

**ADRIAN I. VLAD**

**BASIC**

**PROGRAMME  
DE  
INSTRUIRE**

ing. Adrian I. Vlad

**PROGRAMME DE INSTRUIRE**

PROGRAMME  
DE  
INSTRUIRE



© 1988 by Editura Tehnică  
București, România

**Nota editurii**

Această lucrare a fost elaborată, tehnoredactată și culeasă în întregime cu ajutorul calculatorului.

ing. ADRIAN I. VLAD

# PROGRAME DE INSTRUIRE



EDITURA ȘTIINȚIFICĂ ȘI ENCICLOPEDICĂ

București, 1989

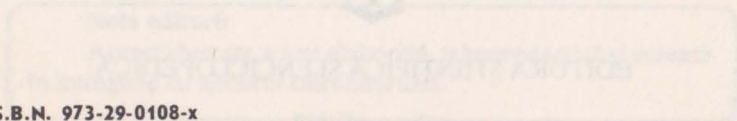


de ADRIAN I. VLAD

PROGRAMME

DE

INSTRUIRE



I.S.B.N. 973-29-0108-x

## CUVÎNT ÎNAINTE

În mai puțin de cinci decenii de la apariția lor, mijloacele tehnice moderne de calcul electronic au devenit instrumente obișnuite de lucru în cele mai diverse domenii ale activității umane. Aceasta a fost posibilă atât datorită creșterii cu totul deosebite a performanțelor cât și a dezvoltării concomitente a unei diversități de limbaje de programare cu ajutorul cărora s-a realizat o gamă largă de programe de aplicații. Între aceste limbaje de programare, un loc aparte îl ocupă limbajul BASIC (Beginners All-purpose Symbolic Instruction Code).

Ușor de învățat și simplu de utilizat, acest limbaj este implementat pe aproape toate tipurile de calculatoare, permițând rezolvarea unor probleme dintre cele mai diverse și complexe.

În lume sînt realizate, în prezent, numeroase dialecte ale limbajului BASIC. Lucrarea de față prezintă cea mai uzuală versiune (BASIC Microsoft), pornind de la considerentul că acest dialect este foarte larg răspîndit.

Limbajul BASIC este implementat pe numeroase tipuri de calculatoare produse de cele mai cunoscute firme, inclusiv pe calculatorul FELIX-PC realizat de Întreprinderea de Calculatoare București.

Această carte se dorește a fi un instrument de lucru la îndemîna utilizatorilor calculatoarelor personale. Pornind de la acest scop, se vor prezenta în cele cinci capitole ale lucrării atât informațiile generale necesare în procesul programării cât și instrucțiunile BASIC în ordine alfabetică. În acest fel va fi ușurată atât lectura cât și identificarea rapidă a informațiilor utile în timpul sesiunii de lucru la calculator.

Lucrarea a fost în întregime realizată pe calculator, iar exemplele prezentate au fost atent verificate și rulate pentru a se elimina eventualele erori.

Autorul



# CUPRINS

<b>CUVÎNT ÎNAINTE</b> .....	5
<b>1. SESIUNEA DE LUCRU BASIC</b> .....	9
1.1. Opțiuni în linia de comenzi BASIC .....	9
1.2. Redirecționarea intrării și ieșirii standard ....	11
<b>2. MODURI DE OPERARE</b> .....	13
2.1. Modul direct .....	13
2.2. Modul program .....	13
<b>3. INFORMAȚII GENERALE DESPRE PROGRAMAREA IN BASIC</b> .....	14
3.1. Formatul liniei .....	14
3.2. Constante .....	15
3.3. Precizia numerică .....	15
3.4. Variabile .....	16
3.5. Conversia numerelor dintr-o precizie în alta .	16
3.6. Matrici .....	18
3.7. Expresii numerice și operatori .....	19
3.8. Operatori aritmetici .....	19
3.9. Operatori relaționali .....	20
3.10. Operatori logici .....	21
3.11. Funcții numerice .....	22
3.12. Expresii și operatori șir .....	23
<b>4. INTRĂRI ȘI IEȘIRI</b> .....	24
4.1. Fișiere .....	24
4.2. Numele de fișier .....	24
4.3. Numele fișierului .....	25
<b>5. COMENZI, INSTRUCȚIUNI ȘI FUNCȚII BASIC</b> .	26
<b>BIBLIOGRAFIE</b> .....	129
<b>INDEX</b> .....	131





# 1. SESIUNEA DE LUCRU BASIC

## 1.1. Opțiuni în linia de comenzi BASIC

O sesiune de lucru BASIC este perioada de timp în care se lucrează sub acest limbaj de programare. Având în vedere că interpretorul sau compilatorul limbajului este rezident pe disc, este necesar ca la debutarea sesiunii de lucru să se apeleze limbajul de pe suportul magnetic în memoria calculatorului, iar la sfârșitul sesiunii să se execute întoarcerea la sistemul de operare.

Apelarea limbajului se face din linia de comandă a sistemului de operare cu numele lui (BASIC, BASICA, GWBASIC, etc). Numele acestuia poate fi urmat de opțiunile:

```
BASIC [numefis] [stdin] [<] [stdout] [/F:files] [/S:bsize]  
[C:combuffer] [/M:[max workspace] [,max blockspace]] [/D]
```

unde:

**numefis** rezezintă numele programului care va fi încărcat în memorie și rulat imediat. Dacă nu se va găsi o altă extensie se va lua .BAS drept extensie de fișier, aceasta permițând programelor BASIC să fie rulate în modul "batch processing" prin punerea acestei linii de comandă în fișierul AUTOEXEC.BAT.

**stdin** un program BASIC are drept periferic de intrare, în mod normal, claviatura. Utilizarea opțiunii **stdin** permite limbajului să recepționeze intrarea de la un fișier specificat.

**stdout** un program BASIC atribuie ecranul drept periferic de ieșire. Utilizând opțiunea **stdout** se informează limbajul asupra perifericului sau fișierului de ieșire.

• **/F:files** atribuie numărul maxim de fișiere ce pot fi deschise simultan în timpul rulării unui program BASIC. Fiecare fișier necesită 62 octeți de memorie pentru "blocul de control al fișierelor" (FCB), plus 128 octeți pentru memoria tampon de date. Dimensiunea memoriei tampon poate fi modificată cu ajutorul opțiunii **/S:**. În cazul în care opțiunea **/F:** este omisă, numărul prestabilit de fișiere este 3.

**/S:bsize** atribuie mărimea memoriei tampon pentru fișierele de tip

"aleator". Lungimea parametrului "record" în instrucțiunea OPEN nu poate depăși această valoare. Mărimea memoriei tampon standard este de 128 octeți. Valoarea maximă acceptată este de 32767 octeți.

**/C:commbuffer** atribuie mărimea tamponului de memorie pentru recepția datelor în cazul utilizării "Adaptorului de comunicații asincrone" (Asynchronous Communications Adapter). Această opțiune nu are nici un efect atîta timp cît nu există atașat sistemului acest adaptor. Memoria tampon de transmisie de date este de 128 octeți. Valoarea maximă care se poate atribui este 32767. Dacă se omite opțiunea **/C:commbuffer**, pentru tamponul de memorie de recepție se vor atribui 256 octeți. În cazul în care sistemul are atașată o linie de transmisie de mare viteză se poate utiliza opțiunea **/C:1024**.

**/M:max workspace** atribuie valoarea maximă a locației care va fi utilizată de către limbaj. Limbajul BASIC atribuie 64Ko de memorie pentru date și stivă. Dacă se intenționează să se utilizeze subrutine în limbaj masină, se recomandă utilizarea opțiunii **/M:** în linia de comandă BASIC. De exemplu, **/M:32768** alocă 32768 octeți pentru datele și stiva programului BASIC și rezervă 32768 octeți pentru rutinele limbaj masină.

**:max blockspace** specifică numărul maxim de paragrafe alocate pentru BASIC, plus spațiul de memorie aflat deasupra stivei și care poate fi utilizat pentru rutinele în limbaj masină. Paragrafele sînt blocuri de cîte 16 octeți fiecare. Dacă **:max blockspace** este omis, se atribuie valoarea **&H1000 (4096)** și, deci, 65536 (4096\*16) octeți pentru zona de date și stiva BASIC. Spre exemplificare, dacă se doresc 65535 octeți pentru BASIC și 512 octeți pentru subrutinele în limbaj masină, se va utiliza **/M:,&H1020** (4096 paragrafe pentru BASIC + 32 paragrafe pentru rutinele limbaj masină).

**/D** indică limbajului că funcțiile matematice dublă precizie vor rămîne rezidente. La specificarea opțiunii **/D** aproximativ 3000 octeți vor rămîne rezidenți în BASIC pentru utilizarea funcțiilor transcendente. Funcțiile care pot fi convertite în dublă precizie sînt: ATN, COS, EXP, LOG, SIN, SQR și TAN.

Cîteva exemple de linii de comandă BASIC:

**BASIC prog1.bas**

încarcă și execută fișierul **prog1.bas**.

**BASIC prog2.bas/F:6**

încarcă și execută programul **prog2.bas** utilizînd 64 Ko de memorie și 6 fișiere.

**BASIC /C:0/M:32768**

dézafectează interfața RS232 și utilizează 32 Ko pentru spațiul de memorie de lucru BASIC.



În lucrare, la prezentarea sintaxei diferitelor instrucțiuni ori comenzi, s-au folosit următoarele convenții:

- cuvintele scrise cu litere mari sînt cuvinte cheie și trebuie introduse așa cum sînt prezentate, cu precizarea că ele pot fi scrise cu litere mari sau mici sau cu orice combinație a lor. Limbajul BASIC face conversia literelor mici în litere mari, exceptînd cazul în care ele fac parte dintr-un șir de caractere sau o instrucțiune DATA.
- orice termeni prezentați mai sus cu litere mici trebuie înlocuiți cu o valoare.
- termenii prezentați între paranteze drepte [ ] sînt opționali.
- parantezele rotunde ( ) indică faptul că termenii interiori pot fi repetați de cîte ori este nevoie.
- toate semnele de punctuație cu excepția parantezelor pătrate (virgule, paranteze, punct și virgulă, sau semnul egal), trebuiesc incluse acolo unde sînt indicate.

## 1.2 Redirecționarea intrării și ieșirii standard

Limbajul BASIC permite ca intrarea și/sau ieșirea standard să fie redirecționată. De obicei, pentru intrare se folosește claviatura, iar pentru ieșire ecranul. Acestea pot fi înlocuite, după dorință, cu un fișier sau un alt periferic de intrare sau ieșire. Pentru aceasta sintaxa este:

**BASIC numefis[ < stdin][ < ][ < stdout]**

- în timpul redirecționării, instrucțiunile INPUT, INPUT\$, INKEY\$ și LINE INPUT vor citi datele de la fișierul sau perifericul de intrare, în loc de claviatură.
- instrucțiunile PRINT și eventualele mesaje de eroare se vor scrie pe fișierul sau perifericul de ieșire, în locul ecranului.
- cînd un fișier este redirecționat doar pentru ieșire, toate datele ce apar pe ecran vor fi trimise la fișierul de ieșire specificat.
- cînd se redirecționează doar ieșirea, mesajele de eroare vor merge atît la ecran cît și la fișierul de ieșire, după care vor fi închise toate fișierele, programul se va termina, iar controlul va fi preluat de către sistemul de operare DOS.
- fișierul de intrare "KYBD:" va citi datele de la claviatură.
- fișierul de ieșire "SCRN:" va afișa datele pe ecran.
- comanda Ctrl-Break va acționa la ieșirea standard, va închide toate fișiere și va da controlul sistemului de operare DOS.

Cîteva exemple de redirecționare:

**BASIC prog > date.out**

În exemplul de mai sus, ieșirea va fi redirecționată spre fișierul **date.out**,

înlocuindu-se astfel ecranul.

### **BASICA unprog < date.in**

În acest exemplu, instrucțiunile de intrare (INPUT, INPUT\$, LINE INPUT și INKEY\$) vor fi primite de la fișierul **date.in**, înlocuindu-se claviatura.

### **BASIC unprog < date.in > prog.fin**

Datele citite cu instrucțiunile INPUT, INPUT\$, INKEY\$ sau LINE INPUT vor fi recepționate de la fișierul **date.in**, iar datele scrise cu instrucțiunea PRINT vor fi adăugate fișierului **prog.fin**.

## 2. MODURI DE OPERARE

În momentul inițializării, limbajul BASIC va afișa pe ecran mesajul **Ok**, indicînd faptul că este gata pentru recepționarea instrucțiunilor. Această stare se numește nivel de comandă. Comunicația cu BASIC-ul se poate realiza în două moduri: modul direct și modul program.

### 2.1. Modul direct

În modul direct, se poate instrui limbajul să execute o comandă imediat după ce a fost introdusă. Aceasta se realizează prin omiterea numărului de linie. Astfel, se pot afișa rezultatele unor operații aritmetice sau logice sau se pot memora pentru utilizarea lor ulterioară, singura remarcă fiind că instrucțiunile respective nu sînt memorate după afișarea rezultatului. Acest mod este avantajos pentru depanarea programelor, cît și pentru unele calcule rapide, care nu necesită realizarea unui program. De exemplu:

```
PRINT 10*4.52  
45.2
```

### 2.2. Modul program

În modul program (sau modul indirect), se instruește limbajul BASIC că liniile introduse alcătuiesc un program. Pentru aceasta se va începe o linie cu un număr de linie. Astfel, această linie va fi memorată ca parte a unui program în memorie. Programul astfel memorat poate fi rulat cu comanda RUN. De exemplu:

```
10 PRINT 10*4.52  
RUN  
45.2
```



## 3. INFORMAȚII GENERALE DESPRE PROGRAMAREA ÎN BASIC

### 3.1. Formatul liniei

O linie BASIC are următorul format general:

*nnnnn instrucțiune BASIC[: instrucțiune BASIC...][' comentariu]*

și se termină cu caracterul ENTER.

**nnnnn** reprezintă numărul liniei de program BASIC și este un număr de 1 pînă la 5 cifre lungime. Acesta indică ordinea în care au fost memorate liniile de program BASIC și servește, de asemenea, drept referința pentru salturile de program precum și pentru editare. Numărul liniei poate fi cuprins între 0 și 65529.

**instrucțiune BASIC** O instrucțiune BASIC poate fi executabilă sau neexecutabilă. Instrucțiunile executabile indică limbajului ce acțiune trebuie îndeplinită în momentul rulării. De exemplu instrucțiunea PRINT este o instrucțiune executabilă. Instrucțiunile neexecutabile ca de exemplu DATA sau REM, conțin doar informații și nu pot cauza o anumită acțiune directă în timpul rulării. Toate instrucțiunile BASIC vor fi descrise în capitolele viitoare.

După dorință, o linie poate include mai mult de o instrucțiune, dar fiecare instrucțiune trebuie separată de celelalte prin două puncte (:), cu mențiunea că o linie de program nu poate avea mai mult de 255 de caractere inclusiv numărul liniei. De exemplu:

```
10 FOR I = 1 TO 5: PRINT I: NEXT I
RUN
1
2
3
4
5
```

comentariu Comentariul poate fi inclus la sfîrșitul unei linii. Simbolul ' separă comentariul de restul liniei.

## 3.2. Constante

Constantele sînt valori ce se utilizează în timpul elaborării unui program, dar care nu se schimbă în timpul execuției unui program. Acestea pot fi: constante șir (de caractere) și constante numerice.

O constanta șir este o secvență de pînă la 255 de caractere incluse între ghilimele ("). Caracterele pot fi litere, cifre sau simboluri. Exemple de constante șir de caractere:

"Abcd"

"a12.123"

"1"

"2#\$\$%&"

Constantele numerice sînt numere pozitive sau negative. Constantele numerice în BASIC nu pot conține virgula zecimală drept delimitator ci (după notația engleză) punctul. Există cinci moduri de reprezentare a constantelor numerice:

**întregi** toate numerele între -32768 și +32767. Constantele întregi nu pot conține punctul zecimal.

**virgulă fixă** numerele reale pozitive sau negative.

**virgulă mobilă** numerele pozitive sau negative reprezentate în forma exponențială (similară notației științifice). În calculele în simplă precizie o constantă în virgulă mobilă constă dintr-un întreg pozitiv sau negativ sau un număr în virgulă fixă (mantissa) urmată de litera E, urmată de un întreg pozitiv sau negativ (exponentul). Constantele de tip dublă precizie utilizează litera D în locul lui E. E sau D înseamnă "ori zece la puterea". Se poate reprezenta, în acest mod, orice număr pozitiv sau negativ cuprins între  $2.9E-39$  la  $1.7E+38$ .

## 3.3. Precizia numerică

Intern, numerele pot fi memorate ca numere întregi, simplă precizie sau dublă precizie. Constantele declarate în format întreg, hexazecimal sau octal sînt memorate în doi octeți și sînt interpretate ca numere întregi. În calculele de dublă precizie, numerele sînt memorate cu 17 cifre precizie și sînt afișate cu pînă la 16. În calculele de simplă precizie se memorează 7 cifre și se afișează pînă la 7 cifre, din care doar 6 pot fi de această precizie. Exemple de constante de simplă și dublă precizie:

**simplă precizie**

46.8

-1.09E-6

3489.0

**dublă precizie**

235468274

-1.09432D-6

11.22334567871346



### 3.4. Variabile

Variabilele sînt nume utilizate pentru a reprezenta valori în programele BASIC. Ca și în cazul constantelor, există două tipuri de variabile: numerice și șir (de caractere). O variabilă numerică are totdeauna atașată o valoare numerică. Unei variabile șir i se atribuie drept valoare un șir de caractere. Șirul poate avea pînă la 255 de caractere.

Unei variabile i se poate atribui o valoare fixă sau rezultată în urma unor calcule. În orice caz tipul de variabilă trebuie să concorde cu datele care i se atribuie.

Dacă se utilizează o variabilă înainte de a i se atribui o valoare, aceștia i se atribuie, din oficiu, valoarea zero pentru variabilele numerice și valoarea nulă (șir de lungime zero) pentru variabilele șir.

BASIC-ul permite nume de variabile de pînă la 40 de caractere lungime. Caracterele pot fi litere, cifre sau punctul zecimal, cu condiția ca primul caracter al numelui variabilei să fie o literă. Caracterele speciale care identifică tipul variabilei declarate trebuie să se afle pe ultima poziție a numelui variabilei. O variabilă nu poate avea un nume rezervat de instrucțiunile BASIC. Astfel instrucțiunea:

**10 COS = 1**

nu este corectă deoarece COS este un nume rezervat limbajului și nu poate fi, deci, numele unei variabile. În schimb:

**10 COSINUS = 1**

este o instrucțiune corectă. De notat faptul că o variabilă care începe cu prefixul FN se așteaptă a fi o variabilă declarată funcție, definită de utilizator (vezi declarația DEF FN).

### 3.5. Conversia numerelor dintr-o precizie în alta

Un nume de variabilă determină tipul (șir de caractere, numerică) sau în cazul variabilelor numerice, precizia. După cum am spus, un caracter special aflat la sfîrșitul numelui unei variabile indică tipul acesteia. Astfel, caracterul:

- declară o variabilă numerică simplă precizie A = 12.345

! - declară o variabilă numerică simplă precizie A! = 11.23412

% - declară o variabilă numerică întregă A% = 123

# - declară o variabilă numerică dublă precizie A# = 1.123472238123

\$ - declară variabilă șir de caractere A\$ = "ABCD"

Tipul variabilei numerice poate fi declarat și pe altă cale. Astfel, instrucțiunile DEFINT, DEFSNG, DEFDBL, DEFSTR pot fi și ele utilizate

în program pentru a declara tipul de variabilă (vezi instrucțiunile DEFtip).

Cînd este necesar, limbajul BASIC convertește numerele dintr-o precizie în alta după următoarele reguli:

1. Dacă o valoare numerică de o precizie este atribuită unei variabile numerice de precizie diferită, numărul este memorat cu precizia variabilei țintă. Exemplu:

```
10 A% = 4.3256
```

```
20 PRINT A%
```

```
RUN
```

```
4
```

2. Dacă se atribuie unei variabile de precizie mai mică o valoare numerică de precizie mai mare se execută o rotunjire și nu o truncchiere:

```
10 B = 7.394920883132416
```

```
20 PRINT B
```

```
RUN
```

```
7.394921
```

Această regulă afectează nu numai instrucțiunile de atribuire (de ex.  $I\% = 2.5$  implică  $I\% = 3$ ), dar, de asemenea, funcțiile și instrucțiunile de evaluare. De exemplu  $TAB(4.5)$  va executa  $TAB(5)$ ;  $A(2.5)$  va avea drept rezultat  $A(3)$ , iar  $X = 11.5 \text{ MOD } 4$  va fi evaluat  $X = 0$ .

3. Dacă se face conversia dintr-o precizie mai scăzută, într-o precizie mai ridicată numărul rezultat nu poate avea valoare mai precisă. Exemplu:

```
10 A = 2.04
```

```
20 B# = A
```

```
30 PRINT A;B#
```

```
RUN
```

```
2.04 2.039999961853027
```

Eroarea poate fi limitată utilizînd următoarea formulă:

```
ABS(B#-A) < 6.3E-8 * A
```

4. Cînd este evaluată o expresie, toți operanzii unei operații aritmetice sau relaționale sînt convertiți la același grad de precizie, adică la precizia maximă. De asemenea, rezultatul unei operații aritmetice va fi afișat în această precizie. De exemplu:

```
10 D# = 6#/7
```

```
20 PRINT D#
```

```
RUN
```

```
.8571428571428571
```

Operația aritmetică se face în dublă precizie, iar rezultatul este atribuit variabilei  $D\#$  ca o valoare în dublă precizie.

```
10 D = 6#/7
```

```
20 PRINT D
```



RUN  
.857429

Operația aritmetică se face în dublă precizie, iar rezultatul se atribuie variabilei D ca o valoare în simplă precizie.

5. Operatorii logici convertesc operanzii în întregi, iar rezultatul va fi un număr întreg. Operanzii trebuie să fie cuprinși între -32768 și 32767, în caz contrar se va semnala eroare de depășire (**Overflow**).

### 3.6. Matrici

O matrice (sau vector) este o listă sau un tablou de valori la care se pot face referiri cu un singur nume. Fiecare valoare dintr-o matrice se numește element. Elementele sînt șiruri sau valori numerice și pot fi utilizate în expresii sau în instrucțiuni BASIC.

Subdomeniul (numărul din paranteză) indică poziția elementului în matrice. Prima poziție este zero atîta timp cît nu este schimbată cu ajutorul instrucțiunii OPTION BASE. Declararea numelui și tipului unei matrici precum și a numărului de elemente se numește definirea sau dimensionarea matricii. Numărul maxim de dimensiuni ale unei matrici este 255. Se poate utiliza o matrice fără a o dimensiona, în acest caz subdomeniul maxim este de 10 (deci 11 elemente numerotate de la 0 la 10). Pentru definirea unei matrici se utilizează instrucțiunea DIM. De exemplu:

**DIM B\$(5)**

va crea o matrice de tip șir unidimensională cu numele B\$, cu 6 elemente maxim:

B\$(0)

B\$(1)

B\$(2)

B\$(3)

B\$(4)

B\$(5)

Un alt exemplu:

**DIM A(2,3)**

Această instrucțiune crează o variabilă numerică A, bidimensională. Atîta timp cît matricea nu include un caracter special de declarație de tip de variabilă, aceasta se consideră a fi simplă precizie.

Matricea A poate fi gîndită ca un tablou de 3 rînduri și patru coloane:

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

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

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

O variabilă scalară poate avea același nume cu o matrice deoarece



acestea sînt reprezentate intern diferit. Astfel:

$$A = 7 \quad \text{si} \quad A(3,3)$$

nu se confundă deoarece A este numele unei variabile scalare, iar A(...) este numele unui vector.

### 3.7. Expresii numerice și operatori

O expresie numerică poate fi o variabilă sau o constantă numerică. De asemenea, poate fi un operator, combinînd constante și variabile pentru a produce o singură valoare numerică.

Operatorii numerici realizează operații numerice sau logice de cele mai multe ori asupra valorilor numerice, dar și asupra valorilor șir de caractere. Ei sînt numiți operatori numerici deoarece au drept rezultat o valoare numerică, adică un număr. Operatorii numerici pot fi divizați în următoarele categorii:

- aritmetici
- relaționali
- logici
- funcții

### 3.8. Operatori aritmetici

Operatorii aritmetici realizează operațiile aritmetice uzuale în ordinea matematică standard:

<u>Operator</u>	<u>Operație</u>	<u>Exemplu</u>
^	Exponențial	$X \wedge Y$
-	Negație	-X
*,/	Multiplicare	$X * Y$
	Diviziune în virgulă mobilă	$X / Y$
\	Diviziune întregă	$X \setminus Y$
MOD	Modulo aritmetic	$X \text{ MOD } Y$
+, -	Adunare	$X + Y$
	Scădere	$X - Y$

Cele mai multe dintre aceste operații fiind, probabil, cunoscute de către cititor, ne vom opri doar la operațiile de diviziune întregă și modulo aritmetic.

**Diviziunea întregă** se notează cu simbolul (\). Operanzii sînt rotunjiți la întregi cu valori între -32768 și 32767 înainte de a fi realizată

diviziunea, cîtul fiind trunchiat la un întreg. De exemplu:

```
10 A = 10\4
20 B = 25.68\6.99
30 PRINT A;B
RUN
2 3
```

**Modulo aritmetic** se notează cu operatorul MOD. El dă valoarea întregă a restului unei împărțiri întregi. De exemplu:

```
PRINT 7 MOD 4
3
```

deoarece  $7/4 = 1$  rest 3.

```
PRINT 25.68 MOD 6.99
5
```

rezultă 5 deoarece  $26/7 = 3$  rest 5.

### 3.9. Operatori relaționali

Operatorii relaționali compară două valori. Valorile pot fi atît de tip șir cît și numerice. Rezultatul comparării va fi *adevărat* (-1) sau *fals* (0).

Operator	Relația testată	Exemplu
=	Egalitate	$X = Y$
< > sau > <	Inegalitate	$X < > Y$
<	Mai mic ca	$X < Y$
>	Mai mare ca	$X > Y$
< = sau = <	Mai mic sau egal ca	$X < = Y$
= sau = >	Mai mare sau egal ca	$X > = Y$

**Comparări de valori numerice.** Cînd operatorii aritmetici și relaționali se combină într-o expresie, mai întîi se execută operațiile aritmetice. De exemplu expresia:

```
X + Y < (A-1)/B
```

este *adevărată* (-1) dacă valoarea X adunată cu valoarea Y este mai mică decît valoarea A-1 împărțită la B.

**Comparări de șiruri.** Comparările de șiruri pot fi gîndite ca "alfabetice". Astfel un șir este *mai mic ca* un altul dacă o literă de pe o poziție a primului șir vine alfabetic după o literă aflată pe o poziție echivalentă în șirul al doilea. Literele mici sînt *mai mari decît* aceleași litere mari. De exemplu:

```
"AA" < "AB"
"kg" < "KG"
```

"123" < "678"

"abc" > "123"

### 3.10. Operatori logici

Operatorii logici execută operații logice (sau booleene) asupra valorilor numerice. Un operator logic combină valori adevărate - false și atribuie un rezultat la rîndul lui adevărat sau fals. Un operand este considerat a fi *adevărat* dacă este diferit de zero și *fals* dacă este egal cu zero. Asupra numerelor operația se efectuează bit cu bit. Operatorii logici sînt:

NOT	complement logic
AND	conjuncție
OR	disjuncție
XOR	sau exclusiv
EQV	echivalență
IMP	implicație

Fiecare operator atribuie rezultatul după cum este ilustrat în tabelul alăturat (unde A - *adevărat*, F - *fals*). Operatorii sînt listați în ordinea precedenței:

NOT	X		NOT X
	A		F
	F		A
AND	X	Y	X AND Y
	A	A	A
	A	F	F
	F	A	F
	F	F	F
OR	X	Y	X OR Y
	A	A	A
	A	F	A
	F	A	A
	F	F	F
XOR	X	Y	X XOR Y
	A	A	F
	A	F	A
	F	A	A
	F	F	F



EQV	X	Y	X EQV Y
	A	A	A
	A	F	F
	F	A	F
	F	F	A
IMP	X	Y	X IMP Y
	A	A	A
	A	F	F
	F	A	A
	F	F	A

Operatorii sînt convertiți la întregi între -32768 și 32767. Dacă valoarea operandilor este în afara acestui domeniu se semnalează eroare de depășire (**Overflow**). Dacă operandul este negativ se folosește forma complementului față de doi. Aceasta transformă operandul în secvențe de 16 biți. Astfel operația se execută asupra acestor secvențe. Următorul exemplu ilustrează modul în care lucrează operatorii logici:

$$A = 63 \text{ AND } 16$$

În acest caz A ia valoarea 16. Deoarece 63 în binar este 111111, iar 16 este 10000, rezultă că 63 AND 16 este egal cu 010000, care este egal cu 16 în decimal.

$$B = -1 \text{ AND } 8$$

B va lua valoarea 8 deoarece -1 în binar este 1111111111111111 și 8 în binar este 1000, -1 AND 8 va fi egal cu 0000000000001000, sau 8 în zecimal.

$$C = 4 \text{ OR } 2$$

C va fi egal cu 6 deoarece 4 în binar este 100 și 2 este 010, 4 OR 2 va fi în binar 110, deci 6.

### 3.11. Funcții numerice

O funcție este utilizată ca o variabilă într-o expresie pentru a chema o operație predeterminată ce urmează a fi executată pe unul sau mai mulți operanzi. Limbajul BASIC are cîteva funcții numerice încorporate ce rezidă în sistem, ca de exemplu SQR (rădăcina pătrată), SIN (sinus), etc. De asemenea utilizatorul poate defini propriile funcții cu ajutorul instrucțiunii DEF FN.

Anterior, am discutat operațiile numerice în ordinea lor de precedență. Pe scurt:

1. Funcția chematoare este evaluată prima.
2. Operațiile aritmetice sînt executate în ordinea:
  - ^ (ridicarea la putere)
  - (scăderea)

\*/ (înmulțirea și împărțirea)

\ (împărțirea întregă)

MOD (modulo aritmetic)

+,- (adunarea și scăderea)

3. Operațiile relaționale

4. Operațiile logice sînt executate ultimele în ordinea: NOT, AND, OR, XOR, EQV SI IMP.

Operațiile cu același nivel de execuție sînt realizate în ordinea "de la stînga la dreapta". Pentru a schimba ordinea de execuție a operațiilor se vor folosi parantezele. În interiorul parantezelor nivelul uzual al ordinii de execuție se păstrează.

### 3.12. Expresii și operatori șir

O expresie șir poate fi o constantă șir, o variabilă șir, sau o combinație a acestora prin utilizarea operatorilor pentru a produce o singură valoare șir. Operatorii șir sînt folosiți pentru aranjarea șirurilor de caractere în diferite moduri. Cele două categorii de operatori șir sînt:

- concatenarea
- funcția

De notat că se pot folosi operatorii relaționali (=, <, >, <=, >=) pentru compararea a două șiruri, însă aceștia nu sînt considerați "operatori șir" deoarece au ca rezultat o valoare numerică și nu o valoare șir.

Operația de legare a două șiruri de caractere se numește concatenare. Această operație se realizează cu ajutorul simbolului (+). De exemplu:

10 A\$ = "Calculatorul"

20 B\$ = " Felix PC"

30 C\$ = A\$ + B\$

40 PRINT C\$

RUN

Calculatorul Felix PC

Funcția șir este identică cu funcția numerică cu deosebirea că rezultatul primeia este o valoare numerică. O funcție șir poate fi folosită într-o expresie pentru chemarea unei operații predeterminate ce urmează a fi executată pe unul sau mai mulți operanzi. Limbajul BASIC are înglobate mai multe funcții șir ce rezidă în sistem, acestea urmînd a fi explicate și detaliate în unul din capitolele viitoare.



## 4. INTRĂRI ȘI IEȘIRI

### 4.1. Fișiere

Un fișier este o colecție de informații ce este "păstrat" pe un suport, altul decât RAM (Random Access Memory), de exemplu disc, bandă magnetică, etc. Pentru a avea acces la informațiile fișierului este necesară deschiderea acestuia cu ajutorul instrucțiunii OPEN. După aceasta fișierul poate fi utilizat pentru operații de intrare-ieșire. Limbajul suportă conceptul de **fișiere generale Input/Output**. Aceasta înseamnă că orice tip de intrare/ieșire (I/O) poate fi tratat ca I/O către un fișier, indiferent dacă se utilizează un disc, o bandă, sau un canal de comunicație cu un alt calculator.

BASIC-ul execută operațiile de I/O utilizând un număr de fișier. În momentul deschiderii unui fișier cu ajutorul instrucțiunii OPEN se atribuie un număr de fișier sau periferic. Numărul de fișier poate fi orice număr, variabilă sau expresie în domeniul de la 1 la numărul maxim de fișiere ce pot fi deschise simultan.

### 4.2. Numele de fișier

Acesta trebuie să fie în conformitate cu convențiile DOS (Disk Operating System), astfel:

- numele poate consta din două părți distincte separate prin punct (.):

**nume.extensie**

Numele poate fi unul pînă la opt caractere lungime, iar extensia de unul la trei caractere. Dacă se folosesc mai mult de trei caractere pentru extensie restul vor fi ignorate. Dacă sînt folosite mai mult de opt caractere pentru nume, iar extensia nu este inclusă, limbajul va include după opt caractere un punct, iar restul de caractere (pînă la trei) vor fi folosite pentru extensie. Dacă numele este mai mare de opt caractere și extensia prezentă se va semnala eroare.

- următoarele caractere sînt permise în numele și extensia fișierului:

A la Z, 0 la 9, ( ) { } @ # \$ % ^ & \* ! - \_ / ' "

### 4.3. Numele perifericului

Constă din pînă la patru caractere alfanumerice urmate de două puncte (:). Este numele unui periferic de intrare/ieșire:

- KYBD:** Claviatura. Doar pentru intrare.
- SCRN:** Ecranul. Doar pentru ieșire.
- LPT1:** Primul printer. Ieșire sau acces aleator.
- LPT2:** Al doilea printer. Ieșire sau acces aleator.
- LPT3:** Al treilea printer. Ieșire sau acces aleator.

#### PERIFERICE DE COMUNICAȚIE

- COM1:** Primul adaptor de comunicație asincronă. Intrare și ieșire.
- COM2:** Al doilea adaptor de comunicație asincronă. Intrare și ieșire.

#### PERIFERICE DE MEMORARE

- A:** Primul cititor de disc. Intrare și ieșire.
- B:** Al doilea cititor de disc. Intrare și ieșire.
- C:** Primul disc fix. Intrare și ieșire.
- D:** Al doilea disc fix. Intrare și ieșire.

## 5. COMENZI, INSTRUCȚIUNI ȘI FUNCȚII

### BASIC

În acest capitol vor fi descrise complet comenzile, instrucțiunile și funcțiile limbajului BASIC. Acestea vor fi prezentate în ordine alfabetică. Distincția dintre comenzile și instrucțiunile BASIC este o problemă care ține mai mult de tradiție. Comenzile, deoarece, în mod general, operează asupra programelor, pot fi introduse în modul direct. Spre deosebire de acestea, instrucțiunile urmează un algoritm în cadrul unui program, de aceea ele pot fi introduse în modul indirect, deci ca parte a unui program. În dialectele BASIC moderne (ca și cel de față) cele mai multe dintre comenzile și instrucțiunile limbajului pot fi introduse atât în modul direct cât și în modul program.

Descrierea fiecărei comenzi, instrucțiuni, funcții sau variabile se va face după următorul algoritm:

**Scopul:** Indică utilitatea comenzii, instrucțiunii funcției sau variabilei.

**Formatul:** Prezintă formatul corect de scriere a comenzii, instrucțiunii, sau a variabilei. De reținut următoarele reguli generale:

- cuvintele scrise cu majuscule sînt cuvinte cheie și trebuie introduse exact cum sînt prezentate, cu excepția că ele pot fi introduse cu minusculă, majuscule sau orice combinație a acestora, limbajul făcînd automat conversia în majuscule.
- se pot înlocui orice termeni prezentați în caractere minuscule.
- termenii în paranteze pătrate ([ ]) sînt opționali.
- parantezele rotunde (...) indică faptul că termenii din interior pot fi repetați de cîte ori se dorește.
- toate semnele de punctuație - cu excepția parantezelor pătrate (ca de exemplu, virgule, paranteze, punct și virgulă, semnul egal, semnul negativ) trebuie introduse unde sînt arătate.

**Comentariu:** Descrie în detaliu modul de utilizare a instrucțiunii, comenzii, funcției sau variabilei.

**Exemple:** Prezintă instrucțiunile în mod direct, exemple de programe sau segmente de programe ce ilustrează modul de utilizare a co-



menzilor, instrucțiunilor, funcțiilor și variabilelor.

## ABS

Funcție

**Scop:** Atribuie unei variabile valoarea absolută a unei expresii.

**Format:**  $v = ABS(x)$

**Comentariu:**  $x$  poate fi orice expresie numerică. Rezultatul este, totdeauna, un număr pozitiv sau zero.

**Exemplu:**

```
PRINT ABS(3*(-4))  
12
```

## ASC

Funcție

**Scop:** Atribuie unei variabile codul ASCII al primului caracter al șirului  $x\$$ .

**Format:**  $v = ASC(x\$)$

**Comentariu:**  $x\$$  poate fi orice expresie șir. Rezultatul funcției **ASC** este o valoare numerică ce reprezintă codul ASCII al primului caracter din șirul  $x\$$ . Dacă  $x\$$  este un șir nul se va semnala eroare **Illegal function call**.

**Exemplu:**

```
10 X$ = "ALFA"  
20 PRINT ASC(X$)  
RUN  
65
```

## ATN

Funcție

**Scop:** Calculează arctangentă de  $x$ .

**Format:**  $v = ATN(x)$

**Comentariu:**  $x$  poate fi orice expresie. Funcția **ATN** calculează unghiul a cărei tangentă este  $x$ . Rezultatul este o valoare în radiani în domeniul  $-PI/2$  la  $PI/2$ , unde  $PI = 3.141593...$  Pentru conversia din radiani în grade hexazecimale se înmulțește cu  $180/PI$ .

Valoarea poate fi calculată în dublă precizie specificând argumentul **/D** în linia de comandă **BASIC** la inițializarea limbajului

**Exemplu:**

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

Al doilea exemplu calculează unghiul a cărei tangentă este 1. Valoarea calculată este .7853983 radiani sau 45 grade.

```
10 PI = 3.141593
20 RAD = ATN(1)
30 GRAD = RAD*180/PI
40 PRINT RAD,GRAD
RUN
.7853983          45
```

## AUTO

Comandă

**Scop:** Generează în mod automat numere de linie de program.

**Format:** *AUTO[număr][],[increment]*

**Comentariu:** *număr* este numărul de start al liniei. Se poate folosi un punct (.) în locul numărului de linie pentru a indica linia curentă.

*increment* este valoarea adăugată fiecărei linii pentru a da numărul liniei următoare.

Numărătoarea începe cu numărul indicat și crește, apoi, cu valoarea incrementului. Dacă se omit ambele valori, numărătoarea va începe de la linia 10 și va avea increment 10. Dacă este indicat un număr de linie urmat de virgulă, însă incrementul nu este specificat, se va atribui incrementul specificat la ultima comandă **AUTO**. Dacă numărul este omis, însă incrementul este specificat, numărătoarea liniilor va începe de la 0. Comanda **AUTO** este utilizată la numerotarea liniilor, ceea ce salvează timpul de scriere manuală a liniilor. Dacă **AUTO** generează un număr de linie deja existent în program se va tipări un asterisc (\*) după numărul de linie, pentru a semnala faptul că linia respectivă se va suprapune peste una deja existentă. Astfel, în cazul în care se apasă tasta Enter imediat după asterisc, linia marcată va rămâne nemodificată, iar **AUTO** va genera următoarea linie de program. **AUTO** se încheie cu Ctrl-Break. Această ultimă linie nu va fi salvată.

**Exemplu:**

```
AUTO
```

va genera, în ordine, liniile 10, 20, 30, 40,...

```
AUTO 100,50
```

va genera, în ordine, liniile 100, 150, 200,...

```
AUTO ,30
```

va genera liniile 0, 30, 60, 90,...



## BEEP

Instrucțiune

**Scop:** Produce un sunet în difuzor.

**Format:** BEEP

**Comentariu:** Instrucțiunea BEEP produce în difuzor un sunet de 800 Hz timp de 1/4 secunde. Instrucțiunea BEEP are același efect cu :  
**PRINT CHR\$(7);**

## BLOAD

Comandă

**Scop:** Încarcă un fișier imagine de memorie în spațiul indicat de memorie.

**Format:** BLOAD numefis[,offset]

**Comentariu:** numefis este o expresie șir care indică numele fișierului.

offset este o expresie întreagă între 0 și 65535. Indică offset- ul la care fișierul va fi încărcat în segmentul curent de memorie, specificat, anterior, cu instrucțiunea DEF SEG.

Dacă offset-ul este omis, se va atribui offset-ul specificat la comanda BSAVE, aceasta însemnând că fișierul va fi încărcat în memorie la aceeași locație la care a fost salvat cu ajutorul comenzii BSAVE. Aceste două comenzi sînt utile în cazul manipulării programelor în limbaj masină.

**Observație:** BASIC-ul nu verifică offset-ul segmentului curent de memorie la care se va încărca fișierul. Aceasta înseamnă că încărcarea se va putea face în orice zonă de memorie, inclusiv în zona stack-ului limbajului BASIC, a zonei de variabile sau a celei de program, ceea ce poate duce la serioase probleme în timpul rulării.

**Exemplu:** Exemplul următor încarcă tamponul de memorie de ecran a adaptorului de grafică color, care se găsește la adresa &HB800. Linia 40 încarcă fișierul POZA la offset-ul 0, segmentul &HB800:

```
10 ' incarca segmentul de memorie de ecran
20 DEF SEG = &HB800
30 ' incarca POZA in memoria tampon de ecran
40 BLOAD "POZA",0
```

Exemplul de la comanda BSAVE ilustrează modul în care a fost salvat fișierul "POZA".

## BSAVE

Comandă

**Scop:** Salvează porțiuni din memoria calculatorului pe un pe-



periferic specificat.

**Format:** *BASVE numefis,offset,lungime*

**Comentariu:** *numefis* este o expresie șir care indică numele fișierului.

*offset* este o expresie întreagă în domeniul 0 la 65535. Acesta este offset-ul segmentului declarat anterior prin DEF SEG. Salvarea începe de la această locație.

*lungime* este o expresie întreagă cuprinsă între 1 și 65535. Aceasta este lungimea imaginii de memorie care urmează a fi salvată.

Dacă se omit *lungimea* și *offset*-ul, se va semnala eroare, iar salvarea nu se mai execută.

**Exemplu:** În acest exemplu vom vedea modul în care este salvată imaginea de memorie a ecranului pentru adaptorul de grafică color începând cu locația de memorie &HB800. Este necesară utilizarea instrucțiunii DEF SEG pentru a indica adresa de start a tamponului de memorie a ecranului (&HB800). Offset-ul 0 și lungimea &H4000 specifică faptul că tot tamponul de memorie de 16Ko al ecranului urmează să fie salvat.

```
10 DEF SEG = &HB800
20 BSAVE "POZA",0,&H4000
```

## CALL

Instrucțiune

**Scop:** Cheamă subrutine în limbaj masină.

**Format:** *CALL varnum[(variabila [,variabila]...)]*

**Comentariu:** *varnum* este numele unei variabile numerice. Valoarea variabilei indică offset-ul subrutinei în segmentul de memorie curent, așa cum a fost definit în ultima instrucțiune DEF SEG.

*variabila* este numele unei variabile ce va fi trecut drept argument subrutinei în limbaj masină.

Instrucțiunea CALL este unul din modurile în care limbajul BASIC se poate interfața cu limbajul masină.

**Exemplu:** Linia 10 din program atribuie valoarea segmentului BASIC. Linia 30 declară toate valorile scalare și matricile utilizate în program. Linia 40 calculează offset-ul variabilei ARRAY în segmentul de date BASIC. Linia 50 încarcă fișierul într-o matrice întreagă, iar linia 60 cheamă rutina. Variabilele Q, B\$ și C sînt trecute ca argumente ale rutinei în limbaj masină.

```
10 DEF SEG: OPTION BASE 1
20 DEFINT A-Z
30 DIM ARRAY(512)
```

```

35 P = 0: Q = 5: C = 0: B$ = "TRUE"
40 P = VARPTR(ARRAY(1))
50 BLOAD "ASM.FIL",P
60 CALL P(Q,B$,C)

```

## CDBL

Funcție

**Scop:** Convertește argumentul x într-un număr în dublă precizie.

**Format:**  $v = CDBL(x)$

**Comentariu:** x poate fi orice expresie numerică.

**Exemplu:**

```

10 A = 234.56
20 PRINT A;CDBL(A)
RUN
234.56 234.559982910156

```

## CHAIN

Instrucțiune

**Scop:** Transferă controlul unui alt program și trece variabilele din programul curent către acesta.

**Format:** *CHAIN* [*MERGE*]numefis[,*[linie]*],[*ALL*],[*DELETE* domeniu]]

**Comentariu:** *MERGE* aduce un fișier ASCII în programul BASIC curent ca "overlay". Dacă urmează a fi unit prin opțiunea *MERGE*, programul de legătură trebuie să fie un fișier de tip ASCII. Exemplu:

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

*numefis* este o expresie șir care conține numele fișierului. Numele fișierului este numele programului către care se va transfera controlul. Exemplu:

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

*linie* este un număr de linie sau o expresie care evaluează un număr de linie în programul de legătură, Specifică numărul liniei la care se va da controlul execuției. Dacă este omis, controlul se da la prima linie de program. Exemplu:

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

*ALL* specifică faptul că toate variabilele din programul curent vor fi trecute programului de legătură. Dacă opțiunea *ALL* este omisă, este necesară includerea în programul curent al unei instrucțiuni *COMMON* pentru trecerea variabilelor dorite programului de legătură. Exemplu:

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



**DELETE** lucrează identic cu instrucțiunea **DELETE**. Exemplu:

**CHAIN MERGE "A:OVERLAY1",1000,DELETE  
1500-5000**

**Observații:**

1. Instrucțiunea **CHAIN** lasă fișierele deschise.
2. Instrucțiunea **CHAIN** conservă instrucțiunea **OPTION BASE**.
3. Fără argumentul **MERGE**, instrucțiunea **CHAIN** nu conservă tipurile de variabilă sau funcțiile definite de utilizator pentru a fi trecute programului de legătură. Astfel, oricare din instrucțiunile **DEFINT**, **DEFSNG**, **DEFDBL**, **DEFSTR** sau **DEF FN**, conținând variabile active trebuie redeclarată în programul de legătură.
4. Instrucțiunea **CHAIN** execută un **RESTORE** înainte de a da controlul programului de legătură.

## CHDIR

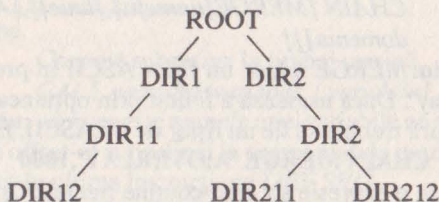
**Comandă**

**Scop:** Schimbă directorul curent.

**Format:** *CHDIR <cale>*

**Comentariu:** *cale* este o expresie șir, mai mică de 63 caractere; cu scopul de a identifica noul director care va deveni directorul curent.

**Exemplu:**



Pentru trecerea din directorul rădăcină (**ROOT**) în oricare din sub-directoare se utilizează:

**CHDIR "\"**

Pentru trecerea din directorul rădăcină în directorul **DIR12** se utilizează:

**CHDIR "DIR1\DIR11\DIR12"**

Pentru trecerea, de exemplu, din directorul **DIR21** în directorul **DIR212** se va utiliza:

**CHDIR "DIR212"**

Pentru trecerea înapoi din directorul **DIR11** în directorul **DIR1**, se va utiliza:

**CHDIR ".."**



## CHR\$

Funcție

**Scop:** Converteste un cod ASCII în caracterul său echivalent.

**Format:**  $v\$ = CHR\$(n)$

**Comentariu:** n trebuie să fie în domeniul 0 la 255. CHR\$ este utilizat de obicei pentru a trimite un caracter special la imprimantă sau ecran. Complementara ei este funcția ASC.

**Exemplu:**

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

```
B
```

Următorul exemplu este un program care tipărește pe ecran toate caracterele reprezentabile împreună cu codul lor ASCII:

```
10 CLS
```

```
20 FOR I = 1 TO 255
```

```
30 ' ignora toate caracterele nerepresentabile
```

```
40 IF (I > 6 AND I < 14) OR (I > 27 AND I < 32)
```

```
THEN 100
```

```
50 COLOR 0,7
```

'negru pe alb

```
60 PRINT USING "###";I;
```

'3 cifre cod ASCII

```
70 COLOR 7,0
```

'alb pe negru

```
80 PRINT " ";CHR$(I);" ";
```

```
90 IF POS(0) > 75 THEN PRINT
```

```
100 NEXT I
```

## CINT

Funcție

**Scop:** Converteste argumentul x într-un întreg.

**Format:**  $v = CINT(x)$

**Comentariu:** x poate fi orice expresie numerică. Dacă x este în afara domeniului de la -32768 la 32767 va fi semnalată eroare de tip "Overflow". x este convertit la un întreg prin rotunjirea (în sus) a părții fracționare.

**Exemplu:** Observați, în exemplele următoare cum se realizează rotunjirea:

```
PRINT CINT(45.499)
```

```
45
```

```
PRINT CINT(-2.89)
```

```
-3
```

## CIRCLE

### Instrucțiune

**Scop:** Desenează, pe ecran, o elipsă cu centrul  $(x,y)$  și raza  $r$ .

**Format:** `CIRCLE(x,y),r[culoare[,început,sfârșit[aspect]]]`

**Comentariu:**  $(x,y)$  reprezintă coordonatele elipsei. Acestea pot fi date atât sub forma absolută cât și relativă.

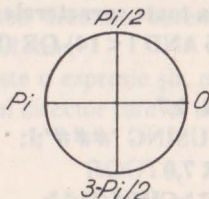
$r$  este raza (axei majore), în puncte, a elipsei.

*culoare* este o expresie numerică întreagă care indică atributul de culoare din gama atributelor de culoare a modului ecran curent.

*început,sfârșit* reprezintă unghiurile în radiani în domeniul  $-2*PI$  la  $2*PI$ , unde  $PI = 3.141593$ .

*aspect* este o expresie numerică.

*început* și *sfârșit* specifică de unde va începe și unde se va termina desenarea elipsei. Unghiurile sînt poziționate în standardul matematic, cu 0 la dreapta și crescător în sens trigonometric.



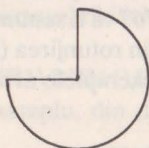
Dacă unghiul de început sau de sfârșit este negativ, elipsa este legată de punctul de centru cu o linie. Unghiul de început poate fi mai mare sau mai mic decât unghiul de sfârșit. De exemplu:

`10 PI = 3.141593`

`20 SCREEN 1`

`30 CIRCLE (160,100),60,,-PI,-PI/2`

desenează o parte a unui cerc similar cu:



Argumentul *aspect* afectează raportul dintre raza mare și raza mică. Dacă nu este specificată valoarea, aspectul este  $5/6$  în rezoluție medie și  $5/12$  în înaltă rezoluție, din aceste valori rezultînd un raport de  $4/3$  pentru ecranul standard.

## CLEAR

Comandă

**Scop:** Atribuire tuturor variabilelor valoarea zero și anulează toate variabilele șir. Opțiunile comenzii atribuie numărul maxim de octeți pe care îi va adresa limbajul, precum și mărimea stivei.

**Format:** `CLEAR[, [n][, m]]`

**Comentariu:** *n* specifică numărul maxim de octeți ai segmentului de date BASIC. Dacă nu se specifică, se ia pentru *n* 65535. Se poate specifica o valoare mai mică pentru *n* pentru a descrește spațiul adresabil total BASIC. Aceasta face să crească, în schimb, totalul memoriei disponibile dincolo de segmentul de date BASIC în memoria superioară, operație necesară, de exemplu, pentru rezervarea de spațiu de memorie pentru programele în limbaj masină.

*m* setează spațiul stivei BASIC. Dacă nu se specifică, se ia valoarea de 512 octeți, sau 1/8 din memoria disponibilă. Parametrul *m* trebuie inclus în cazul în care sînt folosite mai multe înlănțuiri de instrucțiuni GOSUB sau bucle FOR ... NEXT, sau în cazul în care se utilizează instrucțiunea PAINT pentru crearea unor scene complexe.

CLEAR eliberează întreaga memorie de date, fără a șterge însă programul curent.

**Exemple:**

`CLEAR ,32768`

șterge datele și setează segmentul de date la 32 Kocteți.

`CLEAR ,,2000`

șterge datele și setează mărimea stivei la 2000 de octeți.

## CLOSE

Instrucțiune

**Scop:** Încheie activitatea de I/O (intrare-ieșire) cu un periferic sau fișier.

**Format:** `CLOSE [[#]nrfis[, [#]nrfis]...]`

**Comentariu:** *nrfis* este un număr utilizat în instrucțiunea OPEN. Asocierea între un fișier sau periferic particular și un număr de fișier se încheie odată cu executarea instrucțiunii CLOSE. Dacă instrucțiunea CLOSE nu este urmată de nici un parametru, vor fi închise toate fișierele sau perifericele.

**Exemplu:**

`9900 CLOSE #1,#2,#3`



## CLS

Instrucțiune

**Scop:** Șterge ecranul.

**Format:** CLS

**Comentariu:** Dacă ecranul este în modul text, este ștersă pagina activă. Dacă ecranul este în modul grafic este ștersă toată memoria tampon de ecran. Instrucțiunea CLS aduce, de asemenea cursorul în poziția inițială, stînga-sus,

## COLOR

Instrucțiune

**Scop:** Atribuie culorile pentru fundal ,prim-plan și margine. Sintaxa instrucțiunii COLOR depinde de modul de lucru (text sau grafic), mod care a fost setat cu ajutorul instrucțiunii SCREEN.

**In modul text:**

**prim-plan** 1 la 16 atribute de culoare sau caracter clipitor.

**fundal** 1 la 8 atribute de culoare.

**margine** 1 la 16 atribute de culoare.

**In modul grafică medie rezoluție:**

**fundal** 1 la 16 atribute de culoare

**paleta** 1 la 2 palete cu trei culori fiecare.

**margine** la fel ca la fundal

## COLOR (modul Text)

**Format:** COLOR [primplan][,[fundal]][,margine]

**Comentariu:** *primplan* este o expresie numerică cu valori între 0 și 31, reprezentînd culoarea caracterului.

*fundal* este o expresie numerică cu valori între 0 și 7, pentru fundal.

*margine* este o expresie numerică cu valori între 0 și 15, pentru marginea ecranului.

Culorile permise pentru prim-plan sînt:

0 negru 8 gri

1 albastru 9 albastru deschis

2 verde 10 verde deschis

3 bleu 11 bleu deschis

4 roșu 12 roșu deschis

5 magenta 13 magenta deschis

6 maro 14 galben

7 alb 15 alb strălucitor

Caracterele pot fi făcute să clipească prin adăugarea de 16 unități la

valoarea culorii alese.

Pentru fundal pot fi alese culorile de la 0 la 7.

Cu adaptorul de grafică monocrom paralel, pentru caractere pot fi folosite următoarele culori:

0 negru

1 caracter subliniat

2-7 culoarea standard primplan

Pentru adaptorul grafic/color, adăugînd 8 la numărul culorii dorite, se va obține aceeași culoare în intensitate mărită. Dacă la culoarea dorită se adaugă 16 se obține aceeași culoare, dar clipitoare.

### În modul grafic:

**Format:** *COLOR [fundal][,][paleta]*

**Comentariu:** *fundal* este o expresie întreagă cu valori între 0 și 15. Specifică atributul de culoare al fundalului.

*paleta* este o expresie întreagă. Selectează una din cele două palete de culoare.

În modul grafic, instrucțiunea *COLOR* atribuie culoarea fundalului și alege una din cele două palete, fiecare cu cîte 4 atribute de culoare (0-3). Atributul de culoare 0 este totdeauna culoarea fundalului curent. Instrucțiunea *COLOR* are înțeles, în modul grafic, doar pentru rezoluția medie. Utilizarea acestei instrucțiuni în rezoluție înaltă duce la semnalarea unei erori de tip "**Illegal function call**". Culorile care pot fi selectate sînt următoarele:

Culoare	Paleta 1	Paleta 0
1	verde	bleu deschis
2	roșu	magenta
3	maro	alb

Dacă paleta este un număr par, va fi selectată paleta 0, iar dacă paleta este un număr impar oarecare, se va selecta paleta 1.

Schimbarea culorii prim-plan a caracterelor de la 3 la 2 la 1 se poate realiza introducînd:

**DEF SEG: POKE &HFE,CULOARE**

unde *CULOARE* este atributul de culoare dorit (1, 2 sau 3).

### COM(n)

Instrucțiune

**Scop:** Activează sau dezactivează comunicațiile prin adaptorul de comunicații specificat.

**Format:** *COM(n) ON*

*COM(n) OFF*  
*COM(n) STOP*

**Comentariu:** *n* este numărul adaptorului de comunicație (1 sau 2).

Instrucțiunea **COM(n) ON** trebuie executată înainte de lansarea instrucțiunii **ON COM(n)**. Dacă **COM(n)** este **OFF** nu se poate realiza activitatea de comunicații, aceasta fiind declarată închisă. Instrucțiunea **COM(n) STOP** dezafectează, pentru moment, linia de transmisie, dar fără închiderea definitivă a activității, urmînd ca la lansarea instrucțiunii **COM(n) ON** aceeași activitate să fie reluată.

## COMMON

Instrucțiune

**Scop:** Trece variabilele către un program de legătură.

**Format:** *COMMON var[,var]...*

**Comentariu:** *var* este numele variabilei ce urmează a fi trecută în programul de legătură. Vectorii sînt specificați prin adăugarea "( )" la numele vectorului.

Instrucțiunea **COMMON** poate apare oriunde în program, dar este preferabil să fie amplasată la începutul acestuia. Se pot folosi oricîte instrucțiuni **COMMON**, dar este obligatoriu ca variabilele să fie numite o singură dată. Dacă în programul de legătură sînt necesare toate variabilele din programul curent, instrucțiunea **CHAIN** se va folosi cu opțiunea **ALL** și va fi omisă utilizarea instrucțiunii **COMMON**.

**Exemplu:**

```
100 COMMON A,B1,C,D0,S$
```

```
110 CHAIN "A:PROG3"
```

## CONT

Comandă

**Scop:** Reia execuția unui program după o comandă de "break".

**Format:** *CONT*

**Comentariu:**

Comanda **CONT** poate fi utilizată pentru reluarea execuției unui program, după ce programul a fost oprit cu Ctrl-Break. Execuția continuă de la punctul în care a avut loc întreruperea.

Comanda **CONT** este de obicei utilizată cu comanda **STOP** pentru depanarea programelor. Cînd execuția este oprită, valorile variabilelor pot fi examinate sau schimbate utilizînd instrucțiunile de mod direct.

Comanda **CONT** nu este executabilă dacă programul sau linia din programul curent au fost editate.



## COS

**Funcție**

**Scop:** Calculează funcția cosinus trigonometric.

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

**Comentariu:**  $x$  este unghiul al cărui cosinus urmează a fi calculat.

Valoarea lui  $x$  trebuie să fie în radiani.

**Exemplu:**

10 PI = 3.141593

20 GRAD = 180

30 RAD = GRAD\*PI/180

40 PRINT COS(RAD)

RUN

-1

## CSNG

**Funcție**

**Scop:** convertește argumentul  $x$  în simplă precizie.

**Format:**  $v = \text{CSNG}(x)$

**Comentariu:**  $x$  este o expresie numerică ce va fi convertită în simplă precizie.

**Exemplu:**

10 A# = 975.3421222#

20 PRINT A#; CSNG(A#)

RUN

975.3421222 975.3421

## CSRLIN

**Variabilă**

**Scop:** Citește coordonata verticală a cursorului.

**Format:**  $v = \text{CSRLIN}$

**Comentariu:** Variabila CSRLIN redă poziția cursorului liniei (rîndului) curente în pagina activă, putînd lua valori între 1 și 25. Funcția POS redă poziția cursorului coloanei. Vezi "Funcția POS".

**Exemplu:** În acest exemplu se salvează coordonatele cursorului în variabilele X și Y, apoi cursorul este mutat pe linia 24, unde este afișat cuvîntul "SCOR =", după care cursorul este mutat la vechea poziție.

10 Y = CSRLIN 'salveaza linia curenta

20 X = POS(0) 'salveaza coloana curenta

30 LOCATE 24,1: PRINT "SCOR ="

40 LOCATE Y,X 'inapoi la vechea pozitie

## CVI, CVS, CVD

Funcții

**Scop:** Convertesc variabilele șir în variabile numerice.

**Format:**  $v = CVI(\text{șir } 2 \text{ octeți})$   
 $v = CVS(\text{șir } 4 \text{ octeți})$   
 $v = CVD(\text{șir } 8 \text{ octeți})$

**Comentariu:** Valorile numerice citite dintr-un fișier aleator trebuie convertite din șiruri în numere. CVI convertește un șir de 2 octeți într-un întreg. CVS convertește un șir de 4 octeți într-un număr simplă precizie. CVD convertește un șir de 8 octeți într-un număr dublă precizie.

**Exemplu:** Acest exemplu utilizează un fișier aleator (#1) care are câmpurile definite în linia 100. Linia 110 citește o înregistrare din fișier. Linia 120 utilizează funcția CVS pentru interpretarea primilor 4 octeți (N\$) ai înregistrării, ca un număr în simplă precizie. N\$ era la origine, probabil, un număr scris în fișier utilizând funcția MKS\$.

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

## DATA

Instrucțiune

**Scop:** Memorează constantele șir și numerice ce sînt adresate de instrucțiunea READ.

**Format:** *DATA const[,const]...*

**Comentariu:** *const* poate fi orice constantă șir sau numerică.

În lista constantelor nu sînt permise expresii. Constantele numerice pot fi de orice formă - întregi, virgulă fixă, virgulă mobilă, hexazecimale sau octale. Constantele șir nu trebuie incluse între ghilimele atîta timp cît nu conțin virgule, două puncte sau spații semnificative.

Instrucțiunile DATA nu sînt executabile și pot fi plasate oriunde în program. O instrucțiune DATA poate conține oricîte constante în limita lungimii unei linii de program, iar un program poate conține oricîte instrucțiuni DATA. Instrucțiunea READ accesează instrucțiunile DATA în ordinea numărului de linie. Pentru recitirea informațiilor dintr-o instrucțiune DATA trebuie folosită instrucțiunea RESTORE.

**Exemplu:** Vezi exemplul de la instrucțiunea READ.

## DATE\$

Variabilă și Instrucțiune

**Scop:** Instaurează sau alocă data calendaristică.

**Format:**

Ca variabilă:  $v\$ = DATE\$$

Ca instrucțiune:  $DATE\$ = x\$$

**Comentariu:** Pentru variabilă ( $v\$ = DATE\$$ ):

Este alocat un șir de 10 caractere în forma **ll-zz-aaaa** unde:

ll este luna;

zz este ziua;

aaaa este anul.

Pentru instrucțiune ( $DATE\$ = x\$$ ):

$x\$$  este o expresie șir utilizată pentru instaurarea datei calendaristice curente.  $x\$$  poate lua una din formele:

ll-zz-aa

ll/zz/aa

ll-zz-aaaa

ll/zz/aaaa

Anul poate lua valori în domeniul 1980-2099.

**Exemplu:**

**10** DATE\$ = "12/31/1988"

**20** PRINT DATE\$

**RUN**

**12-31-1988**

## DEF FN

**Instrucțiune**

**Scop:** Definește și numește o funcție.

**Format:**  $DEF FN\text{nume}[(arg,[arg]...)] = \text{expresie}$

**Comentariu:** nume este numele variabilei. Aceste nume precedate de prefixul FN devin numele funcției.

$arg$  este un argument. Reprezintă numele unei variabile din definiția funcției, ce va fi înlocuit cu o valoare în momentul apelării funcției. Argumentele din listă reprezintă, într-o bază unu-la-unu, valorile ce vor fi atribuite în momentul apelării funcției.

$expresie$  definește și redă valoarea funcției. Tipul expresiei (numerică sau șir) trebuie să coincidă cu tipul declarat în *nume*.

Definirea unei funcții este limitată la o singură instrucțiune. Argumentele ( $arg$ ) ce apar în definirea funcției servesc doar la definiție; ele nu afectează variabilele de program ce au același nume. Un nume de variabilă utilizat în  $expresie$  nu trebuie să apară în lista argumentelor, altfel, chiar această valoare va fi furnizată în momentul apelării.

Instrucțiunea DEF FN trebuie executată înaintea apelării funcției pe



care o definește.

Nu sînt suportate funcțiile recursive.

DEF FN nu este permisă în modul direct.

**Exemplu:**

```
10 PI = 3.141593
20 DEF FNARIA(R) = PI*R ^ 2
30 INPUT "Raza? ",RAZA
40 PRINT "Aria este ";FNARIA(RAZA)
RUN
Raza?
```

(Să spunem că răspundem cu 2)

```
Raza?2
Aria este 12.56637
```

Și acum un exemplu de două argumente:

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

## DEF SEG

Instrucțiune

**Scop:** Definește segmentul curent de memorie.

**Format:** DEF SEG[ =segment]

**Comentariu:** *segment* este o expresie numerică cu valori între 0 și 65535.

Valoarea inițială a segmentului, la începutul sesiunii BASIC, este valoarea segmentului de date (DS) BASIC, sau, cu alte cuvinte, începutul spațiului de memorie de lucru a utilizatorului. Valoarea segmentului de date BASIC poate fi găsită la segmentul 0, offset &H510 și &H11. Dacă se dă valoarea segmentului, aceasta trebuie să fie cuprinsă în limite de câte 16 octeți, deoarece segmentele încep doar în limitele paragrafelor. Valoarea este deplasată la stînga 4 biți (multiplicat cu 16) pentru a forma adresa segmentului operației următoare. Aceasta înseamnă că, dacă segmentul este în hexazecimal, este adăugat un 0 (zero) pentru a da adresa actuală a segmentului. Limbajul BASIC nu execută nici o verificare pentru validarea valorii segmentului.

Cuvintele DEF și SEG trebuie separate printr-un spațiu; altfel instrucțiunea DEFSEG = 100 va fi interpretată "atribuie valoarea 100 variabilei DEFSEG".

**Exemplu:** Primul exemplu restaurează un segment în segmentul de

date BASIC:

### DEF SEG

În al doilea exemplu, segmentul este atribuit memoriei tampon a adaptorului de grafică color (&HB800, offset 0).

**DEF SEG = &HB800**

### DEFtip

Instrucțiune

**Scop:** Declară tipul variabilei ca întreg, simplă precizie, dublă precizie sau șir.

**Format:** *DEFtip literă[-literă][.literă[-literă]]...*

**Comentariu:** *tip* este INT, SNG, DBL sau STR.

*literă* este o literă a alfabetului (A-Z).

O instrucțiune DEFtip declară că variabilele începând cu litera sau literele specificate vor fi de tipul descris. Totuși, caracterele de declarare a tipului (% , ! , # , \$) au, întotdeauna, precedenta mai mare asupra instrucțiunii DEFtip în declararea unei variabile. Dacă nu se specifică tipul unei variabile, se presupune a fi în simplă precizie.

**Exemplu:**

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

### DEFUSR

Instrucțiune

**Scop:** Specifică locația din memorie a unei subrutine în limbaj masină, care va fi, ulterior, apelată de funcția USR.

**Format:** *DEFUSR[n] = offset*

**Comentariu:** *n* trebuie să fie un număr între 1 și 9. Identifică numărul subrutinei USR. Dacă se omite *n*, se presupune DEFUSR0.

*offset* este o expresie întregă cuprinsă între 0 și 65535. Valoarea offset-ului este adăugată la valoarea segmentului curent pentru a obține adresa curentă de start a rutinei USR.

**Exemplu:** Acest exemplu încarcă o subrutină limbaj masină într-un

vector întreg. Valoarea lui n din linia 60 este determinată de mărimea subrutinei.

```
10 OPTION BASE 1
20 DEFINT A-Z
30 'Se definesc toate variabilele de VARPTR
40 SUBRT = 0: I = 0: J = 0
50 'Dimensionarea vectorului pentru subrutina
60 DIM VECTOR(N)
70 'Obținem offset-ul primului element al vectorului
in segmentul de date BASIC
80 SUBTR = VARPTR(VECTOR(1))
90 'Incarca rutina in vector
100 BLOAD "ASMFILE",SUBTR
.
.
.
1000 'Trece offset-ul în DEFUSR
1010 DEFUSR0 = VARPTR(VECTOR(1))
1020 'Executa subrutina
1030 J = USR0(I)
```

## DELETE

Comandă

Scop: Șterge linii de program.

Format: *DELETE [linie1][-linie2]*  
*DELETE [linie1-]*

Comentariu: *linie1* este numărul primei linii ce urmează a fi ștersă.  
*linie2* este numărul ultimei linii ce urmează a fi ștersă.

**DELETE line1-**

șterge toate liniile de la linia specificată la sfârșitul programului.

**DELETE -linie1**

șterge toate liniile de la începutul programului la linia specificată.

Poate fi utilizat un punct (.) în locul unui număr de linie pentru a indica linia curentă de program.

Exemple:

**DELETE 10**

șterge linia 10 din program;

**DELETE 20-70**

șterge toate liniile cuprinse între linia 20 și linia 70;

**DELETE .**



Presupunând că ultima linie editată a fost 220, comanda de mai sus șterge această linie;

#### DELETE 100-

șterge toate liniile de program începînd cu linia 100, inclusiv, pînă la sfîrșitul programului;

#### DELETE -80

șterge toate liniile de program, de la început pînă la linia 80 inclusiv.

### DIM

#### Instrucțiune

**Scop:** Specifică valoarea maximă a elementelor unei matrici (vector) și alocă memorie în concordanță cu numărul de elemente.

**Format:** *DIM variabilă(elemente)[,variabilă(elemente)]...*

**Comentariu:** *variabilă* este numele matricii.

*element* este o listă de expresii numerice, separate prin virgule, elemente care definesc dimensiunile matricii.

La execuție, instrucțiunea DIM atribuie, inițial, valoarea zero tuturor elementelor matricii. Elementele unei matrici șir sînt, toate, de lungime variabilă, cu valoare inițială nulă (șir de lungime zero).

Dacă numele unei variabile vectoriale este utilizat fără a se fi declarat dimensiunea acesteia cu ajutorul instrucțiunii DIM, numărul maxim de elemente este presupus a fi de 10. Numărul minim de elemente al unei matrici este totdeauna 0, în afara cazului cînd a fost menționat expres cu ajutorul instrucțiunii OPTION BASE. Numărul maxim de dimensiuni pentru o matrice este 255.

Redimensionarea unei matrici este prohibită, ea putîndu-se realiza, în cazul unui program doar după execuția instrucțiunii ERASE.

**Exemplu:** Acest exemplu crează două matrici; una unidimensională (SIS) cu 13 elemente și una bidimensională (LIT\$)

```
10 MAX=2
```

```
20 DIM SIS(12), LIT(MAX,2)
```

```
30 DATA 26.5, 37, 8, 29, 80, 9.9, &H800
```

```
40 DATA 7, 18, 55, 12, 5, 43
```

```
50 FOR I=0 TO 12
```

```
60 READ SIS(I)
```

```
70 NEXT I
```

```
80 DATA ION, VASILE, "A:"
```

```
90 DATA "Buna ziua", SALUT, Bonjour
```

```
100 DATA NOAPTE BUNA, BUCURESTI, PLOIESTI
```

```

110 FOR I=0 TO 2: FOR J=0 TO 2
120 READ LIT$(I,J)
130 NEXT J,I
140 PRINT SIS(3); LIT$(2,0)
RUN
29 NOAPTE BUNA

```

## DRAW

Instrucțiune

**Scop:** Desenează un obiect specificat într-un șir de caractere.

**Format:** *DRAW* șir

**Comentariu:** Instrucțiunea DRAW desenează obiecte utilizând un limbaj de definiții grafice. Comenzile limbajului sînt conținute într-o expresie șir. În timpul execuției BASIC-ul examinează valorile șirului. Cînd este dată o comandă de deplasare, se desenează o linie de la ultimul punct de referință. Valoarea lui  $n$  din următoarele comenzi de deplasare indică mărimea deplasării. Numărul de puncte de deplasare este  $n$  ori factorul de scalare. (atribuit prin comanda S). În continuare, sînt prezentate, detaliat, comenzile de deplasare.

**Un** Deplasare în sus.

**Dn** Deplasare în jos.

**Ln** Deplasare la stînga.

**Rn** Deplasare la dreapta.

**En** Deplasare pe diagonală sus și dreapta.

**Fn** Deplasare pe diagonală jos și dreapta.

**Gn** Deplasare pe diagonală jos și stînga.

**Hn** Deplasare pe diagonală sus și stînga.

**Mx,y** Deplasare absolută sau relativă. Dacă  $x$  are un semn plus (+) sau un semn minus (-) în fața sa, mișcarea este relativă, în alt caz este absolută.

Următoarele două prefixe de comandă pot precede oricare din aceste comenzi:

**B** Deplasare, dar fără plotare.

**N** Deplasare, dar cu întoarcere în poziția inițială.

Mai sînt disponibile următoarele comenzi:

**An** Atribuie unghiul  $n$ . Valoarea lui  $n$  poate varia între 0 și 3, unde 0 înseamnă 0 grade, 1 înseamnă 90 grade, 2 înseamnă 180 grade, iar 3, 270 grade. Figurile rotărite cu 90 și, respectiv, 270 grade sînt scalate, astfel încît



ele apar ca fiind de aceeași dimensiune cu cele la 0 și 180 grade, pe un display standard care are raportul de aspect 4/3.

**TAn** Întoarcere cu unghiul  $n$ . Valoarea lui  $n$  este cuprinsă între - 360 și 360. Dacă  $n$  este pozitiv (+), unghiul de întoarcere este contrar sensului acelor de ceasornic și invers pentru  $n$  negativ (-).

**Cn** Atribuie culoarea  $n$ . Valoarea lui  $n$  este cuprinsă între 0 și 3 în medie rezoluție, și 0 sau 1 în rezoluție înaltă. În medie rezoluție  $n$  selectează atributul de culoare specificat de instrucțiunea COLOR. Zero (0) este totdeauna atributul pentru fundal. Atributul inițial de culoare pentru prim-plan este atributul maxim de culoare pentru modul ecran curent: 3 în medie rezoluție și 1 în rezoluție înaltă.

**Sn** Atribuie factorul de scală. Valoarea lui  $n$  poate varia între 1 și 255. Factorul de scală este  $n$  împărțit la 4. De exemplu, dacă  $n = 1$ , atunci factorul de scală este 1/4. Valoarea inițială este 4, deci factorul de scală este 1.

**X variabilă**; Execută subșirul. Aceasta permite execuția unui al doilea șir în cadrul primului.

**P m,n** Setează culoarea figurii la  $m$  și culoarea ramei la  $n$ .

În toate comenzile descrise mai sus, argumentele  $m, n, x$  sau  $y$  pot fi constante (de ex. 123) sau variabile = **variabilă**; unde **variabilă** este numele unei variabile numerice. Pentru utilizarea unei astfel de variabile sau în cazul comenzii X, este necesar ca, după numele variabilei, să apară semnul (;). Se pot, deasemenea, specifica nume de variabile sub forma **VARPTR\$(variabilă)**, în loc de = **variabilă**; De exemplu:

**DRAW "XA\$;"** se poate scrie **DRAW "X" + VARPTR\$(A\$)**

**DRAW "S = SC;"** se poate scrie **DRAW "S = " + VARPTR\$(SC)**

Raportul de aspect al ecranului determină spațiul între punctele orizontale, verticale și diagonale. Instrucțiunea DRAW nu ține cont de raportul de aspect al modului ecran curent; astfel, **DRAW "R50 U50"**, va plota 50 de puncte la dreapta și 50 puncte în sus, dar liniile rezultate nu au lungime egală. Explicația rezidă din faptul că ecranul nu are un număr egal de puncte pe orizontală și pe verticală. Astfel, pentru a calcula raportul de aspect vom împărți numărul de puncte pe orizontală cu numărul de puncte pe verticală. Deci:

- pentru medie rezoluție : 320 puncte/200 puncte = 5/6

- pentru înaltă rezoluție: 640 puncte/320 puncte = 5/12

Dacă dorim să desenăm un pătrat în oricare din modurile de rezoluție trebuie scalată axa  $y$  cu raportul de aspect corespunzător, sau axa  $x$  1/raportul de aspect.

Pentru a desena un pătrat cu latura de 100 se scalează axa  $x$  cu:



10 '100\*6/5 inseamna 120

20 DRAW "U100 R120 D100 L120"

## EDIT

Comandă

**Scop:** Afișează o linie pentru editare.

**Format:** *EDIT linie*

**Comentariu:** *linie* este numărul unei linii existente în program.

Comanda **EDIT** afișează linia pe ecran poziționând cursorul pe primul caracter al liniei.

## END

Instrucțiune

**Scop:** Termină execuția programului, închide toate fișierele și dă controlul nivelului de comandă.

**Format:** *END*

**Comentariu:** Instrucțiunea **END** poate fi plasată oriunde într-un program pentru a termina execuția. **END** diferă de **STOP** în două moduri:

- **END** nu cauzează tipărirea mesajului **Break**.
- **END** închide toate fișierele.

Instrucțiunea **END** la sfârșitul programului este opțională.

**Exemplu:** În exemplu de mai jos, programul se termină dacă variabila **K** este mai mare decât 100; altfel, programul face salt la linia 20.

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

## ENVIRON

Instrucțiune

**Scop:** Modifică parametrii tabloului de referințe globale **BASIC**. **ENVIRON** este utilizat pentru a schimba parametrul "**PATH**" a unui proces descendent sau de a trece parametrii unui proces descendent prin inventarea unui nou tabel de referințe globale.

**Format:** *ENVIRON param = șir*

**Comentariu:** *param* este numele unui parametru ca de exemplu "**PATH**".

*șir* este textul care definește noul parametru.

*param* trebuie separat de *șir* prin semnul egal sau printr-un spațiu. **ENVIRON** ia drept *param* toate caracterele pînă la primul spațiu. Primul caracter care nu este spațiu sau semnul egal după *param* este luat drept *șir*.

**Exemplu:** Pentru a crea o cale de acces spre directorul rădăcină din

cititorul de disc A, folosim următoarea instrucțiune:

**ENVIRON "PATH = A:\".**

## ENVIRON\$

**Funcție**

**Scop:** Recuperează și afișează șirul specificat din tabelul de referințe globale BASIC.

**Format:**  $v\$ = ENVIRON\$(param)$

**sau**  $v\$ = ENVIRON\$(n)$

**Comentariu:** *param* este o expresie șir conținând parametrii de recuperat.

*n* este o expresie întreagă cu valori între 1 și 255.

Dacă este utilizat un argument șir, **ENVIRON\$** recuperează, din tabelul de referințe globale, un șir conținând textul dat de *param*.

În cazul în care este utilizat un argument numeric, **ENVIRON\$** recuperează un șir conținând al *n*-lea parametru din tabelul de referințe globale, pentru tot șirul *param* = text.

**ENVIRON\$** face distincție între majuscule și minuscule.

**Exemplu:** În momentul încărcării inițiale, sistemul de operare DOS atribuie un parametru numit "COMSPEC" care indică sistemului de operare unde să localizeze fișierul COMMAND.COM și atribuie o cale de acces nulă. Pentru a observa conținutul tabelului de referințe globale la momentul inițializării, se introduce următoarea instrucțiune BASIC:

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

ceea ce va avea drept rezultat tipărirea pe ecran:

```
PATH =
```

Dacă se introduce:

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

vom vedea afișat:

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

Dar dacă se introduce:

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

răspunsul calculatorului va fi:

```
A:COMMAND.COM
```

Programul următor salvează tabelul de referințe globale BASIC într-un vector, acest tabel putând fi modificat într-un proces descendent. După ce acest proces este încheiat, referințele globale sînt realocate:

```
10 DIM TABEL$(10) 'atribuie 10 parametri  
20 PARAM = 1 'numarul initial de parametri  
30 WHILE LEN(ENVIRON$(PARAM)) > 0
```

```

40 TABEL$(PARAM) = ENVIRON$(PARAM)
50 PARAM = PARAM + 1
60 WEND
70 PARAM = PARAM - 1 'ajusteaza parametrii
80 'acum se salveaza noile referinte globale
90 ENVIRON "DATAIN = C:\DATAIN\INP.FIL"
100 ENVIRON
"SORT.DAT = SORT.DAT < " + ENVIRON$("DATAIN"
) + "> LPT1:"
.
.
.
1000 SHELL ENVIRON$("SORT.DAT")
1010 FOR I = 1 TO PARAM
1020 ENVIRON TABEL$(I) 'resalveaza parametrii
1030 NEXT I
.
.
.

```

## EOF

**Funcție**

**Scop:** Indică condiția de sfârșit de fișier (**end-of-file**).

**Format:**  $v = EOF(numfis)$

**Comentariu:** *numfis* este numărul fișierului specificat în instrucțiunea OPEN.

Funcția EOF este utilă în evitarea erorii de tip **Input past end**. EOF întoarce valoarea -1 (adevărat) dacă a fost întâlnit sfârșitul fișierului și 0 (zero) dacă sfârșitul nu a fost identificat.

EOF are semnificație doar pentru fișierele deschise pentru intrare de pe disc sau pentru un fișier de comunicație. O valoare de -1 pentru fișierele de comunicație înseamnă că buffer-ul este gol.

**Exemplu:** Acest exemplu citește informații de la un fișier secvențial "DATE". Valorile sînt citite într-o matrice M pînă în momentul în care este depistat sfârșitul fișierului:

```

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

```



## ERASE

### Istrucțiune

**Scop:** Elimină matrici din program.

**Format:** *ERASE numemat[,numemat]...*

**Comentriu:** *numemat* este numele matricii de șters.

După ștergerea matricii din memorie spațiul alocat pentru acea matrice poate fi utilizată în alte scopuri.

Instrucțiunea **ERASE** este utilă pentru redimensionarea matricilor într-un program. Dacă se redimensionează matricile fără a se utiliza instrucțiunea **ERASE**, va fi semnalată eroare de tip **Duplicate definition**.

### Exemplu:

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

## ERDEV si ERDEV\$

### Variabile

**Scop:** Variabile de citire. Reține codul de eroare generat de **INTerrupt 24** a unei erori de periferic, precum și numele perifericului care a generat eroarea.

**Format:**  $v = ERDEV$   
 $v\$ = ERDEV\$$

**Comentariu:** În momentul detectării în sistemul de operare DOS a unei erori, **ERDEV** reține codul de eroare generat de **INTerrupt 24** în cei mai de jos 8 biți, iar cei mai de sus 8 biți conțin biții 13, 14 și 15 pentru atributul blocului de început al perifericului. **ERDEV\$** conține în două caractere numele perifericului (A:, B:, etc).

**Exemplu:** Deschizând fereastra cititorului de disc B: și introducând:  
**FILES "B:"**

se va afișa mesajul:

**Disk not ready**

și introducând:

**PRINT ERDEV, ERDEV\$**

se obține:

**2**

**B:**

Exemplul următor simulează o eroare la imprimantă:

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

Dacă se rulează acest exemplu cu imprimanta oprită, calculatorul va afișa:

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

## ERR si ERL

Variabile

**Scop:** Atribuie unei variabile codul de eroare și numărul liniei asociate cu o eroare.

**Format:**  $v = ERR$   
 $v = ERL$

**Comentariu:** Variabila **ERR** conține codul de eroare pentru ultima eroare, iar variabila **ERL** conține numărul liniei de program în care a fost semnalată eroarea.

**Exemplu:** Acest exemplu testează dacă fereastra cititorului de disc este deschisă:

```
10 ON ERROR GOTO 100
20 OPEN "DATE" FOR INPUT AS #1
30 END
.
.
.
100 IF ERR = 71 THEN LOCATE 23,1
110 PRINT "FEREASTRA DESCHISA !": RESUME
```

## ERROR

Instrucțiune

**Scop:** Simulează apariția unei erori, sau permite definirea unor coduri de erori ale utilizatorului.

**Format:** *ERROR n*

**Comentariu:**  $n$  trebuie să fie cuprins între 0 și 255.

Dacă  $n$  are aceeași valoare cu un cod de eroare folosit de limbajul BASIC, instrucțiunea ERROR simulează apariția aceluși tip de eroare. Dacă a fost definită o rutină de manipulare a erorilor de către instrucțiunea ON ERROR, este introdusă acea rutină de eroare. Altfel, este afișat mesajul de eroare corespunzător codului și se oprește execuția programului. Pentru definirea unor erori ale utilizatorului este necesară utilizarea unor valori diferite de cele BASIC.

**Exemplu:** Primul exemplu simulează o eroare de tip **String too long**:

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

Exemplul următor generează un cod de eroare (210) care nu este utilizat de BASIC:

```
100 ON ERROR GOTO 1000
110 INPUT "Deplasarea pe axa X = ", X
120 IF X > 639 THEN ERROR 210
.
.
.
1000 IF ERR = 210 THEN PRINT
"Limita pentru axa X = 639"
1010 IF ERR = 210 THEN RESUME 110
```

## EXP

**Funcție**

**Scop:** Calculează funcția exponențială.

**Format:**  $v = \text{Exp}(x)$

**Comentariu:**  $x$  poate fi orice expresie numerică.

Funcția atribuie variabilei  $v$  numărul matematic  $e$  ridicat la puterea  $x$ , unde  $e$  este baza logaritmilor naturali. Dacă  $x$  este mai mare decât 88.02969 se va semnala eroare.

**Exemplu:**

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



## FIELD

Instrucțiune

**Scop:** Aloca spațiu variabilelor dintr-un fișier aleator.

**Format:** *FIELD* [#*nrfis*,*lung AS varsir*],*lung AS varsir*...

**Comentariu:** *nrfis* este numărul sub care a fost deschis fișierul.

*lung* este o expresie numerică ce specifică numărul de caractere alocat variabilei *varsir*.

*varsir* este o variabilă șir utilizată pentru accesarea fișierului aleator.

**Exemplu:**

```
10 OPEN "FOO" AS #1
```

```
20 FIELD 1, 100 AS A#, 200 AS B$
```

```
30 FIELD 1, 300 AS DMY$, 40 AS C$
```

## FILES

Comandă

**Scop:** Afișează numele fișierelor din directorul discului curent.

**Format:** *FILES* [*filespec*]

**Comentariu:** *filespec* este o expresie șir. Dacă se omite *filespec* sînt afișate toate fișierele de pe directorul curent. Regulile de citire sînt aceleași ca în cazul comenzii *dir* din sistemul de operare DOS.

## FIX

Funcție

**Scop:** Trunchiază  $x$  la un întreg.

**Format:**  $v = \text{FIX}(x)$

**Comentariu:**  $x$  poate fi orice expresie numerică. Diferența între *FIX* și *INT* este că, în cazul funcției *FIX*, nu este evaluat următorul număr mai mic pentru un  $x$  negativ.

**Exemple:**

```
PRINT FIX(33.82)
```

```
33
```

```
PRINT FIX(-2.77)
```

```
-2
```

## FOR și NEXT

Instrucțiuni

**Scop:** Execută o serie de instrucțiuni într-o buclă, de un număr dat de ori.

**Format:** *FOR var = x TO y [STEP z]*

•  
•  
•  
**NEXT [var[,var]...]**

**Comentariu:** *var* este o variabilă întreagă sau simplă precizie utilizată drept contor.

*x* este o expresie numerică care reprezintă valoarea inițială a contorului.

*y* este o expresie numerică care reprezintă valoarea finală a contorului.

*z* este o expresie numerică utilizată drept increment.

Liniile de program care urmează instrucțiunii **FOR** sînt executate pînă la înlînirea instrucțiunii **NEXT**. Apoi contorul este incrementat cu valoarea incrementului *z*. Dacă nu se specifică o valoare pentru *z*, incrementarea se face cu o unitate. După incrementare se realizează o verificare pentru a se vedea dacă valoarea contorului nu este mai mare decît valoarea finală. Dacă nu este mai mare, BASIC-ul face salt la instrucțiunea imediat următoare instrucțiunii **FOR** și procesul se repetă. În cazul în care contorul este mai mare decît valoarea finală, se dă controlul instrucțiunii care urmează instrucțiunii **NEXT**.

Dacă *z* este negativ se execută decrementarea contorului pînă cînd acesta este mai mic decît valoarea finală.

Buclele **FOR-NEXT** pot fi întrepătrunse, aceasta însemnînd ca o buclă poate fi plasată în interiorul alteia. Dacă buclele sînt întrepătrunse, fiecare din ele trebuie să aibă un nume de variabilă de contor unic. Instrucțiunea **NEXT** pentru o buclă interioară trebuie să apară înaintea instrucțiunii **NEXT** pentru o buclă exterioară. Dacă buclele întrepătrunse au același punct de terminare poate fi utilizată o singură instrucțiune **NEXT** pentru toate. De exemplu, o instrucțiune de forma:

**NEXT var1, var2, var3,...**

este echivalentă cu:

**NEXT var1**

**NEXT var2**

**NEXT var3**

•

•

•

Variabila sau variabilele din instrucțiunea **NEXT** pot fi omise, în care caz instrucțiunea **NEXT** se referă la cea mai recentă instrucțiune **FOR**. Este, totuși, o idee bună de a include variabilele pentru a preîntîmpina

orice confuzie.

Dacă instrucțiunea **NEXT** este întâlnită înaintea instrucțiunii **FOR** corespunzătoare, va fi semnalată eroare de tip **Next without FOR**.

**Exemplu:**

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

În acest exemplu, bucla se va executa de 10 ori. Spre deosebire de alte versiuni BASIC (în care bucla se execută doar de 6 ori), valoarea finală a variabilei de buclă este atribuită înaintea valorii inițiale.

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

## FRE

**Funcție**

**Scop:** Atribuie unei variabile numărul de octeți neutilizați din spațiul de date BASIC.

**Format:**  $v = FRE(x)$   
 $V = FRE(x\$)$

**Comentariu:**  $x$  și  $x\$$  sînt argumente false.

Avînd în vedere că în BASIC șirurile pot avea lungime variabilă, acestea pot fi manipulate dinamic. Din această cauză spațiul de memorie alocat șirurilor poate deveni fragmentat, ceea ce are repercursiuni asupra performanțelor programului. Funcția **FRE** de un argument șir oarecare produce o "curățenie generală" înainte de a evalua numărul neutilizat de octeți. În acest timp, BASIC-ul colectează toate datele utile și eliberează spațiul neutilizat de memorie folosit cîndva, în timpul rulării, pentru șiruri.



Zona de date este comprimată, astfel putîndu-se continua pînă în momentul depășirii memoriei de lucru.

**Exemplu:**

```
PRINT FRE(0)
22325
```

Evident că rezultatul acestui exemplu poate diferi în timpul rulării pe calculatorul dumneavoastră.

## GET

Instrucțiune (fișiere)

**Scop:** Citește o înregistrare dintr-un fișier aleator într-un tampon de memorie aleator.

**Format:** *GET [#]nrfis[,număr]*

**Comentariu:** *nrfis* este numărul sub care a fost deschis fișierul.

*număr* este numărul înregistrării de citit, între 1 și 16 megaoceteți.

Dacă argumentul *număr* este omis, în tamponul de memorie se va citi următoarea înregistrare (după ultima instrucțiune GET).

Instrucțiunea GET poate fi utilizată de asemenea în cazul fișierelor de comunicații, în care caz argumentul *număr* reprezintă numărul de oceteți ce poate fi citit din memoria tampon de comunicații. Acest număr nu poate depăși valoarea atribuită prin opțiunea LEN din instrucțiunea OPEN "COM...

**Exemplu:** Acest exemplu deschide fișierul "FISA" pentru acces aleator, cu cîmpurile definite în linia 20. Instrucțiunea GET de la linia 30 citește o înregistrare într-un tampon de memorie de fișier. Linia 40 afișează info mații din înregistrarea citită.

```
10 OPEN "A:FISA" AS #1
20 FIELD 1,30 AS NUME$, 30 AS ADR$, 35 AS
ORAS$
30 GET 1
40 PRINT NUME$, ADR$, ORAS$
```

## GET

Instrucțiune (Grafică)

**Scop:** Citește puncte dintr-o arie a ecranului.

**Format:** *GET(x1,y1)-(x2,y2),vector*

**Comentariu:** *x1,x2,y1,y2* sînt coordonate în forma relativă sau absolută.

*vector* este numele vectorului în care va fi păstrată informația. GET citește atributele punctelor dintr-un patruleter specificat într-un vector.

Patrulaterul are colțurile  $(x_1, y_1)$  și  $(x_2, y_2)$ .

Instrucțiunile GET și PUT pot fi utilizate pentru animarea obiectelor în modul grafic. Aceste instrucțiuni pot fi gândite ca niște operații de "pompare a biților" spre (PUT) și dinspre (GET) ecran. Matricea (vectorul) este utilizată pentru păstrarea imaginii, ea trebuind să fie numerică. Mărimea matricei în octeți este dată de relația:

$$4 + \text{INT}((x * \text{bitiperpixel} + 7)/8) * y$$

unde  $x$  și  $y$  sînt lungimea orizontală respectiv verticală a patrulelului. Valoarea lui *bitiperpixel* este 2 în medie rezoluție și 1 în înaltă rezoluție. Să presupunem că dorim să utilizăm instrucțiunea GET pentru a "prinde" o imagine de 10 pe 12 pixeli în medie rezoluție. Numărul de octeți necesar este  $4 + \text{INT}((10*2 + 7)/8) * 12$ , sau 40 octeți. Numărul de octeți per element de matrice este:

- 2 pentru șirul întreg
- 4 pentru șirul simplă precizie
- 8 pentru șirul dublă precizie

De aici rezultă că în cazul nostru se poate folosi o matrice întreagă cu cel puțin 20 de elemente. Informația de pe ecran este memorată în vector după cum urmează:

1. 2 octeți pentru dimensiunea lui  $x$  în biți.
2. 2 octeți pentru dimensiunea lui  $y$  în biți.
3. datele.

De reținut că întregii sînt memorați: octetul inferior primul, apoi octetul superior. Datele pentru fiecare rînd de puncte sînt aliniate la stînga în interiorul unui octet, astfel că, dacă este memorat mai puțin de un multiplu de 8 biți, restul octetului este umplut cu zerouri. GET și PUT lucrează sensibil mai rapid în medie rezoluție dacă  $x_1 \text{ MOD } 4$  este egal cu zero și în înaltă rezoluție dacă  $x_1 \text{ MOD } 8$  este de asemenea egal cu zero.

**Exemplu:** Vezi instrucțiunea (grafică) PUT.

## GOSUB și RETURN

Instrucțiuni

**Scop:** Execută salt și întoarcere dintr-o subrutină.

**Format:** *GOSUB linie1*  
*RETURN [linie2]*

**Comentariu:** *linie1* este numărul liniei de început a unei subrutine.

*linie2* este un număr opțional de linie la care se dă controlul după executarea subrutinei. Dacă *linie2*, se omite controlul programului este dat instrucțiunii imediat următoare celei mai recente instrucțiuni GOSUB.

O subrutină poate fi apelată de ori de cîte ori este nevoie într-un program, sau poate fi apelată din cadrul altei subrutine, această întrețesere

fiind limitată doar de memoria disponibilă.

**Exemplu:**

```
10 GOSUB 70
20 PRINT "TERMINAT SUBRUTINA 1"
30 GOSUB 90
40 PRINT "TERMINAT SUBRUTINA 2"
50 PRINT "TERMINAT TOTUL"
60 END
70 PRINT "*** SUBRUTINA 1 ***"
80 RETURN
90 PRINT "*** SUBRUTINA 2 ***"
100 RETURN
RUN
** SUBRUTINA 1 **
TERMINAT SUBRUTINA 1
** SUBRUTINA 2 **
TERMINAT SUBRUTINA 2
TERMINAT TOTUL
```

## GOTO

Instrucțiune

**Scop:** Execută salt necondiționat din secvența normală de program la un număr de linie specificat.

**Format:** *GOTO linie*

**Comentariu:** *linie* este un număr de linie din program.

Dacă se specifică numărul liniei unei instrucțiuni executabile, este executată acea instrucțiune precum și cele care urmează. Dacă numărul de linie se referă la o instrucțiune neexecutabilă (REM sau DATA), programul continuă cu prima instrucțiune executabilă care urmează.

Instrucțiunea **GOTO** poate fi utilizată în modul direct pentru a da execuția programului la o linie dorită (de exemplu în timpul depanării unui program).

**Exemplu:** În acest exemplu instrucțiunea **GOTO** din linia 60 pune programul într-o buclă infinită, ce se termină în momentul în care nu mai pot fi citite date din instrucțiunea **DATA**.

```
10 DATA 5,7,12
20 READ R
30 PRINT "R = ";R,
40 A = 3.14*R^2
50 PRINT "ARIA = ";A
```



60 GOTO 10

RUN

R = 5

ARIA = 78.5

R = 7

ARIA = 153.86

R = 12

ARIA = 452.16

OUT OF DATA IN 20

## HEX\$

Funcție

**Scop:** Evaluează un șir ce reprezintă valoarea hexazecimală a argumentului zecimal.

**Format:**  $v\$ = \text{HEX}\$(n)$

**Comentariu:**  $n$  este o expresie numerică cu valori cuprinse în intervalul -32768 la 65535. Dacă  $n$  este negativ este folosită forma complementului față de doi. Astfel  $\text{HEX}\$(-n)$  este echivalent cu  $\text{HEX}\$(65535-n)$ .

**Exemplu:**

```
10 INPUT X
```

```
20 A$ = HEX$(X)
```

```
30 PRINT X " ZECIMAL INSEAMNA " A$ "  
HEXAZECIMAL"
```

```
40 GOTO 10
```

```
RUN
```

```
? 32
```

```
32 ZECIMAL INSEAMNA 20 HEXAZECIMAL
```

```
? 1023
```

```
1023 ZECIMAL INSEAMNA 3FF HEXAZECIMAL
```

## IF

Instrucțiune

**Scop:** Permite luarea unei decizii după evaluarea unei expresii.

**Format:** *IF expresie [,] THEN clauză [ELSE clauză]*  
*IF expresie [,] GOTO linie [[,] ELSE clauză]*

**Comentariu:** *expresie* poate fi orice expresie numerică.  
*clauză* poate fi o instrucțiune sau o secvență de instrucțiuni BASIC (separate prin două puncte), sau un număr de linie pentru salt.  
*linie* este numărul unei linii de program.

Dacă *expresia* este adevărată (nonzero) atunci se va executa clauza **THEN** sau **GOTO**.

Dacă rezultatul *expresiei* este fals (zero), clauza **THEN** sau **GOTO** este ignorată, executându-se clauza **ELSE**, dacă este prezentă. Execuția

continuă cu următorul număr de linie ce conține o instrucțiune executabilă.

Instrucțiunile **IF-THEN-ELSE** pot fi întrepătrunse. Întrepătrunderea este limitată doar de lungimea liniei. De exemplu:

```
IF X > Y THEN PRINT "MAI MARE" ELSE IF Y > X  
THEN PRINT "MAI MIC" ELSE PRINT "EGAL"
```

este o instrucțiune validă. În cazul în care instrucțiunea nu conține același număr de clauze **ELSE** și **THEN**, fiecărei clauze **ELSE** i se opune cea mai apropiată clauză **THEN**, astfel:

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

nu va afișa "A < > C" când "A < > B".

**Exemplu:** În exemplul următor dacă I este cuprins între 10 și 20, este calculat DB și se execută salt la linia 300. Dacă I nu este cuprins în acest domeniu se va afișa mesajul "Greseala !". De notat utilizarea a două instrucțiuni în clauza **THEN**:

```
100 IF (I > 10) AND (I < 20) THEN  
DB = 1988-I: GOTO 300 ELSE  
PRINT "Greseala !"
```

În exemplul următor, în linia 20, tot ceea ce urmează după **THEN** este parte a clauzei. Aceasta înseamnă că instrucțiunea **NEXT** nu este executată pînă cînd  $N = I$ . Cînd se execută linia 20,  $N$  nu este egal cu  $I$  astfel că evaluarea **IF** dă rezultat fals. Deci, instrucțiunea **NEXT** nu va fi executată, iar programul se va termina la linia 30. Dacă se dorește execuția buclei pînă cînd  $N = I$ , instrucțiunea **NEXT** trebuie să apară sau pe o linie separată, sau anticipată de o clauza **ELSE**:

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

## INKEY\$

Variabilă

**Scop:** Citește un caracter de la tastatură.

**Format:**  $v\$ = INKEY\$$

**Comentariu:** **INKEY\$** citește doar un singur caracter, chiar dacă în tamponul de memorie al tastaturii așteaptă mai multe caractere. Valoarea rezultantă este un șir de zero, unu sau două caractere.

- un șir nul (lungime zero) înseamnă că nu a fost citit nici un caracter de la tastatură.
- un șir de lungime un caracter conține caracterul actual citit de la tastatură.
- un șir de lungime două caractere indică un cod special extins.

Înainte de utilizarea acestui caracter într-o funcție sau instrucțiune BASIC, este necesară atribuirea rezultatului la o variabilă șir.

**Exemplu:** Următoarea secțiune a unui program oprește execuția pînă cînd este apăsată o tastă:

```
100 PRINT "Apasati o tasta pentru continuare"
110 A$ = INKEY$: IF A$ = "" THEN 110
```

Următorul exemplu poate fi utilizat pentru testarea codului de două caractere rezultat:

```
10 K$ = INKEY$
20 IF LEN(K$) = 20 THEN K$ = RIGHT$(K$,1)
```

## INP

Funcție

**Scop:** Evaluează un octet citit de la portul  $n$ .

**Format:**  $v = INP(n)$

**Comentariu:**  $n$  trebuie să fie cuprins între 0 și 65535.

INP este complementara funcției OUT

**Exemplu:**

```
100 A = INP(255)
```

## INPUT

Instrucțiune

**Scop:** Primește informații de la tastatură în timpul execuției programului.

**Format:**  $INPUT[;][\text{"mesaj"};]variabilă[,variabilă]...$

**Comentariu:** "mesaj" este un șir constant care indică cerința intrării. *variabilă* este numele unei variabile numerice, șir sau element al unei matrici ce va recepționa informația.

În timpul execuției, cînd programul ajunge la o instrucțiune INPUT, se afișează pe ecran un semn de întrebare pentru a indica faptul că programul așteaptă introducerea unor date. Dacă este inclus un "mesaj" se va afișa acel șir de caractere. Dacă mesajul este urmat de punct și virgulă după mesaj se va tipări și un semn de întrebare. Dacă mesajul este urmat de virgulă, semnul de întrebare nu se va mai tipări. De exemplu, în instrucțiunea INPUT "Data nasterii",DN\$, mesajul va fi afișat fără să fie urmat de un semn de întrebare.



În cazul în care programul așteaptă introducerea a mai mult de un element de la tastatură, elementele vor fi introduse cu virgulă între ele.

Tipul elementelor introduse trebuie să coincidă cu tipul declarat al variabilelor cărora li se vor atribui.

Dacă instrucțiunea INPUT este imediat urmată de punct și virgulă, apăsând tasta ENTER, acesta nu va executa un carriage return/line feed, ceea ce înseamnă că cursorul va rămîne pe aceeași linie.

**Exemplu:**

```
10 PI = 3.14
20 INPUT "Introduceți raza ";R
30 A = PI*R ^ 2
40 PRINT "Aria cercului este ";A
50 END
RUN
Introduceți raza ?7.4
Aria cercului este 171.9464
```

## INPUT #

Instrucțiune

**Scop:** Citește date de la un periferic secvențial sau fișier și le atribuie unor variabile din program.

**Format:** INPUT #*nrfis*,*variabilă*[,*variabilă*]...

**Comentariu:** *nrfis* este numărul folosit în momentul deschiderii fișierului pentru intrare.

*variabilă* este numele unei variabile ce va avea drept corespondent un element al fișierului.

Fișierul secvențial poate fi pe disc sau casetă; el poate fi un șir de date secvențial de la un adaptor de comunicație sau chiar tastatură (KYBD:).

Tipul datelor din fișier trebuie să corespundă cu tipul variabilei declarate în program. Elementele fișierului trebuie să apară întocmai ca și în cazul în care acestea ar fi fost citite cu instrucțiunea INPUT. În cazul valorilor numerice vor fi ignorate spațiile libere precum și "carriage return" și "line feed". Numerele se termină cu un spațiu, o virgulă, un "carriage return" sau "line feed".

## INPUT\$

Funcție

**Scop:** Evaluează un șir de *n* caractere citite de la claviatură sau de la fișier.

**Format:**  $v\$ = INPUT\$(n[, \#nr\text{fis}])$

**Comentariu:**  $n$  este numărul de caractere ce urmează a fi citit.

$nr\text{fis}$  este numărul fișierului utilizat în instrucțiunea precedentă OPEN. Dacă se omite  $nr\text{fis}$  citirea se va face de la tastatură.

Dacă se folosește pentru intrare tastatura, caracterele corespunzătoare introduse, nu vor fi afișate pe ecran.

Funcția INPUT\$ permite citirea caracterelor cu semnificație pentru editorul BASIC ca de exemplu Backspace (cod ASCII 8). Dacă se dorește citirea acestor caractere se vor folosi INPUT\$ sau INKEY\$ (niciodată INPUT sau LINE INPUT).

Pentru fișierele de comunicație, funcția INPUT\$ este de preferat înstrucțiunilor INPUT # sau LINE INPUT # pentru care toate caracterele pot fi semnificative în comunicație.

**Exemplu:** În exemplul următor se citește un singur caracter de la tastatură ca răspuns la mesaj:

```
100 PRINT "Apasati C pentru continuare sau S pentru stop"
110 X$ = INPUT$(1)
120 IF X$ = "C" THEN GOTO 200
130 IF X$ = "S" THEN STOP ELSE GOTO 100
```

## INSTR

Funcție

**Scop:** Caută prima apariție a șirului  $y\$$  în șirul  $x\$$ , rezultatul funcției fiind poziția în șir pe care se găsește acesta. Parametrul opțional  $n$  reprezintă poziția în șirul  $x\$$  de la care va începe operația de căutare.

**Format:**  $v = INSTR([n, ]x\$, y\$)$

**Comentariu:**  $n$  este o expresie numerică cu valori cuprinse între 1 și 255.

$x\$, y\$$  pot fi orice variabile șir, expresii șir sau constante șir.

Dacă  $n$  este mai mare decât  $LEN(x\$)$ , dacă  $x\$$  este nul sau dacă  $y\$$  nu poate fi găsit, rezultatul funcției INSTR este  $n$  (sau 1 dacă  $n$  nu este specificat).

**Exemplu:** În exemplul următor se caută șirul "B" în șirul "ABCDEB". Când căutarea se face de la primul caracter al șirului, "B" va fi găsit pe poziția 2; când căutarea începe de la poziția 4, "B" va fi găsit la poziția 6:

```
10 A$ = "ABCDEB"
20 B$ = "B"
30 PRINT INSTR(A$, B$); INSTR(4, A$, B$)
```

**INT****Funcție****Scop:** Evaluează cel mai mare întreg care este mai mic sau egal cu argumentul  $x$ .**Format:**  $v = INT(x)$ **Comentariu:**  $x$  poate fi orice expresie numerică.

Vezi de asemenea funcțiile FIX și CINT (care evaluează la rîndul lor valori întregi).

**Exemplu:** Acest exemplu arată modul în care funcția INT trunchează numerele pozitive, dar rotunjește "în sus" numerele negative (în direcția negativă).

**PRINT INT(45.77)**

45

**PRINT INT(-2.89)**

-3

**IOCTL****Instrucțiune****Scop:** Permite trimiterea unui șir de control spre un "driver" de periferic, după ce acesta a fost, în prealabil, deschis.**Format:** *IOCTL [#]nrfis,sir*

**Comentariu:** *nrfis* este numărul fișierului pentru "driver"-ul de periferic.

*sir* este o expresie șir conținînd datele de control.

Sistemul de manipulare a fișierelor de intrare/ieșire BASIC permite crearea și instalarea unor programe de control a perifericelor (driver) proprii utilizatorului. Instrucțiunea IOCTL și funcția IOCTL\$ trimit și recepționează date de control de la periferice.

Un șir de comandă IOCTL poate fi de pînă la 255 octeți lungime. Sînt acceptate comenzi multiple în cadrul șirului, separate unele de altele prin punct și virgulă:

**"LF;PL66;LW132"**

Posibilele comenzi sînt determinate de tipul "driver"-ului instalat.

**Exemplu:** În exemplul care urmează se modifică lungimea paginii la imprimantă:

**10 OPEN "LPT1" FOR OUTPUT AS #1****20 IOCTL #1,"PL60"**



## IOCTL\$

Funcție

**Scop:** Citește un șir de date de control de la un "driver" de periferic care a fost, anterior, deschis.

**Format:**  $v\$ = IOCTL\$( / \# / nrfis )$

**Comentariu:** *nrfis* este numărul fișierului deschis către periferic.

Funcția **IOCTL\$** poate fi utilizată pentru a confirma succesul executării comenzii **IOCTL**.

**Exemplu** Exemplul următor verifică dacă un șir de control a fost recepționat corect.

```
10 OPEN "COM" AS #1
20 IOCTL #1,"SW132;GW"
30 IF IOCTL$(1) = "132" THEN
PRINT "INSTALAT CORECT"
```

## KEY

Instrucțiune

**Scop:** Instalează sau afișează tastele programabile (notate cu F1...F10).

**Format:** *KEY ON*  
*KEY OFF*  
*KEY LIST*  
*KEY n,x\$*  
*KEY n,CHR\$(KBflag) si CHR\$(scan code)*

**Comentariu:** **KEY ON** determină afișarea valorilor tastelor programabile pe linia 25. Când lățimea ecranului este de 40 de caractere, se vor afișa 5 din cele 10 taste programabile. Când lățimea este de 80, toate cele 10 taste vor fi afișate.

**KEY OFF** șterge afișarea acestor valori de pe linia 25, permițând utilizarea prin program și a acestei linii.

**KEY LIST** listează valorile celor 10 taste programabile

**KEY n,x\$** permite programarea acestor taste cu o secvența de caractere ce va fi reprodușă automat la apăsarea tastei respective.

*n* este numărul tastei programabile. Poate lua orice valoare între 1 și 10.

*x\$* este o expresie șir ce va fi atribuită tastei *n*

**Exemple:**

```
50 KEY ON
```

afișează conținutul tastelor programabile pe linia 25.

```
10 KEY OFF
```

șterge afișarea acestor taste de pe linia 25, dar ele rămân active.

**10 KEY 1,"FILES" + CHR\$(13)**

atribuie tastei numărul 1 conținutul șirului "FILES" + Enter.

## KEY(n)

Instrucțiune

**Scop:** Activează sau dezactivează tastele specifice într-un program BASIC.

**Format:** KEY(n) ON  
KEY(n) OFF  
KEY(n) STOP

**Comentariu:** *n* este o expresie numerică cuprinsă între 1 și 20, după cum urmează:

**1-10** tastele programabile F1 la F10.

**11** Cusor sus.

**12** Cusor stînga.

**13** Cusor dreapta.

**14** Cusor jos.

**15-20** taste definite sub forma:

**KEY n,CHR\$(KBflag) + CHR\$(scan code).**

## KILL

Comandă

**Scop:** Șterge un fișier de pe disc. Comanda KILL din BASIC este similară comenzii ERASE din DOS.

**Format:** KILL numefis

**Comentariu:** *numefis* este o expresie șir care specifică numele fișierului.

Comanda KILL poate fi utilizată pentru toate tipurile de fișiere. Numele trebuie să conțină și extensia dacă există. De exemplu, se poate salva un program BASIC cu comanda:

**SAVE "TEST"**

Limbajul adaugă automat extensia .BAS la numele fișierului după comanda SAVE, dar nu și după comanda KILL. Dacă se dorește ștergerea acestui fișier trebuie introdusă comanda:

**KILL "TEST.BAS"**

## LEFT\$

Funcție

**Scop:** Extrage cele mai din stînga *n* caractere din șirul *x\$*.

**Format:**  $v\$ = LEFT\$(x\$,n)$

**Comentariu:**  $x\$\$  este orice expresie șir.

$n$  este o expresie numerică ce trebuie să fie cuprinsă între 0 și 255.<sup>f</sup> Specifică numărul de caractere ce urmează a fi citite. Dacă  $n$  este mai mare sau egal cu  $LEN(x\$\)$ , atunci se va extrage întregul șir  $x\$\$ . Dacă  $n = 0$ , va fi extras șirul nul (de lungime zero).

**Exemplu:**

```
10 A$ = "MULT/PUTIN"  
20 B$ = LEFT$(A$,4)  
30 PRINT B$  
RUN  
MULT
```

## LEN

**Funcție**

**Scop:** Evaluează numărul de caractere din șirul  $x\$\$ .

**Format:**  $v = LEN(x\$\)$

**Comentariu:**  $x\$\$  este orice expresie șir.

Afît caracterele neprintabile cît și spațiile sînt incluse în numărul total de caractere.

**Exemplu:**

```
10 X$ = "ABCD DCBA"  
20 PRINT LEN(X$)  
RUN  
9
```

## LET

**Instrucțiune**

**Scop:** Atribuie unei variabile valoarea unei expresii.

**Format:**  $[LET] \text{ variabilă} = \text{expresie}$

**Comentariu:** *variabilă* este numele unei variabile sau al unui element vectorial.

*expresie* este expresia a carei valoare este atribuită *variabilei*. Tipul expresiei (șir sau numerice) trebuie să concorde cu cel al variabilei, în caz contrar semnalîndu-se eroare.

Cuvîntul **LET** este opțional, ceea ce înseamnă că semnul de egalitate este suficient pentru a atribui unei variabile o expresie.

**Exemplu:**

```
10 LET DORI = 12  
20 LET E = DORI + 2  
30 LET FDANCE$ = "HORA"
```

Aceleași instrucțiuni mai pot fi scrise și:



10 DORI = 12  
20 E = DORI + 2  
30 FDANCE\$ = "HORA"

## LINE

Instrucțiune

**Scop:** Desenează o linie sau un dreptunghi pe ecran.

**Format:** *LINE [(x1,y1)-(x2,y2)[, [color][, B[F]][, stil]]*

**Comentariu:**  $(x1,y1), (x2,y2)$  sînt coordonatele în forma relativă sau absolută.

*color* este o expresie întregă. Alege atributul de culoare al modului ecran curent.

*stil* este un număr întreg de 16 biți folosit pentru a pune puncte pe ecran. Opțiunea *stil* poate fi folosită pentru a desena o linie punctată. Deoarece *stil* are lungimea de 16 biți, un model de linie punctată poate arăta în felul următor:

1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

ceea ce înseamnă &HAAAA în notație hexazecimală.

*B* desenează un dreptunghi avînd colțurile opuse de coordonate  $(x1,y1)$  respectiv  $(x2,y2)$ .

*F* umple dreptunghiul cu o culoare.

**Exemple:** Programul următor desenează o linie punctată:

10 SCREEN 1,0

20 LINE (0,0)-(319,199),,,&HAAAA

Pentru a folosi forma relativă de introducere a celei de-a doua perechi de coordonate față de prima pereche, se va introduce de exemplu:

LINE (100,100)-STEP (10,-20)

care va desena o linie de la (100,100) la (110,80).

Exemplul următor va desena dreptunghiuri aleatoare umplute cu culori aleatoare:

10 CLS

20 SCEEN 1,0: COLOR 0,0

30 LINE -(RND\*319,RND\*199),RND\*2 + 1,BF

40 GOTO 30 'dreptunghiurile se vor suprapune

## LINE INPUT

Instrucțiune

**Scop:** Citește o linie întregă (pînă la 255 de caractere) de la tastatură într-o variabilă șir, ignorînd delimitatorii.

**Format:** *LINE INPUT[;][ "comentariu"; ] varsir*

**Comentariu:** "comentariu" este o constantă șir ce va fi afișată pe

ecran. Semnul întrebării nu este afișat, doar dacă acesta face parte din șirul comentariu.

*varsir* este numele variabilei șir sau al elementului matricei la care se va atribui linia.

Dacă **LINE INPUT** este urmat imediat de punct și virgulă, atunci apăsînd tasta Enter nu se va produce deplasarea cursorului la începutul liniei următoare; cu alte cuvinte cursorul rămîne pe aceeași linie.

## LINE INPUT #

Instrucțiune

**Scop:** Citește o linie întreagă (pînă la 255 de caractere), ignorînd delimitatorii, de la un fișier secvențial într-o variabilă șir.

**Format:** *LINE INPUT #nrfis,varsir*

**Comentariu:** *nrfis* este numărul sub care a fost deschis fișierul.

*varsir* este numele variabilei șir sau elementului matricial căruia i se atribuie linia.

**LINE INPUT #** citește toate caracterele din fișierul secvențial pînă la un "carriage return".

**Exemplu:**

```
10 OPEN "LST" FOR OUTPUT AS #1
```

```
20 LINE INPUT "Adresa? ";C$
```

```
30 PRINT #1,C$
```

```
40 CLOSE 1
```

```
50 OPEN "LST" FOR INPUT AS #1
```

```
60 LINE INPUT #1,C$
```

```
70 PRINT C$
```

```
80 CLOSE 1
```

```
RUN
```

```
Adresa? Bdul MAGHERU, nr. 111
```

```
Bdul MAGHERU, nr. 111
```

## LIST

Comandă

**Scop:** Afișează programul curent din memorie pe ecran sau pe alt periferic specificat.

**Format:** *LIST [linie1][-[linie2]][,numefis]*

**Comentariu:** *linie1,linie2* reprezintă un număr de linie cuprins între 0 și 65529. *linie1* este prima linie de listat, iar *linie2* este ultima. Se poate folosi un punct (.) pentru a înlocui oricare din liniile de mai sus cu linia de program curentă.

*numefis* este o expresie șir ce indică numele fișierului.

**Exemple:** Exemplul de mai jos listează tot programul pe ecran:

**LIST**

Următorul exemplu listează linia 35:

**LIST 35**

Acest exemplu listează toate liniile de la 100 la sfârșitul programului către primul adaptor de comunicații cu 1200 bps (biți pe secundă), fără paritate, 8 biți de date, și 1 bit de stop:

**LIST 100- , "COM1:1200,N,8,1"**

Ultimul exemplu listează programul curent de la început și pînă la linia 150:

**LIST -150**

## LLIST

Comandă

**Scop:** Tipărește la imprimantă o parte dintr-un program sau un program întreg.

**Format:** *LLIST [lin1][-[lin2]]*

**Comentariu:** Vezi comanda LIST.

## LOAD

Comanda

**Scop:** Încarcă un program de la un periferic specificat în memorie și, opțional, dă comanda de rulare.

**Format:** *LOAD numefis[,R]*

**Comentariu:** *numefis* este o expresie șir care indică numele fișierului.

Comanda **LOAD** închide toate fișierele deschise anterior și șterge toate variabilele și toate liniile de program rezidente în memorie anterior comenzii de încărcare.

**LOAD numefis,R** este echivalentă comenzii **RUN numefis**.

**Exemple:** Comanda următoare încarcă și rulează programul PROG2.BAS:

**LOAD "PROG2.BAS",R**

aceasta fiind echivalentă cu comanda:

**RUN "PROG2.BAS"**

## LOC

Funcție

**Scop:** Evaluează poziția curentă într-un fișier.

**Format:**  $v = LOC(nrfis)$



**Comentariu:** *nrfis* este numărul fișierului utilizat la deschidere.

În cazul fișierelor aleatoare, **LOC** evaluează numărul ultimei înregistrări scrise sau citite din momentul deschiderii.

Pentru fișierele secvențiale, **LOC** evaluează numărul de înregistrări scrise sau citite din momentul deschiderii fișierului (o înregistrare pentru fișierele secvențiale înseamnă un bloc de date de 128 octeți).

Pentru fișierele de comunicații, **LOC** evaluează numărul de caractere ce așteaptă să fie citit din memoria tampon de intrare.

**Exemple:** Exemplul următor oprește execuția programului după ce se contorizează 50 de înregistrări:

```
100 IF LOC(1) > 50 THEN STOP
```

Următorul exemplu descrie o înregistrare după citire:

```
100 PUT #1,LOC(1)
```

## LOCATE

Instrucțiune

**Scop:** Poziționează cursorul pe ecranul activ. Parametrii opționali activează sau dezactivează cursorul și redefinesc dimensiunile acestuia.

**Format:** `LOCATE [rînd][,[col]][,[cursor]][,[start][,stop]]`

**Comentariu:** *rînd* este o expresie numerică cuprinsă între 1 și 25. Indică numărul liniei de ecran unde va fi plasat cursorul.

*col* este o expresie numerică cuprinsă între 1 și 40 sau între 1 și 80, depinzînd de modul ecran curent. Indică numărul coloanei de ecran unde va fi plasat cursorul.

*cursor* este o valoare ce indică starea cursorului (vizibil sau nu). Zero (0) indică dezactivat, iar unu (1) cursor activ.

*start* este linia de start a cursorului. Este o expresie numerică cuprinsă între 0 și 31.

*stop* este linia de stop a cursorului. Este o expresie numerică cuprinsă între 0 și 31.

Opțiunile *cursor*, *start* și *stop* nu se aplică în modurile grafice.

**Exemplu:**

```
10 LOCATE 5,1,1,0,7
```

## LOF

Funcție

**Scop:** Evaluează numărul de octeți alocat unui fișier (lungimea fișierului).

**Format:**  $v = LOF(nrfis)$

**Comentariu:** *nrfis* este numărul fișierului utilizat la deschidere.

## LOG

Funcție

**Scop:** Evaluează logaritmul natural al argumentului  $x$ .

**Format:**  $v = LOG(x)$

**Comentariu:**  $x$  este orice expresie numerică mai mare ca zero.

Logaritmul natural este logaritmul în baza  $e$ . Pentru calculul în dublă precizie, se va folosi comanda **/D** în linia de comandă BASIC.

**Exemple:** Primul exemplu calculează logaritmul expresiei 45/7:

```
PRINT LOG(45/7)
```

```
1.860752
```

Al doilea exemplu calculează logaritmul lui  $e$  și  $e^2$ :

```
E = 2.718282
```

```
? LOG(E)
```

```
1
```

```
? LOG(E*E)
```

```
2
```

## LPOS

Funcție

**Scop:** Evaluează poziția curentă a capului imprimantei (LPT1:) în timpul tipăririi.

**Format:**  $v = LPOS(n)$

**Comentariu:**  $n$  este o expresie numerică ce indică imprimanta testată, după cum urmează:

0 sau 1 LPT1:

2 LPT2:

3 LPT3:

**Exemplu:**

```
100 IF LPOS(0) > 60 THEN LPRINT CHR$(13)
```

## LPRINT și LPRINT USING

Instrucțiuni

**Scop:** Tipăresc date pe o imprimantă.

**Format:** *LPRINT* [lista de expresii [;]]

*LPRINT USING*  $v\$$ ;lista de expresii[;]

**Comentariu:** lista de expresii Este o listă de expresii șir și/sau numerice care urmează a fi tipărită. Expresiile trebuie să fie separate prin virgulă sau punct și virgulă.

$v\$$  este o constantă șir sau o variabilă care identifică formatul folosit la tipărire. Acesta este explicat în detaliu la instrucțiunea PRINT.

## LSET și RSET

Instrucțiuni

**Scop:** Mută date într-o memorie tampon de fișier aleator în pregătirea execuției instrucțiunii PUT (pentru fișiere).

**Format:** *LSET varsir = x\$*  
*RSET varsir = x\$*

**Comentariu:** *varsir* este numele unei variabile definite în instrucțiunea FIELD.

*x\$* este o expresie șir utilizată pentru plasarea informației în câmpul identificat de *varsir*.

Dacă *x\$* necesită mai puțini octeți decît cei specificați pentru *varsir* în instrucțiunea FIELD, LSET aliniază la stînga șirul în câmp, în timp ce RSET aliniază la dreapta.

## MERGE

Comandă

**Scop:** Unește linii de program dintr-un fișier ASCII, cu programul curent din memorie.

**Format:** *MERGE numefis*

**Comentariu:** *numefis* este o expresie șir care indică numele fișierului.

Fișierul este căutat după numele său. Dacă este găsit, liniile de program din fișierul respectiv sînt unite cu liniile din programul curent. Dacă în programul încărcat există linii cu același număr cu cel din memorie, liniile din fișier înlocuiesc pe cele deja existente în memorie.

**Exemplu:** Exemplul următor unește fișierul "PROG.OLD" aflat pe cititorul de disc "B:" cu programul rezident:

**MERGE "B:PROG.OLD"**

## MID\$

Funcție și Instrucțiune

**Scop:** Evaluează partea dorită a unui șir de caractere dat. Cînd este folosit ca instrucțiune, înlocuiește o porțiune a unui șir cu un alt șir.

**Format:**

Ca funcție:  $v\$ = MID\$(x\$,n[,m])$

Ca instrucțiune:  $MID\$(v\$,n[,m]) = y\$$

**Comentariu:**

Pentru funcție ( $v\$ = MID\$...$ ):

*x\$* este orice expresie întreagă;

*n* este o expresie întreagă cuprinsă între 1 și 255;



$m$  este o expresie întreagă cuprinsă între 0 și 255.

Funcția evaluează, din șirul  $x\$,$  un șir de lungime  $m$  caractere, începînd cu al  $n$ -lea caracter. Dacă se omite  $m$  sau dacă în dreapta se află mai puțin de  $m$  caractere începînd cu caracterul  $n,$  atunci se vor evalua cele mai din dreapta caractere din șirul  $x\$$  începînd cu caracterul  $n.$

Pentru instrucțiune (**MID** $\$... = y\$:$

$v\$$  este o variabilă șir în care se va opera înlocuirea.

$n$  este o expresie întreagă cuprinsă între 1 și 255.

$m$  este o expresie întreagă cuprinsă între 0 și 255.

$y\$$  este o expresie șir.

Caracterele din șirul  $v\$,$  începînd din poziția  $n$  sînt înlocuite cu caracterele din șirul  $y\$.$  Parametrul opțional  $m$  se referă la numărul de caractere din șirul  $y\$$  ce urmează a fi înlocuit. Dacă  $m$  este omis atunci tot șirul  $y\$$  este înlocuit.

**Exemple:** Primul exemplu utilizează funcția **MID** $\$$  pentru a selecta porțiunea din mijlocul șirului **B** $\$:$

```
10 A$ = "BUNA"
```

```
20 B$ = "DIMINEATA ZIUA SEARA"
```

```
30 PRINT A$;MID$(B$,10,5)
```

```
RUN
```

```
BUNA ZIUA
```

Următorul exemplu ilustrează utilizarea instrucțiunii **MID** $\$:$

```
10 FISA$ = "POPESCU - 97.39.66"
```

```
20 TEL.NOUS$ = "94.15.26"
```

```
30 MID$(FISA$,11,8) = TEL.NOUS$
```

```
40 PRINT FISA$
```

```
RUN
```

```
POPESCU - 94.15.26
```

## **MKDIR**

**Comandă**

**Scop:** Crează un director pe discul specificat.

**Format:** *MKDIR cale*

**Comentariu:** *cale* este o expresie șir de pînă la 63 de caractere, ce specifică noul director ce urmează a fi creat.

**Exemple:**

```
MKDIR "UTIL"
```

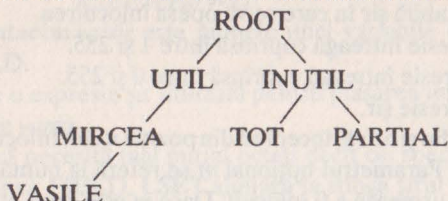
```
MKDIR "UTIL\MIRCEA"
```

```
MKDIR "UTIL\MIRCEA\VASILE"
```

```
MKDIR "INUTIL"
```

CHDIR "INUTIL"  
 MKDIR "TOT"  
 MKDIR "PARTIAL"  
 MKDIR "INUTIL\TOT"  
 MKDIR "INUTIL\PARTIAL"

Exemplele de mai sus au creat o structură arbore de tipul:



### MK\$,MK\$\$,MKD\$

**Funcții**

**Scop:** Converteste valori de tip numeric în valori de tip șir.

**Format:**  
*v\$ = MKI\$(expresie întreagă)*  
*v\$ = MKS\$(expresie simplă precizie)*  
*v\$ = MKD\$(expresie dublă precizie)*

**Comentariu:** Orice valoare numerică ce urmează a fi plasată într-o memorie tampon de fișier aleator cu instrucțiunile LSET sau RSET trebuie să fie convertite într-un șir. MKI\$ convertește un întreg într-un șir de 2 octeți, MKS\$ convertește un număr în simplă precizie într-un șir de 4 octeți, iar MKD\$ convertește un număr în dublă precizie într-un șir de 8 octeți.

Aceste funcții diferă de funcția STR\$ deoarece ele nu schimbă în mod real valoarea lor prin conversia în caractere, ci doar prin modul în care limbajul interpretează acești octeți.

**Exemplu:**

```

100 FIELD #1, 4 AS D$, 20 AS N$
110 LSET D$ = MKS$(J)
120 LSET N$ = A$
130 PUT #1
```

### NAME

**Comandă**

**Scop:** Schimbă numele unui fișier de pe disc. Comanda NAME în BASIC este echivalentă comenzii RENAME din sistemul de operare DOS.

**Format:** *NAME numefis1 AS numefis2*

**Comentariu:** *numefis1* este numele fișierului ce va fi schimbat. *numefis2* este noul nume de fișier.

**Exemplu:**

NAME "PROG.OLD" AS "PROG.NEW"

## NEW

Comandă

**Scop:** Șterge programul curent din memorie precum și toate variabilele.

**Format:** *NEW*

**Comentariu:** Comanda **NEW** este utilizată de obicei pentru a "îndepărta" un program din memorie înainte de a introduce un altul.

## OCT\$

Funcție

**Scop:** Evaluează un șir ce reprezintă valoarea octală a argumentului zecimal.

**Format:**  $v\$ = OCT\$(n)$

**Comentariu:** *n* este o expresie numerică cu valori între -32768 și 65535.

Dacă *n* este negativ, este utilizată forma complementului față de doi, ceea ce înseamnă că **OCT\$(-n)** este identic cu **OCT\$(65535-n)**.

**Exemplu:** Exemplul de mai jos transformă în octal numărul 24 zecimal:

? OCT\$(24)

30

## ON COM(n)

Instrucțiune

**Scop:** Indică un număr de linie pentru salt condiționat în momentul în care sosesc informații în memoria tampon de comunicații.

**Format:** *ON COM(n) GOSUB linie*

**Comentariu:** *n* este numărul adaptorului de comunicație (1 sau 2). *linie* este numărul liniei la care se execută saltul.

Instrucțiunea **COM(n) ON** trebuie executată pentru activarea adaptorului de comunicații *n*. După **COM(n) ON**, dacă în instrucțiunea **ON COM(n)** se specifică un număr de linie diferit de zero, atunci, de fiecare dată când programul începe o nouă instrucțiune, limbajul verifică dacă au sosit informații pe adaptorul de comunicații. În caz afirmativ se execută un salt (**GOSUB**) la linia respectivă.



**Exemplu:**

```
150 ON COM(1) GOSUB 500
```

```
160 COM(1) ON
```

```
.
```

```
.
```

```
.
```

```
500 'dupa sosirea informatiei
```

```
.
```

```
.
```

```
590 RETURN 300
```

## ON ERROR

**Instrucțiune**

**Scop:** Permite detectarea erorilor și specifică linia de început a subrutinei de manipulare a erorilor.

**Format:** *ON ERROR GOTO LINIE*

**Comentariu:** *linie* este numărul primei linii din rutina de manipulare a erorii.

Pentru a înceta detecția erorilor, se execută instrucțiunea **ON ERROR GOTO 0**.

**Exemplu:** Exemplul de mai jos testează dacă fereastra cititorului de disc curent este deschisă:

```
10 ON ERROR GOTO 100
```

```
20 OPEN "DATA" FOR INPUT AS #1
```

```
30 END
```

```
.
```

```
.
```

```
.
```

```
100 IF ERR = 71 THEN LOCATE 23,1
```

```
PRINT "DISK NOT READY"
```

```
110 RESUME NEXT
```

## ON-GOSUB si ON-GOTO

**Instrucțiuni**

**Scop:** Execută salt la una sau mai multe linii de program în funcție de valoarea unei expresii.

**Format:** *ON n GOTO linie[,linie]...*

*ON n GOSUB linie[,linie]...*

**Comentariu:** *n* este o expresie numerică, rotunjită la un întreg, dacă este necesar. Este cuprinsă între 0 și 255, în caz contrar semnalându-se

eroare.

*linie* este numărul liniei la care se execută saltul.

Valoarea lui  $n$  determină la care linie de program se va executa saltul. Dacă, de exemplu, valoarea lui  $n$  este 3, înseamnă că al 3-lea număr de linie de program din listă este cel la care se va face saltul.

Dacă valoarea lui  $n$  este 0 sau mai mare decât numărul de linii din listă (dar mai mic sau egal cu 255), se continuă execuția programului cu următoarea instrucțiune din program.

**Exemplu:**

```
100 REM meniu
110 PRINT "1. Rutina 1"
120 PRINT "2. Rutina 2"
130 PRINT "3. Rutina 3"
140 PRINT "4. Rutina 4"
150 INPUT "Alegeti optiunea";C
160 ON C GOSUB 200,300,400,500
170 GOTO 100 'dupa executia rutinelor
    din nou la meniu
200 REM incepe prima subrutina
.
.
.
290 RETURN
300 REM incepe a doua subrutina
.
.
.
```

## ON KEY(n)

Instrucțiune

**Scop:** Atribuie un număr de linie când este detectată apăsarea pe o tastă programabilă sau de control a cursorului.

**Format:** *ON KEY(n) GOSUB linie*

**Comentariu:**  $n$  este o expresie numerică cuprinsă între 1 și 20 care indică tasta ce urmează a fi detectată după cum urmează:

- |      |                                |
|------|--------------------------------|
| 1-10 | tastele programabile F1 la F10 |
| 11   | cursor sus                     |
| 12   | cursor stînga                  |
| 13   | cursor dreapta                 |
| 14   | cursor jos                     |

15-20 taste definite sub forma:  
KEY n,CHR\$(KBflag) + CHR\$(scan code).  
linie este numărul liniei de program unde se execută saltul.

**Exemplu:**

```
10 KEY 15,CHR$(&H04) + CHR$(70) 'Detecteaza  
Ctrl-Break  
20 KEY 16,CHR$(&H04 + &H08) + CHR$(83)  
'Detecteaza Ctrl-Alt-Del  
30 ON KEY(15) GOSUB 1000  
40 ON KEY(16) GOSUB 2000  
50 KEY(15) ON: KEY(16) ON  
. . .  
1000 PRINT "Detectare Ctrl-Break"  
1010 RETURN  
2000 T = T + 1  
2010 ON T GOTO 2100,2200,2300,2400,2500  
2100 PRINT "PAS 1 REINITIALIZARE SISTEM":  
RETURN  
2200 PRINT "PAS 2 REINITIALIZARE SISTEM":  
RETURN  
2300 PRINT "PAS 3 REINITIALIZARE SISTEM":  
RETURN  
2400 PRINT "PAS 4 REINITIALIZARE SISTEM":  
RETURN  
2500 KEY(16) OFF 'Dezafecteaza detectia  
reinitializarii sistemului
```

## ON PEN

Instrucțiune

**Scop:** Atribuie un număr de linie de program pentru a transfera controlul în momentul activării creionului luminos (**light pen**).

**Format:** ON PEN GOSUB linie

**Comentariu:** linie este numărul liniei de program la care se dă controlul în momentul în care se detectează activarea creionului luminos. Dacă linie este egal cu 0 se încetează detecția.

Pentru activarea acestei instrucțiuni trebuie folosită instrucțiunea PEN ON. După executarea instrucțiunii PEN OFF nu se mai verifică



prezența creionului luminos.

**Exemplu:**

**10 ON PEN GOSUB 500**

**20 PEN ON**

.

.

.

**500 'Subrutina pentru creion luminos**

.

.

.

**599 RETURN**

### ON PLAY(n)

Instrucțiune

**Scop:** Generează continuu muzică în modul fundal (**background**) în timpul execuției programului.

**Format:** *ON PLAY(n) GOSUB linie*

**Comentariu:** *n* este o expresie întreagă cuprinsă între 1 și 32 indicând notele care trebuie detectate.

*linie* este numărul liniei subrutinei de detecție pentru instrucțiunea **PLAY**. Valoarea 0 oprește detectarea.

Pentru activarea acestei instrucțiuni trebuie folosită instrucțiunea **PLAY ON**.

**Exemplu:**

**10 ON PLAY(5) GOSUB 500**

**20 PLAY ON**

.

.

.

**500 'Subrutina pentru muzica in mod fundal**

.

.

.

**599 RETURN**

### ON STRIG(n)

Instrucțiune

**Scop:** Atribuire un număr de linie de program pentru detectarea apăsării butoanelor de la "joystick".

**Format:** *ON STRIG(n) GOSUB linie*

**Comentariu:** *n* poate lua valorile 0,2,4 sau 6 și indică butonul apăsat, după cum urmează:

0 butonul A1

2 butonul B1

4 butonul A2

6 butonul B2

*linie* este numărul liniei la care se execută saltul. Valoarea 0 încetează detectarea apăsării butoanelor de la joystick.

Pentru activarea acestei instrucțiuni trebuie folosită instrucțiunea STRIG(n) ON. După executarea instrucțiunii STRIG(n) OFF nu se mai verifică apăsarea butonului *n* de la joystick.

**Exemplu:**

```
10 ON STRIG(0) GOSUB 500
```

```
20 STRIG(0) ON
```

```
.
```

```
.
```

```
.
```

```
500 'Subrutina pentru primul buton de la joystick
```

```
.
```

```
.
```

```
.
```

```
599 RETURN
```

## ON TIMER

**Instrucțiune**

**Scop:** Transferă controlul unei linii de program după scurgerea unei perioade de timp definite.

**Format:** *ON TIMER(n) GOSUB linie*

**Comentariu:** *n* este o expresie numerică cuprinsă între 1 și 86.400 (1 secundă la 24 ore).

*linie* este numărul liniei la care se execută saltul după scurgerea intervalului de timp precizat.

Pentru activarea acestei instrucțiuni trebuie folosită instrucțiunea TIMER ON. Încetarea detecției se face cu ajutorul instrucțiunii TIMER OFF.

**Exemplu:**

```
10 CLS
```

```
20 ON TIMER(60) GOSUB 10000
```

```
30 TIMER ON
```

10000 RIND = CSRLIN 'salveaza rindul curent  
 10010 COL = POS(0) 'salveaza coloana curenta  
 10020 LOCATE 1,1: PRINT TIMES\$  
 10030 LOCATE RIND,COL 'din nou acasa  
 10040 RETURN

## OPEN

Instrucțiune

**Scop:** Permite intrarea sau ieșirea către un fișier sau periferic.

**Format:** *OPEN numefis [FOR mod1] AS [#]nrfis [LEN = recl]*  
 sau: *OPEN mod2,[#]nrfis,numefis[,recl]*

**Comentariu:** *mod1* de forma:

**OUTPUT** specifică modul secvențial de ieșire.

**INPUT** specifică modul secvențial de intrare.

**APPEND** specifică modul secvențial de ieșire, în care fișierul este poziționat la sfârșitul șirului de date în momentul deschiderii acestuia.

*mod2* este o expresie șir, primul caracter fiind unul din următoarele:

**O** specifică modul secvențial de intrare.

**I** specifică modul secvențial de ieșire.

**R** specifică modul aleator de intrare/ieșire.

Pentru ambele formate:

*nrfis* este o expresie întreagă care indică numărul fișierului deschis.

*numefis* este o expresie șir care indică numele fișierului.

*recl* este o expresie întreagă care, dacă este inclusă, indică lungimea înregistrării pentru fișierele aleatoare. Poate lua valori între 1 și 32767. Dacă nu se specifică lungimea, se ia 128 octeți.

**Exemplu:**

10 OPEN "DATA" FOR OUTPUT AS #1

10 OPEN "B:FISIER" AS 1 LEN = 256

## OPEN "COM..."

Instrucțiune

**Scop:** Deschide un fișier de comunicații. Este valid doar în prezența Adaptorului de comunicații asincrone.

**Format:** *OPEN "COMn:[viteză],[paritate],[data],[stop],[RS] [,CS[n]][,DS[n]][,CD[n]][LF][,PE]" AS #nrfis LEN = număr*



**Comentariu:** *n* este 1 sau 2, indicînd numărul Adaptörului de comunicații asincron.

*viteză* este rata de transmisie/recepție în biți pe secundă (bps). Vitezele acceptate sînt 75, 110, 150, 300, 600, 1200, 2400, 4800 și 9600. Dacă nu este specificată, se consideră 300 bps.

*paritate* este o constantă de lungime un caracter și indică tipul parității la transmisie sau recepție, după cum urmează:

**S SPACE:** Bit de paritate transmis și recepționat ca un spațiu (bit 0).

**O ODD:** Paritate de tip impară de transmitere sau/și recepție.

**E EVEN:** Paritate de tip pară de transmisie sau/și recepție.

**M MARK:** Biții de paritate transmiși și recepționați marcați (1 bit).

**N NONE:** Nu se transmite și nu se verifică paritatea la recepție.

Dacă nu se specifică, se consideră EVEN (E).

*data* este o constantă întregă ce indică numărul de biți de date transmiși/recepționați.

Valorile permise sînt: 5, 6, 7 sau 8. Dacă nu este specificată, se consideră 7.

*stop* este o constantă întregă ce indică numărul de biți de stop. Valorile permise sînt 1 sau 2.

*nrfis* este o expresie întregă ce indică numărul fișierului de comunicație deschis.

*număr* este numărul maxim de octeți ce poate fi citit din memoria tampon de comunicație. Dacă nu se specifică se consideră 128 octeți.

Instrucțiunea **OPEN "COM..."** alocă o memorie tampon pentru intrări/ieșiri la fel ca **OPEN** pentru fișierele de disc.

Opțiunile **RS, CS, DS, CD, LF** și **PE** afectează semnalele de pe linia de comunicație după cum urmează:

**RS** suprimă RTS (Request To Send)

**CS[n]** controlează CTS (Clear To Send)

**DS[n]** controlează DSR (Data Set Ready)

**CD[n]** controlează CD (Carrier Detect)

**LF** trimite un "line feed" după fiecare "carriage return"

**PE** activează verificarea parității

Argumentul *n* din opțiunile **CS, DS** și **CD** specifică numărul de milisecunde de așteptare a semnalului înainte de a indica eroare "**Device timeout**". *n* poate lua o valoare cuprinsă între 0 și 65535. Dacă *n* este omis sau este egal cu zero nu se mai execută verificarea.

**Exemplu:**

**10 OPEN "COM1:" AS #1**

sau

**10 OPEN "COM1:9600,N,8,,CS,DS,CD" AS #1**

## OPTION BASE

Instrucțiune

**Scop:** Declară valoarea minimă a elementelor unei matrici.

**Format:** *OPTION BASE n*

**Comentariu:** *n* este 1 sau 0.

Dacă nu se specifică, valoarea lui *n* este 0. După execuție, instrucțiunea:

**OPTION BASE 1** face ca valoarea celui mai mic element al unei matrici să aibă indicele 1. De exemplu elementele matricii *A(5)* vor fi:

*A(1)....A(5)*

## OUT

Instrucțiune

**Scop:** Trimite un octet la unul din porturile de ieșire al calculatorului.

**Format:** *OUT n,m*

**Comentariu:** *n* este o expresie numerică indicînd numărul portului. Este cuprinsă între 0 și 65535.

*m* este o expresie numerică indicînd valoarea ce urmează a fi trimisă.

Poate lua valori între 0 și 255.

Instrucțiunea **OUT** este complementara funcției **INP**.

**Exemplu:** Exemplul următor trimite valoarea 100 la portul de ieșire 32:

**100 OUT 32,100**

## PAINT

Instrucțiune

**Scop:** Umple o arie de ecran cu o culoare selectată. Are semnificație doar în modul grafic.

**Format:** *PAINT (x,y)[[,pic][,margine][,fundal]]*

**Comentariu:** *(x,y)* sînt coordonatele punctului din aria ce urmează a fi umplută. Coordonatele pot fi indicate în forma absolută sau relativă. Acest punct este utilizat ca punct de început.

*pic* poate fi o expresie numerică sau șir. Este utilizat pentru a umple aria respectivă cu o culoare sau un model. Dacă *pic* este un număr atunci umplerea se face cu unul din atributele de culoare permise. Dacă *pic* este o expresie șir atunci umplerea se face cu un model.

*margine* este o expresie numerică întreagă. Definește atributul de culoare pentru marginile figurii pictate.

*fundal* este o expresie șir de 1 octet utilizat în pictarea modelelor.



În rezoluție medie se poate umple interiorul sau exteriorul unei arii de ecran definite cu una dintre cele patru culori din paleta curentă definită de instrucțiunea COLOR. Un exemplu este umplerea cu roșu a unui cerc verde sau înconjurarea unui cerc roșu cu verde.

Pentru pictarea modelelor, atributul *pic* trebuie să fie o expresie șir de forma:

**CHR\$(&Hnn) + CHR\$(&Hnn) + CHR\$(&Hnn)...**

Secvența CHR\$ specifică o mască de lungime 1 octet. În momentul afișării, masca specificată va lua locul culorii. Expresia șir poate conține pînă la 64 octeți:

	<b>x crește ----&gt;</b>	
	<b>7 6 5 4 3 2 1 0</b>	
0,0	x x x x x x x x	Model 0
0,1	x x x x x x x x	Model 1
0,2	x x x x x x x x	Model 2
.		
.		
0,63	x x x x x x x x	Model 63 (maxim permis)

Modelul ales este apoi repetat uniform peste aria definită de *margină*.

Următorul tabel indică valorile binare și hexazecimale asociate cu fiecare atribut în medie rezoluție:

<b>Paleta de culoare</b>	<b>Atributul în binar</b>	<b>Modelul pentru desenarea unei linii în binar</b>	<b>Modelul pentru desenarea unei linii în hex.</b>
<b>0</b>			
verde	01	01010101	&H55
roșu	10	10101010	&HAA
maro	11	11111111	&HFF

<b>Paleta de culoare</b>	<b>Atributul în binar</b>	<b>Modelul pentru desenarea unei linii în binar</b>	<b>Modelul pentru desenarea unei linii în hex.</b>
<b>1</b>			
turquoise	01	01010101	&H55
mov	10	10101010	&HAA
alb	11	11111111	&HFF

**Exemplu:**

**10 CLS: SCREEN 1,0: KEY OFF**

**20 MODEL\$ = CHR\$(&HAA) + CHR\$(&HAA) +**

**CHR\$(&HAA) + CHR\$(&H55) + CHR\$(&H55) +**



### CHR\$(&HFF)

30 COLOR 0,0 'se alege paleta 0

40 VIEW (1,1)-(150,100),0,2

50 GOSUB 1000

60 COLOR 0,1 'se alege paleta 1

70 GOTO 1020

1000 PAINT (125,50),MODEL\$,2

1010 RETURN

1020 GOTO 1020

### PEEK

Funcție

**Scop:** Evaluează un octet citit de la o locație de memorie indicată.

**Format:**  $v = \text{PEEK}(n)$

**Comentariu:**  $n$  este un întreg cu valori cuprinse între 0 și 65535.  $n$  este offset-ul din segmentul curent de memorie definit de instrucțiunea DEF SEG.

Valoarea evaluată este un întreg cuprins între 0 și 255.

Funcția PEEK este complementară instrucțiunii POKE.

**Exemplu:** Următorul exemplu de program testează care adaptor de terminal este atașat la sistem:

10 'testeaza tipul adaptorului

20 DEF SEG = 0

30 IF (PEEK(&H410) AND &H30) = &H30

THEN IBMMONO = 1

ELSE IBMMONO = 0

### PEN

Instrucțiune și Funcție

**Scop:** Citește creionul luminos.

**Format:**

Ca instrucțiune: *PEN ON*

*PEN OFF*

*PEN STOP*

ca funcție:  $v = \text{PEN}(n)$

**Comentariu:** Funcția  $v = \text{PEN}(n)$  citește coordonatele creionului lu-

minos.

*n* este o expresie numerică cuprinsă între 0 și 9 și afectează valoarea evaluată de funcție după cum urmează:

**0** Indică dacă creionul era activ în timpul ultimei validări. Atribuie variabilei *v* valoarea -1 dacă creionul era activ și 0 în cealaltă situație.

**1** Evaluează coordonata *x* când creionul a fost ultima oară activ. Valoarea va fi cuprinsă între 0 și 319 în medie rezoluție și 0 la 639 în rezoluție înaltă.

**2** Evaluează coordonata *y* când creionul a fost ultima oară activ. Valoarea va fi cuprinsă între 0 și 199.

**3** Evaluează valoarea curentă a creionului; activ -1, iar dezactivat 0.

**4** Evaluează ultima coordonată *x* validă cunoscută. Valoarea evaluată este cuprinsă între 0 și 319 în rezoluție medie și respectiv 639 în rezoluție înaltă.

**5** Evaluează ultima coordonată *y* validă cunoscută. Valoarea evaluată este cuprinsă între 0 și 199.

**6** Evaluează poziția rîndului de caractere unde creionul a fost ultima oară activat. Ia valori între 1 și 24.

**7** Evaluează poziția coloanei de caractere unde creionul a fost ultima oară activat. Cuprins între 1 și 40 sau 80, funcție de WIDTH.

**8** Atribuie ultimul rînd de caractere valid. Ia valori între 1 și 24.

**9** Atribuie poziția ultimei coloane de caractere valide. Ia valori între 1 și 40 sau 80, funcție de WIDTH.

Instrucțiunea **PEN ON** permite activarea funcției de citire **PEN**. După utilizarea creionului luminos se va executa instrucțiunea **PEN OFF**, ceea ce va crește considerabil viteza de execuție, în continuare, a programului.

**Exemplu:** Acest exemplu afișează valoarea creionului luminos de la ultima validare, precum și valoarea curentă:

```
10 PEN ON
20 FOR I = 1 TO 500
30 X = PEN(0): X1 = PEN(3)
40 PRINT X,X1
50 NEXT
60 PEN OFF
```

## PLAY

Instrucțiune

**Scop:** Generează muzică specificată de un șir.

**Format:** *PLAY* șir

**Comentariu:** PLAY implementează un concept similar cu DRAW prin introducerea unui "limbaj de definiții armonice".

*șir* este o expresie șir conștind dintr-o succesiune de comenzi muzicale.

Comenzile în instrucțiunea PLAY sînt:

**A la G** (opțional cu #, + sau -)

Cînta nota indicată în octava curentă. Semnul număr (#) sau semnul plus (+) indică un diez; semnul minus (-) indică un bemol. Semnele #, + și - sînt permise doar dacă au corespondent în notația muzicală (de ex. B# este o notă invalidă).

**On** Octava. Atribuie octava curentă pentru notele ce urmează. Sînt 7 octave numerotate de la 0 la 6. Dacă nu se specifică, se atribuie octava 4.

> **n** Se deplasează o octavă în sus și cînta nota *n*.

< **n** Se deplasează în jos o octavă și cînta nota *n*.

**Nn** Cînta nota *n* care poate varia între 0 și 84. În 7 octave posibile sînt 84 note, 0 însemnînd "pauză".

**Ln** Atribuie lungimea notei care urmează. Lungimea notei este  $1/n$ , unde *n* poate lua valori între 1 și 64.

**Pn** Pauză. Lungimea pauzei este  $1/n$ , unde *n* poate lua valori între 1 și 64.

. Punct. cînd este plasat după o nota face ca acea notă să fie cîntată ca notă accentuată.

**Tn** Tempo. Atribuie numărul de sferturi de notă pe minut. *n* poate varia între 32 și 255. În lipsă se consideră 120.

**MF** Muzică solist. Muzica (creată prin SOUND sau PLAY) rulează în modul solist.

**MB** Muzică fundal. Muzica (creată prin SOUND sau PLAY) rulează în modul fundal. Fiecare sunet sau notă este plasată într-un tampon de memorie, permițînd programului să ruleze în continuare în timp ce muzica este interpretată.

**MN** Muzică în modul normal. Fiecare notă este interpretată 7/8 din timpul specificat de L.

**ML** Muzică legato. Fiecare notă este interpretată la lungimea specificată de L.

**MS** Muzică stacato. Fiecare notă este interpretată 3/4 din lungimea specificată de L.

**Xvar**; Execută șirul specificat de *var*.

În toate aceste comenzi argumentul *n* poate fi o constantă (de ex. 6) sau = **var**; unde *var* este numele unei variabile. După numele variabilei



este necesar să apară punct și virgulă (;). Spațiile sînt ignorate.

**Exemplu:**

```
10 'octave
20 SC$ = "CDEFGAB"
30 PLAY "00 XSC$;"
40 FOR I = 1 TO 6
50 PLAY "> XSC$;"
60 NEXT
70 PLAY "06 XSC$;"
80 FOR I = 1 TO 6
90 PLAY "< XSC$;"
100 NEXT
```

## PLAY(n)

**Funcție**

**Scop:** Evaluează numărul de note curent din tamponul de memorie alocat pentru muzica de fundal.

**Format:**  $v = \text{PLAY}(n)$

**Comentariu:**  $n$  este un argument fals ce poate lua orice valoare.

**PLAY(n)** are ca rezultat valoarea 0 dacă programul rulează în modul solist. Valoarea maximă pe care o poate evalua este 32, care este numărul de note maxim ce poate fi alocat în memoria tampon.

**Exemplu:**

```
10 PLAY "MB CDEFGAB"
20 IF PLAY(1) = 5 GOTO 1000
30 GOTO 2000
.
.
.
1000 PLAY "MB 04 T200 L4 MS GG#GE"
2000 END
```

## PMAP

**Funcție**

**Scop:** Transformă coordonatele fizice în coordonate universale sau invers. Are semnificație doar în modul grafic.

**Format:**  $v = \text{PMAP}(x, n)$

**Comentariu:**  $x$  coordonata punctului ce urmează a fi transformat.

$n$  tipul transformării, după cum urmează:

1 transformă coordonata universală  $x$  în coordonată fizică.

2 transformă coordonata universală  $y$  în coordonată fizică.

3 transformă coordonata fizică  $x$  în coordonată universală.

4 transformă coordonata fizică  $y$  în coordonată universală.

Funcția **PMAP** este utilizată pentru translatarea coordonatelor între sistemul universal definit în instrucțiunea **WINDOW** și sistemul de coordonate fizice.

## POINT

Funcție

**Scop:** Prima formă evaluează atributul unui punct de pe ecran. Cea de-a doua formă evaluează coordonatele grafice curente  $x$  și  $y$ . Are semnificație numai în modul grafic.

**Format:**  $v = POINT(x,y)$   
 $v = POINT(n)$

**Comentariu:**  $(x,y)$  sînt coordonatele punctului dorit. Ele trebuie date în forma absolută. Dacă punctul dat se află în afara domeniului, este evaluată valoarea -1. Valorile evaluate sînt 0, 1, 2 și 3 în medie rezoluție și 0 sau 1 în rezoluție înaltă.

$n$  evaluează coordonatele grafice curente  $x$  și  $y$ .

0 evaluează coordonata fizică  $x$ .

1 evaluează coordonata fizică  $y$ .

2 evaluează coordonata universală  $x$  dacă instrucțiunea **WINDOW** este activă. În caz contrar evaluează coordonata fizică  $x$ .

3 evaluează coordonata universală  $y$  dacă instrucțiunea **WINDOW** este activă. În caz contrar evaluează coordonata fizică  $y$ .

**Exemplu:** Acest exemplu ilustrează valoarea evaluată de către funcția **POINT**:

```
10 CLS: SCREEN 1,0: KEY OFF
20 PRINT "POINT(n) cu WINDOW inactiv"
30 GOSUB 110
40 WINDOW (0,0)-(319,199)
50 PRINT "POINT(n) cu WINDOW activ"
60 GOSUB 110
70 PRINT "POINT(n) cu WINDOW si SCREEN
active"
80 WINDOW SCREEN (0,0)-(319,199)
90 GOSUB 110
100 END
110 PSET (5,15)
```

```

120 FOR I = 1 TO 3
130 PRINT POINT(I);
140 NEXT
150 PRINT: PRINT
160 RETURN
RUN
POINT(n) cu WINDOW inactiv
5 15 5 15
POINT(n) cu WINDOW activ
5 184 5 15
POINT(n) cu WINDOW si SCREEN active
5 15 5 15

```

## POKE

Instrucțiune

**Scop:** Scrie un octet într-o locație de memorie.

**Format:** *POKE n,m*

**Comentariu:** *n* trebuie să fie cuprins între 0 și 65535. Indică offset-ul în segmentul curent unde datele urmează a fi scrise. Segmentul curent este definit de instrucțiunea DEF SEG.

*m* este valoarea ce urmează a fi scrisă în locația de memorie. Poate lua valori între 0 și 255.

**ATENȚIE !**

*Limbajul BASIC nu verifică offset-ul specificat. Va rămâne în grija programatorului să nu scrie date în zona stivei, a variabilelor sau a programului BASIC.*

## POS

Funcție

**Scop:** Evaluează poziția din rînd (numărul coloanei) a cursorului.

**Format:**  $v = POS(n)$

**Comentariu:** *n* este un argument fals.

**Exemplu:** Exemplul următor mută cursorul pe linia următoare dacă acesta depășește coloana 60:

```
IF POS(0) > 60 THEN PRINT CHR$(13)
```

## PRINT

Instrucțiune

**Scop:** Afișează date pe ecranul calculatorului.

**Format:** *PRINT [lista de expresii][;]*



sau ? [lista de expresii][;]

**Comentariu:** lista de expresii este o listă de expresii șir sau/și numerice separate prin virgule, spații sau punct și virgulă. Orice constante șir din listă trebuie închise între ghilimele.

Dacă lista este omisă se va afișa un rînd gol.

### Pozițiile de afișare

Poziția fiecărui termen afișat este determinată de punctuația folosită pentru separarea termenelor din listă. Limbajul divide linia în zone de afișare de cîte 14 coloane fiecare. În lista de expresii:

- tipărind o virgulă între expresii determină că următoarea valoare să fie afișată la începutul următoarei zone libere.
- tipărind un punct și virgulă determină ca următoarea valoare să fie afișată imediat după ultima valoare.
- tipărind unul sau mai multe spații între expresii are același efect ca punct și virgulă.

Dacă lista se termină cu o virgulă, un punct și virgulă sau funcțiile TAB sau SPC, următoarea instrucțiune PRINT va începe afișarea pe aceeași linie.

### Exemple:

```
10 X = 5
```

```
20 PRINT X + 5, X - 5, X * (-5)
```

```
RUN
```

```
10          0          -25
```

```
10 INPUT X
```

```
20 PRINT X; " la patrat este"; X ^ 2; " si";
```

```
30 PRINT X; " la cub este"; X ^ 3
```

```
RUN
```

```
?9
```

```
9 la patrat este 81 si 9 la cub este 729
```

## PRINT USING

Instrucțiune

**Scop:** Afișează pe ecran șiruri sau numere utilizînd un format specificat.

**Format:** PRINT USING v\$; lista de expresii[;]

**Comentariu:** v\$ este o constantă șir sau variabilă constînd din caractere speciale de formatare. Aceste caractere de formatare determină cîmpul și formatul de afișare a șirurilor sau numerelor.

lista de expresii constă din expresii numerice sau șiruri ce urmează a fi afișate, separate prin virgulă sau punct și virgulă.

### Cîmpuri șir

Cînd instrucțiunea PRINT USING este folosită pentru afișarea șirurilor, poate fi folosit unul dintre următoarele formate:

! Specifică faptul că va fi afișat doar primul caracter din șirul dat.

\n spatii\ Specifică faptul că 2 + n caractere din șirul dat urmează a fi afișate.

#### Exemplu:

```
10 A$ = "ABC": D$ = "DEF"  
20 PRINT USING "!A$;D$;  
30 PRINT USING "\ \";A$;D$  
RUN  
AD  
ABCDEF
```

& Specifică un cîmp șir de lungime variabilă. În acest caz șirul va fi afișat exact cum este introdus.

#### Exemplu:

```
10 A$ = "ABC": D$ = "DEF"  
20 PRINT USING "!";A$;  
30 PRINT USING "&";B$  
RUN  
ADEF
```

### Cîmpuri numerice

Cînd instrucțiunea PRINT USING este folosită pentru a afișa numere, se pot folosi următoarele caractere speciale:

# Semnul număr este folosit pentru reprezentarea poziției fiecărei cifre. Dacă numărul de reprezentat are mai puține cifre decît pozițiile specificate se execută automat alinierea la dreapta (precedat de spații) a numărului în cîmp. Punctul zecimal poate fi plasat în orice poziție din cîmp. Dacă este necesar, numerele sînt rotunjite.

#### Exemplu:

```
PRINT USING "###.## ";10.2;5.3;66.789;.234  
10.20 5.30 66.79 0.23
```

+ Semnul plus la începutul sau sfîrșitul formatului implică reprezentarea semnului numărului (plus sau minus) la începutul sau, respectiv, la sfîrșitul său.

#### Exemplu:

```
PRINT USING "+###.## ";-68.95;15.33  
-68.95 +15.33  
PRINT USING "###.## + ";-68.95;15.33  
68.95- 15.33 +
```

- Semnul minus la sfârșitul formatului implică reprezentarea numerelor negative cu semnul minus după numărul respectiv.

**Exemplu:**

```
PRINT USING "###.##-";-68.95,22.449
68.95- 22.45
```

\*\* Un asterisc dublu la începutul formatului face ca spațiile din fața câmpului numeric să fie umplute cu asteriscuri. De asemenea semnul \*\* specifică pozițiile pentru încă două cifre.

**Exemplu:**

```
PRINT USING "***.# ";12.39;-0.9;765.4
*12.4 *-0.9 765.4
```

\$\$ Semnul dolar dublu face ca semnul dolarului să fie afișat imediat în stînga numărului formatat. Semnul \$\$ specifică pozițiile a încă două cifre, dintre care una este a semnelui dolar. Formatul exponențial nu poate fi folosit cu semnul \$\$.

**Exemplu:**

```
PRINT USING "$$###.## ";456.78;0.9;-765.1
$456.78 $0.90 -$765.10
```

\*\*\$ Acest semn la începutul formatului combină efectele celor două simboluri de mai sus.

**Exemplu:**

```
PRINT USING "***$###.##";2.34
**$2.34
```

, O virgulă plasată la stînga punctului zecimal afișează o virgulă la fiecare grup de trei cifre semnificative din stînga punctului zecimal. Virgula plasată la sfârșitul formatului este tipărită ca parte a șirului.

**Exemple:**

```
PRINT USING "#####.##";1234.5
1,234.50
```

```
PRINT USING "#####.##";1234.5
1234.50,
```

^^^ ^ Acest semn plasat după format specifică reprezentarea în forma exponențială.

**Exemple:**

```
PRINT USING "##### ^ ^ ^ ^";234.56
2.35E02
```

```
PRINT USING ".## ^ ^ ^ ^ -";-88888
.889E05-
```



```
PRINT USING "+.## ^ ^ ^ ^";123
+.12E03
```

\_ Semnul sublinierii face ca următorul caracter să fie afișat drept caracter literal.

**Exemplu:**

```
PRINT USING "_!##.##_!";12.34
!12.34!
```

Dacă numărul de afișat este mai mare decât câmpul numeric specificat se va afișa automat semnul procentului în fața numărului respectiv.

**Exemple:**

```
PRINT USING "##.##";111.22
%111.22
```

```
PRINT USING ".##";0.999
%1.00
```

## PRINT # și PRINT # USING

Instrucțiuni

**Scop:** Scriu date secvențiale într-un fișier.

**Format:** *PRINT #nrfis,[USING x\$;] lista de expresii[;]*

**Comentariu:** *nrfis* este numărul fișierului deschis pentru ieșire.

*x\$* este o expresie șir ce conține caractere de formatare descrise la instrucțiunea PRINT USING.

*lista de expresii* este o listă de expresii șir sau/și numerice ce urmează a fi scrise în fișier.

**Exemplu:**

```
10 A = 123
20 B = 6789
30 C = 22.33
40 OPEN "DATA" FOR OUTPUT AS #1
50 PRINT #1,USING "$$###.##";A;B;C
60 CLOSE
70 OPEN "DATA" FOR INPUT AS #1
80 INPUT #1,A$,B$,C$
90 CLOSE
100 PRINT A$,B$,C$
```

## PSET și PRESET

Instrucțiuni

**Scop:** Desenează un punct la poziția specificată de pe ecran. Are înțeles doar în modul grafic.

**Format:** *PSET (x,y)[,culoare]*  
*PRESET (x,y)[,culoare]*

**Comentariu:**  $(x,y)$  sînt coordonatele punctului de afișat. Aceste coordonate pot fi date atît în forma absolută cît și în forma relativă.

*culoare* este o expresie întregă ce alege un atribut din domeniul de atribute de culoare a modului ecran curent.

**PRESET** este aproape identic cu **PSET**. Singura diferență constă în faptul că dacă nu este specificat parametrul *culoare* în instrucțiunea **PRESET**, este selectat automat pentru fundal atributul (0). Dacă este inclus parametrul *culoare* cele două instrucțiuni sînt identice.

**Exemplu:**

```
10 CLS: SCREEN 1: KEY OFF
20 FOR I = 0 TO 100
30 PSET (I,I)
40 NEXT
50 'sterge linia
60 FOR I = 100 TO 0 STEP -1
70 PRESET (I,I)
80 NEXT
```

## PUT

Instrucțiune (Fișiere)

**Scop:** Scrie o înregistrare dintr-o zonă de memorie aleatoare într-un fișier aleator.

**Format:** *PUT [#]nrfs[,număr]*

**Comentariu:** *nrfs* este numărul sub care a fost deschis fișierul.

*număr* este numărul înregistrării ce urmează a fi scrisă, de la 1 octet la 16 Mocteți.

Dacă parametrul *număr* este omis, înregistrarea va avea următorul număr disponibil de înregistrare.

Instrucțiunea **PUT** poate fi utilizată pentru un fișier de comunicații. În acest caz parametrul *număr* este numărul de octeți ce urmează a fi scris în fișierul de comunicații. Acest număr trebuie să fie mai mic sau cel mult egal cu valoarea stabilită de opțiunea **LEN** din instrucțiunea **OPEN "COM...**

## PUT

Instrucțiune (Grafică)

**Scop:** Plotează imagini pe o arie specificată a ecranului. Are semnificație doar în modul grafic.

**Format:** *PUT (x,y),vector[,acțiune]*

**Comentariu:**  $(x,y)$  sînt coordonatele colțului din stînga sus a imaginii de transferat.

*vector* este numele unei matrici (vector) numerice conținînd informația de transferat. Pentru mai multe informații în legatură cu acest vector vezi instrucțiunea GET (pentru grafică).

*acțiune* poate fi de tipul PSET, PRESET, XOR, OR sau AND. Dacă nu se specifică se consideră XOR.

Instrucțiunea PUT este opusă instrucțiunii GET în sensul că prima ia date din vector și le plotează pe ecran.

PSET ia datele din matrice și le transpune pe ecran.

PRESET este la fel cu PSET cu excepția că este produsă imaginea complementară. De exemplu, în medie rezoluție, care are un atribut maxim 3, atributul 0 al imaginii va determina plotarea lui cu atributul 3 și viceversa, iar un atribut 1 va produce o imagine cu atributul 2 și, evident, viceversa.

AND, OR și XOR specifică operații logice pe biții fiecărei imagini.

AND este utilizat în cazul în care există deja o imagine în aria de ecran pe care urmează a se face transferul.

OR este folosit pentru a suprapune imaginea de transferat peste o imagine existentă.

XOR este un mod special care poate fi folosit pentru animație. Aceasta permite mișcarea unei imagini fără alterarea fundalului.

În rezoluție medie AND, OR și XOR au următoarele efecte asupra culorii:

#### AND

	val. vector
ecran	0 1 2 3
0	0 0 0 0
1	0 1 0 1
2	0 0 2 2
3	0 1 2 3

#### OR

	val. vector
ecran	0 1 2 3
0	0 1 2 3
1	1 1 3 3
2	2 3 2 3
3	3 3 3 3



## XOR

	val. vector
ecran	0 1 2 3
0	0 1 2 3
1	1 0 3 2
2	2 3 0 1
3	3 2 1 0

Animarea unui obiect poate fi realizată după cum urmează:

1. Se pune obiectul pe ecran cu instrucțiunea PUT (cu XOR);
2. Se recalculează noua poziție a obiectului;
3. Se pune din nou obiectul pe ecran la vechea locație cu instrucțiunea PUT (cu XOR);
4. Se repetă pasul 1, de data aceasta punând obiectul la noua locație.

Mișcările realizate în acest fel lasă fundalul neschimbat. Evențualele licăriri se pot reduce prin reducerea timpului între pasul 1 și pasul 4 și asigurând un timp de întârziere suficient între pasul 1 și 3.

**Exemplu:** Acest exemplu arată cum poate fi mutat un cerc de-a lungul ecranului cu acțiunea XOR:

```
10 CLS: DEFINT A-Z: SCREEN 1: KEY OFF
20 DIM A(404)
30 CIRCLE (160,100),20,3
40 PAINT (160,100),2,3
50 GET (140,80)-(180,120),A: CLS
60 X = 30: Y = 50
70 FOR I = 1 TO 20
80 PUT (X,Y),A,XOR
90 PUT (X,Y),A,XOR
100 X = X + 10
110 NEXT
```

## RANDOMIZE

Instrucțiune

**Scop:** Inițializează generatorul de numere aleatoare.

**Format:** *RANDOMIZE [n]*  
*RANDOMIZE TIMER*

**Comentariu:** *n* este o expresie întreagă, simplă sau dublă precizie ce este utilizată ca "sămînța" de număr aleator.

Dacă *n* este omis, limbajul BASIC suspendă execuția și cere o valoare cu mesajul:

**Random Number Seed (-32768 to 32767)?**

Dacă generatorul de numere aleatoare nu este inițializat cu o valoare, funcția RND va returna aceeași secvență de numere aleatoare de fiecare dată de câte ori se începe rularea programului. Pentru a schimba secvența de numere aleatoare de câte ori se începe rularea este necesară introducerea unei instrucțiuni **RANDOMIZE** la începutul programului și schimbarea valorii inițiale ("sămînța") la fiecare rulare.

Ceasul intern poate fi folositor pentru a da "sămînța" numărului aleator. Pentru a nu se mai afișa mesajul de mai sus este suficientă utilizarea funcției **TIMER**.

**Exemple:**

```
10 RANDOMIZE
20 FOR I = 1 TO 4
30 PRINT RND;
40 NEXT I
RUN
Random Number Seed (-32768 to 32767)?
```

Să presupunem că se răspunde cu 3. Programul continuă:

```
Random Number Seed (-32768 to 32767)? 3
.7655695 .3558607 .3742327 .1388798
RUN
Random Number Seed (-32768 to 32767)?
```

De data aceasta să presupunem că se răspunde cu 4. Programul continuă:

```
Random Number Seed (-32768 to 32767)? 4
.1719568 .5273236 .6879686 .713297
RUN
Random Number Seed (-32768 to 32767)?
```

Dacă se răspunde, din nou, cu 3, se va obține aceeași secvență ca în cazul primei rulări:

```
Random Number Seed (-32768 to 32767)? 3
.7655695 .3558607 .3742327 .1388798
```

Următorul exemplu utilizează funcția **TIMER**. De notat că de fiecare dată cînd se rulează programul se obțin secvențe diferite de numere:

```
10 RANDOMIZE TIMER
20 FOR I = 1 TO 4
30 PRINT RND;
40 NEXT I
RUN
.9590051 .1036786 .1464037 .7754918
```

**READ**

Instrucțiune

**Scop:** Citește valori dintr-o instrucțiune DATA și le atribuie unor variabile. Vezi instrucțiunea DATA.

**Format:** *READ var[,var]...*

**Comentariu:** *var* este o variabilă de tip numeric sau șir sau element al unei matrici care va recepționa valorile citite din tabloul DATA.

Instrucțiunea **READ** trebuie folosită împreună cu instrucțiunea DATA.

O instrucțiune **READ** poate accesa mai multe instrucțiuni DATA, precum și mai multe instrucțiuni **READ** pot accesa o instrucțiune DATA. Dacă numărul variabilelor din lista de variabile depășește numărul elementelor din instrucțiunile DATA se va semnala eroare de tip "Out of data".

**Exemplu:**

```
10 FOR I = 1 TO 10
20 READ A(I)
30 NEXT I
40 DATA 3.08,4.23,5.74,9.02,3.15
50 DATA 6.19,8.34,9.29,3.68,5.92
```

**REM**

Instrucțiune

**Scop:** Inserează comentarii într-un program.

**Format:** *REM comentariu*

**Comentariu:** *comentariu* poate fi orice secvență de caractere.

Instrucțiunile **REM** nu sînt executate, ci doar afișate în locul în care au fost inserate.

Comentariile pot fi adăugate prin precedarea comentariului respectiv de o ghilimea ('). Dacă se dorește inserarea unui comentariu pe o linie care conține și alte instrucțiuni este necesar ca acel comentariu să fie plasat la sfîrșitul liniei de program.

**Exemplu:**

```
10 REM calculeaza viteza medie
20 SUM = 0 'initializeaza suma
30 FOR I = 1 TO 20
40 SUM = SUM + V(I)
```



Linia 20 mai pote fi scrisă:

**20 SUM = 0: REM initializeaza suma**

## RENUM

Comandă

**Scop:** Renumerotează liniile de program.

**Format:** *RENUM [nrnou][,][nrvechi][,increment]]*

**Comentariu:** *nrnou* este primul număr de linie de utilizat în noua secvență.

*nrvechi* este numărul liniei din programul curent de unde începe renumerotarea. Dacă nu se specifică se ia prima linie din program.

*increment* este pasul de numerotare în noua secvență. Dacă nu se specifică se consideră 10.

**RENUM** realizează schimbarea tuturor numerelor de linii ce urmează instrucțiunilor **ERL**, **ELSE**, **GOSUB**, **GOTO**, **ON...GOSUB**, **ON...GOTO**, **RESTORE**, **RESUME**, **RETURN** și **THEN**. Dacă într-una din aceste instrucțiuni apare un număr de linie inexistent, va fi semnalată eroare de tip "**Undefined line number xxxxx in yyyy**".

Instrucțiunea **RENUM** nu poate fi folosită pentru schimbarea ordinii liniilor de program, sau pentru a crea linii de program cu numere mai mari decât 65529.

**Exemple:**

**RENUM 300,,50**

Acest exemplu renumerotează, de asemenea, întregul program. Prima linie a noului program va fi 300, iar incrementul între linii 50.

**RENUM 1000,900,20**

Exemplul de mai sus renumerotează liniile începând cu linia 900, prima linie din noul program va fi 1000, iar incrementul între linii 20.

## RESET

Comandă

**Scop:** Închide toate fișierele de disc și golește memoria tampon de sistem.

**Format:** *RESET*

**Comentariu:** Dacă toate fișierele deschise se află pe disc, **RESET** este identic cu **CLOSE**.

## RESTORE

Instrucțiune

**Scop:** Permite recitirea instrucțiunilor **DATA** de la o linie specificată.

**Format:** *RESTORE [linie]*

**Comentariu:** *linie* este numărul de linie a unei instrucțiuni DATA din program.

**Exemplu:**

```
10 READ A,B,C
20 RESTORE
30 READ D,E,F
40 DATA 57,68,79
50 PRINT A;B;C;D;E;F
RUN
57 68 79 57 68 79
```

## RESUME

**Instrucțiune**

**Scop:** Continuă programul după execuția unei proceduri de recuperare a erorii.

**Format:** *RESUME [0]*  
*RESUME NEXT*  
*RESUME linie*

**Comentariu:**

**RESUME** sau **RESUME 0**

Execuția reîncepe la instrucțiunea care a cauzat eroarea.

**RESUME NEXT**

Execuția reîncepe la linia imediat următoare celei ce a cauzat eroarea.

**RESUME linie**

Execuția reîncepe la numărul de linie specificat.

**Exemplu:**

```
10 ON ERROR GOTO 1000
.
.
.
1000 IF (ERR = 230) AND (ERL = 90) THEN PRINT
      "Incerca din nou": RESUME 80
```

## RETURN

**Instrucțiune**

**Scop:** Termină execuția unei subrutine și face trimiterea în programul principal. Vezi GOSUB și RETURN.

**Format:** *RETURN [linie]*

**Comentariu:** *linie* este numărul liniei de program la care se dorește întoarcerea.

## RIGHT\$

Funcție

**Scop:** Atribue șirului  $v$  cele mai din dreapta  $n$  caractere din șirul  $x$ .

**Format:**  $v = RIGHT$(x, n)$

**Comentariu:**  $x$  este orice expresie șir.

$n$  este o expresie întregă ce specifică numărul de caractere ce urmează a fi citite.

**Exemplu:**

10 N\$ = "ION POPESCU"

20 PRINT RIGHT\$(N\$,7)

RUN

POPESCU

## RMDIR

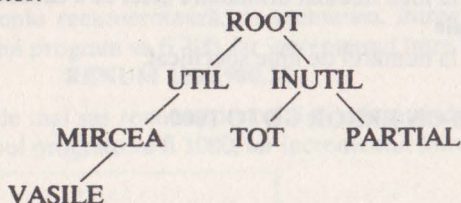
Comandă

**Scop:** Șterge un director de pe discul specificat.

**Format:** *RMDIR cale*

**Comentariu:** *cale* este o expresie șir ce identifică subdirectorul ce urmează a fi șters din directorul existent.

Înainte ca subdirectorul respectiv să poată fi șters trebuie ca acel subdirector să nu conțină nici un fișier. **Exemplu:** Să considerăm următorul arbore director:



Pentru a șterge subdirectorul "PARTIAL", se va executa:

**RMDIR "INUTIL\PARTIAL"**

## RND

Funcție

**Scop:** Atribue unei variabile un număr aleator cuprins între 0 și 1.

**Format:**  $v = RND[(x)]$

**Comentariu:**  $x$  este o expresie numerică ce afectează valoarea atribuită.



La fiecare rulare a programului este generată aceeași secvență de numere aleatoare. Pentru a preîntîmpina acest lucru trebuie reinițializat generatorul de numere aleatoare cu ajutorul instrucțiunii **RANDOMIZE**. Același lucru se poate face și cu ajutorul funcției **RND** cu condiția ca  $x$  să fie un număr negativ. Aceasta generează totdeauna o secvență particulară pentru un  $x$  dat. Secvența nu este afectată de către instrucțiunea **RANDOMIZE**; astfel, pentru a genera secvențe diferite la fiecare rulare a programului, este necesar să se utilizeze diferite valori pentru  $x$ . Dacă  $x$  este pozitiv sau dacă lipsește, va fi generat următorul număr aleator din secvența. **RND(0)** repetă ultimul număr generat. Pentru a avea numere aleatoare cuprinse între 0 și  $n$ , se va folosi formula:

$$\text{INT}(\text{RND} * (\text{N} + 1))$$

Exemplu:

```

10 FOR I = 1 TO 3
20 PRINT RND(I);      ' x > 0
30 NEXT I
40 PRINT: X = RND(-6) ' x < 0
50 FOR I = 1 TO 3
60 PRINT RND(I);      ' x > 0
70 NEXT I
80 RANDOMIZE 853
90 PRINT X = RND(-6)  ' x < 0
100 FOR I = 1 TO 3
110 PRINT RND;        ' la fel ca x > 0
120 NEXT I
130 PRINT: PRINT RND(0)
RUN
.6291626 .1948297 .6305799
.6818615 .4193624 .6215937
.6818615 .4193624 .6215937
.6215937

```

## RUN

Comandă

**Scop:** Începe execuția unui program.

**Format:** *RUN [linie]*  
*RUN numefis[,R]*

**Comentariu:** *linie* este numărul liniei din programul curent de la care se dorește începerea execuției.

*numefis* este o expresie șir care indică numele fișierului de executat.

**RUN** sau **RUN linie** începe execuția programului curent din memorie.

**RUN numefis** încarcă un fișier de pe disc în memorie și începe execuția lui. Înainte de a încărca programul specificat, această comandă închide toate fișierele deschise anterior și șterge conținutul memoriei, după care va executa încărcarea noului program în memorie. Dacă se adaugă opțiunea **R**, toate fișierele de date rămân deschise.

**Exemple:**

```
10 PI = 3.141593
```

```
20 PRINT PI
```

```
RUN
```

```
3.141593
```

```
RUN 20
```

```
0
```

## SAVE

**Comandă**

**Scop:** Salvează pe disc un fișier BASIC.

**Format:** *SAVE numefis[,A]*  
*SAVE numefis[,P]*

**Comentariu:** *numefis* este o expresie șir care indică numele sub care va fi salvat programul curent din memorie pe disc.

Opțiunea **A** salvează programul în format ASCII. Dacă nu se specifică această opțiune, programul va fi salvat în format binar comprimat. Fișierele de tip ASCII ocupă puțin mai mult spațiu pe disc, dar sînt cazuri cînd este necesară salvarea fișierelor de program sub acest format. Programele salvate în ASCII pot fi citite ca fișiere de date.

Opțiunea **P** (protejat) salvează programul într-un format binar codat. Un program protejat poate fi încărcat și rulat, dar nu mai poate fi listat sau editat. Nu există modalități de "deprotejare" a unui program după ce acesta a fost în prealabil salvat cu opțiunea **P**.

**Exemple:**

```
SAVE "PROG1"
```

```
SAVE "FISIER",A
```

```
SAVE "SECRET.TOT",P
```

## SCREEN

**Funcție**

**Scop:** Atribuie unei variabile codul ASCII (0-255) corespunzător caracterului din rîndul și coloana specificată.

**Format:**  $v = \text{SCREEN}(\text{rînd}, \text{col}, [\text{z}])$

**Comentariu:** *rînd* este o expresie numerică cu valori între 1 și 25.

*col* este o expresie numerică cu valori între 1 și 40 sau 1 și 80.

*z* este o expresie numerică evaluată la adevărat sau fals.

În modul text, dacă se include parametrul opțional *z* și acesta este adevărat (diferit de zero), atunci se va atribui variabilei atributul de culoare al caracterului de la poziția respectivă în locul codului caracterului respectiv. Atributul de culoare este un număr cuprins între 0 și 255. Acest număr este:

$v \text{ MOD } 16$  atributul de culoare al caracterului;

$((v - \text{c.car})/16) \text{ MOD } 128$  atributul de culoare pentru fundal, unde **c.car** este atributul de culoare al caracterului, calculat ca mai sus;

$V > 127$  este adevărat (-1) dacă caracterul clipește și fals (0) invers.

**Exemple:** Presupunînd că la poziția (10,10) de pe ecran se află caracterul A, atunci:

```
10 X = SCREEN(10,10)
```

```
20 PRINT X
```

```
RUN
```

```
65
```

Următorul exemplu atribuie variabilei *x* valoarea atributului de culoare din partea stînga sus a ecranului:

```
100 X = SCREEN(1,1,1)
```

## SCREEN

Instrucțiune

**Scop:** Specifică atributul de culoare al ecranului pentru instrucțiunile ce urmează.

**Format:**  $\text{SCREEN} [\text{mod}][, [\text{culoare}][, [\text{apag}][, [\text{vpag}]]]$

**Comentariu:** *mod* este o expresie numerică care are ca rezultat o valoare întregă, după cum urmează:

0 modul text la lățimea curentă (40 sau 80 de caractere);

1 rezoluție medie în modul grafic (320x200).

2 rezoluție înaltă în modul grafic (640x200).

*culoare* este o expresie numerică ce are ca rezultat o valoare adevărată sau falsă. Activează sau dezactivează culoarea. În modul text ( $\text{mod} = 0$ ), o valoare falsă (zero) dezactivează culoarea (doar imaginile monocrome sînt afișate); o valoare adevărată (diferită de zero) activează culoarea (sînt afișate imaginile color). În modul grafic, rezoluție medie ( $\text{mod} = 1$ ), o valoare adevărată (diferită de zero) va dezafecta culoarea, iar



o valoare falsă (zero) activează culoarea. În modul grafic, rezoluție înaltă fiind doar două culori (alb și negru), acest parametru nu are semnificație.

*apag* (pagina activă) selectează pagina ce urmează a fi afișată pe ecran; are valori cuprinse între 0 și 7 pentru 40 de coloane și 0 și 3 pentru 80 de coloane. Este validă doar pentru modul text ( $\text{mod} = 0$ ).

*vpag* (pagina vizuală) selectează pagina ce urmează a fi afișată pe ecran în același mod ca și *apag* de mai sus. Pagina vizuală poate fi diferită de pagina activă. Ca și *apag*, *vpag* este validă doar în modul text ( $\text{mod} = 0$ ). Dacă este omisă *vpag* este egală cu *apag*.

Dacă toți parametrii sînt corecți, noul mod ecran este memorat; ecranul este curățat; se va selecta culoarea alb pentru caracter și negru pentru fundal și margine.

## SGN

**Funcție**

**Scop:** Atribuie unei variabile semnul lui  $x$ .

**Format:**  $v = \text{SGN}(x)$ .

**Comentariu:**  $x$  este o expresie numerică oarecare.

$\text{SGN}(x)$  este funcția matematică signum:

- dacă  $x$  este pozitiv,  $\text{SGN}(x)$  va avea ca rezultat 1;
- dacă  $x$  este zero,  $\text{SGN}(x)$  va avea ca rezultat 0;
- dacă  $x$  este negativ,  $\text{SGN}(x)$  va avea ca rezultat -1.

**Exemplu:** Exemplul următor realizează salt la linia 100 dacă  $x$  este negativ; la 200 dacă  $x$  este zero și la linia 300 dacă  $x$  este pozitiv:

```
ON SGN(X) + 2 GOTO 100,200,300
```

## SHELL

**Instrucțiune**

**Scop:** Încarcă și execută un alt fișier de program (ca de exemplu fișiere cu extensie .COM, .BAT, .EXE). Orice program executat sub limbajul BASIC este considerat un proces descendent. La terminarea procesului descendent, controlul este preluat din nou de ascendentul lui BASIC, respectiv de instrucțiunea imediat următoare instrucțiunii **SHELL**.

**Format:** *SHELL* [*șir de comandă*]

**Comentariu:** *șir de comandă* este o expresie șir conținând numele programului de rulat și, opțional, orice parametru ce trebuie trecut procesului descendent.

Procesul descendent este executat de instrucțiunea **SHELL** prin încărcarea și rularea fișierului **COMMAND.COM** cu opțiunea **/C**. Dacă

instrucțiunea **SHELL** nu conține șirul de comandă, se va încărca o copie a fișierului **COMMAND.COM**, va apare promptul **DOS**, iar din acest moment utilizatorul va putea introduce orice comandă validă în sistemul de operare **DOS**. Pentru întoarcerea la procesul ascendent **BASIC** se va introduce în linia de comandă **DOS** comanda **EXIT**.

**Exemple:**

```
10 OPEN "SORTIN.DAT" FOR OUTPUT AS #1
20 'scrie datele de sortat
.
.
.
100 CLOSE 1
110 SHELL "SORT < SORTIN.DAT >
SORTOUT.DAT"
120 OPEN "SORTOUT.DAT" FOR INPUT AS #1
130 'proceseaza datele sortate
.
.
.
```

Exemplul următor afișează directorul discului A: din **BASIC**:

```
SHELL
A > DIR (tipareste comanda dir sub promptul DOS)
.
.
.
A > EXIT (tipareste EXIT pentru reintoarcere în
BASIC)
```

Același rezultat se poate obține mai simplu cu:

```
SHELL "DIR A:"
```

## SIN

**Funcție**

**Scop:** Calculează funcția sinus trigonometric.

**Format:**  $y = SIN(x)$

**Comentariu:**  $x$  este unghiul în radiani.

Pentru conversia gradelor în radiani se va multiplica cu  $PI/180$ , unde  $PI = 3.141593$ . Pentru calculul corect al acestei funcții în dublă precizie se va specifica opțiunea **/D** în linia de comandă **BASIC**, în momentul inițializării limbajului.

**Exemplu:** Considerînd că limbajul **BASIC** a fost inițializat cu opțiu-

nea /D și că dorim să calculăm sinus de 90 de grade, vom executa:

```
10 DEFDBL G,P,R
20 PI = ATN(1#)*4
30 GRADE = 90
40 RADIANI = GRADE*PI/180 ' PI/2
50 PRINT SIN(RADIANI)
RUN
1
```

## SOUND

Instrucțiune

**Scop:** Generează sunete în difuzor.

**Format:** *SOUND frecv,durată*

**Comentariu:** *frecv* este frecvența dorită în Herți (cicli pe secundă). Trebuie să fie o expresie numerică cu valori cuprinse între 37 și 32767.

*durata* este durata dorită în eșantioane de timp. Într-o secundă sînt 18,2 eșantioane de timp. *durata* trebuie să fie o expresie numerică cu valori cuprinse între 0,0015 și 65535.

După ce instrucțiunea **SOUND** generează un sunet, programul continuă execuția pînă în momentul în care se ajunge la o nouă instrucțiune **SOUND**. Dacă *durata* noii instrucțiuni **SOUND** este 0, se oprește execuția instrucțiunii **SOUND** anterioare. Altfel, programul așteaptă pînă cînd se termină execuția primului sunet și apoi dă execuția celui de-al doilea.

Nota	Frecventa	Nota	Frecventa
DO	130.810	DO	523.250
RE	146.830	RE	587.330
MI	164.810	MI	659.260
FA	174.610	FA	698.460
SOL	196.000	SOL	783.990
LA	220.000	LA	880.000
SI	246.940	SI	987.770
DO	261.630	DO	1046.500
RE	293.660	RE	1174.700
MI	329.630	MI	1318.500
FA	349.230	FA	1396.900
SOL	392.000	SOL	1568.000
LA	440.000	LA	1760.000
SI	493.880	SI	1975.500

Durata unei măsurii poate fi calculată divizînd numărul de măsurii pe minut la 1092 (numărul de eșantioane de timp pe minut).



Tabelul următor indică tempo-urile tipice funcție de numărul de eșantioane de timp:

	Tempo	Măsuri/ Minut	Eșantioane/ Măsura
foarte lent	Larghissimo		
	Largo	40-60	27.3-18.2
	Larghetto	60-66	18.2-16.55
	Grave		
	Lento		
lent	Adagio	66-76	16.55-14.37
	Adagietto		
	Andante	76-108	14.37-10.11
mediu	Andantino		
rapid	Moderato	108-120	10.11-9.1
	Allegretto		
	Allegro	120-168	9.1-6.5
	Vivace		
	Veloce		
foarte rapid	Presto	168-208	6.5-5.25
	Prestissimo		

**Exemplu:** Următorul program crează un glisando:

```

10 FOR I = 440 TO 1000 STEP 5
20 SOUND I,0.5
30 NEXT
40 FOR I = 1000 TO 440 STEP -5
50 SOUND I,0.5
60 NEXT

```

## SPACES

**Instrucțiune**

**Scop:** Are ca rezultat un șir de  $n$  spații.

**Format:**  $v\$ = SPACE\$(n)$

**Comentariu:**  $n$  este cuprins între 0 și 255.

**Exemplu:**

```

10 FOR I = 1 TO 5
20 X$ = SPACE(I)
30 PRINT X$;I
40 NEXT I
RUN

```

```

1
  2
    3
      4
        5

```

## SPC

Funcție

**Scop:** Sare  $n$  spații într-o instrucțiune PRINT.

**Format:** *PRINT SPC( $n$ )*

**Comentariu:**  $n$  este o valoare cuprinsă între 0 și 255.

Dacă  $n$  este mai mare decât lățimea perifericului de afișare specificat, valoarea utilizată va fi  $n \text{ MOD lățime}$ .

**Exemplu:**

```

PRINT "A" SPC(15) "B"
A                               B

```

## SQR

Funcție

**Scop:** Atribuie unei variabile valoarea rădăcinii pătrate din  $n$

**Format:**  $v = \text{SQR}(x)$

**Comentariu:**  $x$  este o expresie numerică mai mare sau cel puțin egală cu 0.

Pentru calculul corect al acestei funcții în dublă precizie se va specifica opțiunea /D în linia de comandă BASIC, în momentul inițializării limbajului.

**Exemplu:**

```

10 FOR X = 10 TO 25 STEP 5
20 PRINT X,SQR(X)
30 NEXT
RUN
10      3.162278
15      3.872984
20      4.472136
25      5

```

## STICK

Funcție

**Scop:** Atribuie unei variabile coordonatele  $x$  și  $y$  a două joystick-uri.

**Format:**  $v = \text{STICK}(n)$

**Comentariu:** *n* este o expresie numerică cu valori între 0 și 3, ce afectează rezultatul după cum urmează:

0 atribuie coordonata x pentru joystick-ul A.

1 atribuie coordonata y pentru joystick-ul A.

2 atribuie coordonata x pentru joystick-ul B.

3 atribuie coordonata y pentru joystick-ul B.

**Exemplu:** Exemplul următor afișează 100 de eșantioane ale coordonatelor joystick-ului B:

```
10 PRINT "Joystick B"
```

```
20 PRINT "Coordonata x","Coordonata y"
```

```
30 FOR J = 1 TO 100
```

```
40 TMP = STICK(0)
```

```
50 X = STICK(2): Y = STICK(3)
```

```
60 PRINT X,Y
```

```
70 NEXT
```

## STOP

**Instrucțiune**

**Scop:** Oprește execuția programului și dă controlul la nivelul de comandă.

**Format:** STOP

**Comentariu:** Instrucțiunea STOP poate fi folosită oriunde în program pentru a întrerupe execuția.

Spre deosebire de instrucțiunea END, instrucțiunea STOP nu închide fișierele.

Din nivelul de comandă se poate reîncepe execuția programului de la instrucțiunea care urmează instrucțiunii STOP cu ajutorul comenzii CONT.

**Exemplu:**

```
10 INPUT A,B
```

```
20 TEMP = A*B
```

```
30 STOP
```

```
40 TFIN = TEMP + 200: PRINT FINAL
```

```
RUN
```

```
? 26, 2.1
```

```
Break in 30
```

```
PRINT TEMP
```

```
54.6
```

```
CONT
```

```
254.6
```



## STR\$

Funcție

**Scop:** Transformă o valoare într-un șir de caractere ASCII.

**Format:**  $v\$\$ = STR\$(x)$

**Comentariu:**  $x$  este orice expresie numerică.

Dacă  $x$  este pozitiv, șirul rezultat va conține în față un spațiu (spațiul rezervat pentru semnul plus). De exemplu:

? STR\$(321); LEN(STR\$(321))

321 4

Funcția VAL este complementară funcției STR\$.

## STRIG

Instrucțiune și Funcție

**Scop:** Citește statutul butoanelor de la joystick.

**Format:**

Ca instrucțiune: *STRIG ON*

*STRIG OFF*

Ca funcție:  $v = STRIG(n)$

**Comentariu:**  $n$  este o expresie numerică cu valori între 0 și 7. Afec-tează valoarea atribuită după cum urmează:

0 Are ca rezultat -1 dacă butonul A1 a fost apăsat de la ultima apelare a funcției STRIG(0); dacă nu, are ca rezultat 0.

1 Are ca rezultat -1 dacă butonul A1 este în acel moment apăsat; dacă nu, are ca rezultat 0.

2 Are ca rezultat -1 dacă butonul B1 a fost apăsat de la ultima apelare a funcției STRIG(2); dacă nu, are ca rezultat 0.

3 Are ca rezultat -1 dacă butonul B1 este în acel moment apăsat; dacă nu, are ca rezultat 0.

4 Are ca rezultat -1 dacă butonul A2 a fost apăsat de la ultima apelare a funcției STRIG(4); dacă nu, are ca rezultat 0.

5 Are ca rezultat -1 dacă butonul A2 este în acel moment apăsat; dacă nu, are ca rezultat 0.

6 Are ca rezultat -1 dacă butonul B2 a fost apăsat de la ultima apelare a funcției STRIG(6); dacă nu, are ca rezultat 0.

7 Are ca rezultat -1 dacă butonul B2 este în acel moment apăsat; dacă nu, are ca rezultat 0.

Instrucțiunea **STRIG ON** trebuie executată înainte de a fi apelată funcția **STRIG(n)**. După **STRIG(n)**, de fiecare dată când se începe o nouă instrucțiune, limbajul verifică dacă a fost apăsat un buton de la joystick. Pentru a dezafecta testul se apelează instrucțiunea **STRIG OFF**.

## STRIG(n)

Instrucțiune

**Scop:** Activează și dezactivează testarea butoanelor de la joystick.

**Format:** *STRIG(n) ON*  
*STRIG(n) OFF*  
*STRIG(n) STOP*

**Comentariu:** **STRIG(n) ON** trebuie executată pentru a activa testarea butoanelor prin instrucțiunea **STRIG(n)**

Instrucțiunii **STRIG(n) OFF** dezafectează testarea.

Instrucțiunea **STRIG(n) STOP** dezafectează temporar testarea. În acest caz, dacă butonul este apăsat în acest interval, acțiunea este memorată și, în momentul în care este activată instrucțiunea **STRIG(n) ON**, acțiunea respectivă este executată.

## STRING\$

Funcție

**Scop:** Atribuie unei variabile un șir de lungime  $n$  ale cărui caractere au codul ASCII  $m$  sau primul caracter al șirului  $x$ .

**Format:**  $v\$ = \text{STRING}\$(n,m)$   
 $v\$ = \text{STRING}\$(n,x\$)$

**Comentariu:**  $n, m$  sînt cuprinse între 0 și 255.

$x$  este orice expresie șir.

**Exemple:**

```
10 X$ = STRING$(10,45)
20 PRINT X$ "RAPORT LUNAR" X$
RUN
-----RAPORT LUNAR-----
```

```
10 X$ = "ABC"
20 S$ = STRING$(5,X$)
30 PRINT S$
RUN
AAAAA
```

## SWAP

Instrucțiune

**Scop:** Schimbă între ele valoarea a două variabile.

**Format:** *SWAP var1,var2*



**Comentariu:** *var1, var2* sînt numele a două variabile sau elemente ale unei matrici.

Operația de schimbare poate fi efectuată pe orice tip de variabilă (simplă precizie, dublă precizie sau șir), cu condiția ca cele două variabile să fie de același tip.

**Exemplu:**

```
10 A$ = " UNUL ": B$ = " TOTI ": C$ = "PENTRU"  
20 PRINT A$,C$,B$  
30 SWAP A$,B$  
40 PRINT A$,C$,B$  
RUN  
UNUL PENTRU TOTI  
TOTI PENTRU UNUL
```

## SYSTEM

Comandă

**Scop:** Iese din limbajul BASIC în sistemul de operare DOS.

**Format:** *SYSTEM*

**Comentariu:** Comanda **SYSTEM** închide toate fișierele înainte de întoarcerea în sistemul de operare. Dacă nu este salvat în prealabil, programul curent din memorie se pierde.

## TAB

Funcție

**Scop:** Mută cursorul pe linia curentă în coloana *n*.

**Format:** *PRINT TAB(n)*

**Comentariu:** *n* este cuprins între 1 și 255.

Funcția **TAB** poate fi utilizată în instrucțiunile **PRINT**, **LPRINT** și **PRINT #**.

**Exemplu:**

```
10 PRINT "NUME" TAB(25) "DATA NASTERII"  
20 READ A$,B$  
30 PRINT A$ TAB(25) B$  
40 DATA "G. IONESCU","15 MAI 1930"  
RUN  
NUME DATA NASTERII  
G. IONESCU 15 MAI 1930
```

## TAN

Funcție

**Scop:** Atribuie unei variabile valoarea tangentei.



**Format:**  $v = TAN(x)$

**Comentariu:**  $x$  este unghiul în radiani.

Pentru conversia gradelor în radiani se va multiplica cu  $PI/180$ , unde  $PI = 3.141593$ . Pentru calculul corect al acestei funcții în dublă precizie se va specifica opțiunea **/D** în linia de comandă **BASIC**, în momentul inițializării limbajului.

**Exemplu:** Acest program calculează tangenta de 45 grade:

```
10 PI = 3.141593
```

```
20 GRADE = 45
```

```
30 PRINT TAN(GRADE*PI/180)
```

```
RUN
```

```
1
```

## TIME

**Variabilă și Instrucțiune**

**Scop:** Stabilește sau citește timpul curent.

**Format:**

Ca variabilă:  $v\$ = TIME\$$

Ca instrucțiune:  $TIME\$ = x\$$

**Comentariu:**

**Pentru variabilă ( $v\$ = TIME\$$ ):**

Timpul curent este citit ca un șir de 8 caractere. Șirul este de forma  $hh:mm:ss$ , unde  $hh$  sînt orele (0 la 23),  $mm$  sînt minutele (0 la 59), iar  $ss$  reprezintă secundele (0 la 59).

**Pentru instrucțiune ( $TIME\$ = x\$$ ):**

Stabilește timpul curent "potrivind" ceasul calculatorului.  $x\$$  este o expresie șir (de forma  $hh:mm:ss$ ) indicînd timpul ce urmează a fi stabilit.

**Exemplu:** Exemplul de mai jos afișează continuu timpul pe ecran:

```
10 CLS
```

```
20 LOCATE 10,15
```

```
30 PRINT TIME$
```

```
40 GOTO 20
```

## TIMER

**Funcție**

**Scop:** Are ca rezultat un număr în simplă precizie reprezentînd numărul de secunde trecute de la miezul nopții sau de la inițializarea sistemului.

**Format:**  $v = TIMER$

**Exemplu:**

```
10 TIME$ = "23:59:59"
```

```

20 FOR I = 1 TO 70
30 PRINT "TIME$ = ";TIME$;" TIMER = ";TIMER
40 NEXT
RUN
TIME$ = 23:59:59  TIMER = 86399.06
TIME$ = 23:59:59  TIMER = 86399.11
TIME$ = 23:59:59  TIMER = 86399.18
.
.
.
TIME$ = 24:00:00  TIMER = 0
TIME$ = 00:00:00  TIMER = .05
TIME$ = 00:00:00  TIMER = .16
TIME$ = 00:00:00  TIMER = .23

```

## TRON și TROFF

Comenzi

**Scop:** Trasează execuția instrucțiunilor din program.

**Format:** *TRON*  
*TROFF*

**Comentariu:** Ca un ajutor în depanarea programelor, comanda **TRON** poate fi introdusă atât în interiorul programului cât și în modul direct, activând ca trasor numărul liniei de program în execuție. Numerele liniilor de program apar în paranteze drepte. Trasarea se dezafectează cu comanda **TROFF**.

**Exemplu:**

```

10 K = 10
20 FOR I = 1 TO 2
30 L = K + 10
40 PRINT J;K;L
50 K = K + 10
60 NEXT
70 END
TRON
RUN
[10][20][30][40] 0 10 20
[50][60][30][40] 0 20 30
[50][60][70]
TROFF

```

## USR

**Funcție**

**Scop:** Apelează subrutina în cod mașina indicată de argumentul *arg*.

**Format:**  $v = USR[n](arg)$

**Comentariu:** *n* este un număr cuprins între 0 și 9 corespunzând numărului stabilit de instrucțiunea DEF USR pentru rutina dorită. Dacă se omite *n* se ia USR0.

*arg* este orice expresie numerică sau variabilă șir ce va trece ca argument în subrutina limbaj masină. Dacă subrutina nu cere argumente, este necesară prezența unui argument fals.

**Exemplu:**

```
10 DEFINT A-Z
20 OPTION BASE 1
30 X = 0: Y = 0: Z = 0
40 DIM M(&H14)
50 Z = VARPTR(M(1))
60 BLOAD "SUBRT.COM",Z
70 DEF USR0 = Z
80 X = 5
90 Y = USR0(X)
100 PRINT Y
```

## VAL

**Funcție**

**Scop:** Atribuie unei variabile valoarea numerică a șirului *x\$*.

**Format:**  $v = VAL(x\$)$

**Comentariu:** *x\$* este o expresie șir.

Funcția VAL scoate toate spațiile din șirul argument pentru a determina rezultatul. De exemplu:

```
VAL(" -3")
```

va avea ca rezultat -3.

Dacă primul caracter al șirului *x\$* nu este numeric, rezultatul funcției VAL(*x\$*) va fi 0 (zero).

## VARPTR

**Funcție**

**Scop:** Are ca rezultat offset-ul segmentului curent de memorie a variabilei argument.

**Format:**  $v = VARPTR(var)$



**Comentariu:** *var* este numele unei variabile șir' sau numerice sau al unui element de matrice din program.

Valoarea rezultată este cuprinsă între 0 și 65535. Acest număr este offset-ul din segmentul de date BASIC al primului octet de date identificat cu variabila.

Funcția **VARPTR** este folosită pentru a obține offset-ul unei variabile din segmentul de date BASIC, astfel că ea poate fi folosită pentru apelarea subrutinelor în limbaj mașină în scopul trecerii variabilelor spre acestea.

**Exemplu:**

```
10 DEFINT A-Z
20 DATA1 = 500
30 P = VARPTR(DATA1)
40 V = PEEK(P) + 256*PEEK(P + 1)
50 PRINT V
```

## VARPTR\$

Funcție

**Scop:** Atribuie unei variabile, sub formă de caractere, offset-ul unei variabile din memorie. Este folosită, de obicei, cu instrucțiunile **PLAY** și **DRAW**.

**Format:**  $v\$ = \text{VARPTR}\$(var)$

**Comentariu:** *var* este numele unei variabile din program.

Se poate folosi **VARPTR\$** pentru a indica numele unei variabile în șirul de comenzi ale instrucțiunilor **PLAY** sau **DRAW**. De exemplu:

```
PLAY "X$;"
```

se mai poate scrie:

```
PLAY "X" + VARPTR$(A$)
```

sau:

```
PLAY "O = I;"
```

se mai poate scrie:

```
PLAY "O =" + VARPTR$(I)
```

## VIEW

Instrucțiune

**Scop:** Definește un subset rectangular din ecran la care face apel instrucțiunea **WINDOW**. Are înțeles doar în modul grafic.

**Format:**  $\text{VIEW} [ [\text{SCREEN}][x1,y1)-(x2,y2)[,[culoare][,[margine]]]] ]$

**Comentariu:**  $(x1,y1)-(x2,y2)$  sînt coordonatele sus-stînga, respectiv

jos-dreapta ale ferestrei definite. x și y trebuie să fie în limitele actuale definite ale ecranului.

*culoare* permite umplerea ferestrei cu o culoare. *culoare* este o expresie întreagă care poate fi unul dintre atributele de culoare permise.

*margin* permite încadrarea ferestrei cu o linie. *margin* este o expresie întreagă cu valori între limitele descrise la parametrul *culoare*.

Dacă argumentul SCREEN este omis, toate punctele plotate sînt relative la fereastră. Dacă se include argumentul SCREEN, toate punctele plotate sînt absolute și pot fi atît în interiorul cît și în exteriorul limitelor ecranului.

VIEW fără parametrii definește tot ecranul ca fereastră.

Prin această metodă se pot defini mai multe ferestre, dar numai una poate fi activă la un moment dat.

**Exemplu:** Următorul exemplu definește patru ferestre:

```
10 SCREEN 1: VIEW: CLS: KEY OFF
20 VIEW(1,1)-(151,91),,1
30 VIEW(165,1)-(315,91),,2
40 VIEW(1,105)-(151,195),,2
50 VIEW(165,105)-(315,195),,1
60 LOCATE 2,4:PRINT "Fereastră 1"
70 LOCATE 2,25: PRINT "Fereastră 2"
80 LOCATE 15,4: PRINT "Fereastră 3"
90 LOCATE 15,25: PRINT "Fereastră 4"
100 VIEW(1,1)-(151,91): GOSUB 1000
200 VIEW(165,1)-(315,91): GOSUB 2000
300 VIEW (1,105)-(151,195): GOSUB 3000
400 VIEW (165,105)-(315,195): GOSUB 4000
900 END
1000 CIRCLE (65,50),30,2
1010 ' Deseneaza un cerc în prima fereastră
1020 RETURN
2000 LINE (45,50)-(90,75),1,B
2010 ' Deseneaza un patrat in a doua fereastră
2020 RETURN
3000 FOR D = 0 TO 360: DRAW "ta = d;nu20": NEXT
3010 ' Deseneaza spite in a treia fereastră
3020 RETURN
4000 PSET (60,50),2: DRAW "e15 f15 l30"
4010 ' Deseneaza un triunghi in ultima fereastră
4020 RETURN
```



Următorul exemplu demonstrează scalarea cu ajutorul instrucțiunii

View:

```
10 KEY OFF: CLS: SCREEN 1,0: COLOR 0,0
20 WINDOW SCREEN (320,0)-(0,200)
30 GOTO 80
40 C = 1
50 CIRCLE (160,100),60,C,,,5/18
60 CIRCLE (160,100),60,C,,,1
70 RETURN
80 GOSUB 40: FOR I = 1 TO 1500: NEXT I: CLS
90 VIEW (1,1)-(160,90),,2: GOSUB 40
100 GOTO 100
```

## WAIT

Instrucțiune

**Scop:** Suspendă execuția programului în timpul monitorizării unuia dintre porturile calculatorului.

**Format:** *WAIT port,n[,m]*

**Comentariu:** *port* este numărul unuia dintre porturile calculatorului, între 0 și 65535.

*n,m* sînt expresii întregi cu valori între 0 și 255.

Instrucțiunea **WAIT** suspendă execuția programului pînă la primirea de la portul dorit al calculatorului a unui șir de biți specificat.

Datelor citite de la port li se aplică operația XOR cu expresia întregă *m*, apoi operația AND cu *n*. Apoi se execută salt înapoi și se citește, din nou, datele de la port. Dacă rezultatul este diferit de zero, execuția continuă cu următoarea instrucțiune. Dacă *m* se omite, se ia 0.

Instrucțiunea **WAIT** permite testarea a unuia sau mai mulți biți de la portul de intrare. Se pot testa pozițiile biților atît pentru 1 cît și pentru 0. Pozițiile biților de testat sînt specificate punînd 1-uri pentru acele poziții în *n*. Dacă nu se specifică *m*, biții portului de intrare sînt testați pentru 1. Dacă *m* este specificat, un 1 în orice poziție a lui *m* (pentru care există un bit de 1 în *n*) face ca **WAIT** să testeze biții de 0 la intrare.

**Exemplu:** Acest exemplu suspendă execuția programului pînă în momentul în care recepționează un bit de 1 pe poziția 2 de la portul 32:

```
WAIT 32,2
```

## WHILE si WEND

Instrucțiuni

**Scop:** Execută o serie de instrucțiuni într-o buclă atîta timp



cît o condiție dată este adevărată.

**Format:**

**WHILE** *exp*

.

.

.

(**buclă de instrucțiuni**)

.

.

.

**WEND**

**Comentariu:** *exp* este orice expresie numerică.

Dacă *exp* este adevărată (diferită de zero), se execută bucla de instrucțiuni pînă cînd este înregistrată instrucțiunea **WEND**. Dacă *exp* este în continuare adevărată, procesul se repetă. Dacă *exp* devine falsă (egală cu zero), execuția continuă cu instrucțiunea imediat următoare instrucțiunii **WEND**.

**Exemplu:** Exemplul de mai jos sortează elementele vectorului A în ordine alfabetică. Matricea A a fost definită cu J elemente.

100 F = 1

110 WHILE F

120 F = 0

130 FOR I = 1 TO J-1

140 IF A(I) > A(I + 1) THEN SWAP A(I), A(I + 1):

F = 1

150 NEXT I

160 WEND

## WIDTH

**Instrucțiune**

**Scop:** Fixează numărul de caractere pe o linje. După tipărirea numărului respectiv de caractere, limbajul așează cursorul pe o linie nouă în prima coloană din stînga, prin adăugarea unui "carriage return".

**Format:** *WIDTH* *mărime*  
*WIDTH* *periferic*, *mărime*  
*WIDTH* #*nrfis*, *mărime*

**Comentariu:** *mărime* este o expresie numerică cuprinsă între 0 și 255. Aceasta este noua lățime. **WIDTH 0** are aceeași semnificație cu **WIDTH 1**.

*periferic* este o expresie șir pentru identificatorul perifericului. Perifericele valide sînt SCR:N:, LPT1:, LPT2:, LPT3:, COM1: sau COM2:.

*nrfis* este o expresie numerică cu valori între 0 și 15. Este numărul fișierului deschis ca periferic de ieșire.

În funcție de perifericul specificat, sînt posibile următoarele acțiuni:

**WIDTH mărime sau WIDTH "SCRN:",mărime**

Fixează lățimea ecranului. Pentru *mărime* sînt permise doar valorile 40 sau 80. **WIDTH 40** nu este permis pentru adaptorul monocrom.

Specificînd lățimea la 255 se consideră lățime "infinită". Această lățime este utilizată de obicei cu "COM1:" sau "COM2:" pentru fișierele de comunicații.

**Exemple:**

**WIDTH "LPT1:",75**

fixează lățimea de 75 caractere la imprimantă.

**WIDTH 40**

fixează lățimea ecranului la 40 caractere și este același cu:

**WIDTH "SCRN:",40**

## WINDOW

Instrucțiune

**Scop:** Redefinește coordonatele ferestrei. Are semnificație doar în modul grafic.

**Format:** *WINDOW [ [SCREEN](x1,y1)-(x2,y2)]*

**Comentariu:**  $(x1,y1)$ ,  $(x2,y2)$  sînt coordonatele definite de programator și se numesc coordonate universale. Aceste coordonate sînt numere reale în simplă precizie. Ele definesc spațiul coordonatelor universale ce va fi transformat în spațiu de coordonate fizice cu ajutorul instrucțiunii VIEW.

Instrucțiunea **WINDOW** permite desenarea unor obiecte (în coordonate univereale) ce nu sînt limitate de marginile ecranului (coordonate fizice). Aceasta se face specificînd perechea de coordonate universale  $(x1,y1)$  și  $(x2,y2)$ . Limbajul convertește, apoi, perechea de coordonate universale pentru a putea fi plotată în fereastră. Pentru a putea face această transformare din spațiul universal în spațiul fizic al ecranului, limbajul trebuie să cunoască care porțiune a spațiului universal nemărginit conține informația de plotat. Această regiune rectangulară în spațiul coordonatelor universale se numește *fereastră*.

În sistemul de coordonate fizice, dacă se rulează:

**NEW**

**SCREEN 2**

ecranul apare în coordonatele standard:

0,0	320,0	639,0
	320,100	
0,199	320,199	639,199

Dacă se omite atributul **SCREEN**, ecranul este văzut în coordonate carteziene. De exemplu introducând:

**WINDOW (-1,-1)-(1,1)**

ecranul va apare:

1,1	0,1	1,1
	0,0	
-1,-1	0,-1	1,-1

De notat faptul că coordonata  $y$  este inversată, astfel  $(x_1, y_1)$  este coordonata jos-stînga, iar  $(x_2, y_2)$  este coordonata dreapta-sus.

Dacă se include atributul **SCREEN**, coordonatele nu sînt inversate, astfel  $(x_1, y_1)$  este coordonata sus-stînga, iar  $(x_2, y_2)$  este coordonata jos-dreapta. De exemplu:

**WINDOW SCREEN (-1,-1)-(1,1)**

definește ecranul:

1,-1	0,-1	1,-1
	0,0	
-1,1	0,1	1,1



Este important de notat faptul că instrucțiunea **WINDOW** sortează perechile de argumente  $x$  și  $y$ , plasând primele valorile mai mici pentru  $x$  și  $y$ . De exemplu:

**WINDOW (100,100)-(5,5)**

devine:

**WINDOW (5,5)-(100,100)**

Un alt exemplu:

**WINDOW (-4,4)-(4,-4)**

devine:

**WINDOW (-4,-4)-(4,4)**

Orice pereche de  $x$  și  $y$  este validă cu singura restricție că  $x_1$  nu poate fi egal cu  $x_2$  și  $y_1$  nu poate fi egal cu  $y_2$ .

Instrucțiunea **WINDOW** permite crearea efectelor de "lupă" și "decupaj".

Dacă nu se specifică nici un argument, se trece din nou la coordonatele fizice.

**Exemple:** Exemplul de mai jos ilustrează efectul de "decupaj" cu ajutorul instrucțiunii **WINDOW**:

```
10 SCREEN 2: CLS
20 WINDOW (-6,-6)-(6,6)
30 CIRCLE (4,4),5,1
40 ' cercul este mare si doar în parte vizibil
50 WINDOW (-100,-100)-(100,100)
60 CIRCLE (4,4),5,1 ' cercul este foarte mic
70 END
```

Exemplul următor ilustrează efectul de "zoom":

```
10 KEY OFF: CLS: SCREEN 1,0
20 '
30 GOTO 160
40' = = = = =
50 ' procedura afisaj
60 '
70 LINE (X,0)-(-X,0),,,&HAA00 'axa x
80 LINE (0,X)-(0,-X),,,&HAA00 'axa y
90 '
100 CIRCLE (X/2,X/2),R 'cercul are raza r
110 FOR P= 1 TO 500: NEXT P 'bucla de intirziere
120 '
130 RETURN
140' = = = = =
```

```

150 '
160 X = 1000: WINDOW(-X,-X)-(X,X): R = 20
170 '
180 GOSUB 50: FOR P = 1 TO 1000: NEXT P: CLS
190 '
200 X = 60: WINDOW (-X,-X)-(X,X): R = 20
210 '
220 GOSUB 50: FOR P = 1 TO 1000: NEXT P: CLS
230 '
240 X = 100: WINDOW (-5,-5)-(X,X): R = 20
250 '
260 GOSUB 50: FOR P = 1 TO 1000: NEXT P: CLS
270 '
280 PRINT "... un exemplu": PRINT ". de zooming.."
290 FOR P = 1 TO 2000: NEXT P
300 CLS: T = -50: U = 100: X = U: R = 20
310 FOR I = 1 TO 45
320 T = T + 1: U = U - 1: X = X - 1
330 WINDOW (T,T)-(U,U): CLS: GOSUB 50
340 NEXT I
350 END

```

## WRITE

Instrucțiune

**Scop:** Trimite date spre ecran.

**Format:** *WRITE [lista de expresii]*

**Comentariu:** *lista de expresii*

este o listă de expresii șir și/sau numerice, separate prin punct și virgulă sau virgule.

Dacă lipsește lista de expresii, se afișează o linie liberă.

La afișare, valorile expresiilor numerice sînt separate prin virgule, iar expresiile șir sînt delimitate de ghilimele. După ultimul termen al listei de expresii limbajul adaugă un "carriage return/line feed".

Instrucțiunea **WRITE** este similară cu instrucțiunea **PRINT**, singura diferență între cele două instrucțiuni fiind că **WRITE** inserează virgule între termeni, iar șirurile sînt delimitate de ghilimele. De asemenea numerele pozitive nu sînt precedate de spații.

**Exemplu:**

```

10 A = 80: B = 90: C$ = "ABCD"
20 WRITE A,B,C$

```

## RUN 80,90,"ABCD"

### WRITE #

Instrucțiune

**Scop:** Scrie date către un fișier secvențial.

**Format:** WRITE #*nrfis*,*lista de expresii*

**Comentariu:** *nrfis* este numărul sub care a fost deschis pentru ieșire un fișier.

*lista de expresii* este o listă de expresii șir și/sau numerice, separate prin punct și virgulă sau virgulă.

La afișare, valorile expresiilor numerice sînt separate prin virgule, iar expresiile șir sînt delimitate de ghilimele. După ultimul termen al listei de expresii limbajul adaugă un "carriage return/line feed". Dacă lipsește lista de expresii, se scrie o linie liberă.

Instrucțiunea WRITE # este similară cu instrucțiunea PRINT #, singura diferență între cele două instrucțiuni fiind că WRITE # inserează virgule între termeni, iar șirurile sînt delimitate de ghilimele. De asemenea numerele pozitive nu sînt precedate de spații.

**Exemplu:** Considerînd că A = 1.23 și B\$ = "ABCD", instrucțiunea:

WRITE #1,A,B\$

va scrie următoarea imagine în fișier:

1.23,"ABCD"



# BIBLIOGRAFIE

- \*\*\* - IBM BASIC Handbook, General Programming Information, 1984
- \*\*\* - IBM, BASIC Reference, 1984
- Clarance B. Germain** - Programming the IBM PC & XT, Robert J. Brady Co., 1984
- L.J. Goldstein, M.D. Bowie** - Advanced BASIC and beyond for the IBM PC, Robert J. Brady, 1984
- G. Held** - IBM PC User's reference manual, Hayden Book, 1985
- D. Simon** - IBM BASIC from the ground up, Hayden Book, 1985
- H. Simson** - Serious programming in BASIC, TAB Books INC., 1986



# INDEX

- ABS funcție, 27
- ASC funcție, 27
- ATN funcție, 27
- AUTO comandă, 28
  
- BEEP instrucțiune, 29
- BLOAD comandă, 29
- BSAVE comandă, 29
  
- CALL instrucțiune, 30
- CDBL funcție, 31
- CHAIN instrucțiune, 31
- CHDIR comandă, 32
- CHR\$ funcție, 33
- CINT funcție, 33
- CIRCLE instrucțiune, 34
- CLEAR comandă, 35
- CLOSE instrucțiune, 35
- CLS instrucțiune, 36
- COLOR instrucțiune, 36
- COM(n) instrucțiune, 37
- COMMON instrucțiune, 38
- CONT comandă, 38
- COS funcție, 39
- CSNG funcție, 39
- CSRLIN variabilă, 39
- CVD funcție, 40
- CVI funcție, 40
- CVS funcție, 40
  
- DATA instrucțiune, 40
- DATE\$ instrucțiune, 40
- DATE\$ variabilă, 40
- DEF FN instrucțiune, 41
- DEF SEG instrucțiune, 42
- DEF USR instrucțiune, 43
- DEFtp instrucțiune, 43
  
- DELETE comandă, 44
- DIM instrucțiune, 45
- DRAW instrucțiune, 46
  
- EDIT comandă, 48
- END instrucțiune, 48
- ENVIRON instrucțiune, 48
- ENVIRON\$ funcție, 49
- EOF funcție, 50
- ERASE instrucțiune, 51
- ÉRDEV variabilă, 51
- ERDEV\$ variabilă, 51
- ERL variabilă, 52
- ERR variabilă, 52
- ERROR instrucțiune, 52
- EXP funcție, 53
  
- FIELD instrucțiune, 54
- FILES comandă, 54
- FIX funcție, 54
- FOR instrucțiune, 54
- FRE funcție, 56
  
- GET instrucțiune (fișlere), 57
- GET instrucțiune (grafică), 57
- GOSUB instrucțiune, 58
- GOTO instrucțiune, 59
  
- HEX\$ funcție, 60
  
- IF instrucțiune, 60
- INKEY\$ variabilă, 61
- INP funcție, 62
- INPUT # instrucțiune, 63
- INPUT instrucțiune, 62
- INPUT\$ funcție, 63
- INSTR funcție, 64



- INT funcție, 65  
 IOCTL instrucțiune, 65  
 IOCTL\$ funcție, 66
- KEY instrucțiune, 66  
 KEY(n) instrucțiune, 67  
 KILL comandă, 67
- LEFT\$ funcție, 67  
 LEN funcție, 68  
 LET instrucțiune, 68  
 LINE INPUT # instrucțiune, 70  
 LINE INPUT instrucțiune, 69  
 LINE instrucțiune, 69  
 LIST comandă, 70  
 LLIST comandă, 71  
 LOAD comandă, 71  
 LOC funcție, 71  
 LOCATE instrucțiune, 72  
 LOF funcție, 72  
 LOG funcție, 73  
 LPOS funcție, 73  
 LPRINT instrucțiune, 73  
 LPRINT USING instrucțiune, 73  
 LSET instrucțiune, 74
- MERGE comandă, 74  
 MID\$ funcție, 74  
 MID\$ instrucțiune, 74  
 MKD\$ funcție, 76  
 MKDIR comandă, 75  
 MKI\$ funcție, 76  
 MKS\$ funcție, 76
- NAME comandă, 76  
 NEW comandă, 77  
 NEXT instrucțiune, 54
- OCT\$ funcție, 77  
 ON COM(n) instrucțiune, 77  
 ON ERROR instrucțiune, 78  
 ON KEY instrucțiune, 79  
 ON PEN instrucțiune, 80  
 ON PLAY instrucțiune, 81  
 ON STRIG instrucțiune, 81
- ON TIMER instrucțiune, 82  
 ON-GOSUB instrucțiune, 78  
 ON-GOTO instrucțiune, 78  
 OPEN "COM..." instrucțiune, 83  
 OPEN instrucțiune, 83  
 OPTION BASE instrucțiune, 85  
 OUT instrucțiune, 85
- PAINT instrucțiune, 85  
 PEEK funcție, 87  
 PEN funcție, 87  
 PEN instrucțiune, 87  
 PLAY instrucțiune, 88  
 PLAY(n) funcție, 90  
 PMAP funcție, 90  
 POINT funcție, 91  
 POKE instrucțiune, 92  
 POS funcție, 92  
 PRESET instrucțiune, 96  
 PRINT # instrucțiune, 96  
 PRINT # USING instrucțiune, 96  
 PRINT instrucțiune, 92  
 PRINT USING instrucțiune, 93  
 PSET instrucțiune, 96  
 PUT instrucțiune (grafică), 97  
 PUT instrucțiune (fișiere), 97
- RANDOMIZE instrucțiune, 99  
 READ instrucțiune, 101  
 REM instrucțiune, 101  
 RENUM comandă, 102  
 RESET comandă, 102  
 RESTORE instrucțiune, 102  
 RESUME instrucțiune, 103  
 RETURN instrucțiune, 58, 103  
 RIGHT\$ funcție, 104  
 RMDIR comandă, 104  
 RND funcție, 104  
 RSET instrucțiune, 74  
 RUN comandă, 105
- SAVE comandă, 106  
 SCREEN funcție, 106  
 SCREEN instrucțiune, 107  
 SGN funcție, 108

SHELL instrucțiune, 108  
SIN funcție, 109  
SOUND instrucțiune, 110  
SPACE\$ instrucțiune, 111  
SPC funcție, 112  
SQR funcție, 112  
STICK funcție, 112  
STOP instrucțiune, 113  
STR\$ funcție, 114  
STRIG funcție, 114  
STRIG instrucțiune, 114  
STRIG(n) instrucțiune, 115  
STRING\$ funcție, 115  
SWAP instrucțiune, 115  
SYSTEM comandă, 116  
  
TAB funcție, 116  
TAN funcție, 116  
TIME instrucțiune, 117  
TIME variabilă, 117

TIMER funcție, 117  
TROFF comandă, 118  
TRON comandă, 118

USR funcție, 119

VAL funcție, 119  
VARPTR funcție, 119  
VARPTR\$ funcție, 120  
VIEW instrucțiune, 120

WAIT instrucțiune, 122  
WEND instrucțiune, 122  
WHILE instrucțiune, 122  
WIDTH instrucțiune, 123  
WINDOW instrucțiune, 124  
WRITE # instrucțiune, 128  
WRITE instrucțiune, 127

#### PROGRAME DE INSTRUIRE

Redactor: ION MURAI

Tehnoredactor: MARIANA DAMASCHINOPU

Coli tipar: 4 1/2

Num de tipar: 2500 (1982)

Tiparul executat sub Cod, 07113 la

Intreprinderea Poligrafică Nouă

Republica Socialistă România

#### PROGRAME DE INSTRUIRE

REDACTOR: ION MURAI

TEHNOREDACTOR: MARIANA DAMASCHINOPU

COLI TIPAR: 4 1/2

NUM DE TIPAR: 2500 (1982)

TIPARUL EXECUTAT SUB COD, 07113 LA

INTREPRINDEREA POLIGRAFICA NOUA

REPUBLICA SOCIALISTA ROMANIA





PROGRAME DE INSTRUIRE

Redactor: ION MIHAI

Tehnoredactor: MARILENA DAMASCHINOPOL

---

Coli tipar: 8  $\frac{1}{2}$

Bun de tipar: 25.09.1989

---

Tiparul executat sub Cda. 019113 la

Întreprinderea Poligrafică Sibiu

Republica Socialistă România

PROGRAME DE INSTRUIRE

REDACTOR: ION MIHAI

TEHNOREDACTOR: MARILENA DAMASCHINOPOL

---

COLI TIPAR: 8  $\frac{1}{2}$

BUN DE TIPAR: 25.09.1989

---

TIPARUL EXECUTAT SUB CDA. 019113 LA

ÎNTEPRINDEREA POLIGRAFICĂ SIBIU

REPUBLICA SOCIALISTA ROMÂNIA











**EDITURA  
ȘTIINȚIFICĂ ȘI ENCICLOPEDICĂ**

**ISBN 973-29-0108-X**

**Lei 11,50**