza alcun limite sulla sua precisione (8 cifre nella versione standard), il metodo BCD, è il più accurato metodo di calcolo aritmetico applicabile, che èpossibile utilizzare attualmente.

LA VALUTAZIONE IF ... THEN

In NSB8, se la condizione valutata, risulta essere falsa, il blocco di istruzioni presenti dopo il THEN, viene saltata e si procede l’esecuzione, dalla istruzione successiva, la quale, si può trovare sia sulla stessa linea dell’IF, che nella riga successiva.
Per esempio il seguente programma, non salta l’istruzione B=7, a differenza di altri BASIC:

```
10    A=0
20    B=0
30    IF A<>0 THEN A=7 \ B=7
40    PRINT B
```

Come risultato, si ottiene:

```
      7
```

INDIRIZZI SPECIALI DI NSB8

La seguente lista, rappresenta gli indirizzi speciali di NSB8, ed il risultato, di ogni modifica inserita (da ricordare, che l'indirizzo di inizio di NSB8 corrisponde a E00H).

ORG+00H (E00H)

Il BASIC è inizializzato.
Un'automatica pulizia della memoria programma/dati è effettuata, e qualsiasi informazione, esistente in quella area di RAM, viene cancellata.
Da notare che questo indirizzo, è utilizzato dal comando GO BASIC del DOS.

ORG+04H (E04H)

Il programma esistente, non è influenzato, ma tutte le variabili ed i dati associato ad esso, sono cancellati.

ORG+14H (E14H)

Effettuando un salto a tale indirizzo, si riprende a lavorare in NSB8 dopo che si e usciti da quest'ultimo (il programma ed i dati rimangono invariati, in modo che il lavoro si possa riprendere esattamente da dove lo si era lasciato).
Perciò, si può effettuare una interruzione di un programma BASIC, con control-C, uscire dal BASIC con BYE, usare il DOS, e rientrare in BASIC all'indirizzo ORG+14 e usare il comando CONT, per fare riprendere l'esecuzione del programma corrente, esattamente da dove si era interrotto (naturalmente, se non si è cambiato nulla in memoria, durante l'interruzione).

BASIC PERSONALIZZATO

E' possibile cambiare certe caratteristiche di tale BASIC, in modo da renderlo più efficace per uno specifico lavoro. Per esempio, i limiti dell'area di memoria, possono essere variati, in modo da ottimizzare il lavoro, è possibile cioe, variare i limiti che configurano l'area dati e l'area codice. Tali modifiche, si possono effettuare, con l'istruzione FILL, e con la funzione interna EXAM. Se si vuole che le modifiche, siano permanenti, bisogna eseguire tutti i passi descritti di seuito, altrimenti le modifiche, durano per una sessione di lavoro soltanto (eseguendo solo il punto **C**).

A) Effettuare una verifica della memoria del sistema usando la funzione di test-memoria del MONITOR per essere sicuro di fare una copia del BASIC da una memoria errata.
In particolare, l'area dove risiedono BASIC e DOS deve essere verificata completamente.

B) A questo punto, si deve esssere sicuri che il DOS sia operante, e che tu sia nel suo modo di COMANDO (specificat dal prompt dello stesso DOS).Ora inserisci il tuo dischetto di sistema protetto (supportato dal North Star) nel drive 1 e scrivi:

GO BASIC <CR> quindi attendi fino a quando non viene visualizzato READY

C) Ora sei pronto per eseguire le varie modifiche al BASIC. Per effettuare ciò vengono mostrati i passi da seguire in ordine esatto. Se desideri saltare qualcuno di questi passi puoi farlo ma non cambiare mai la sequenzialità di essi! In ogni caso, devi sempre effettuare il passo 2 prima di utilizzare ed eseguire un numero più alto di passi.

PERSONALIZZAZIONE DEL BASIC NSB8

1. MEMORY SIZE

Inizialmente, la versione standard del BASIC NSB8, è realizzata per darti solo 16.384 Bytes, cioè 16 KBtyte, di memoria di lavoro detta anche area di programma/dati. Per darti la possibilità di scrivere e di eseguire programmi ragionevolmente lunghi, devi avere una quantità maggiore di memoria rispetto a tale limite. Inoltre, bisogna informare il BASIC della possibilità di utilizzare questa memoria extra grazie al comando **MEMSET**. Tu puoi utilizzare questo comando per allargare o stringere la area di programma/dati che il BASIC è abilitato ad utilizzare.
Semplicemente, si determinerà l'indirizzo (in decimale) della cella di memoria più alta che il BASIC è abilitato ad utilizzare, ed impiegare questo numero come argomento del comando MEMSET.
Per esempio,se si estenderà la memoria di 48 K,e si vorrà abilitare il BASIC a poterla utilizzare si scriverà:

MEMSET 49151

N.B.
L'argomento del MEMSET, tra l'altro, viene portato in binario e sistemato nei bytes **ORG+09H** e **ORG+10H**,dove "ORG" è l'origine del BASIC (indirizzo di partenza) nel tuo sistema usualmente 2D00H. Nella versione standard del BASIC, questi indirizzi sono rispettivamente 2D09H e 2D0AH. Il normale indirizzo più alto per la area di programma/dati è 5FFFH.

2. FISSARE UNA VARIABILE COME ORIGINE DEL BASIC

Per molti dei passi mostrati di seguito, la parola FILL viene utiliuzzata per modificare le locaziooni di memoria nel BASIC. Nell'esempio quì proposto, esso assumerà come indirizzo di partenza del BASIC il valore decimale assunto dalla variabile S. Se tu hai una versione del BASIC che parte a 2D00H, allora verrà usato 11520 per origine del BASIC; altrimenti, se il tuo BASIC parte da qualche alto indirizzo si determinerà il valore decimale equivalente dell'origine, e si utilizzerà tale valore. Utilizzate S in una assegnazione diretta. Per esempio, per la versione standard del BASIC,scrivere:

S=11520

3. LUNGHEZZA DI LINEA

La versione standard del BASIC NSB8 prevede che la console abbia una lunghezza di Input/Output delle linee di 80 caratteri.
Se la capacità di linea del tuo terminale è minore o maggiore rispetto a questa, assegna alla variabile L la sua appropriata lunghezza di linea. Se questa è 64, per ottenerla, scrivere:

L=64

Così il valore di L sarà corretto, quindi scrivere:

FILL S+14 , L

4. IMPAGINAZIONE VIDEO

Avendo un terminale video (CRT), è desiderabile spedire, dal BASIC, solo una "pagina video" per volta quando si desidera vedere il listato di un programma,e quindi attendere fino a che non si voglia vedere la pagina successiva. Assegna quindi alla variabile P il valore appropriato per il tuo terminale, inserendo il numero massimo di linee che questo può mostrare contemporaneamente.
Quindi scrivere:

FILL S+19 , P

La versione standard del BASIC NSB8 prevede come capacità video 24 linee alla volta; se il tup terminale risponde a queste caratteristiche, puoi saltare questo passo.
Da notare che che le linee visualizzate di un listato son sempre P-1in quanto nella parte bassa del video vi sarà la scritta:

PRESS RETURN TO CONTINUE

Per avere una altra pagina di listato, dovrai premere il tasto RETURN, mentre se desideri terminare la visualizzazione del listato a questo punto, dovrai premere control-C.

5. CARATTERE DI "BACKSPACE"

Nella versione standard del BASIC NSB8, quando vengono premuti i tasti di UNDERLINE, control-Q, BACKSPACE (control-H) o RUB/DEL per cancellare l'ultimo carattere inserito, il BASIC visualizza un UNDERLINE per confermare la cancellazione.
E' possibile cambiare questa configurazione di cancellazione di caratteri in qualsiasi altra che tu desideri assegnando alla variabile D il valore decimale ASCII del carattere desiderato.
Per esempio, il valore ASCII del carattere di BACKSPACE è 8 così scrivi:

D=8

Quindi inserire:

FILL S+23 , D

N.B.
Non tutti i terminali video usano il codice ASCII 8 come BACKSPACE, quindi è meglio consultare il manuale del tuo video per vedere quale valore ASCII consente di effettuare questa operazione.

6. INIBIZIONE DEL CONTROL-C

Per alcune applicazioni,si utilizza il comando control-C per interrompere volutamenete o meno un programma. Se per qualche ragione desideri disabilitare questo comando inserisci:

FILL S+24 , 1

Mentre per riabilitare il control-C scrivi:

FILL S+24 , 0

Nella versione standard del BASIC NSB8 il comando contol-C è abilitato; esso, se desiderato, può anche essere attivato o disattivato durante l'esecuzione del programma, usando gli stessi metodi.

7. IL BOOTSTRAP PROM NON STANDARD

Se il tuo sistema utilizza un BOOTSTAP di controllo della PROM non standard, devi convertire il byte alto, di quello che è l'indirizzo a cui si trova la tua PROM, in decimale e quindi assegnare questo valore a una variabile, chiamata B. Per esempio,se la tua PROM e indirizzata a partire da FC00H, avrai che il byte alto è FCH, che espresso in decimale equivale a 252, quindi inserirai:

B=252

Quindi scrivere:

FILL S+16 , B

Da notare che avendo una PROM non standard, se questa modifica viene sbagliata, la funzione RND non lavorerà correttamente quando gli viene passato un argomento negativo.

8. BASIC RISTRETTO

Esistono molte applicazioni che non richiedono l'utilizzo, delle funzioni matematiche speciali: SIN, COS, ATN, LOG e EXP; esse però occupano spazio in memoria, a scapito della quantità disponibile per l'area dati/programma. Il BASIC NSB8 fornisce la possibilità di ovviare a questo inconveniente, infatti queste funzioni possono essre eliminate.
Facendo riferimento alla tabella seguente, si può notare che non è possibile eliminate certe funzioni senza elinare quelle che precedono; così, per esempio, se si vuole eliminare LOG, verrano eliminate anche le funzioni SIN-COS ed ATN.
Per effettuare l'eliminazione delle funzioni, si procede, inserendo nella variabile C i valori espressi dalla seguente tabella:

FUNZIONI DA ELIMINARE	VALORE DA ASSEGNARE A C
ATN	1
SIN-COS	2
LOG	3
EXP	4

Supponiamo, di voler cancellare, tutte e quattro le funzioni, per fare questo, occorre inserire nella variabile C, il valore 4 e successivamente digitare:

```
FILL S+6 , EXAM(S+24+(C*2)-1)
FILL S+7 , EXAM(S+24+(C*2))
```

Un tentativo di utilizzo di una delle quattro funzioni, dopo che sono state eliminate, provoca un "crollo del sistema".

TABELLA DI RIFERIMENTO DEGLI INDIRIZZI DI NSB8

Gli indirizzi, sono forniti in relazione all'inizio di NSB8 (ORG) che si trova all'indirizzo E00H.

ORG+6 & ORG+7 [ENDBAS]
Queste due locazioni, contengono il byte alto e il byte basso dell'ultimo indirizzo usato dallo stesso BASIC e può essere modificato per contenere l'indirizzo più basso, in relazione al BASIC "ristretto".

ORG+9 & ORG+10 [HIGHMEM]
Contengono il byte basso ed il byte alto, rispettivamente, dell'indirizzo della RAM più alto, che il BASIC, può utilizzare per l'area dati/codice.
I valori standard: 255 e 95 rispettivamente (corrispondente a 5FFFH).

ORG+14 [LINE]
Lunghezza iniziale della linea.
Valore standard: 80

ORG+15 [AUTOS]
Controlla l'AUTO-START. Il byte a 0, attiva l'auto-start.
Valore standard: 1.

ORG+16 [BOOTPROM]
Corrisponde ai primi due digit esadecimali, nell'indirizzo PROM del bootstrap.
Valore standard: 224 (E8H)

ORG+19 [PAGES]
Controlla il modo-pagina per il listaggio del programma.
Se si desidera l'impaginazione, esso, deve contenere, il numero di linee, nella pagina terminale.
Se si inserisce il valore 0, non avviene l'impaginazione.
Valore standard: 24

ORG+23 [DELECHO]
Carattere da ripetere in risposta ad un cancellazione di un carattere.

ORG+24 [PANICOK]
Controlla l'uso del control-C per l'interruzione dei programmi BASIC.
Se questo byte è a 0, il control-C è abilitato, in caso contrario questo comando è disabilitato.
Valore standard: 0

NOTE DI IMPLEMENTAZIONE

FORMATI DELLA MEMORIA-DATI DEI DISCHETTI

Tutti i numeri, aventi una determinata precisione, hanno, in associazione, una relativa lunghezza di memorizzazione fissata in bytes. Naturalmente, un numero, avente 6 digit occuperà uno spazio minore, sul disco, di un valore che invece ne possiede 10.
Di seguito, viene riportata una tabella che indica il numero bytes, occupati sul disco, in relazione ad una specifica precisione:

PRECISIONE	BYTES
6	4
8	5
10	6
12	7
14	8

I numeri, sono memorizzati in blocchi, mediante il codice BCD:

Primo byte: bits 7-4 = digit più signifiativi del valore, nella codifica BCD.
bits 3-0 = successivi digit più significativi del valore.

Byte intermedio: bits 7-4 = successivi digit significativi del valore in BCD.
bits 3-0 = successivi digit significativi del valore

Ultimo byte: bit 7 = segno (1=negativo, 0=positivo)
bits 6-0 = esponente in eccesso nella rappresentazione binaria a 64
(Se tutti i bit dell'ultimo byte sono a 0, l'intero numero è 0)

Il valore decimale del primo byte, nel numero memorizzato sul disco, deve sempre essere maggiore di 15, perfino quando il numero è 0. Da ricordare, che la funzione TYP utilizza questo metodo per determinare se il successivo elemento, è numerico.
Le stringhe, sono memorizzate, usando un numero di bytes corrispondente alla lunghezza della stringa più 2 o 3 byte in alto, più specificatamente, le stringhe aventi, lunghezza minore o uguale a 255, sono memorizzate con due byte in alto, il primo contiene dal valore decimale 3, ed il secondo contiene il numero di caratteri presente nella stringa.
I bytes contenenti la stringa vera e propria, sono quelli che seguono appunto, questi byte inziali.

I valori stringa, aventi invece una lunghezza maggiore di 255, sono memorizzati con 3 byte in alto. Il primo contiene il valore 2, mentre i successivi, contengono rispettivamente il byte alto e il byte basso della lunghezza della stringa, espressa come un intero a 16 bit.
Come per il precedente caso, i bytes che contengono la stringa, sono successivi a quelli di testa.
L'ENDMARK per un file sequenziale è rappresentato da un singolo byte, di valore 1.

DIMENSIONI DEL BUFFER DEL FILE - "TEMPO DI VITA" DEI BUFFERS

L'apertura di un file, implica la riservazione di una area di memoria RAM che viene utilizzata come un buffer di trasferimento dati ad alta velocità, tra il BASIC, ed il drive.
Se si apre un file con singola densità, la dimensione del buffer corrisponde a 256 bytes, altrimenti, doppia densità, a 512 bytes. I buffer sono utilizzati, per accedere ai dischi in modo rapido ed efficiente. Dopo la chiusura, il buffer, non diventa memoria libera, infatti rimane riservato per un successivo uso, da parte di altri file, aventi il numero file, associato a tale buffer.

TABELLA DEI DISPOSITIVI DI STAMPA

Alla memoria, indirizzata da **ORG+17** e **ORG+18** esiste un puntatore, contenente il byte alto e il byte basso dell'indirizzo dove il BASIC ha memorizzato, la "tabella dei dispositivi di stampa".
Ognuno degli 8 bytes presenti, in questa tabella, contiene la posizione corrente del cursore di ognuno degli 8 possibili dispositivi di I/O, che il BASIC può avere (il primo dispositivo è indicato con #0).
Pe molte applicazioni, l'utente, può estrarre ed esaminare tali bytes per evitare messaggi tipo LENGTH ERROR o per evitare il CR automatico, in caso di "sbordamento" dalla riga.
L'utente, può utilizzare le seguenti funzioni, per manipolare tali byte:

```
DEF FNH(D)=EXAM (3601)+(EXAM(3602)*256)+D
REM D è il numero del dispositivo (da 0 a 7).
```

TABELLA DI GESTIONE DEI FILE

Tale tabella è lunga 80 bytes e contiene un ingresso di 10 byte, per ognuno degli 8 possibili file aperti (0-7). Ogni ingresso, ha il seguente formato:

a) byte 0: byte di stato
b) bytes 1-2: indirizzo del buffer per il file (basso/alto)
c) bytes 3-4: indirizzo del disco del file aperto
d) bytes 5-6: lunghezza del file in blocchi.
e) bytes 7-9: puntatore del file corrente

PROGRAMMI BASIC PRE-ESEGUITI

Dopo che le linee di programma sono inserite, esse sono pre-eseguite, automaticamente in modo da ottenere una forma più efficiente e compatta.Tale operazione, permette una esecuzione più rapida, ed un efficiente utilizzo della memoria.Se si effettua un list, invece accade che il compattamento, si dissolve, in modo che l'utente possa riavere il programma nella sua forma originale.
Questo metodo di compattamento, è applicato anche all'istruzione REM.
Da ricordare che istruzioni REM che contengono parole chiavi tipo FOR, NEXT ecc. avranno un compattamento maggiore di quelle che non le contengono, questo è dovuto al fatto che ogni singola parola chiave, può essere compattata in un singolo byte.

FORMA INTERNA DI UN PROGRAMMA

In RAM e su disco, un programma è rappresentato, come una serie di linee di programma le quali sono state compattate come spiegato prima.Ogni linea quindi contiene:

a) Byte 0: contiene la rappresentazione binaria, del numero di bytes che compongono la linea (qui chiamato N).

b) Byte 1-2: il numero linea, espresso come un intero binario a 16 bit (byte basso e byte alto).

c) Byte sopra a N-2: la linea del programma, nella sua forma compatta

d) Byte N-1: un carattere di carriage return (valore del byte 13 o 0DH).

Da ricordare che dopo l'ultima riga, esiste un ENDMARK, costituito da un byte contenente 1.

USO DELLA RAM DURANTE L'ESECUZIONE DEL PROGRAMMA

Quando il programma è in esecuzione, NSB8, mantiene due aree di memoria per dati, di dimensione variabile alla fine della memoria. Queste sono denominate, AREA DATI GENERALE e STACK DI CONTROLLO DEL BASIC. La prima area di memoria, comincia, immediatamente dopo, l'ultimo byte del programma corrente, e contiene la tabella dei simboli e la memoria statica per l'allocazione delle variabili, dei vettori e delle stringhe. Tale area di memoria, può crescere dalla memoria bassa a quella alta. La seconda area di memoria, cioè quella riguardante lo stack di controllo comincia al byte più alto che il BASIC, può indirizzare e decresce verso la memoria bassa.
Questo stack, contiene informazioni transienti, riguardanti chiamate a funzioni utente, a istruzioni di FOR-NEXT e GOSUB ecc..
Se il programma corrente, agisce su tale area di memoria, e provoca quindi dei problemi, il BASIC, risponde, con il messaggio di errore **MEMORY FULL ERROR**.

MESSAGGI DI ERRORE DEL BASIC NSB8

Nelle seguenti pagine, sono riportati tutti gli errori che il BASIC NSB8, riconosce.
Da notare, che per ogni errore viene presentato la sua denominazione, il suo codice di riconoscimento racchiuso tra parentesi, ed infine una descrizione delle possibili cause, che hanno provocato tale errore, in modo che l'utente, possa correggere il programma.

ARG ERROR (1)

E' stato fornito, un argomento, non valido, ad un comando o ad una funazione.

ARG MISMATCH ERROR (13)

Il numero di argomenti nella chiamata ad una funzione definita dall'utente, non corrisponde al numero di parametri richiesti dalla funzione stessa.

CONTINUE ERROR (non codificato)

E' stato fatto un tentativo illegale, per fare continuare, l'esecuzione del programma.
L'esecuzione non può continuare, se si è fermata precedentemente, causa il riconoscimento di un errore, o si è giunti all'esecuzione dell'istruzione END.

CONTROL STACK ERROR (non codificato)

Si provoca questo errore, quando nel programma, esiste un innesto errato di FOR-NEXT, di GOSUB-RETURN. Esso accade inoltre, quando FOR, è l'ultima istruzione del programma.

DIMENSION ERROR (2)

E'stato fatto un tentativo di ridimensionamento di un array o di una stringa, o è stata usata l'istruzione DIM, in modo non corretto.

DIVIDE ZERO ERROR (9)

È stata tentata l'esecuzione di una operazione di divisione per zero.

DOUBLE DEF ERROR (non codificato)

Esistono più definizioni di una stessa fuzione, nel programma. Le funzioni, sono definite, all'atto del RUN, perciò tale errore avviene prima che l'esecuzione inizi.

FILE ERROR (7)

Il programma ha provato ad accedere ad un file inesistente o di tipo incorretto oppure, è stata fatta una operazione di apertura del file, utilizzando un numero-file, già in uso. Questo errore, inoltre, accade, quando si tenta di caricare un programma BASIC, da un file di tipo 2, che non ha mai memorizzato, un programma BASIC o quando, vengono utilizzati numeri file, che sono maggiori di 7, o minori di 0. L'errore 7, si provoca anche, quando si effettuano tentativi di creare o di salvare file su dischetti già pieni o protetti in scrittura.

FORMAT ERROR (5)

È stato utilizzato un formato stringa illegale, in una istruzione PRINT, questo significa, che il formato stringa è formato in modo non corretto, o che il campo della specificazione, è troppo grande, o inconsistente. Inoltre, tale errore, viene provocato se si tenta di scrivere, un valore non intero usando il formato I, o un valore che non è inserito, in un campo specifico.

FUNCTION DEF ERROR (non codificato)

Questo significa, che il BASIC ha incontrato l'inizio di una nuova definizione di una funzione utente (istruzione DEF), ma non è stata chiusa, la precedente, mediante l'istruzione FNEND.
Tale errore, viene provocato inoltre, se si effettua un tentativo di chiamata ad una funzione-utente non definita.

HARD DISK ERROR (8)

È stato tentato, un accesso impossibile al disco.
Questo, può essere dovuto, ad un inserimento non corretto del disco nel drive, o ad un dischetto contenente dati illeggibili.

ILLEGAL DIRECT ERROR (non codificato)

È stato usata una istruzione, in modo diretto, che può essere eseguita solo se si trova in un programma.
Da ricordare, che le funzioni utente, non possono essere utilizzate, nel modo diretto.

INPUT ERROR (12)

Durante l'esecuzione di una istruzione di INPUT, è stata inserita una costante numerica illegale in risposta ad una richiesta del programma, di un input numerico.

INTERNAL STACK OV (non codificato)

Questo messaggio, non viene presentato nei programmi BASIC normali.
Una inanticipata quantità di memoria interna, è stata richiesta, per eseguire una istruzione o un comando.

LENGTH ERROR (16)

Questo errore viene provocato, se viene effettuato un tentativo di scrittura di una linea troppo lunga. Tale limite, può essere resettato usando l'istruzione LINE.
Tipicamente, questo errore avviene, quando si risponde ad una istruzione di INPUT, o quando si inseriscono. Come default, il BASIC, fissa, la lunghezza linea, a 80 caratteri.

LINE NUMBER ERROR (6)

C'è un numero di linea mancante , oppure non corretto.
Tale errore, può avvenire, anche quando è stato specificato in una istruzione o in un comando, un numero di linea, non presente nel programma corrente.

MEMORY FULL ERROR (non codificato)

La memoria non è sufficiente a contenere il programma corrente ed i suoi dati.
Il comando MEMSET può essere utilizzato per effettuare una espansione di tale memoria.
Notare che quando si effettuano concatenazioni di stringhe, il BASIC riserva un'area di memoria temporanea, grande quanto la concatenazione stessa.
Il BASIC NSB8 riserva inoltre, questa memoria temporanea, anche quando si effettuano operazioni di scrittura, perciò la stampa, di espressioni molto lunghe, può provocare tale errore.

MISSING NEXT ERROR (non codificato)

E' stato omesso il NEXT in un ciclo FOR.

NO PROGRAM ERROR (non codificato)

Questo errore avviene quando si tenta di effettuare una esecuzione senza però che esista il programma corrente.

NUMERIC OV ERROR (14)

Questo errore accade quando il risultato di una operazione aritmetica supera il valore 9.9999999E+62.
Da ricordare che i numeri minori di 1E-64, sono convertiti a 0.

OUT OF BOUNDS ERROR (3)

Questo errore, è causato da un argomento numerico che non è compreso nel range legale, per esempio, può accadere nelle istruzioni CALL, EXAM, FILL, INP o OUT, quando l'argomento non è compreso nel range corretto.
Si causa infine tale errore, se si tenta di effettuare scritture, o letture, dopo il fine file, di un file su dischetto.

READ ERROR (11)

È stato fatto un tentativo di lettura (READ), di un valore numerico, dove era richiesto invece una stringa, o viceversa.
Inoltre viene causato, quando non ci sono più dati disponibili, e viene effettuata una lettura (READ).

STOP (15)

Questo, non è realmente un errore, infatti, la sua comparsa, sta a significare che è stato premuto un control-C, mentre era in azione ERRSET.

SYNTAX ERROR (10)

Questo è l'errore più comune. Esso è provocato, quando nelle linee del programma, esistono istruzioni, che non hanno una sintassi corretta.

TOO LARGE OR NO PROGRAM ERROR (non codificato)

Si causa questo errore, quando si tenta di effettuare una delle seguenti operazioni: LOAD, APPEND o CHAIN, caricando un programma avente una dimensione, maggiore di quella memorizzabile, dall'area dati.
Viene inoltre causato, se le precedenti operazioni, vengono tentate, su programmi BASIC non validi.

TYPE ERROR (4)

Tale errore, accade, quando una stringa appare in una operazione che necessitava invece un valore numerico, o viceversa.
L'errore, è provocato inoltre, se si tenta di aprire un file, di tipo diverso da quello specificato, o se si legge un valore diverso dal tipo richiesto dalla variabile a cui verrà assegnato.

APPENDICE A: ESECUZIONE PROGRAMMI DA GDOS

Se si desidera creare un **"Command File"** (programma direttamente eseguibile) per il sistema operativo **GDOS**, utilizzando un programma scritto in NSB8, devono essere eseguiti i seguenti passi:

a) Da NSB8 si realizza il programma BASIC di cui creare l' autorun, lo si verifica in ogni aspetto eliminando tutti gli errori di sintassi, logica ecc. fino ad ottenere la versione definitiva dello stesso.

b) Da NSB8 si fornisce il comando:

PSIZE<CR>

con cui si ottiene la lunghezza del programma realizzato espresso in pagine.
Da ora in poi chiameremo SIZE_APPL questo valore da utilizzare nelle fasi sucessive.

c) Da NSB8 con il programma realizzato presente in memoria, si eseguono i seguenti comandi, sfruttando direttamente l'interprete di linea del BASIC:

FILL 3599,00<CR>
FILL 256,195<CR>
FILL 257,00<CR>
FILL 258,14<CR>

d) Si esce da NSB8 con il comando BYE portandosi quindi sotto il controllo del sistema operativo **GDOS**, indicato dall'apposito prompt **ABACO°>**.

e) Si fornisce il comando:

ABACO°>**SAVE nn d:nome.G80**<CR>

dove **nn** coincide con il risultato della somma **66+SIZE_APPL** espresso in **esadecimale**.
A questo punto, quando ricompare il prompt del sistema operativo locale, il programma autorun é stato realizzato e coincide con il file "nome.G80" creato sul drive d. Tale programma può essere verificato direttamente tramite il sistema operativo nella modalita` di esecuzione di un command file del **GDOS**.
Se ad esempio in questa fase si volesse creare un programma in Autorun in RAM Disk, supponendo che la lunghezza dello stesso programma BASIC sia SIZE_APPL=3, si dovra` fornire il comando:

ABACO°>**SAVE 45 M:AUTORUN.G80**<CR>

A questo punto, in corrispondenza di ogni power ON o reset della scheda remota, eseguirà automaticamente il programma di autorun impostato.

Vedi anche:

Comando **PSIZE**
Manuale del **GDOS**

APPENDICE B: GUIDA RAPIDA AL BASIC NSB8

PARAMETRI E FLAGS DEL BASIC NSB8

INDIRIZZO	CODIFICA	FUNZIONE	DEFAULT
3584	ENTRY1	Cancellazione programma	
3588	ENTRY2	Salvataggio programma	
3589	ENDBAS	Fine del BASIC	
3595	ENDMEM	Fine memoria	
3598	LINECT	Lunghezza linea iniziale	80
3599	AUTOS	Flag di auto-start	1
3600	PROM	Origine controllore disco	E8H
3601	LINETB	Stampa origine tabella	
3603	PAGES	Linee per pagina del CRT	24
3604	ENTRY3	Salvataggio dati	
3607	BSPC	Cancellazione carattere	8
3608	CONC	Flag di gestione control-C	0
3993	FILE-OPEN	Numero di file aperti (max 5)	0

SETTAGGIO MEMORIA PER NSB8

MEMSET VALORE		EQUIVALENTE IN HEX	MEMORIA DA 0H
MEMSET	24575	5FFF	24K
MEMSET	28671	6FFF	28K
MEMSET	32767	7FFF	32K
MEMSET	36863	8FFF	36K
MEMSET	40959	9FFF	40K
MEMSET	45055	AFFF	44K
MEMSET	49151	BFFF	48K
MEMSET	53247	CFFF	52K
MEMSET	57343	DFFF	56K
MEMSET	59391	E7FF	58K
MEMSET	61439	EFFF	60K
MEMSET	63487	F7FF	62K
MEMSET	65535	FFFF	64K

TABELLA DELLE ISTRUZIONI

NIBBLE PIU' SIGNIFICATIVO

NIBBLE MENO SIGNIFICATIVO	8	9	A	B	C	D	E	F
0	LET	FN	CLS	STEP			(	<=
1	FOR	DEF		TO			^	<>
2	PRINT	!		THEN		ATN	*	
3	NEXT	ON		TAB		FILESIZE	+	
4	IF	OUT		ELSE	SORT	FILEPTR		<
5	READ	FILL		CHR$		ADDR	-	=
6	INPUT	EXIT		ASC	INT			>
7	DATA	OPEN	APPEND	VAL			/	NOT
8	GOTO	CLOSE		STR$		FREE		
9	GOSUB	WRITE		NONENDMARK		INP		
A	RETURN			INCHAR$	SGN	EXAM		
B	DIM	CHAIN		FILE	SIN	ABS		
C	STOP	LINE			LEN	COS	AND	
D	END				CALL	LOG	OR	
E	RESTORE				RND	EXP		
F	REM	ERRSET				TYP	>=	

SINTASSI FILE PER NSB8

I file con i programmi eseguibili da NSB8 devono avere la seguente sintassi:

<nome file>.B

Dove <nome file> è una sequenza di 6 caratteri validi per il sistema operativo in uso.

SINTASSI FILE DATI DI NSB8

I file contenenti dei dati da gestire con NSB8 devono avere la seguente sintassi:

<nome file>.<est>

Dove <nome file> e <est>, sono due sequenze di, rispettivamente, 6 e 3 caratteri validi per il sistema operativo in uso.

OPERATORI ARITMETICI

OPERATORE	FUNZIONE	ESEMPIO
^	esponenziale	9^2=81
*	moltiplicatore	5*1.5=7.5
/	divisore	3/2=1.5
-	sottrattore	3.2-2=1.2
+	sommatore	7.9+2.1=10
-	negatore	-3, -27

OPERATORI RELAZIONALI

OPERATORE	RELAZIONE	ESEMPI
>	maggiore di	(6>1)=1 (vero) (2>3)=0 (falso)
<	minore di	(0<0)=0 (falso) (1<3)=1 (vero)
<=	minore o uguale a	(5<=5)=1 (3<=5)=1 (6<=5)=0
>=	maggiore o uguale a	(8>=7)=1 (7>=7)=1 (6>=7)=0
=	uguale a	(9=9)=1 (9=7)=0
<>	diverso da	(4<>5)=1 (2<>2)=0

OPERATORI BOOLEANI

AND
OR
NOT

ORDINE DI VALUTAZIONE DEGLI OPERATORI

NOT , -	Operatore logico, Negazione di un numero
^	Esponenziale
*** , /**	Moltiplicatore e Divisore
+ , -	Sommatore e Sottrattore
= , < , > ,<= , >= , <>	Operatori relazionali
AND	Operatore logico
OR	Operatore logico

FORMATO DI STAMPA

F format	nFm
I format	nI
E format	nEm

Sintassi da seguire:

PRINT %<formato carattere><specificazione formato>,<variabili>

FORMATO CARATTERI

A	Conteggio
C	Virgola
Z	Omissione degli 0
+	Numero positivo
$	Segno di dollaro
#	Formato di default

EDITOR DI LINEA NSB8

Control **G**	Copia fino alla fine della linea
Control **N**	Cancella per una nuova scrittura
Control **A**	Copia un carattere
Control **Q**	Indietreggia di un carattere
Control **Z**	Cancella un carattere
Control **D X**	Copia sino al carattere X
Control **Y**	Attiva e disattiva il modo insert

REALIZZAZIONE DI UNA FUNZIONE INTERNA

FUNZIONI MATEMATICHE

ABS	**ABS**olute value (<espressione>)
ATN	**A**rc**T**a**N**gent (<espressione numerica>)
COS	**COS**ine (<espressione numerica>)
EXP	**EXP**onential value (<espressione numerica>)
INT	**INT**eger value (<espressione numerica>)
LOG	**LOG**arithmic value (<espressione numerica>)
SIGN	**SIGN** of a number (<espressione numerica>)
SIN	**SIN**e (<espressione numerica>)
SQRT	**SQ**uare **R**oo**T** (<espressione numerica>)

FUNZIONI PER STRINGHE

ASC	**ASC**ii value (<espressione di tipo stringa>)
CHR$	**CHR**aracter**$** (<espressione numerica>)
LEN	**LEN**gth of a string (<nome stringa>)
STR$	**STR**ing**$** (<espressione numerica>)
VAL	**VAL**ue (<espressione di tipo stringa>)

FUNZIONI PER INPUT

INCHAR$	**IN**put a **CHAR**acter$ (<dispositivo#>)
INP	**INP**ut a byte (<numero port>)

FUNZIONI PER FILE

FILE	**FILE** type (<nomefile>)
FILEPTR	**FILEP**oin**T**e**R** position (<canale#>)
FILESIZE	**FILESIZE** (<canale#>)
TYP	**TYP**e of file pointer (<canale#>)

FUNZIONI DI UTILITY

CALL	**CALL** machine language (<indirizzo memoria> [,<argomento>])
EXAM	**EXAM**ine memory (<locazione memoria>)
FREE	**FREE** memory (<argomento>)
TAB	**TAB**ulate (<#espressione>)
RND	**R**a**ND**om (<#espressione>)

FUNZIONI DEFINITE DALL' UTENTE

DEF <NOME FUNZIONE> (<LISTA PARAMETRI) [=<ESPRESSIONE>]
RETURN <VARIABILE NUMERICA O DI TIPO STRINGA>
FNEND

MNEMONICO

LINE#	Numero linea BASIC
DEVICE#	Numero di un dispositivo di input o output
DRIVE#	Numero del disk drive
FILENAME	Nome del file
#EXPR	Espressione numerica
LOGEXPR	Espressione logica
TYPEXPR	Un tipo di espressione
CHANNEL#	Numero canale del disco
[ ]	Ciò che racchiude è opzionale

COMANDI DIRETTI

ALOAD <FILE NAME>
ASCSAV <FILENAME>
AUTO [<LINE>] [,<INCREMENTAL VALUE>]
BYE
CAT [#<DEVICE#>] [,<DRIVE#>]
CONT
DEL <LINE#>,<LINE#>
LIST [#<DEVICE#>] [,<LINE#>] [,<LINE#>]
LOAD <FILENAME>
MEMSET <MEMORY ADDRESS>
PSIZE
REM [<LINE#>] [,<INCREMENTAL VALUE>]
RUN [<LINE#>]
SAVE <FILENAME>
SCR

ISTRUZIONI

***** PUO' ESSERE USATO COME UN COMANDO IN MODO DIRETTO

ISTRUZIONI PER MANIPOLAZIONE DATI INTERNI AL PROGRAMMA

DATA <LISTA DI COSTANTI>
READ <LISTA DI VARIABILI>
***RESTORE** [<LINE#>]

ISTRUZIONI DI INPUT E OUTPUT

INPUT [<DEVICE#>] [,<STRING PROMPT>,] <VARIABLE LIST>
INPUT1 [<DEVICE#>] [,<STRING PROMPT>,] <VARIABLE LIST>
***OUT** <PORT NUMBER>,<BYTE VALUE>
***PRINT** [@,(ROW,COLUMN)][#<DEVICE#>] [,<LIST OF EXPRESSIONS>]
***!** [@(ROW,COLUMN)] [#<DEVICE#>] [,<LIST OF EXPRESSION>]

ISTRUZIONI DI CONTROLLO

FOR <CONTROL>=<INITIAL> **TO** <LIMIT> [**STEP** <VALUE>]
GOSUB <LINE#>
GOTO <LINE#>
***IF** <LOGEXPR> **THEN** <STATEMENT> [**ELSE** <STATEMENT>]
ON <#EXPR> **GOSUB** <LIST OF LINE NUMBERS>
ON <#EXPR> **GOTO** <LIST OF LINE NUMBERS>
RETURN

ISTRUZIONI PER FILE

***APPEND** [LINE#>,]<FILENAME>
***CHAIN** <FILENAME>
***CLOSE #**<CHANNEL#>
***OPEN #**<CHANNEL#>[%<TYPEXPR>],<FILENAME>[,<SIZEVAR>]
***READ# #**<CHANNEL#>[%<RANDOM ADDRESS>],<LIST OF VARIABLES>
***WRITE# #**<CHANNEL#>[%<RANDOM ADDRESS>],<LIST OF EXPRESSIONS>

ISTRUZIONI VARIE

***CLS** [#<DISPOSITIVO#>]
***DIM** <NOME VARIABILE (<DIMENSIONE DELL' ARRAY O DELLA STRINGA>)
END
ERRSET [<LINEA#>,<NUMERO LINEA D'ERRORE>,<NUMERO D' ERRORE>]
***FILL** <LOCAZIONE>,<VALORE DEL BYTE>
***[LET]** <VARIABILE NUMERICA>=<ESPRESSIONE NUMERICA>
***[LET]** <VARIABILE STRINGA>=<ESPRESSIONE DI TIPO STRINGA>
***LINE** [#<DISPOSITIVO#>,]<#ESPRESSIONE>[,<#ESPRESSIONE>]
REM <QUALSIASI LINEA DEL PROGRAMMA>
STOP

ERRORI RICONOSCIBILI (TRAPPABLE ERRORS)

ARGument	Error 1
DIMENSION	Error 2
OUT OF BOUNDS	Error 3
TYPE	Error 4
FORMAT	Error 5
LINE NUMBER	Error 6
FILE	Error 7
HARD DISK	Error 8
DIVIDE by ZERO	Error 9
SYNTAX	Error 10
READ	Error 11
INPUT	Error 12
ARGument MISMATCH	Error 13
NUMERIC OVerflow	Error 14
STOP/control C	Error 15
LENGTH	Error 16

APPENDICE C: INDICE ANALITICO

A

ABS, funzione matematica **62**
ACCESSO DEI DATI **95**
ACCESSO RANDOM DEI DATI **98**
ACCESSO SEQUENZIALE **95**
ACCESSO SEQUENZIALE AI BYTE **97**
AGGIUNTA DI NUOVI DATI IN UN FILE SEQUENZIALE **97**
ALOAD, comando **1**
ALTRI FORMATI CARATTERI **86**
ANNIDAMENTI DI CICLI FOR-NEXT **89**
APERTURA DEI FILES **94**
APERTURA E CHIUSURA DI UNA SOTTO-STRINGA **79**
APPEND, comando **2**
ARG ERROR, messaggio di errore **119**
ARG MISMATCH ERROR, messaggio di errore **119**
ARGOMENTI **61**
ARITMETICA IN BCD DEL BASIC NSB8 **109**
ASC, funzione stringa **63**
ASCSAVE, comando **3**
ASSEGNAMENTI A STRINGHE E A SOTTO-STRINGHE **81**
ATN, funzione matematica **62**
ATTIVARE O DISATTIVARE IL MODO DI INSERZIONE (Control-Y) **107**
AUTO, comando **4**

B

BASIC PERSONALIZZATO **111**
BASIC RISTRETTO, passo 8 **114**
BYE, comando **5**

C

CALL, funzione **65**
CANCELLARE E REINSERIRE UNA NUOVA LINEA (Control-N) **107**
CANCELLAZIONE DI UN CARATTERE DALLA OLD LINE (Control-Z) **106**
CARATTERE DI “BACKSPACE”, passo 5 **113**
CAT, comando **6**
CHAIN, istruzione **21**
CHIUSURA DEI FILE **94**
CHR$, funzione stringa **63**
CLOSE, istruzione **22**
COMPARAZIONE TRA STRINGHE **80**
COMPATIBILITA’ CON ALTRI BASIC **108**
COMUNICAZIONE TRA PROGRAMMI **101**

segue C

CONCATENAZIONE (Sequenza automatica del programma) **101**
CONCATENAZIONE DI STRINGHE **79**
CONT, comando **8**
CONTINUE ERROR, messaggio di errore **119**
CONTROL STACK ERROR, messaggio di errore **119**
CONTROL-C **59**
COPIA DEL RESTO OLD LINE ALLA FINE DELLA NEW LINE (Control-G) **105**
COPIA DI UN CARATTERE DALLA OLD LINE (Control-A) **106**
COPIA SU UNO SPECIFICO CARATTERE (Control-D) **106**
CORPO DEL CICLO **88**
COS, funzione matematica **62**
COSA RAPPRESENTA IL FORMATO DI UN NUMERO ? **83**
COSTANTI **69**
COSTANTI DI TIPO STRINGA **77**

D

DATA, isrtuzione **23**
DEF, istruzione **24**
DEL, comando **9**
DICHIARAZIONI DELLE STRINGHE **108**
DIM, istruzione **25**
DIMENSION ERROR, messaggio di errore **119**
DIMENSIONAMENTO DI DEFAULT **75**
DIMENSIONAMENTO VARIABILI DI TIPO STRINGA **78**
DIMENSIONI DEL BUFFER DEL FILE - “TEMPO DI VITA” DEI BUFFERS **117**
DISPOSITIVI DI I/O MULTIPLI **60**
DIVIDE ZERO ERROR, messaggio di errore **119**
DOUBLE DEF ERROR, messaggio di errore **119**

E

END, istruzione **26**
ESECUZIONE PROGRAMMI DA GDOS, appendice A **123**
ESEMPI DI ESPRESSIONI NUMERICHE ILLEGALI **72**
ESEMPI DI ESPRESSIONI NUMERICHE LEGALI **72**
ESPRESSIONI **72**
ESPRESSIONI DI TIPO STRINGA **79**
EXAM, funzione **64**
EXIT, istruzione **28**
EXP, funzione matematica **62**

F

FILE ERROR, messaggio di errore **120**
FILE, funzione per File **64**
FILL, istruzione **29**
FISSARE UNA VARIABILE COME ORIGINE DEL BASIC, passo 2 **112**
FLUSSO DI ESECUZIONE E DI CONTROLLO **87**
FNEND, istruzione **31**
FOR, istruzione **32**
FORMA INTERNA DI UN PROGRAMMA **118**
FORMAT ERROR, messaggio di errore **120**
FORMATI DELLA MEMORIA-DATI DEI DISCHETTI **116**
FORMATO DI DEFAULT E FORMATO CORRENTE **85**
FORMATO DI STAMPA **83**
FRAZIONI **69**
FREE, funzione **64**
FUNCTION DEF ERROR, messaggio di errore **120**
FUNZIONI DI CARATTERE GENERALE **64**
FUNZIONI INTERNE **61**
FUNZIONI MATEMATICHE **62**
FUNZIONI PER LA MANIPOLAZIONE DI FILE SU DISCO **64**
FUNZIONI PER OPERAZIONI SU STRINGA **63**
FUNZIONI PER PARTICLARI INPUT **63**
FUNZIONI SU STRINGHE **79**
FUNZIONI-UTENTE **65**
FUNZIONI-UTENTE COMPOSTE DA PIU' LINEE **67**
FUNZIONI-UTENTE COMPOSTE DA UNA SOLA LINEA **66**

G

GLI ERRORI E LE SOLUZIONI **102**
GOSUB, istruzione **33**
GOTO, istruzione **34**
GUIDA RAPIDA AL BASIC NSB8, appendice B **125**

H

HARD DISK ERROR, messaggiop di errore **120**

I

I FILE DATI **93**
I NUMERI **69**
I VETTORI **74**
I VETTORI NON POSSONO ESSERE RIDIMENSIONATI **76**
IF, istruzione **35**
IL BOOTSTRAP PROM NON STANDARD, passo 7 **113**
IL COMANDO EDIT **104**
IL LOOP FOR-NEXT **88**

segue I

ILLEGAL DIRECT ERROR, messaggio di errore **120**
IMPAGINAZIONE VIDEO, passo 4 **112**
INCHAR$, funzione di Input **63**
INDICI **74**
INDIRIZZI SPECIALI DI NSB8 **110**
INIBIZIONE DEL CONTROL-C, passo 6 **113**
INP, funzione di Input **63**
INPUT ERROR, messaggio di errore **120**
INPUT, istruzione **36**
INPUT1, istruzione **38**
INSERIMENTO OPZIONALE DELLA VARIABILE DI CONTROLLO **90**
INT, funzione matematica **62**
INTERNAL STACK OV, messaggio di errore **120**
INTRODUZIONE ALL'EDITOR **103**

L

LA LINEA DI EDITOR **103**
LA STRINGA NULLA **77**
LA VALUTAZIONE IF ... THEN **109**
LA VARIABILE DI CONTROLLO ED IL VALORE LIMITE **88**
LE FUNZIONI **61**
LE PROCEDURE **92**
LE STRINGHE **77**
LEN, funzione stringa **63**
LENGTH ERROR, messaggio di errore **121**
LET, istruzione **39**
LINE, istruzione **40**
LINE NUMBER ERROR, messaggio di errore **121**
LIST, comando **10**
LOAD, comando **11**
LOG, funzione matematica **62**
LUNGHEZZA DI LINEA, passo 3 **112**

M

MANIPOLAZIONE DELLE STRINGHE **108**
MASSIMA LUNGHEZZA DI STRINGHE E LUNGHEZZA CORRENTE **82**
MEMORY FULL ERROR, messaggio di errore **121**
MEMORY SIZE, passo 1 **111**
MEMSET, comando **13**
MESSAGGI DI ERRORE DEL BASIC NSB8 **119**
MISSING NEXT ERROR, messaggio di errore **121**

N

NEXT, istruzione **42**
NO PROGRAM ERROR, messaggio di errore **121**
NOME DEI FILES **93**
NOME DELLE FUNZIONI-UTENTE **65**
NOTE DI IMPLEMENTAZIONE **116**
NOTE FINALI **68**
NUMERIC OV ERROR, messaggio di errore **121**
NUMERO DI DECIMALI **84**

O

ON, istruzione **43**
OPEN, istruzione **44**
OPERATORI **71**
OPERATORI ARITMETICI **71**
OPERATORI BOOLEANI **73**
OPERATORI RELAZIONALI **72**
ORDINE DI VALUTAZIONE DEGLI OPERATORI **73**
OUT, istruzione **46**
OUT OF BOUNDS ERROR, messaggio di errore **121**

P

PASSAGGIO PARAMETRI ALLE FUNZIONI-UTENTE **66**
PERSONALIZZAZIONE DEL BASIC NSB8 **111**
PRECISIONE **69**
PRENDERE UN CARATTERE (Control-Q) **106**
PRINT, istruzione **47**
PROCEDURE IN LINGUAGGIO MACCHINA **100**
PROGRAMMI BASIC PRE-ESEGUITI **117**
PSIZE, comando **14**

R

RAPPRESENTAZIONE A DESTRA **84**
READ ERROR, messaggio di errore **122**
READ, istruzione **51**
READ#, istruzione **49**
REM, istruzione **52**
REN, comando **15**
RESTORE, istruzione **53**
RETURN, istruzione **54**, **55**
RIFERIMENTI AD ARRAY NELLE ESPRESSIONI NUMERICHE **76**
RND, funzione **64**
RUN, comando **17**

S

SAVE, comando **18**
SCR, comando **20**
SETTAGGIO CARATTERI IN BASIC **82**
SGN, funzione matematica **62**
SIN, funzione matematica **62**
SOTTOSTRINGHE **78**
SPAZIO OCCUPATO DAI FILE (LUNGHEZZA) **94**
SPECIFICHE E FUNZIONI DEL LINE EDITOR **105**
SQRT, funzione matematica **62**
STAMPA DI NUMERI IN FORMATO NORMALE ED ESPONENZIALE **83**
STOP, istruzione **56**
STOP, messaggio di errore **122**
STR$, funzione stringa **63**
SYNTAX ERROR, messaggio di errore **122**

T

TAB, funzione **65**
TABELLA DEI DISPOSITIVI DI STAMPA **117**
TABELLA DI GESTIONE DEI FILE **117**
TABELLA DI RIFERIMENTO DEGLI INDIRIZZI DI NSB8 **115**
TIPI DI DATI NEI FILES **95**
TIPO DEI FILES **94**
TOO LARGE OR NO PROGRAM ERROR, messaggio di errore **122**
TRASLAZIONE DEGLI INPUT **109**
TYP, funzione per File **64**
TYPE ERROR, messaggio di errore **122**

U

USCITA DA LOOP INNESTATI **91**
USO DELLA RAM DURANTE L'ESECUZIONE DEL PROGRAMMA **118**
USO DI EXIT **90**

V

VAL, funzione stringa **63**
VALORE DEL PASSO (OPZIONALE) **88**
VARIABILI **71**
VARIABILI DI TIPO STRINGA **77**
VETTORI A PIU' DIMENSIONI: MATRICI **74**
VETTORI DI TIPO STRINGA **108**

W

WRITE, istruzione **57**