MANUAL BASIC

RED

Tehnical Book

Editura TM

By ALPHA Ltd. @ 1991

MANUAL BASIC

Tehnical Book

Editura TM

By **ALPHA Ltd. @ 1991**

MATRIX BASIC

Technical Book

Editors TM

CUPRINS

LECTIA nr.1

Ai in fata un calculator. Deocamdata el iti apare ca un mic monstru cu multe butoane si multe fire, dar ai sa vezi ca dupa cîtva timp va fi un mielusel blînd sub mîinile tale.

Mai intii,fiecare isi pune intrebarea:"Ce poti face cu un calculator?" Ei bine, poti face aproape orice,de la rezolvarea temelor la matematica, la jocuri si muzica, cu conditia sa il programezi asa cum trebuie. Totul pleaca de la faptul ca un calculator nu intelege limbajul pe care noi il folosim in mod obisnuit (ex. limba romana, franceza sau engleza). De aceea, ca sa poti discuta cu el, trebuie sa inveti "limba" lui, care este **limbajul BASIC**. Asa cum intr-o conversatie cu un coleg folosesti fraze formate din cuvinte, cind stai de vorba cu calculatorul folosesti programe formate din instructiuni.

Lectiile de **BASIC** iti vor spune exact ce poti face si ce nu poti face intr-un program. Vei gasi de asemenea exercitii si exemple pentru fiecare capitol. Nu trece peste ele! Asa vei invata unele lucruri care altfel ar lua ore intregi de explicatii plicticoase.

Foloseste intotdeauna calculatorul! Daca iti pui intrebarea: "Ce va face daca ii spun asta sau asta?" raspunsul e foarte simplu: spune-i si vei vedea. Iar acum partea mai grea - incearca sa-ti explici de ce a facut asa si nu altfel?

Daca intr-un exemplu se spune sa faci intr-un anume fel, pune-ti intrebarea:"Oare nu se poate si altfel?" Cu cit vei scrie mai multe programe ale tale, cu atit vei intelege mai bine si mai repede calculatorul.

Deci, sa pornim la drum! Scrieti orice, nu va fie frica, calculatorul nu se strica!

Tastatura

Daca te uiti la un calculator, primul lucru pe care il vezi este o multime de butoane (taste) aliniate frumos si pe care sint scrise litere, cuvinte si alte semne care nu par sa insemne ceva intr-o limba paminteana. Si totusi, acestea sint "cuvintele" pe care le intelege calculatorul.

Nu trebuie sa te sperii ca sint asa de multe; intr-un timp scurt le vom invata pe toate.

Fiecare tasta are mai multe roluri. In afara de simboluri (litere, cifre, etc.), pentru fiecare tasta mai corespund si asa numitele token (cuvinte-cheie, nume de functii, etc.).

Pentru a alege semnificatia dorita a fiecarei taste, se folosesc niste taste speciale: CAPS SHIFT si SYMBOL SHIFT. De asemenea vom fi atenti la modul de lucru al calculatorului, care este indicat de cursor - o litera pilpiitoare aflata pe ecran in locul unde urmeaza sa scriem ceva.

Modurile de lucru sint:

K (keywords - cuvinte cheie) - calculatorul asteapta un numar de linie sau o comanda.

L (letters - litere) - se asteapta o litera (litere mici).

Daca in aceste doua moduri se apasa o tasta simultan cu CAPS SHIFT sau SYMBOL SHIFT atunci se va afisa un caracter special sau o functie.

C (capitals - litere mari) - este o varianta a modului L care afiseaza litere mari.

E (extended mode - mod extensie) - se obtin alte functii din cele inscrise pe fiecare tasta. Se intra in modul E prin apasarea simultana a tastelor CAPS SHIFT si SHIFT LOCK.

G (graphics - caractere grafice) - se obtin diferite mozaicuri grafice inscrise pe tastele numerice. Se intra in modul grafic apasind CAPS SHIFT si 9 si se iese apasind 9 cu sau fara CAPS SHIFT.

Daca un program a fost introdus, acesta se va afisa pe ecran in ordinea crescatoare a numarului liniei. Una dintre linii este numita linia_curenta si este aratata de semnul >. Aceasta linie poate fi editata (EDIT) si apare in partea de jos a ecranului putind fi modificata. Pentru a ajunge cu cursorul in locul unde vrem sa facem modificarea se folosesc sagetile (CAPS SHIFT +5 sau 8). Pentru a schimba linia curenta se folosesc sagetile verticale (CAPS SHIFT +6 sau 7). O linie intreaga se poate sterge apasind EDIT si apoi ENTER.

Ecranul_TV

Pe ecranul TV se afiseaza programele si rezultatele obtinute. Acesta are 24 de linii cu cite 32 de caractere pe fiecare linie. El este impartit in doua parti. Partea de sus, care contine 22 de linii, afiseaza lista instructiunilor programului si rezultatele, iar partea de jos (2 linii) afiseaza linia care se introduce sau se editeaza. Cind se apasa ENTER linia de jos isi ocupa locul in partea de sus a ecranului.

LECTIA nr. 2

Programe, linii de program, editarea programelor.

Citeva instructiuni.

Limbajul **BASIC** admite doua tipuri de instructiuni: numerotate si nenumerotate. Cele nenumerotate sint imediat dupa apasarea tastei **ENTER**. Instructiunile numerotate sint stocate ca linii de program. Numerele de linii trebuie sa fie intregi, intre 1 si 9999. Listarea si executia unui program se fac prin ordonarea programului dupa numarul de linie. De aceea este indicat ca la scrierea unui program sa se pastreze spatii intre numerele liniilor consecutive, pentru a putea introduce la nevoie linii noi intre cele vechi. O linie de program poate contine mai multe instructiuni separate intre ele prin : (doua puncte).

In continuare vor fi prezentate exemple de programe in care apar citeva din instructiunile **BASIC**, punindu-se accentul pe facilitatile de editare ale sistemului.

Exemplul 1: Sa se tipareasca suma a doua numere.

Dupa ce vei introduce liniile:

```
20 PRINT a

10 LET a=10
```

vei constata ca programul se listeaza pe ecran ordonat dupa numarul de linie.

Pina acum ai introdus primul numar. Ca sa-l introduci pe al doilea trebuie sa scrii:

```
15 LET b=15
```

Pentru tiparirea sumei, este necesar ca ultima linie sa fie:

```
20 PRINT a+b
```

S-ar putea rescrie linia, dar este mai usor sa se foloseasca **EDIT**. Pentru aceasta se coboara cursorul > de la linia 15 la linia 20, apasind tasta ↓. In continuare apesi **EDIT** si linia 20 va apare copiata in partea de jos a ecranului. Vei apasa -> pina cind cursorul ajunge la sfirsitul liniei si vei scrie +b. Apasind acum **ENTER** vechea linie 20 va fi inlocuita cu cea noua.

Se executa programul cu **RUN** si **ENTER**; ca urmare, pe ecran va apare rezultatul. Pentru o noua rulare se apasa din nou **RUN** si **ENTER**, rezultatul fiind identic.

> Dupa executarea programului, ultima valoare a fiecarei variabile ramine memorata si poate fi citita cu o instructiune **PRINT** nenumerotata (acest lucru este deosebit de util la depanarea programelor).

Daca ai scris o linie si vrei sa o stergi, poti proceda in doua feluri:

-apesi **DELETE** pina o stergi complet;

-apesi **EDIT**; pe ultima linie va apare o copie a liniei curente. Cu **ENTER** acum, linia curenta ramine nemodificata, iar linia de jos este stearsa.

Presupunem ca din greseala ai introdus linia:

```
12 LET b=8
```

Ea va putea fi stearsa cu:

```
12          (cu ENTER)
```

Surpriza! A disparut cursorul. Daca apesi ↑ cursorul va apare la linia 10, iar daca apesi ↓ va apare la linia 15. Scrie:

```
12          (cu ENTER)
```

Din nou cursorul "s-a ascuns" intre liniile 10 si 15. Daca apesi acum **EDIT**, linia 15 va apare in zona_de_editare. Cind cursorul este ascuns intre doua linii, **EDIT** aduce in josul ecranului linia cu numarul imediat urmator. Scrie acum:

```
30          (cu ENTER)
```

De data aceasta cursorul este ascuns dupa sfirsitul programului.

Cu comanda: **LIST 15** pe ecran se obtine:

```
15>LET b=15

20 PRINT a+b
```

Instructiunea **LIST 15** produce listarea incepind cu linia 15 si pune cursorul pe linia 15. Pentru un program foarte lung, **LIST** va fi o metoda mai usoara de mutare a cursorului decit ↑ sau ↓.

Aceasta arata o alta utilitate a numerelor de linie: ele actioneaza ca nume ale liniilor de program si se pot face referiri la ele la fel ca la numele de variabile. **LIST** fara numar face listarea de la inceputul programului.

O alta comanda este **NEW** care sterge programul si variabilele din memoria calculatorului.

Exemplul 2: Sa se scrie un program care transforma temperatura din grade Fahrenheit in grade Celsius.

```
10 REM conversia temperaturii

20 PRINT "grade F","grade C"

30 PRINT

40 INPUT "introduceti gradele F.",f

50 PRINT f,(f-32)*5/9

60 GO TO 40
```

Este necesar sa fie introdusa fiecare litera pentru textul "conversia temperaturii" in linia 10. In linia 60 se obtine **GO TO** apasind tasta G (desi contine spatiu **GO TO** este un singur cuvint cheie). Rulind programul, se va vedea pe ecran capul de tabel tiparit de linia 20. Linia 10 este o linie de comentariu care ne ajuta sa stim ce face programul (atunci cind citim listingul) dar este ignorata de calculator. Comanda **INPUT** din linia 40 asteapta sa fie introdusa o valoare pentru variabila f; se introduce un numar si se apasa **ENTER**. Calculatorul afiseaza rezultatul dar nu se opreste din rulare, ci asteapta alt numar (datorita saltului din linia 60). Programul se poate opri apasind **STOP** in momentul in care pe ecran apare mesajul "introduceti gradele F", adica in instructiunea **INPUT** din linia 40.

Calculatorul raspunde cu mesajul:

```
H    STOP in INPUT 40:1
```

care precizeaza de ce si unde s-a oprit programul (prima instructiune din linia 40). Pentru a continua programul se apasa **CONTINUE** si calculatorul asteapta alt numar. Comanda **CONTINUE** determina rularea programului de la linia la care se oprise (linia 40).

Scrie acum:

```
60 GO TO 31
```

In executie, aceasta varianta se comporta exact ca si precedenta. Daca numarul liniei intr-o comanda **GO TO** se refera la o linie inexistenta, atunci se sare la linia imediat urmatoare. Acest lucru e valabil si pentru **RUN** (de fapt **RUN** inseamna **RUN 0**).

Daca tiparim numere pina cind se umple ecranul, calculatorul va muta intreaga parte de sus a ecranului cu o linie pentru a face loc, pierzind astfel capul de tabel. Aceasta mutare a ecranului se numeste **scrolling**. Cind am terminat, programul se opreste cu **STOP** si **ENTER**; daca mai apasam o data **ENTER** va apare

lista de instructiuni.

Sa ne uitam putin la comanda **PRINT** din linia 50. Virgula utilizata aici face ca ce urmeaza dupa ea sa fie tiparit de la mijlocul liniei. Sa incercam acum si cu alte semne in loc de virgula. Daca punem punct si virgula "**;**" atunci al doilea sir de caractere se va tipari imediat dupa primul. Daca punem apostrof "**'**" atunci se sare la linie noua (ca si cum ar fi doua **PRINT**-uri separate). Orice **PRINT** face saltul la linie noua cu exceptia cazului cind am avut un **PRINT** anterior terminat cu "**,**" sau "**;**".

Pentru exemplificare incearca pe rind in locul liniei 50:

```
50 PRINT f, ...

50 PRINT f; ...

50 PRINT f  ...

50 PRINT f' ...
```

Se constata ca varianta cu "**,**" imparte totul in doua coloane, cea cu "**;**" scrie totul compact, iar cele fara semn si cu "**'**" scriu un numar pe o linie.

In memorie pot exista in acelasi timp mai multe programe, cu conditia ca numerele de linie sa nu se suprapuna.

Exemplul 3: Un program politicos care te saluta.

```
100 INPUT n$

110 PRINT "Salut ";n$ " !"

120 GO TO 100
```

Acest program poate exista impreuna cu cel de la exemplul 2 fiindca unul are numerele de linie intre 10...60 iar celalalt intre 100...120. Pentru lansarea_in_executie se foloseste comanda **RUN 100**. Comanda **RUN** sterge ecranul si toate variabilele, iar dupa aceea executa sirul de instructiuni. Daca nu doresti stergerea ecranului si a variabilelor poti folosi pentru lansare comanda **GO TO 100**.

La executia programului din ex.3 pe ecran apare "L" care arata ca se asteapta un sir de caractere. Sistemul admite ca o instructiune **INPUT** sa se comporte similar cu o instructiune de atribuire dar numai pentru cazul citirii unei variabile de tip sir_de_caractere (incadrata intre ghilimele).

De exemplu, daca la prima solicitare a programului din ex.3 se introduce "ANA", valoarea variabilei n$ va deveni **n$="ANA"**. La urmatoarea citire scriem "MARIA" si deci **n$="MARIA"**. Daca la urmatoarea cerere vom scrie n$, atunci valoarea vechii variabile n$ se asociaza noii variabile n$ (ca o instructiune **LET n$=n$**) si

deci vom avea n$="MARIA", adica instructiunea 110 va tipari:

Salut MARIA !

Uneori, din greseala, se scrie un program care ruleaza la infinit:

```
200 GO TO 200
RUN 200
```

Pentru oprirea executiei se apasa BREAK (CAPS SHIFT + SPACE) si calculatorul va raspunde cu mesajul:

L BREAK into program, 200:1

La sfirsitul fiecarei instructiuni calculatorul verifica daca aceste taste sint apasate si daca da, atunci opreste executia.

Comanda BREAK poate fi utilizata de asemenea cind sint conectate casetofonul sau imprimanta si se asteapta ca acestea sa efectueze o comanda. Mesajul in acest caz este:

D BREAK - CONT repeats

Comanda CONTINUE in cazul lucrului cu casetofonul sau imprimanta repeta instructiunea unde programul a fost oprit.

Listingurile automate sint acelea care apar nu dupa o comanda LIST ci dupa introducerea unei linii noi. De retinut ca linia curenta (cu >) apare intotdeauna pe ecran si in general pe pozitie centrala.

Rezumatul lectiei 2

-Programe, linii de program

-Editarea programelor cu EDIT, ↑,↓,<-,->, DELETE

-RUN, LIST, GO TO, CONTINUE, INPUT, NEW, REM, PRINT, STOP IN INPUT DATA, BREAK

LECTIA nr.3

Decizii

Toate programele pe care le-am vazut pina acum treceau pe rind din instructiune in instructiune si apoi din nou de la inceput. Acest lucru nu este prea folositor in practica. Vom dori ca prietenul nostru, calculatorul, sa ia decizii si sa actioneze dupa cum a hotarit. Instructiunea folosita are forma ...IF (daca) ceva e adevarat sau nu e adevarat THEN (atunci) fa altceva.

De exemplu, sterge programele anterioare cu NEW si scrie-l pe urmatorul (e un program pentru doi jucatori).

```
10 REM Ghiceste numarul
20 INPUT a: CLS
30 INPUT "Ghiceste numarul",b
40 IF b=a THEN PRINT "Ai ghicit !":STOP
50 IF b<a THEN PRINT "Prea mic, mai incearca"
60 IF B>A THEN PRINT "Prea mare, mai incearca"
70 GO TO 30
```

Poti vedea ca instructiunea IF are forma:

```
IF conditie THEN ...
```

unde "..." reprezinta o secventa de comenzi, separate in mod obisnuit de doua puncte ":". Conditia este ceva care poate fi adevarat sau fals. Daca este adevarat atunci se executa comenzile din linie aflate dupa THEN, iar in caz contrar acestea sint sarite si se executa instructiunea urmatoare.

Cele mai simple conditii compara doua numere sau doua string-uri (coduri de caractere alfanumerice). Se poate vedea daca doua numere sint egale sau care din ele este mai mare. De asemenea se poate verifica daca doua string-uri sint identice sau care din ele este primul in ordine alfabetica. Pentru asta vom utiliza relatiile =, <, >, <=, >= si <>.

= inseamna "egal". Este acelasi simbol ca si = din instructiunea LET, dar are un inteles diferit.

< (SYMBOL SHIFT + R) inseamna "este mai mic decit" si deci 1<2; -2<-1; -3<1 sint adevarate, iar 1<0; 0<-2 sint false.

Linia 40 compara a cu b. Daca sint egale, atunci programul este oprit de STOP cu mesajul 9 STOP statement, 30:3.

Linia 50 determina daca b este mai mic decit a, iar linia 60 daca b este mai mare decit a. Daca una din aceste conditii este adevarata, atunci se va tipari unul dintre mesaje si se va merge mai departe la linia 70 care trimite executia inapoi la 30. Comanda CLS din linia 20 impiedica partenerul sa vada numarul pe care l-ai introdus.

> (SYMBOL SHIFT + T) inseamna "este mai mare decit", deci exact invers decit <. Poti tine minte usor cum se folosesc daca te gindesti ca virful sagetii arata numarul care ar fi mai mic.

<= (SYMBOL SHIFT + Q - nu se tasteaza < urmat de =) inseamna "este mai mic sau egal cu", adica la fel cu < dar este adevarata si pentrù egalitate. Deci 2<=2 este adevarat, dar 2<2 e fals.

>= (SYMBOL SHIFT + E) inseamna "este mai mare sau egal cu", adica similar cu >.

<> (SYMBOL SHIFT + W) inseamna "nu este egal" sau "e diferit de" si are deci intelesul opus lui =.

Exercitii

1. Incearca programul:

```
10 PRINT "x": STOP: PRINT :"y"
```

Cind ruleaza, el va afisa x si se va opri cu mesajul: 9 STOP statement, 10:2. Acum scrie: CONTINUE. Te-ai fi asteptat poate ca sa sara inapoi la STOP deoarece in mod obisnuit CONTINUE repeta instructiunea din mesaj. Aici insa, acest lucru ar fi destul de neplacut fiindca nu am mai ajunge sa tiparim y. De aceea, in cazul unui mesaj 9, CONTINUE sare la instructiunea de dupa comanda STOP si deci calculatorul va tipari valoarea lui y si va ajunge la sfirsitul programului.

Bucle de program (iteratii)

Sa presupunem ca vrei sa introduci cinci numere si sa le aduni. O posibilitate ar fi sa scrii:

```
10 LET total=0
20 INPUT a
```

```
30 LET total=total+a

40 INPUT a

50 LET total=total+a

60 INPUT a

70 LET total=total+a

80 INPUT a

90 LET total=total+a

100 INPUT a

110 LET total=total+a

120 PRINT total
```

Ei, ce zici? Pare ca si cum ai avea de ispasit o pedeapsa. Si inchipuie-ti ce-ar fi daca ai vrea sa aduni o suta de numere !

Mult mai bine ar fi daca am avea o variabila care sa numere pina la cinci adunari de acelasi fel si apoi sa opreasca programul. Deci acum vei scrie:

```
10 LET total=0

20 LET numar=1

30 INPUT a

40 REM numar = cite numere s-au introdus pina acum

50 LET total=total+a

60 LET numar=numar+1

70 IF numar <=5 THEN GO TO 30

80 PRINT total
```

Vedem aici ce usor putem sa schimbam programul pentru zece sau o suta de numere, doar modificind linia 70.

Acest fel de numarare este atit de des folosit incit s-au prevazut niste instructiuni speciale pentru a-l utiliza mai usor: instructiunile FOR si NEXT. Ele sint folosite intotdeauna impreuna. Programul va arata in felul urmator:

```
10 LET total=0

20 FOR n=1 TO 5
```

```
30 INPUT a
40 REM n=cite numere s-au introdus pina acum
50 LET total=total+a
60 NEXT n
80 PRINT total
```

(Ca sa obtinem acest program din cel precedent trebuie doar editate liniile 20, 40, 60 si 70. TO este SYMBOL SHIFT + F).

Trebuie sa fi atent ca am schimbat variabila_de_control numar cu n, deoarece intr-o bucla FOR - NEXT aceasta trebuie sa aiba o singura litera.

Efectul acestui program este ca c ia valorile 1 (valoarea initiala), 2, 3, 4 si 5 (valoarea limita) si de fiecare data se executa liniile 30, 40 si 50. Dupa ultima valoare a lui c se executa linia 80.

O subtilitate este ca variabila de control poate creste la fiecare pas nu numai cu 1, ci cu orice alta valoare, utilizind comanda STEP in linia FOR. Forma generala a instructiunii FOR este:

```
FOR var.de_control=val.initiala TO val.limita STEP pasul
```

unde variabila de control este o litera, iar valoarea initiala, valoarea finala si pasul sint lucruri pe care calculatorul le poate intelege ca numere (numere, sume, produse, variabile, etc...). Daca vei rescrie linia 20:

```
20 FOR n=1 TO 5 STEP 3/2
```

atunci n va lua valorile 1; 2,5; si 4. Deci nu trebuie sa te restringi la numere intregi si nici nu e nevoie ca ultima valoare a variabilei de control sa fie exact valoarea limita. Raminem in bucla atit timp cit variabila de control e mai mica sau egala cu limita. Sa incercam acum un program care tipareste numerele de la 10 la 1 (in ordine descrescatoare):

```
10 FOR n=10 TO 1 STEP -1
20 PRINT n
30 NEXT n
```

In acest caz regula se modifica: daca pasul e negativ atunci programul ramine in bucla atit timp cit variabila de control este mai mare sau egala cu limita.

Trebuie sa fi foarte atent cind rulezi doua bucle FOR - NEXT in acelasi program. Sa vedem un program care tipareste setul

jocului de domino:

```
10 FOR m=0 TO 6
20 FOR n=0 TO m
30 PRINT m;":";n;" ";
40 NEXT n
50 PRINT
60 NEXT m
```

bucla n — bucla m

Aici bucla n este inclusa complet in bucla m. Daca cele doua bucle "se incaleca" programul va rula incorect. Buclele FOR - NEXT trebuie sa fie sau complet incluse una in cealalta sau complet separate.

```
5 REM acest program e gresit
10 FOR m=0 TO 6
20 FOR n=0 TO m
30 PRINT m;":";n;" ";
40 NEXT m
50 PRINT
60 NEXT n
```

bucla m — bucla n

Un alt lucru ce trebuie evitat este saltul din afara in mijlocul unei bucle, deoarece variabila de control este initializata in instructiunea FOR. Vei primi probabil un mesaj de eroare de forma NEXT without FOR sau variable not found.

Bineinteles ca se poate folosi FOR si NEXT ca o comanda directa:

```
FOR m=0 TO 10: PRINT m: NEXT m
```

Poti folosi acest lucru pentru a ocoli restrictia ca nu poti intra (GO TO) in interiorul unei comenzi directe - care nu are numar de linie. De exemplu:

```
FOR m=0 TO 1 STEP 0: INPUT a: PRINT a: NEXT m
```

Pasul zero face comanda sa se repete la infinit. Totusi aceasta metoda nu este foarte indicata deoarece, in cazul unei erori, comanda se pierde si va trebui scrisa din nou.

Exercitii

1. O variabila de control nu are numai nume si valoare, ci si o valoare initiala, pas si valoare limita, care sint diponibile dupa executia unei instructiuni FOR, si care sint folosite de instructiunea NEXT.

2. Ruleaza al treilea program si apoi scrie:

PRINT c

De ce raspunsul e 6 si nu 5? Ce se va intimpla daca vei pune STEP 2 in linia 20?

3. Schimba al treilea program astfel incit in loc sa adune automat cinci numere, el sa te intrebe cite numere vrei sa aduni. Ce se va intimpla daca vei introduce 0 (sa nu adune nici un numar) si de ce?

4. In linia 10 a programului patru schimba 10 cu 100 si ruleaza-l asa. El va tipari numerele de la 100 la 79 dupa care va intreba scroll? in josul ecranului, pentru a putea citi aceste numere. Daca apesi n, STOP sau BREAK, programul se va opri cu mesajul

D BREAK - CONT repeats. Daca apesi orice alta tasta, atunci se vor tipari alte 22 de numere si te va intreba din nou.

5. Sterge linia 30 din al patrulea program. La rulare se va tipari primul numar si programul se va opri cu mesajul 0 OK. Daca acum vei scrie NEXT n, va mai face inca o bucla si va tipari urmatorul numar.

Rezumatul_lectiei_3

-Decizii,IF, STOP, =, <, >, <=, >=, <>

-Bucle de program,FOR, NEXT, TO, STEP

LECTIA nr.4

Subrutine

Se intimpla citeodata ca diferite parti ale unui program sa faca acelasi lucru. Te vei trezi astfel ca scrii aceleasi linii de doua sau mai multe ori, treaba care bineinteles ca nu e necesara. Rezolvarea e simpla: scrii aceste linii o singura data, insa sub forma unei subrutine pe care o vei apela ori de cite ori vei avea nevoie de ea. Pentru aceasta vom folosi instructiunile **GO SUB** (GO to SUBroutine) si **RETURN**.

Deci apelarea unei subrutine se face cu:

```
GO SUB n
```

unde n este numarul primei linii a subrutinei. Efectul este asemanator cu GO TO n cu diferenta ca in cazul lui **GO SUB** calculatorul tine minte locul de unde s-a facut saltul, pentru a sti apoi unde sa se intoarca dupa executarea subrutinei. El face asta punind numarul liniei si al instructiei din interiorul liniei (acestea constituie adresa_de_reintoarcere) in virful unei asa numite stive (**GO SUB** stack).

La sfirsitul subrutinei trebuie pusa instructiunea

```
RETURN
```

care ia adresa de reintoarcere din virful stivei si reincepe executia programului de la prima instructiune de dupa **GO SUB**.

Sa vedem, de exemplu, cum ar arata programul pentru ghicirea unui numar daca folosim subrutine:

```
10 REM "Joc de ghicit in alta varianta"

20 INPUT a: CLS

30 INPUT "Ghiceste numarul ",b

40 IF a=b THEN PRINT "Ai ghicit !": STOP

50 IF a<b THEN GO SUB 100

60 IF a>b THEN GO SUB 100

70 GO TO 30

100 PRINT "Mai incearca o data"
```

```
110 RETURN
```

Instructiunea GO TO din linia 70 este foarte importanta pentru ca altfel programul ar intra in subrutina direct si ar aparea o eroare 7 RETURN without GO SUB.

Iata acum un alt exemplu:

```
100 LET x=10
110 GO SUB 500
120 PRINT s
130 LET x=x+4
140 GO SUB 500
150 PRINT s
160 LET x=x+2
170 GO SUB 500
180 PRINT s
190 STOP
500 LET s=0
510 FOR y=1 TO x
520 LET s=s+y
530 NEXT y
540 RETURN
```

Poti sa-ti dai seama ce face acest program? Subrutina incepe la linia 500.

In limbajul BASIC o subrutina poate chema alta subrutina sau chiar se poate chema pe ea insasi (in acest caz este recursiva).

READ, DATA, RESTORE

In unele programe am vazut ca putem introduce informatii (date) in calculator cu ajutorul instructiunii . Acest mod este insa obositor daca avem de introdus multe date. Poti lucra mult mai repede daca folosesti instructiunile READ, DATA si RESTORE. De exemplu:

```
10 READ a,b,c
```

```
20 PRINT a,b,c

30 DATA 10,20,30

40 STOP
```

Instructiunea **READ**, urmata de o lista de variabile separate intre ele prin virgule, actioneaza ca un **INPUT** cu diferenta ca in loc sa-ti ceara datele de la tastatura le ia dintr-o lista corespunzatoare de valori din instructiunea **DATA**. Orice instructiune **DATA** este o lista de expresii (numerice sau alfanumerice) separate cu virgule. Instructiunile **DATA** pot fi plasate oriunde in program, ele formind de fapt (daca sint mai multe) o lista lunga ce va fi citita de instructiuni **READ**. Este pierdere de vreme sa folosesti **DATA** ca o comanda directa fiindca o instructiune **READ** care ar urma nu mai gaseste valorile. Instructiunea **DATA** trebuie sa fie inclusa intr-un program.

Sa revenim acum la programul pe care l-ai scris. Ca sa vezi ordinea in care decurg lucrurile, inlocuieste linia 20 cu:

```
20 PRINT b,c,a
```

Informatiile cuprinse in **DATA** pot face parte dintr-o bucla **FOR...NEXT**:

```
10 FOR n=1 TO 6

20 READ D

30 DATA 2,4,6,8,10,12

40 PRINT D

50 NEXT n

60 STOP
```

Cind programul ruleaza vei vedea cum **READ** se plimba peste lista de valori din DATA. Instructiunea **DATA** poate contine de asemenea si variabile alfanumerice (string-uri):

```
10 READ d$

20 PRINT "Data este ",d$

30 DATA "5 iunie 1993"

40 STOP
```

Pina acum am vazut modul cel mai simplu de extragere a

datelor dintr-o instructiune **DATA**. Se poate face un salt in lista de date cu ajutorul instructiunii **RESTORE**. De exemplu:

```
10 READ a,b
20 PRINT a,b
30 RESTORE 10
40 READ x,y,z
50 PRINT x,y,z
60 DATA 1,2,3
70 STOP
```

In acest program, in linia 10 se asigneaza a=1 si b=2. Instructiunea RESTORE 10 reseteaza variabilele si face ca x,y si z sa fie citite incepind cu prima valoare din lista DATA. Ruleaza din nou programul, dar fara linia 30, si vezi ce se intimpla.

Daca folosesti RESTORE fara sa fie urmat de numar de linie, atunci se considera ca si cum ar fi prima linie din program.

Rezumatul_lectiei_nr.4

-Subrutine, GO SUB, RETURN

-READ, DATA, RESTORE

LECTIA nr.5

Expresii

Am vazut pina acum unele moduri in care calculatorul stie sa lucreze cu numere. El poate sa faca si cele patru operatii aritmetice +, -, * si / (* este inmultire si / este impartire) cu numere sau variabile date prin numele lor. De exemplu:

```
LET tax=sum*15/100
```

arata cum pot fi combinate aceste operatii. O astfel de combinatie, ca sum*15/100, se numeste expresie si este o forma foarte scurta prin care se poate spune calculatorului sa faca anumite calcule matematice, unul dupa celalalt. In exemplul de mai sus, i se spune calculatorului "cauta valoarea variabilei numita 'sum', inmulteste-o cu 15 si apoi imparte cu 100". Asa cum ai invatat la aritmetica, inmultirea si impartirea se fac inaintea adunarii si scaderii (au prioritate mai mare). Intre ele, * si / au aceeasi prioritate si se vor efectua in odinea citirii lor (de la stinga spre dreapta), la fel si + si -. La fel cum noi invatam care este ordinea unor operatii fata de altele, si calculatorul trebuie sa stie acest lucru. Pentru asta, el are cite un numar intre 1 si 16 care reprezinta ordinea de prioritate a fiecarei operatii; * si / au prioritatea 8, iar + si - au prioritatea 6.

Aceasta ordine a operatiilor este rigida, dar ea poate fi ocolita folosind parantezele. Expresiile sint utilizate foarte des, deoarece ori de cite ori calculatorul asteapta un numar, ii poti da o expresie a carei valoare si-o calculeaza singur (sint si citeva exceptii).

Exista citeva reguli pentru a da nume unor variabile. Astfel, o variabila alfanumerica (string) va avea ca nume o singura litera urmata de $, iar numele unei variabile de control dintr-o bucla FOR...NEXT va fi o singura litera. Variabilele numerice obisnuite pot avea oricite litere si pot contine chiar si cifre, atita vreme cit incep cu o litera. Poti pune spatii in interiorul numelui (dar acestea nu vor fi luate in considerare); de asemenea nu se face diferenta intre nume scrise cu litere mari sau mici.

Iata citeva exemple de nume de variabile:

```
x
t42
acest nume e atit de lung incit nu il pot scrie fara greseala
acum sintem sase     (aceste ultime doua nume se considera ca
aCUmsINTei1SASE       sint identice - este aceeasi variabila)
```

Iata acum citeva exemple de nume care nu sint corecte:

2001	(incepe cu o cifra)
3 ursuleti	(incepe cu o cifra)
M*A*S*H	(* nu este litera sau cifra)
Scufita-Rosie	(- nu este litera sau cifra)

Numerele pot fi scrise sub forma de mantisa si exponent:

```
PRINT 2.34e0

PRINT 2.34e1

PRINT 2.34e2    si asa mai departe pina la

PRINT 2.34e15
```

Vei vedea ca dupa un timp si calculatorul incepe sa scrie sub aceasta forma, datorita faptului ca el nu stie sa scrie un numar cu mai mult de 14 cifre. Incearca acum:

```
PRINT 2.34e-1

PRINT 2.34e-2   si asa mai departe.
```

Comanda PRINT afiseaza numai 8 cifre semnificative ale unui numar:

```
PRINT 4294867295, 4294867595-429e7
```

Deci calculatorul tine minte cifrele lui 4294867295 chiar daca nu le poate afisa pe toate deodata. Pentru a retine numerele in memorie, calculatorul memoreaza separat cifrele numarului (mantisa) si separat locul unde se pune punctul zecimal (exponentul). Aceasta reprezentare se numeste in_virgula_flotanta si nu este o reprezentare riguros exacta, chiar pentru numere intregi. De exemplu:

```
PRINT 1e10+1-1e10, 1e10-1e10+1
```

Numerele sint retinute cu o precizie de aproximativ noua cifre si jumatate, ceea ce face ca 1e10 sa apara egal cu 1e10+1.

Un alt exemplu de aproximare:

```
PRINT 5e9+1-5e9
```

Aici inexactitatea de reprezentare a lui 5e9 este cam de o unitate, iar adunarea cu 1 duce la rotunjire_in_sus pina la 2. Numerele 5e9+1 si 5e9+2 ii apar calculatorului ca fiind egale. Cel mai mare numar intreg pe care calculatorul il poate retine cu exactitate este 4294967295 (2 la puterea 32 minus 1).

String-ul "" fara nici un caracter se numeste nul sau gol si nu trebuie confundat cu un string ce contine numai spatii.

Incearca acum sa scrii:

```
PRINT "Ai terminat "Winnetou" sau nu?"
```

Cind apesi ENTER vei vedea ca este o eroare in linie. Aceasta apare fiindca atunci cind calculatorul gaseste ghilimelele de la "Winnetou", el crede ca acestea marcheaza sfirsitul stringului "Ai terminat" si dupa aceea nu mai stie ce inseamna Winnetou. Deci, daca vrei sa scrii ceva intre ghilimele in interiorul unui string, va trebui sa pui ghilimelele de doua ori:

```
PRINT "Ai terminat ""Winnetou"" sau nu?"
```

Ca sa apara pe ecran o singura data, ghilimelele trebuie scrise dublat.

Alte_lucruri_despre_stringuri

Daca avem un string oarecare, vom numi un substring al sau un sir de caractere consecutive, luate in_ordine, din sirul initial. Astfel, "tata" este un substring din "tata mare", dar "tama" sau "marta" nu sint.

Exista o notatie numita slicing (taiere in felii) care ne permite sa obtinem un substring dintr-un string mai mare. Forma generala a acestei notatii este:

```
expresie alfanumerica (start TO stop)
```

De exemplu:

```
"abcdef"(2 TO 5)="bcde"
```

Daca nu-i spui startul, atunci il considera 1; daca nu-i spui stopul, il considera pina la sfirsitul stringului. Adica:

```
"abcdef"( TO 5) = "abcdef"(1 TO 5) = "abcde"

"abcdef"(2 TO ) = "abcdef"(2 TO 6) = "bcdef"

"abcdef"( TO ) = "abcdef(1 TO 6) = "abcdef"
```

O forma diferita de scriere este fara TO si cu un singur numar:

```
"abcdef"(3) = "abcdef"(3 TO 3) = "c"
```

In mod normal, valorile pentru start si stop trebuie sa fie incadrate in lungimea stringului. Daca startul e mai mare decit stopul, atunci rezultatul este stringul gol (nul). Deci:

```
"abcdef"(5 TO 7)
```

va da eroare cu mesajul 3 subscript wrong fiindca stringul nu are decit 6 caractere, iar

```
"abcdef"(1 TO 0) = ""
```

```
"abcdef"(8 TO 7) = ""      -nu se mai considera lungimea
```

Daca startul sau stopul sint negative apare mesajul de eroare B integer out of range. Iata citeva exemple:

```
10 LET a$="abcdef"
20 FOR n=1 TO 6
30 PRINT a$(n TO 6)
40 NEXT n
50 STOP
```

Sterge dupa rulare cu NEW si scrie urmatorul program:

```
10 LET a$="SALUTARE !"
20 FOR n=1 TO 10
30 PRINT a$(n TO 10), a$((10-n) TO 10)
40 NEXT n
50 STOP
```

Din variabilele alfanumerice putem nu numai sa extragem substring-uri, dar sa le si asignam:

```
LET a$="Eu sint ZX Spectrum"          iar apoi:
LET a$(5 TO 8)="******"               si, in sfirsit:
PRINT a$
```

S-a intimplat asa pentru ca substring-ul a$(5 TO 8) are doar patru caractere si deci s-au luat in considerare decit primele patru stelute. Daca substring-ul din stinga ar fi fost mai lung decit sirul din dreapta, atunci s-ar fi completat cu spatii. Acest mod de asignare se numeste **procustean** (de la patul lui Procust). Incearca acum:

```
LET a$()="Salut amice"          si apoi:
PRINT a$; "!"
```

De data asta a$() a fost considerat ca substring. Ca sa apara scris corect trebuie folosit:

```
LET s$="Salut amice"
```

Expresiile cu string-uri trebuie incadrate in paranteze inainte de a extrage substring-uri din ele. De exemplu:

```
"abc"+"def"(1 TO 2)="abcde"

("abc"+"def")(1 TO 2)="ab"
```

Exercitiu:

Scrie un program care sa afiseze zilele saptaminii folosind un string de forma LuniMartiMiercuriJoiVineriSimbataDuminica.

Rezumatul_lectiei_nr.5

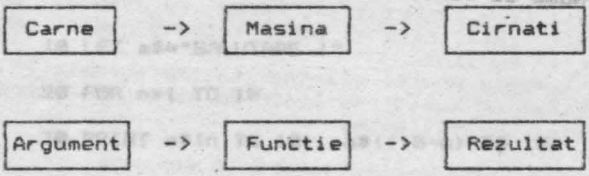
-Operatii aritmetice +, -, *, /
-Expresii matematice, notatia stiintifica a numerelor
-Nume de variabile, variabile alfanumerice, slicing

LECTIA nr.6

Functii

Functiile seamana foarte bine cu o masina de facut cirnati. Pui intr-o parte o bucata de carne, invirti de manivela, iar la capatul celalalt iese cirnatul. Daca pui carne de porc iese cirnat de porc, daca pui carne de vita iese cirnat de vita, iar daca pui carne de peste (bîhhh!!) iese cirnat de peste.

Singura diferenta este ca functiile se "incarca" cu numere sau string-uri (care se numesc argumente), iar ceea ce obtinem se numeste rezultat.



Pentru argumente diferite obtinem rezultate diferite. Daca argumentul nu este potrivit pentru functia pe care o dorim (ca si cum ai vrea sa faci cirnati din pietre), atunci va aparea un mesaj de eroare.

La fel, asa cum exista mai multe feluri de masini care fac diferite lucruri, exista si mai multe functii. Fiecare functie are un nume prin care o deosebim de celelalte. Daca vrei sa folosesti o functie intr-o expresie, n-ai decit sa-i scrii numele urmat de argument, iar calculatorul va sti sa calculeze rezultatul si sa lucreze cu el mai departe.

Sa luam de exemplu functia **LEN**, care da ca rezultat lungimea sirului de caractere din argument. Deci, daca scrii:

```
PRINT LEN "Sinclair"
```

rezultatul va fi 8 (numarul literelor din cuvintul "Sinclair"). Ca sa obtii LEN va trebui sa treci in modul extensie cu **CAPS SHIFT** si **SYMBOL SHIFT**, cursorul va trece din L in E, iar apoi sa apesi tasta K.

Daca formezi expresii cu functii si operatii, functiile vor fi executate inaintea operatiilor. Daca in evaluarea unei ex-

presii este necesara o alta ordine de executie a operatiilor si functiilor decit cea determinata de prioritatile lor, atunci se folosesc paranteze. Iata acum doua exemple in care vei vedea ordinea de executare a calculelor:

```
LEN "Ana" + LEN "Maria"          LEN ("Ana" + "Maria")

4 + LEN "Maria"                  LEN ("AnaMaria")

3+5                              LEN "AnaMaria"

8                                8
```

Mai sint si alte functii, pe care le vom incerca:

Functia STR$, converteste numere in siruri. Argumentul este un numar, iar rezultatul este sirul care ar apare pe ecran daca numarul ar fi afisat cu PRINT. Se observa ca numele functiei se sfirseste cu "$" pentru a arata ca rezultatul ei este un string. De exemplu:

```
LET a$=STR$ 1e2     va avea acelasi efect ca

LET a$="100"
```

Sau, daca scrii

```
PRINT LEN STR$ 100.0000
```

vei obtine raspunsul 3, fiindca STR$ 100.0000 ="100"

Functia VAL converteste siruri de caractere in numere, fiind intr-un fel inversa functiei STR$:

```
VAL "3.5"=3.5
```

Daca se aplica functiile STR$ si VAL asupra unui numar, totdeauna se va obtine numarul initial, pe cind daca se aplica VAL urmat de STR$ asupra unui sir de caractere nu se obtine totdeauna sirul initial. Evaluarea functiei VAL se face in 2 pasi:

1. argumentul este evaluat ca sir

2. ghilimelele sint indepartate si caracterele ramase sint evaluate ca numere.

```
VAL "2*3"=6     dar chiar si

VAL ("2"+"*3")=6
```

Poti sa te incurci foarte usor printre atitea ghilimele, daca nu-ti pastrezi cumpatul. De exemplu:

```
PRINT VAL "VAL ""VAL""""2"""""""
```

Sa nu uiti ca pentru un string in interiorul altui string, ghilimelele trebuie scrise dublat. Daca te scufunzi in string-uri, ai sa vezi ca ghilimelele vor trebui scrise de patru ori sau chiar de opt ori.

Alta functie similara lui **VAL** dar mai putin utilizata este **VAL$**. Si aceasta functie se evalueaza tot in 2 pasi; primul pas este la fel cu al functiei **VAL**, dar dupa inlaturarea ghilimelelor caracterele sint evaluate ca un alt string.

```
VAL$ """fructe"""="fructe"
```

(Vezi cum ghilimelele se-nmultesc iar ca ciupercile dupa ploaie)

Scrie acum:

```
LET a$="99"
```

si tipareste apoi: **VAL a$, VAL "a$", VAL """a$""", VAL$ a$, VAL$ "a$", VAL$ """a$"""**. Pastreaza-ti calmul si incearca sa-ti explici de ce unele merg si altele nu merg.

Functia **SGN** (se mai numeste si signum) aplicata asupra variabilei x are urmatoarea definitie:

1, daca x>0 ; 0, daca x=0 ; -1, daca x<0

Functia **ABS** produce valoarea_absoluta a numarului pe care-l are ca argument.

```
ABS -3.2 = ABS 3.2 = 3.2
```

Functia **INT** furnizeaza partea_intreaga a argumentului sau.

```
INT 3.9 = 3
```

Trebuie sa fii atent la numerele negative, fiindca acestea se rotunjesc in jos:

```
INT -3.9 = -4
```

Functia **SQR** calculeaza radacina_patrata a argumentului sau, care trebuie sa fie un numar pozitiv. De exemplu:

```
SQR 4 = 2              deoarece 2 * 2 = 4

SQR 0.25=0.5           deoarece 0.5 * 0.5 = 0.25

SQR 2 = 1.4142136 (aproximativ)
                deoarece 1.4142136 * 1.4142136 = 2.0000001

SQR -4     mesaj de eroare: An Invalid Argument
```

Sistemul permite definirea de functii ale utilizatorului. Numele posibile pentru acestea sint **FN** urmat de o litera (daca rezultatul e un numar), sau **FN** urmat de o litera si **$** (daca rezultatul e un string). Obligatoriu argumentul trebuie sa fie inclus in paranteze. Definirea functiilor utilizator se face cu functia predefinita **DEF** pusa undeva in program. Definirea functiei de ridicare la patrat se poate face astfel:

```
10 DEF FN s(x) = x*x : REM patratul lui x
```

DEF se obtine din modul extensie, apasind **SYMBOL SHIFT** si 1. Cind scrii aceasta, calculatorul pune automat **FN**, dupa care vei completa cu s si vei obtine numele complet al functiei **FN** s. Litera x dintre paranteze este numele prin care te referi la argumentul functiei. Acest nume nu poate sa aiba decit o singura litera, sau, daca argumentul este un sir de caractere, o litera urmata de $. Dupa semnul = urmeaza definitia functiei care poate fi orice expresie care contine x ca si cum ar fi o variabila obisnuita. Odata definita o functie, o poti folosi exact la fel ca si functiile calculatorului, fara sa uiti insa sa pui argumentul in paranteza. De exemplu:

```
PRINT FN s(2)

PRINT FN s(3+4)

PRINT 1+INT FN s (LEN "cocosel"/2+3)
```

Alt exemplu de functie definita: rotunjirea unui numar la cel mai apropiat intreg poate fi facuta prin aplicarea functiei **INT** asupra argumentului marit cu 0.5:

```
20 DEF FN r(x)=INT(x+0.5) : REM rotunjeste la cel mai
     apropiat intreg
```

Rezultatul va fi, de exemplu:

```
FN r(2.9) = 3          FN r(2.4) = 2

FN r(-2.9) = -3        FN r(-2.4) = -2
```

Compara aceste rezultate cu cazul cind aplici direct **INT**.

Sa vedem acum un exemplu de program:

```
10 LET x=0:  LET y=0:  LET a=10

20 DEF FN p(x,y)=a+x*y

30 DEF FN q()=a+x*y

40 PRINT FN p(2,3), FN q()
```

Aici sint citeva subtilitati:

In primul rind, o functie nu trebuie sa aiba un singur argument, ci poate avea mai multe sau chiar nici unul (trebuie insa scrise parantezele).

In al doilea rind, nu conteaza unde pui in program instructiunile DEF. Dupa ce a executat linia 10, calculatorul sare peste liniile 20 si 30, ajungind la linia 40. Ele trebuie sa fie puse undeva in program si nu pot fi intr-o comanda.

In al treilea rind, x si y sint atit nume de variabile in program, cit si nume ale argumentelor pentru functia FN p. FN p uita pentru moment de variabilele numite x si y, si chiar daca nu are un argument numit a, isi aminteste de variabila a. Cind este evaluata FN p(2,3), a are valoarea 10, deoarece e variabila libera, x are valoarea 2 deoarece este primul argument si y are valoarea 3 deoarece este al doilea argument.

Rezultatul este deci: 10+2*3=16.

Cind este evaluata functia fara argumente FN q(), a,x si y sint variabile libere si au valorile: 10, 0 respectiv 0.

Raspunsul in acest caz este : 10+0*0=10.

Schimbind acum linia 20 cu

```
20 DEF FN p(x,y)=FN q()
```

de aceasta data FN p(2,3) va avea valoarea 10.

Exista unele variante de BASIC (nu si cel ZX-Spectrum) care au functiile LEFT$, RIGHT$, MID$ si TL$ definite in continuare:

LEFT$ (a$,n) da primele n caractere din a$.

RIGHT$ (a$,n) da caracterele de la al n-lea la sfirsit.

MID$ (a$,n1,n2) da n2 caractere incepind de la al n1-lea.

TL$ (a$) da un substring din a$ fara primul caracter.

Poti defini niste functii care sa faca aceste lucruri si in BASIC-ul de pe Spectrum. De exemplu:

```
10 DEF FN t$(a$)=a$(2 TO ): REM TL$
20 DEF FN l$(a$,n)=a$( TO n): REM LEFT$
```

O functie poate avea pina la 26 argumente numerice (de ce chiar 26?) si in acelasi timp pina la 26 argumente de tip string.

Exercitiu

Utilizeaza functia FN s(x)=x*x ca sa verifici SQR. Vei vedea ca:

FN s(SQR x)=x daca x este pozitiv, si

SQR FN s(x)=ABS x indiferent daca x este pozitiv sau negativ. (De ce trebuie pus ABS?)

Rezumatul_lectiei_nr.6

-functii definite de calculator: LEN, STR$, VAL, SGN, ABS, INT, SQR
-functii definite de utilizator: DEF, FN

LECTIA nr.7

Functii_matematice

In aceasta lectie vei vedea cite ceva din matematica pe care o stie calculatorul. S-ar putea sa nu trebuiasca sa folosesti prea curind aceste functii, dar e bine sa stii ca ele sint definite de calculator.

Functiile definite de calculator au prioritate mai mare decit operatiile. Daca in evaluarea unei expresii este necesara o alta ordine de executie a operatiilor si functiilor decit cea determinata de prioritatile lor, atunci se folosesc paranteze.

Functiile matematice definite in **BASIC** sint ridicarea la putere, functia exponentiala, functia logaritmica si functiile trigonometrice.

Functia ridicare la putere "↑" are prioritate mai mare decit inmultirea si impartirea. Ea necesita 2 operanzi dintre care primul este obligatoriu pozitiv. Intr-o insiruire de ridicari la putere, ordinea evaluarii este de la stinga la dreapta, ceea ce inseamna ca :

```
2↑3↑2 = 8↑2 = 64
```

Functia EXP defineste functia exponentiala:

```
EXP x = e↑x
```

unde e=2,71...

Daca vrei sa afli o valoare mai precisa a lui e, scrie:

```
PRINT EXP 1
```

si vei obtine valoarea (aproximativa) a lui e, fiindca EXP 1 = e↑1 = e.

Functia LN calculeaza logaritmul natural al argumentului. Functia logaritmica este deci inversa functiei exponentiale. Ea poate fi utilizata la calculul unui logaritm in orice baza folosind formula:

```
LOGa x = LN x/ LN a
```

SIN, COS, TAN, ASN, ACS, ATN sint mnemonicele (denumirile prescurtate) ale functiilor sinus, cosinus, tangenta, arcsinus, arccosinus si respectiv arctangenta. Argumentul functiilor trigonometrice trebuie sa fie exprimat in radiani. Legatura intre exprimarea in grade (care ne este mai familiara) si exprimarea in radiani este:

```
360 grade = 2 * PI radiani
```

Sistemul pune la dispozitia utilizatorului numarul "pi", ce poate fi apelat apasind tasta PI (mod extensie si apoi tasta M). Comanda PRINT PI tipareste valoarea numarului "pi"=3.1415926...

Numere_aleatoare_(intimplatoare)

Vom studia acum functia **RND** si cuvintul-cheie **RANDOMIZE**. Ambele se folosesc in legatura cu numerele aleatoare, asa ca va trebui sa fii atent ca sa nu le incurci (mai ales ca sint pe aceeasi tasta - T).

RND este ca o functie, adica face niste calcule si da un rezultat, insa nu are argument. De fiecare data cind o folosesti, vei obtine un numar intre 0 si 1 (se poate atinge valoarea 0, dar valoarea 1 nu se atinge niciodata). Incearca:

```
10 PRINT RND

20 GO TO 10
```

si vei vedea ca nu exista nici o regula dupa care sa apara numerele. De fapt, valorile date de **RND** sint sint luate dintr-o secventa fixa de 65536 numere, si vom spune ca aceasta este o functie pseudo-aleatoare.

Putem obtine usor numere aleatoare intre diferite alte limite decit 0 si 1, folosind **RND** in diferite expresii:

```
5 * RND          - intre 0 si 5
1.3 + 0.7*RND    - intre 1.3 si 2
```

Ca sa obtinem numere aleatoare cu valori intregi putem folosi **INT** (atentie, se rotunjeste in jos), ca de exemplu in expresia 1 + INT (RND * 6) care da valorile 1, 2, 3, 4, 5 si 6, putind sa simuleze un zar:

```
10 REM  Program de simulare a aruncarii zarurilor

20 CLS

30 FOR n=1 TO 2

40 PRINT 1+INT(RND*6);" ";

50 NEXT n

60 INPUT a$:GO TO 20
```

Apasa ENTER cind vrei sa arunci zarurile.

Instructiunea **RANDOMIZE** se foloseste pentru a face ca **RND** sa

porneasca dintr-un anume loc al al secventei de numere, dupa cum vei vedea in programul urmator:

```
10 RANDOMIZE 1

20 FOR n=1 to 5 :PRINT RND :NEXT n

30 PRINT :GO TO 10
```

Dupa fiecare executie a lui RANDOMIZE 1, functia RND va incepe cu valoarea 0.0022735596. Poti incerca cu diferite numere intre 1 si 65535 in instructiunea RANDOMIZE, iar RND va incepe de la alte valori. Daca ai un program care contine RND si nu merge, poti gasi mai usor greselile daca folosesti RANDOMIZE pentru a obtine aceeasi valoare a lui RND la fiecare rulare.

Instructiunea RANDOMIZE (are acelasi efect cu RANDOMIZE 0) este diferita:

```
10 RANDOMIZE

20 PRINT RND:GO TO 10
```

Secventa de numere pe care o vei obtine nu este chiar aleatoare, deoarece RANDOMIZE foloseste timpul scurs de la pornirea calculatorului. vei obtine o secventa aleatoare daca vei inlocui GO TO 10 cu GO TO 20.

Obs. Multe tipuri de BASIC folosesc RND si RANDOMIZE pentru a genera numere aleatoare, dar nu toate le folosesc la fel.

Iata un program care simuleaza aruncarea unei monezi si numara de cite ori cade "capul" sau "pajura":

```
10 LET cap=0:LET pajura=0

20 LET moneda=INT(RND*2)

30 IF moneda=0 THEN LET cap=cap+1

40 IF moneda=1 THEN LET pajura=pajura+1

50 PRINT cap; ",";pajura

60 IF pajura>0 THEN PRINT cap/pajura;

70 PRINT :GO TO 20
```

Raportul dintre "cap" si "pajura" va fi aproximativ 1 dupa un numar suficient de mare de aruncari, deoarece probabilitatea celor doua posibilitati este egala.

Exercitii

1. Incearca acest program care calculeaza ce suma vei avea in cont peste y ani, daca acum depui 100 lei, iar dobinda este de 15% (bineinteles ca acest calcul nu tine seama de inflatie).

```
10 FOR y=0 to 100
20 PRINT y, 100*1.15↑y
30 NEXT y
```

2. Alege un numar intre 1 si 872 si scrie:

```
RANDOMIZE numarul ales
```

Urmatoarea valoare data de RND se poate calcula:

```
(75 * (numarul ales + 1) - 1)/65536
```

Rezumatul_lectiei_nr.7

-Functii: ↑, EXP, LN, SIN, COS, TAN, ASN, ACS, ATN, PI

-Numere aleatoare: RND, RANDOMIZE

LECTIA nr.8

Tablouri

Sa presupunem ca ai o lista de numere, de exemplu matricolele a zece elevi dintr-o clasa. Pentru a le stoca in calculator poti sa atribui o variabila pentru fiecare elev, dar vei vedea ca este un procedeu foarte greoi. Daca te-ai decide sa folosesti variabilele Matric 1, Matric 2, ... , Matric 10 programul pentru inscriera valorilor acestor variabile ar fi destul de lung. Ar fi foarte frumos sa poti scrie:

```
10 FOR n = 1 TO 10

20 READ Matric n

30 NEXT n

40 DATA 10,2,5,19,16,3,11,1,0,6
```

Dar nu poti. Exista totusi un mecanism cu care poti sa folosesti aceasta idee, si anume sa folosesti tablouri. Un tablou este un set de variabile care au toate acelasi nume deosebindu-se intre ele printr-un numar de ordine (indice) care se pune in paranteza imediat dupa nume. Numele unui tablou trebuie sa fie o singura litera. Deci am putea folosi in cazul nostru un tablou ale carui elemente sa fie M(1), M(2),..., M(10). Elementele unui tablou se numesc variabile_indexate spre deosebire de variabilele simple cu care deja am lucrat.

Inainte de a folosi un tablou, va trebui sa-i rezervi un spatiu in memoria calculatorului, utilizind instructiunea **DIM**.

```
DIM M(10)
```

Aceasta initializeaza un tablou cu numele M si cu dimensiunea 10, adica cu 10 elemente indexate, si toate valorile vor fi 0. De asemenea aceasta instructiune va sterge un eventual alt tablou anterior care s-ar fi numit tot M, dar nu sterge o eventuala variabila simpla numita M. Pot exista simultan tablouri si variabile simple cu acelasi nume, fara sa le incurcam intre ele, deoarece variabilele indexate au intotdeauna un indice. Indicele poate fi orice expresie numerica. Acum vei putea scrie:

```
10 FOR n = 1 TO 10

20 READ M(n)

30 NEXT n

40 DATA 10,2,5,19,16,3,11,1,0,6
```

Poti folosi de asemenea tablouri cu mai mult de o dimensiune. Intr-un tablou bidimensional (matrice) va trebui ca fiecare element sa aiba doi indici care sa arate numarul liniei si al coloanei pe care se gaseste elementul respectiv in cadrul matricii. De exemplu,

```
DIM c(3,6)
```

va genera un tablou bidimensional cu 3*6=18 elemente:

```
c(1,1), c(1,2), ... , c(1,6)

c(2,1), c(2,2), ... , c(2,6)

c(3,1), c(3,2), ... , c(3,6)
```

Pe acelasi principiu poti folosi tablouri cu oricite dimensiuni doresti. Trebuie insa sa ai grija ca nu poti folosi doua tablouri cu acelasi nume, chiar daca au un numar diferit de dimensiuni.

Exista de asemenea tablouri de string-uri. Sirurile dintr-un tablou difera de sirurile simple prin aceea ca au lungime fixa si asignarea lor este procusteana. Un alt mod de interpretare al unui tablou de siruri de caractere este ca tablou de caractere simple cu numarul dimensiunilor majorat cu 1 fata de cazul precedent. Un tablou de siruri si o variabila sir simpla nu pot avea acelasi nume (spre deosebire de cazul variabilelor numerice).

Pentru a defini un tablou a$ de 5 siruri, trebuie stabilita mai intii lungimea sirului - spre exemplu 10 caractere. Linia:

```
DIM a$(5,10)
```

defineste 5*10=50 caractere, dar fiecare rind poate fi interpretat ca un sir.

```
a$(1)=a$(1,1) a$(1,2) ... a$(1,10)

a$(2)=a$(2,1) a$(2,2) ... a$(2,10)
  .       .       .       .
  .       .       .       .
a$(5)=a$(5,1) a$(5,2) ... a$(5,10)
```

Daca sint utilizate doua dimensiuni, se obtine un singur caracter, dar daca este omisa a doua dimensiune, atunci se obtine un sir cu lungime fixa. Astfel a$(2,7) e al saptelea caracter in sirul a$(2); o alta notatie a aceluiasi element este a$(2)(7).

Ultimul indice poate avea si forma unui selector de subsir. De exemplu, daca a$(2)="12345667890", atunci

```
a$(2,4 TO 8)= a$(2)(4 TO 8)="45678"
```

Se pot defini variabile de tip tablou de siruri de caractere cu o singura dimensiune:

```
DIM a$(10)
```

In acest caz variabila se comporta ca o variabila simpla cu exceptia faptului ca are totdeauna lungime fixa iar asignarea ei este procusteana.

Exercitiu

Cu ajutorul instructiunilor **READ** si **DATA** incearca sa introduci un tablou l$ format din 12 string-uri, in care l$(n) sa fie numele lunii a n-a. (instructiunea **DIM** va fi **DIM l$(12,10)**)

Conditii

In lectia a 3-a am vazut cum se folosea instructiunea IF cu conditiile =, <, >, <=, >=, si <>. Pentru realizarea unor expresii complexe se pot utiliza si operatiile logice **OR**, **AND** si **NOT** care admit operanzi de tip boolean (expresii ce pot lua valorile fals sau adevarat).

O relatie AND alta relatie este adevarata daca ambele relatii sint adevarate simultan. De exemplu instructiunea

```
IF a$="DA" AND x>0 THEN PRINT x
```

tipareste valoarea numarului "x" daca sint indeplinite simultan conditiile ca a$="DA" si x>0.

Similar se pot realiza expresii cu **OR** daca se doreste identificarea situatiei in care cel putin una dintre conditii este indeplinita. Operatia **NOT** produce ca rezultat inversul valorii_de_adevar a argumentului sau.

Expresiile logice sint realizate din relatii legate prin AND, OR si NOT, exact asa cum expresiile numerice sint alcatuite din numere legate prin operatii (+, -, ... etc.). Daca e nevoie se pot folosi si paranteze. Exista si in acest caz o ordine de prioritate in evaluarea expresiilor logice, si anume: OR are cea mai mica prioritate, apoi, in ordine, **AND**, **NOT**, apoi relatiile si in sfirsit operatiile uzuale.

Iata in continuare citeva aspecte destul de complicate, la care va trebui sa fii mai atent. Incearca sa scrii:

```
PRINT 1=2, 1<>2
```

la care te-ai astepta sa apara eroare de sintaxa. De fapt, calculatorul nu lucreaza efectiv cu valori logice, ci cu niste numere

care se supun unor reguli:

a. =, <, >, <=, >=, si <> dau rezultate numerice: 1 pentru "adevarat" si 0 pentru "fals".

b. In instructiunea

```
IF conditie THEN ...
```

conditia poate fi orice expresie numerica. Daca aceasta expresie era valoarea 0, atunci este considerata ca falsa, iar daca are o valoare diferita de zero (inclusiv 1 dat de o relatie adevarata), atunci este considerata ca adevarata. Deci, IF inseamna

```
IF conditie <>0 THEN ...
```

c. Operatiile OR, AND, NOT pot fi aplicate si unor argumente numerice. Functiile definite astfel sint:

1. x AND y ia valoarea: x , daca y e nenul
   0 , daca y=0

2. x OR y ia valoarea: 1 , daca y e nenul
   x , daca y=0

3. NOT x ia valoarea: 0 , daca x e nenul
   1 , daca x=0

Trebuie sa retii ca "adevarat" inseamna "nenul" atunci cind verificam o valoare deja data, dar inseamna "1" atunci cind producem o valoare noua.

Pentru exemplificare, iata un program care tipareste pe cel mai mare dintre doua numere:

```
10 INPUT a
20 INPUT b
30 PRINT (a AND a>=b) + (b AND a<b)
40 GO TO 10
```

Poti obtine si un string ca rezultat al unei expresii logice dar numai pentru AND:

```
x$ AND y are valoarea:        x$ daca y este nenul
                              "" daca y=0
```

Iata un program care pune in ordine alfabetica doua cuvinte:

```
10 INPUT "scrie doua cuvinte",a$,b$
```

```
20 IF a$>b$ THEN LET c$=a$: LET a$=b$: LET b$=c$
30 PRINT a$;" ";("<" AND a$<b$)+("=" AND a$=b$);" ";b$
40 GO TO 10
```

Rezumatul lectiei nr.8:

- **Tablouri: DIM**
- **Conditii, expresii logice: AND, OR, NOT**

LECTIA nr.9

Setul_de_caractere

Literele, cifrele, semnele de punctuatie si orice apare pe ecran poarta numele de caractere. Luate impreuna, acestea formeaza alfabetul calculatorului care se numeste setul de caractere. Majoritatea caracterelor folosite de ZX-Spectrum sint simboluri (apar pe ecran in spatiul corespunzator unei singure litere), dar mai sint si asa numitele token-uri (apar scrise ca si cuvinte intregi : ex. PRINT, LET, <>...). Exista un numar de 256 caractere, pentru fiecare existind un cod numeric, cuprins intre 0 si 255 (aici trebuie mentionat ca primele 32 de caractere [codurile 0 - 31] nu sint de fapt caractere propriu-zise, ci sint caractere de control, utilizate pentru a facilita dialogul cu imprimanta si in alte scopuri). Conversia intre cod si caracterul corespunzator se face cu functiile **CODE** si **CHR$**.

CODE se aplica unui string si da ca rezultat codul primului caracter al acestuia (rezultatul este 0 daca string-ul este gol).

CHR$ se aplica unui numar si produce caracterul ce are acel cod.

Iata un program care tipareste intregul set de caractere:

```
10 FOR a=32 TO 255: PRINT CHR$ a;: NEXT a
```

Setul de caractere este format din: caracterele ASCII, cuvinte cheie, caractere grafice definite de utilizator.

Un caracter se deseneaza pe o retea de 8*8 puncte, fiecarui punct corespunzindu-i un bit in memorie. Pentru programarea unui caracter definit de utilizator este necesara descrierea starii fiecarui punct al matricii prin care se reprezinta caracterul respectiv:

1. 0 corespunde unui punct alb
2. 1 corespunde unui punct negru

Pentru definirea unui caracter se foloseste de 8 ori functia **BIN**. Functia **BIN** descrie o linie a caracterului, argumentul sau fiind format din 8 cifre binare.

Cei 8 numere sint memorate in 8 octeti care corespund aceluiasi caracter.

Functia **USR** are ca argument o litera intre a si u (sau A si U) intre ghilimele. Rezultatul funtiei **USR** este adresa primului

octet (din cei 8) care formeaza respectivul caracter definit de utilizator (UDG).

POKE memoreaza un numar direct intr-o locatie de memorie. Opusul lui POKE este PEEK, care ne permite sa citim continutul unei locatii de memorie, fara a-l modifica.

Pentru a defini caracterul grecesc pi (care sa apara pe ecran la apasarea tastei P in mod grafic) se utilizeaza urmatoarea secventa de program:

```
10 FOR n=0 TO 7
20 INPUT rind:POKE USR "P"+n, rind
30 NEXT n
```

Datele introduse vor fi (in ordinea prezentata):

```
BIN 00000000
BIN 00000000
BIN 00111100
BIN 01010100
BIN 00010100
BIN 00010100
BIN 00000000
```

V-ati pus cred intrebarea de ce nu am tiparit in primul exemplu decit caracterele incepind cu codul 32. Caracterele 0...31 sint caractere de control ai modului de lucru. Ele nu produc nimic afisabil, unele avind efect asupra modului cum se tipareste, iar altele fiind folosite pentru a controla altceva decit afisarea. De exemplu CHR$ 6 realizeaza tabularea pe orizontala (efect similar unei virgule intr-o instructiune PRINT).

```
PRINT 1; CHR$ 6; 2
```

are acelasi efect cu:

```
PRINT 1,2
```

Un mod mai subtil si util de folosire este:

```
LET a$="1"+CHR$ 6+"2"
PRINT a$
```

CHR$ 8 determina mutarea cursorului inapoi cu o pozitie.

Exemplu:

```
PRINT "1234"; CHR$ 8;"5"
```

tipareste: 1235

CHR$ 13 muta cursorul la inceputul liniei urmatoare.

Utilizind codurile pentru caractere putem extinde conceptul de ordine alfabetica pentru a acoperi siruri ce contin orice caractere, nu numai litere, folosind in locul alfabetului uzual de 26 litere, alfabetul extins de 256 caractere (la codificarea caracterelor s-a avut in vedere ca ordinea crescatoare a codurilor atasate literelor sa coincida cu ordinea alfabetica).

Este prezentata mai departe o regula de gasire a ordinii in care se afla doua siruri. Mai intii se compara primele caractere. Daca sint diferite, unul dintre ele are codul mai mic decit celalalt si, deci, se poate decide care este ordinea alfabetica a sirurilor. Daca aceste coduri sint egale, se compara urmatoarele caractere. Sa retinem ca literele mici sint dupa cele mari ("a" este dupa "Z"); de asemenea conteaza spatiile.

Ordinea alfabetica se exprima prin relatiile =, <, >, <=, >= sau <>, la fel ca si in cazul numerelor. Poti experimenta toate aceste lucruri cu urmatorul program care primeste doua siruri de caractere si le pune in ordinea stricta a codurilor lor:

```
10 INPUT "Scrie doua siruri:",a$,b$
20 IF a$>b$ THEN LET c$=a$: LET a$=b$: LET b$=c$
30 PRINT a$;" ";
40 IF a$<b$ THEN PRINT "<";: GO TO 60
50 PRINT "="
60 PRINT " ";b$
70 GO TO 10
```

(Trebuie introdus c$ in linia 20 pentru a schimba intre ele a$ si b$).

Iata acum un program care initializeaza citeva din caracterele puse la dispozitia utilizatorului cu figurile jocului de sah:

```
5 LET b=BIN 01111100: LET c=BIN 00111000:
```

```
LET d=BIN 00010000
10 FOR n=1 TO 6: READ p$: REM 6 piese
20 FOR f=0 TO 7: REM citeste piesele in octeti
30 READ a: POKE USR p$+f,a
40 NEXT f
50 NEXT n
100 REM Nebunul
110 DATA "B", 0, 0, BIN 001001000, BIN 01000100
120 DATA BIN 01101100, c, b, 0
130 REM Regele
140 DATA "R", 0, d, c, d
150 DATA c, BIN 010001000, c, 0
160 REM Turnul
170 DATA "T", 0, BIN 01010100, b, c
180 DATA c,b,b,0
190 REM Dama (regina)
200 DATA "D",0,BIN 01010100, BIN 00101000, d
210 DATA BIN 01101100, b, b, 0
220 REM Pionul
230 DATA "P", b, 0, d, c
240 DATA c, d, b, 0
250 REM Calul
260 DATA "C", 0, d, c, BIN 01111000
270 DATA BIN 00011000, c, b, 0
```

Dupa ce rulezi programul vei putea afisa figurile respective folosind caracterele grafice ale utilizatorului: P (pion), T (turn), C (cal), N (nebun), D (dama) si R (rege).

LECTIA nr.10

Mai multe despre PRINT si INPUT

Instructiunea **PRINT** a fost folosita deja pentru a scrie variabile numerice sau sir, ori pentru a scrie cifre sau siruri de caractere. Pentru delimitarea lor s-au folosit urmatorii separatori:

- punct-virgula ;
- virgula ,
- apostrof '

Pentru a observa efectul acestora incercati urmatorul program:

```
10 PRINT 1;2

20 PRINT 1,2

30 PRINT 1'2
```

Se pot face urmatoarele observatii:

- ceea ce este scris dupa punct-vigula apare lipit de ceea ce era anterior punct-virgulei

- virgula indica un salt la inceputul urmatoarei jumatati de ecran

- apostroful semnifica salt la inceputul urmatoarei linii

1. Ecranul televizorului are 32 de coloane numerotate de la stinga la dreapta (0 la 31) si 22 de linii numerotate de sus in jos (0 la 21). Pentru a scrie intr-o anumita pozitie pe ecran, se foloseste instructiunea:

```
PRINT AT linie,coloana
```

Incercati urmatoarele programe:

```
10 CLS

20 PRINT AT RND*21,RND*31;"*"

30 GO TO 20
```

sau

```
10 CLS
```

```
20 FOR K = 0 TO 31
30 PRINT AT 0,K;"*"; AT 21,K;"*"
40 NEXT K
50 FOR K = 0 TO 21
60 PRINT AT K,0;"*";AT K,31;"*"
70 NEXT K
```

2. In conjunctie cu PRINT AT se foloseste SCREEN$. Rezultatul functiei SCREEN$ este caracterul care exista pe ecran la linia si coloana specificata. Sintaxa instructiunii este urmatoarea:

```
SCREEN$ (linie,coloana)
```

Introduceti acum urmatoarele comenzi:

```
CLS: PRINT AT 12,14;"$": PRINT SCREEN$(12,14)
```

Functia SCREEN$ nu va recunoaste caractere sau semne obtinute cu PLOT, DRAW, CIRCLE sau prin supraafisare cu OVER 1.

3. Pentru a afisa pe ecran tabele se poate folosi instructiunea:

```
PRINT TAB coloana
```

care are ca efect afisarea la coloana specificata din linia curenta sau urmatoare. Pentru a intelege cum functioneaza aceasta instructiune introduceti urmatorul program:

```
10 PRINT "grade"; TAB 5; "sinus"; TAB 18; "cosinus"
20 FOR i = 0 TO 20
30 LET r = i*PI/180
40 PRINT i; TAB 5; SIN r; TAB 18; COS r
50 NEXT i
```

Daca numarul specificat pentru coloana depaseste 31 se va considera restul impartirii acestuia la 32. Rulati urmatorul program:

```
10 FOR n = 0 TO 20
20 PRINT TAB 8*n;n;
```

```
30 NEXT n
```

4. Atunci cind folositi **INPUT** cu mesaj, scrierea acestuia se face tinind cont de toate regulile pentru **PRINT**, dar afisarea se face in partea de jos a ecranului. Daca in mesajul instructiunii **INPUT** doriti sa figureze variabile numerice sau sir de caractere, va trebui sa le puneti intre paranteze, altfel vor fi considerate variabile de introdus. De exemplu:

```
10 LET V = INT (100*RND)

20 INPUT "Eu am ";(V);" ani. Tu citi ani ai ? ";w

30 PRINT "Tu ai ";w;" ani !"
```

5. Pentru a vedea cum functioneaza **AT** impreuna cu **INPUT** incercati:

```
10 INPUT "Aceasta e linia 1";a$; AT 0,0;"Aceasta e linia 0";a$; AT 2,0;"Aceasta e linia 2";a$; AT 1,0;"Aceasta ramine linia 1";a$
```

Apasati **ENTER** dupa fiecare oprire.

Pentru a vedea cum este influentata partea de sus a ecranului, incercati urmatorul program:

```
10 FOR n= 0 TO 19: PRINT AT n,0 ;n;: NEXT n

20 INPUT AT 0,0; a$; AT 1,0;a$; AT 2,0; a$; AT 3,0; a$; AT 4,0;a$; AT 5,0; a$
```

Un alt rafinament referitor la **INPUT** este introducerea variabilelor sir de caractere cu **LINE**. Introduceti comanda:

```
INPUT LINE a$
```

Se observa ca nu apar ghilimelele, din acest motiv nu se pot introduce expresii sir pentru variabile.

5. Caracterele de control **CHR$ 22** si **CHR$ 23** au acelasi efect ca si **AT** si **TAB**. Incercati urmatorul program:

```
10 LET c=10

20 PRINT AT c,12; "Salut !"
```

Rulati programul, dupa care inlocuiti linia 20 cu:

```
20 PRINT CHR$ 22 + CHR$ c + CHR$ 12
```

Urmatoarele comenzi sint echivalente:

```
PRINT CHR$ 23 + CHR$ a + CHR$ b
```

cu:

```
PRINT TAB a + 256*b
```

6. Puteti opri calculatorul sa va intrebe "scroll?" cu urmatoarea comanda:

```
POKE 23692,255
```

De exemplu:

```
10 FOR n = 0 TO 10000
20 PRINT n: POKE 23692,255
30 NEXT n
```

Exercitiu

Urmatorul program verifica elevii daca cunosc tabla inmultirii:

```
10 LET m$ = ""
20 LET a = INT (RND*10) + 1: LET b = INT (RND*10) + 1
30 INPUT (m$)''"Cit face";(a);"*";(b);"?";c
40 IF c = a*b THEN LET m$ = "Foarte bine": GO TO 20
50 LET m$ = "GRESIT ! MAI INCEARCA !": GO TO 30
```

Daca programul intreaba, de exemplu: 2*3 si se va raspunde chiar cu 2*3, am pacalit programul. Pentru a evita aceasta, introduceti in linia 30 in loc de c pe c$, iar in linia 40 in loc de c, VAL c si introduceti linia:

```
35 IF c$ <> STR$ VAL c$ THEN LET m$ = "Introduceti rezultatul cinstit": GO TO 30
```

Programul mai poate fi pacalit si acum daca in loc de raspuns se introduce STR$(2*3). Pentru a elimina si aceasta posibilitate inlocuiti in linia 30 pe c$ cu LINE c$.

CULORI

Rulati urmatorul program:

```
10 FOR a = 0 TO 1: BRIGHT a
```

```
 20 FOR n = 1 TO 10
 30 FOR c= 0 TO 7
 40 PAPER c: PRINT "    ": REM 4 spatii
 50 NEXT c: NEXT n: NEXT m
 60 FOR m = 0 TO 1: BRIGHT m: PAPER 7
 70 FOR c = 0 TO 3
 80 INK c: PRINT c;"    ";
 90 NEXT c: PAPER 0
100 FOR c = 4 TO 7
110 INK c: PRINT c;"   "; REM 3 spatii
120 NEXT c: NEXT m
130 PAPER 7: INK 0: BRIGHT 0
```

Acest program ne arata cele 8 culori (inclusiv alb si negru) si cele doua nivele de luminozitate pe care calculatorul le poate afisa pe ecranul TV. (Daca dispunem de un televizor alb-negru, se vor observa nuante de gri. Culorile sint codificate astfel:

0 - negru	4 - verde
1 - albastru	5 - albastru dechis (cyan)
2 - rosu	6 - galben
3 - violet	7 - alb

Am vazut anterior ca ecranul are 32 de coloane si 24 de linii, adica un total de maxim 768 de caractere afisabile la un moment dat. Acestor caractere le corespund un numar de 768 de pozitii de pe ecran, fiecarei pozitii fiindu-i asociate doua culori: una pentru simbolul propriu-zis (INK sau cerneala) si cealalta pentru fond (PAPER sau hirtie). In plus fiecarei pozitii ii este asociata o luminozitate care poate fi normala sau marita (BRIGHT). In sfirsit, caracterul care este scris intr-o anumita pozitie poate fi pilpiitor sau nu (FLASH). Prin pilpiire se intelege schimbarea repetata a culorii pentru INK cu cea pentru PAPER.

In concluzie, fiecarei pozitii de caracter de pe ecran ii sint asociate urmatoarele informatii privind culorile:

- FLASH (da = 1, nu = 0)
- BRIGHT (da = 1, nu = 0)

- INK (de la 0 la 7)

- PAPER (de la 0 la 7)

Pentru a seta culoarea verde (de exemplu), dati comanda:

```
PAPER 4
```

si apasati ENTER de doua ori, sau doar:

```
PAPER 4: CLS
```

Numarul 8 nu semnifica o culoare anume, ci inseamna "transparent" in sensul ca se pastreaza vechiul atribut. Incercati:

```
10 PAPER 7: INK 0: CLS
20 PAPER 4
30 PRINT 12345
```

Daca dispuneti de un televizor in culori, veti vedea cum pe ecranul alb va fi scris numarul 12345 cu negru pe verde (in cazul in care aveti un TV alb-negru, acelasi efect va fi perceput pe nuante de gri). Daca modificati linia 30 astfel:

```
30 PRINT PAPER 8;1234
```

atunci numarul 1234 va fi scris tot cu negru pe alb in ciuda liniei 20.

Acelasi lucru este valabil si pentru **INK**, **FLASH**, **BRIGHT**.

Numarul 9 poate fi folosit doar cu **INK** si **PAPER** si inseamna "contrast". Introduceti urmatorul program:

```
10 INK 9
20 FOR c = 0 TO 7
30 PAPER c: PRINT c
40 NEXT c
```

Se observa ca pentru a se obtine contrastul maxim, pe culorile inchise (negru, albastru, rosu si violet) se scrie cu cerneala alba, iar pe culorile deschise (verde, cyan, galben si alb) se scrie cu cerneala neagra.

O alta instructiune utilizabila este **INVERSE**. Daca se scrie **INVERSE 1**, rezultatul va fi schimbarea intre ele a culorilor hirtiei si cernelii; de exemplu daca avem **INK 0** si **PAPER 7**, in urma comenzii **INVERSE 1** se va scrie cu alb pe negru. Comanda **INVERSE 0** readuce totul la starea initiala.

Comanda OVER se refera la supraimprimare. Pentru a vedea cum functioneaza dati comanda NEW, apoi introduceti:

```
10 OVER 1

20 FOR n = 1 TO 32

30 PRINT "o";CHR$ 8;"""";

40 NEXT n
```

Se va obtine un rind de O-uri. Notati ca CHR$ 8 este caracterul de control pentru intoarcerea cu un spatiu (backspace). Daca se modifica linia 10 astfel:

```
10 OVER 0
```

se vor afisa doar ghilimelele.

Colorarea zonei din jurul ecranului se poate realiza cu instructiunea:

```
BORDER culoare
```

unde culoare este un numar intre 0 si 7.

Instructiunile FLASH, BRIGHT, PAPER, INK, INVERSE si OVER pot fi folosite impreuna cu PRINT, INPUT, dar si impreuna cu instructiunile grafice PLOT, DRAW si CIRCLE, dupa cum se va vedea in continuare. Incercati:

```
PRINT PAPER 6;"x";: PRINT "y"
```

si veti observa ca doar x este scris pe fond galben. Se mai poate scrie:

```
INPUT FLASH 1;INK 1;"Introduceti numarul";n
```

La fel ca si pentru AT si TAB avem urmatoarea corespondenta cu caracterele de control:

```
CHR$ 16 . . . INK        CHR$ 19 . . . BRIGHT
CHR$ 17 . . . PAPER      CHR$ 20 . . . INVERSE
CHR$ 18 . . . FLASH      CHR$ 21 . . . OVER
```

De exemplu:

```
PRINT CHR$ 16+ CHR$ 2;"Salut!"
```

are acelasi efect ca si :

```
PRINT INK 2;"Salut!"
```

Foarte interesant este sa introduci culorile direct in program, apasind CS si SS (EXTEND) si apoi o cifra intre 0 si 7 pentru INK daca este apasat CS, respectiv o cifra intre 0 si 7 fara apasarea lui CS pentru PAPER.

De exemplu apasati:

```
CS + SS (EXTEND)
CS + 4
```

pentru a obtine cerneala (INK) verde.

De asemenea in modul extins se poate obtine FLASH si BRIGHT apasind CS + SS (EXTEND) si apoi:

```
8          pentru BRIGHT 0

9          pentru BRIGHT 1

CS + 8     pentru FLASH 0

CS + 9     pentru FLASH 1
```

iar in modul normal (nu EXTEND) se poate obtine:

```
CS + 3     pentru INVERSE 0
CS + 4     pentru INVERSE 1
```

Functia ATTR are forma:

```
ATTR (linie,coloana)
```

unde cele doua argumente au aceeasi semnificatie ca la AT. Rezultatul functiei este un numar compus din suma urmatoarelor 4 numere:

```
128 daca pozitia este pilpiitoare   si 0 in caz contrar
 64 daca pozitia este stralucitoare si 0 in caz contrar
  8 * codul culorii pentru PAPER
      codul culorii pentru INK
```

Daca de exemplu, un caracter albastru este pilpiitor, nestralucitor, pe fond galben, rezultatul functiei ATTR este un numar numit atributul caracterului respectiv (de aici denumirea functiei ATTR. In cazul nostru, acest numar este:

```
128 * 1 + 64 * 0 + 8 * 6 + 1 = 177
```

Testati acest rezultat cu:

```
PRINT AT 0,0; FLASH 1; PAPER 6; INK 1; " "; ATTR(0,0)
```

Exercitii

1. Introduceti:

```
PAPER 0: INK 0: BORDER 0: CLS
```

Va place?

2. Incercati programul:

```
10 POKE 22527 - RND * 704, RND * 127
20 GO TO 10
```

Acest program schimba in mod aleator culoarea unor pozitii de pe ecran, cu o alta culoare, aleasa aleator.

Rezumatul lectiei 10

- separatorii ; , '
- AT
- TAB
- INPUT LINE
- SCREEN$
- culori
- INK, PAPER
- FLASH, BRIGHT

LECTIA nr.11

PLOT, DRAW, CIRCLE, POINT

In aceasta lectie vom vedea cum se deseneaza. Zona din ecran pe care o poti folosi este de 22 de linii si 32 de coloane, adica 22 * 32 = 704 pozitii pentru caractere. Fiecare caracter este compus din 8*8 puncte, numite pixeli (picture elements) - elemente primare de imagine. Fiecarui pixel ii sint asociate 2 numere, numite coordonate. Primul este coordonata x care indica distanta punctului fata de marginea din stinga a ecranului. Al doilea numar este coordonata y, care arata cit de departe se afla punctul fata de marginea de jos a ecranului.

Colturile ecranului au coordonatele:

```
(0,0)     - stinga-jos
(255,0)   - dreapta-jos
(0,175)   - stinga-sus
(255,175) - dreapta-sus
```

Instructiunea

```
PLOT x,y
```

deseneaza un punct la coordonatele x,y.

Incercati urmatorul program:

```
10 PLOT INT (RND*256),INT (RND*176) : GO TO 10
```

El va desena pe ecran puncte aleatoare.

Programul urmator afiseaza graficul functiei sinus pentru valori ale argumentului intre 0 si 2 π.

```
10 FOR n = 0 TO 255
20 PLOT n,88+88*SIN(n/128*PI)
30 NEXT n
```

Urmatorul program va desena graficul functiei √ (SQR), adica un arc de parabola, argumentul luind valori intre 0 si 4.

```
10 FOR n = 0 TO 255
20 PLOT n,80*SQR(n/64)
30 NEXT n
```

De notat ca **PLOT** foloseste alte coordonate decit cele utilizate ca linie si coloana in **AT** la instructiunile **PRINT** si **INPUT**.

Instructiunea **DRAW** care traseaza o dreapta are forma:

DRAW x,y

Inceputul liniei este ultimul pixel desenat cu **PLOT**, **DRAW** sau **CIRCLE** - instructiunile **RUN**, **CLEAR**, **CLS** si **NEW** aduc acest punct in coltul din stinga-jos, de coordonate (0,0). Sfirsitul segmentului de dreapta se va gasi cu x pixeli spre dreapta si cu y pixeli mai sus decit punctul de inceput. Instructiunea DRAW specifica deci directia si lungimea dreptei, dar nu si punctul sau de inceput.

Dati urmatoarele comenzi:

PLOT 0,100: DRAW 80,-35

PLOT 90,150: DRAW 80,-35

Notati ca parametrii instructiunii **DRAW** pot fi si negativi, spre deosebire de instructiunile **PLOT** si **CIRCLE**.

Se pot trasa de asemenea linii si puncte colorate cu ajutorul instructiunilor **PAPER**, **INK**, **FLASH**, **BRIGHT**, **INVERSE** si **OVER** intr-o instructiune **PLOT** sau **DRAW** la fel ca in instructiunile **PRINT** si **INPUT**. De notat ca aceste atribute de culoare afecteaza intreg caracterul care este parcurs de dreptele sau punctele colorate.

O alta forma a instructiunii **DRAW**, care traseaza arce de cerc este:

DRAW x,y,a

unde x si y sint folosite ca si inainte pentru a specifica punctul de sfirsit al arcului de cerc, iar a este unghiul la centru al arcului respectiv exprimat in radiani. Deschiderea arcului este spre stinga daca a este pozitiv si este spre dreapta daca a este negativ. Daca a= π atunci va fi trasat un semicerc.

Urmatoarea instructiune va desena un semicerc care incepe la punctul de coordonate 100,100 si se termina in punctul de coordonate 150,150 si are deschiderea spre dreapta:

10 PLOT 100,100: DRAW 50,50,PI.

Directia de trasare a semicercului este la inceput spre sud-est, iar la sfirsit spre nord-vest; deci ea s-a modifica cu 180 de grade adica π radiani (valoarea lui a). Rulati acest program de mai multe ori, inlocuindu-l pe PI cu -PI, PI/2, 3*PI/2.

Ultima instructiune grafica este **CIRCLE** care traseaza un

cerc intreg. Se specifica coordonatele centrului si raza astfel:

```
CIRCLE x,y,raza
```

Ca si pentru PLOT si DRAW se pot folosi atributele de culoare dupa instructiunea CIRCLE.

Functia POINT da indicatii asupra existentei unui punct de culoare INK diferita de a hirtiei (PAPER). Rezultatul este 0 daca punctul are culoarea hirtiei sau 1 daca punctul are culoarea data de INK.

Incercati:

```
CLS: PRINT POINT (0,0):PLOT 0,0:PRINT POINT (0,0)
```

Pentru a urmarii efectul atributelor de culoare OVER si INVERSE tastati urmatoarele exemple:

```
PLOT 0,0: DRAW OVER 1;255,175
```

sau:

```
PLOT 0,0: DRAW 255,175
```

aceasta dreapta se poate sterge cu:

```
PLOT 0,0: DRAW INVERSE 1;255,175
```

Acum incercati:

```
PLOT 0,0: DRAW OVER 1;255.175
```

si incercati sa o stergeti cu:

```
DRAW OVER 1;-255,-175
```

Stergerea incompleta se datoreaza faptului ca la trasarea dreptei nu se folosesc aceleasi puncte la dus ca si la intors.

O metoda mai putin obisnuita de a obtine nuante este de a suprapune doua culori normale intr-un singur caracter de 8*8, folosind caracterele redefinibile (UDG). Rulati programul:

```
10 FOR n = 0 TO 6 STEP 2
20 POKE USR "a"+n,BIN 01010101:POKE USR "a"+n+1,BIN 10101010
30 NEXT n
```

care va defini un caracter in forma de tabla de sah. Daca afisezi acest caracter (modul grafic, apoi a) cu INK rosu pe PAPER galben se va obtine un patrat portocaliu.

Adeseori in timpul rularii unui program aveti nevoie de comanda PAUSE care opreste sistemul pentru un anumit timp. Forma acestei instructiuni este:

PAUSE n

unde n este perioada de timp exprimata in 1/50 secunde pentru care se opreste sistemul. O pauza poate fi scurtata prin apasarea oricarei taste.

Urmatorul program simuleaza functionarea secundarului unui ceas:

```
10 REM Trasarea cadranului ceasului
20 FOR n = 1 TO 12
30 PRINT AT 10-10*COS(n/6*PI),16+10*SIN(n/6*PI);n
40 NEXT n
50 REM Acum pornim ceasul
60 FOR t = 0 TO 200000: REM t este timpul in secunde
70 LET a = t/30*PI: REM a este unghiul secundarului in rad
80 LET sx = 80*SIN a: LET sy = 80*COS a
200 PLOT 128,88: DRAW OVER 1; sx,sy: REM Traseaza secundarul
210 PAUSE 42
220 PLOT 128,88: DRAW OVER 1; sx,sy: REM Sterge secundarul
400 NEXT t
```

Acest ceas va functiona ...55,5 ore, modificabil in linia 60. In linia 210 v-ati fi asteptat probabil la PAUSE 50 (adica o secunda) dar calculatorul consuma timp si pentru calcule. Acest ceas are o eroare de cca 30 de minute zilnic.

Exista un mod mult mai precis de a masura timpul, folosind continutul anumitor locatii de memorie din zona variabilelor de sistem. Datele stocate pot fi folosite (citite din memorie) cu ajutorul functiei PEEK. Expresia folosita este:

(65536*PEEK 23674 + 256*PEEK 23673 + PEEK 23672)/50,

care ne da numarul de secunde de cind computerul a fost pornit (dupa 3 zile si 21 de ore numarul de mai sus se reia de la zero).

In continuare este dat programul revizuit:

```
10 REM Trasam cadranul ceasului
20 FOR n = 1 TO 12
30 PRINT AT 10-10*COS(N/6*PI),16+10*SIN(N/6*PI);n
40 NEXT n
50 DEF FN t()=INT((65536*PEEK 23674+256*PEEK 23673+
   PEEK 23672)/50): REM nr. de secunde de la pornire
100 REM Acum pornim ceasul
110 LET t1 = FN t()
120 LET a = t1/30*PI: REM a = unghiul secundarului in rad
130 LET sx = 72*SIN a: LET sy = 72*COS a
140 PLOT 131,91:DRAW OVER 1;sx,sy: REM deseneaza secundarul
200 LET t = FN t()
210 IF t<=t1 THEN GO TO 200: REM Asteapta o secunda
220 PLOT 131,91: DRAW OVER 1;sx,sy: REM Sterge secundarul
230 LET t1=t: GO TO 120
```

Ceasul intern folosit de acest program are o acuratete de 0.01%, adica maxim 10 secunde pe zi. Este de mentionat ca ceasul se opreste in timpul instructiunii **BEEP**, sau in timpul operatiilor cu casetofonul, imprimanta sau altor echipamente folosite de calculator.

Numerele PEEK 23674, PEEK 23673 si PEEK 23672 se incrementeaza succesiv (la fiecare 1/50 secunde ultimul, respectiv la fiecare 256*1/50 secunde al doilea si in sfirsit la fiecare 65536*1/50 secunde primul).

Functia **INKEY$** nu are argumente; ea citeste tastatura. Rezultatul sau este caracterul generat prin apasarea tastei respective. Incercati programul:

```
10 LET a$=INKEY$: IF a$="" THEN GO TO 10
20 PRINT AT 10,15;a$
30 GO TO 10
```

In linia 10 se asteapta apasarea unei taste.

Difuzorul calculatorului este actionat de instructiunea

BEEP. Ea are forma:

BEEP durata,inaltime

Durata si inaltimea sint numere sau expresii numerice. Durata este data in secunde, iar inaltimea in semitonuri fata de DO de baza, caruia ii corespunde numarul 0.

Introduceti urmatorul program:

```
10 PRINT "Frere Gustav"

20 DATA 1,0,1,2,.5,3,.5,2,1,0

30 DATA 1,0,1,2,.5,3,.5,2,1,0

40 DATA 1,3,1,5,2,7

50 DATA 1,3,1,5,2,7

60 DATA .75,7,.25,8,.5,7,.5,5,.5,3,.5,2,1,0

70 DATA .75,7,.25,8,.5,7,.5,5,.5,3,.5,2,1,0

80 DATA 1,0,1,-5,2,0

90 DATA 1,0,1,-5,2,0

100 RESTORE: FOR i=1 TO 36: READ a,b

110 BEEP a,b: NEXT i
```

Pentru a urmari cum se schimba cheia in care este cintata melodia trebuie introdusa o variabila ca in programul urmator:

```
10 BEEP 1,key+0:BEEP 1,key+2: BEEP .5,key+3:BEEP .5,key+2:
   BEEP 1,key+0
```

Inainte de a rula programul (cu comanda **GO TO 10**) trebuie initializata variabila key la valoarea corespunzatoare. De exemplu 0 pentru C minor, 2 pentru D minor, 12 pentru C minor de sus samd. La fel se pot obtine melodii mai lente sau mai ritmate introducind o variabila pentru durata sunetului.

Pentru a ne da seama de limitele admise pentru inaltimea sunetului, introduceti urmatorul program:

```
10 FOR n=0 TO 100: BEEP .5,n: NEXT n
```

Programul se va termina cu mesajul de eroare **B integer out of range**. Pentru a afla limita la care s-a ajuns, tastati

```
PRINT n
```

Procedati la fel pentru limita inferioara.

Pentru a obtine confirmarea sonora la tastatura, dati comanda:

```
POKE 23609,n
```

unde n este durata sunetului obtinut la apasarea oricarei taste si poate lua valori intre 0 si 255 (se recomanda 20).

Semnalul obtinut la difuzor poate fi intensificat cu un amplificator.

Exercitii

1. Desenati cercuri si elipse folosind instructiunile SIN si COS. Incercati:

```
10 FOR n=0 TO 2*PI STEP PI/180
20 PLOT 100+80*COS n,87+80*SIN n
30 NEXT n
40 CIRCLE 150,87,80
```

2. Este ilustrat in continuare un program care traseaza graficele pentru majoritatea functiilor:

```
10 PLOT 0,87:DRAW 255,0: REM Trasare axa x
20 PLOT 127,0:DRAW 0,175: REM Trasare axa y
30 INPUT s,e$
35 LET t=0
40 FOR f=0 TO 255
50 LET x=(f-128)*s/128: LET y=VAL e$
60 IF ABS y>87 THEN LET t=0: GO TO 100
70 IF NOT t THEN PLOT f,y+88: LET t=1: GO TO 100
80 DRAW 1,y-oldy
100 LET oldy=INT(y+.5)
110 NEXT f
```

La inceput se introduce numarul n care semnifica limitele intervalului -n,+n iar ulterior se introduce functia sub forma unui sir de caractere.

Luati, de exemplu n=10 si e$="10*TAN x"

3. Folositi functia INKEY$ impreuna cu PAUSE astfel:

```
10 PAUSE 0
20 PRINT INKEY$;
30 GO TO 10
```

4. Realizati un program care sa cinte gama DO major.

Rezumatul lectiei 11

- PLOT, DRAW, CIRCLE, POINT, pixel
- PAUSE, INKEY$, PEEK
- BEEP

LECTIA nr.12

In principal, operatiile de lucru cu banda magnetica (un casetofon obisnuit) sint salvarea programelor sau datelor pe banda si incarcarea lor.

SAVE

Aceasta instructiune serveste pentru stocarea pe banda magnetica a programelor si datelor in ceea ce vom numi fisiere.

1. Pentru programele **BASIC**, trebuie utilizata urmatoarea forma (sintaxa) a instructiunii:

SAVE nume LINE numar

Un program astfel salvat, la incarcare va porni automat de la linia "numar". Daca optiunea **LINE** lipseste, pornirea automata nu mai are loc (se realizeaza salvarea programului, care poate fi apoi incarcat si rulat cu **RUN** sau **GO TO...**).

2. Pentru portiuni de memorie (octeti - programe in cod masina), instructiunea are forma:

SAVE nume CODE start,lungime

Cu aceasta instructiune se salveaza, incepind de la adresa "start", un numar de octeti egal cu "lungime".

3. Pentru a salva pe banda imaginea de pe ecran, folositi:

SAVE nume SCREEN$

care este echivalenta cu:

SAVE nume CODE 16384,6912

4. Tablourile se pot salva astfel:

SAVE nume DATA litera ()

in cazul tablourilor numerice si:

SAVE nume DATA litera$ ()

pentru tablourile sir de caractere.

Aceasta forma permite o serie de subtilitati (schimbarea

numelui tabloului, schimbarea dimensiunilor etc).

Observatii:

- pentru instructiunea SAVE, "nume" este un sir de caractere nevid de maxim 10 caractere (oricare);
- numerele "start" si "lungime" sint obligatorii;
- optiunea LINE este facultativa.

VERIFY

Aceasta instructiune verifica identitatea dintre fisierul citit de pe banda si ceea ce se gaseste in memorie. Sintaxa este asemanatoare cu cea de la instructiunea SAVE.

1. Pentru programe BASIC avem:

```
VERIFY nume
```

2. Pentru portiuni de memorie (octeti) avem:

```
VERIFY nume CODE start,lungime
```

sau:

```
VERIFY nume CODE start
```

sau:

```
VERIFY nume CODE
```

3. De asemenea avem:

```
VERIFY nume SCREEN$
```

care este echivalenta cu:

```
VERIFY nume CODE 16384,6912
```

4. Pentru tablouri avem:

```
VERIFY nume DATA litera ()
```

sau:

```
VERIFY nume DATA litera$ ()
```

Observatie

"nume" poate fi un sir vid, simbolizat prin doua ghilimele apropiate, astfel: "" ; in acest caz se incarca primul fisier de tipul specificat de pe banda.

LOAD

Aceasta instructiune este folosita pentru a incarca de pe banda noi informatii.

1. In cazul programelor BASIC si a variabilelor avem:

```
LOAD nume
```

Folosind o asemenea instructune se sterge complet vechiul program si variabilele (ca si NEW) si le incarca pe cele de pe banda. Daca salvarea respectivului fisier s-a facut cu optiunea LINE numar, dupa incarcare, programul va porni automat de la linia "numar".

2. In cazul portiunilor de memorie vom folosi:

```
LOAD nume CODE start,lungime
```

sau:

```
LOAD nume CODE start
```

sau:

```
LOAD nume CODE
```

3. In cazul tablourilor folosim:

```
LOAD nume DATA litera ()
```

sau:

```
LOAD nume DATA litera$ ()
```

Aceasta instructiune sterge din memorie tabloul cu numele "litera" si il inlocuieste cu cel incarcat de pe banda.

```
MERGE
```

Utilizarea acestei instructiuni se limiteaza exclusiv la programele BASIC si variabile. Se incarca de pe caseta noul program si noile variabile, fara a sterge decit liniile care au acelasi numar de linie si variabilele care au acelasi nume cu cele corespunzatoare din programul incarcat.

Exemple

1. Introduceti urmatorul program:

```
10 REM test1

20 PRINT "program de test banda"

30 OVER 1: PLOT 80,25
40 DRAW 100,100,12345*PI
```

si salvati-l prin comanda:

```
SAVE "test1" LINE 5
```

Programul poate fi acum verificat prin:

```
VERIFY "test1"
```

sau:

```
VERIFY ""
```

Dati comanda:

```
NEW: LOAD ""
```

si observati ca programul porneste automat.

Acum dati din nou **NEW** si apoi introduceti:

```
10 REM "test2"
40 DRAW 100,100,777*PI
```

Nu rulati acest program (oricum nu merge !). Salvati-l cu comanda:

```
SAVE "test2"
```

Acum incarcati programul "test1" cum s-a aratat mai sus si apoi dati comanda:

```
MERGE "test2"
```

incarcind astfel al doilea program.

Observati ca liniile 10 si 40 care existau si in vechiul program (test1) au fost inlocuite, iar celelalte au ramas neschimbate.

Acum puteti da comanda **RUN**.

Incarcati din nou programul "test1". Puteti salva imaginea de pe ecran folosind:

```
SAVE "imagine" SCREEN$
```

sau:

```
SAVE "imagine" CODE 16384,6912
```

Verificarea corectitudinii salvarii imaginii-ecran se poate realiza astfel:

```
VERIFY "" SCREEN
```

sau:

```
VERIFY "" CODE
```

Pentru a salva doar a doua treime din ecran (de exemplu) dati comanda:

```
SAVE "imag 2/3" CODE 18342,2048
```

Acum dati CLS si apoi urmatoarele comenzi:

```
LOAD "" CODE 16384

LOAD "" CODE 18432

LOAD "" CODE 20080
```

LUCRUL CU IMPRIMANTA

Daca folositi un SPECRTUM si o imprimanta ZX Printer, nu veti avea nici o problema deosebita, altfel trebuie sa va asigurati ca instructiunile urmatoare functioneaza si in configuratia Dvs.

Instructiunile de lucru cu imprimanta sint **LPRINT**, **LLIST** si **COPY**.

LPRINT si **LLIST** functioneaza exact ca si **PRINT** si **LIST**, cu deosebirea ca scrierea se va face la imprimanta. (Bineinteles ca nu puteti scrie cu diferite culori sau folosi **FLASH**). Mai trebuie notat ca nu puteti folosi optiunea **AT**, in schimb puteti folosi **TAB**.

In functie de tipul imprimantei pe care lucrati, puteti trimite diferite caractere de control cu ajutorul functiei **CHR$**. Introduceti comanda:

```
LPRINT CHR$(12)
```

si apoi urmatorul program:

```
10 FOR i=0 TO 31

20 FOR j=0 TO i

30 LPRINT j;

40 NEXT j

50 LPRINT

60 NEXT i
```

Incarcati programul "testi" din paragraful precedent apoi dati comanda **COPY**. Imaginea de pe ecran va fi tiparita la imprimanta. Puteti face acest lucru cu ecranele de prezentare (**SCREEN-mode**) ale multor jocuri.

CLEAR

Instructiunea **CLEAR** are urmatoarele efecte:
- sterge toate variabilele
- sterge ecranul (ca si **CLS**)
- reseteaza pozitia **PLOT** in coltul stinga-jos
- sterge stiva pentru apelurile **GOSUB** (deci aceasta instructiune nu se va folosi in subrutine !)

Sintaxa instructiunii este urmatoarea:

CLEAR

sau:

CLEAR numar

in acest ultim caz pe linga efectele enumerate mai sus, variabila de sistem **RAMTOP** va lua valoarea "numar" (maxim 65535). **RAMTOP** este un numar ce ne arata care este ultima locatie de memorie alocata programelor **BASIC**.
Pentru a intelege acest lucru, dati succesiv comenzile:

```
POKE 30000,10

PRINT PEEK 30000

NEW

PRINT PEEK 30000
```

se observa ca utilizind instructiunea **NEW** octetul de la adresa 30000 s-a sters. Incercati acum succesiunea de comenzi:

```
CLEAR 30000-1

POKE 30000,10

PRINT PEEK 30000

NEW

PRINT PEEK 30000
```

Daca programul **BASIC** este prea mare, puteti cistiga inca putina memorie (renuntind la **UDG**-uri) prin:

CLEAR 65535

USR

Pentru a putea folosi programe scrise in cod masina (direct in limbajul lui Z80, microprocesorul in jurul caruia este construit acest tip de calculatoare), se utilizeaza functiile USR astfel:

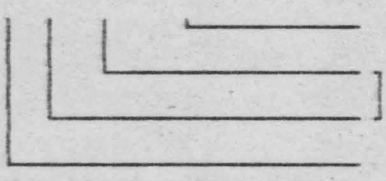
USR adr

in care argumentul "adr" reprezinta adresa de intrare in programul scris in cod masina (nu neaparat inceputul sau). Functia USR executa deci programul scris in cod masina de la adresa "adr", iar la revenirea in BASIC returneaza valoarea registrului BC a microprocesorului Z80.Sa luam, de exemplu, urmatorul program, scris in cod masina:

```
ld      bc,99       ; incarca registrul BC cu 99
ret                 ; revine in BASIC
```

Asamblarea acestui program se reduce la 4 octeti, avind semnificatiile zecimale:

```
1  99  0    201
```



Pentru a rula acest program in cod masina introduceti:

```
 5 CLEAR 30000-1
10 FOR i=0 TO 3
20 READ a: POKE 30000+i,a
30 NEXT i
40 DATA 1,99,0,201
```

Programul scris in cod masina se va gasi la adresa 30000 si va avea o lungime de 4 octeti. Pentru a-l putea rula, dati comanda:

```
PRINT USR 30000
```

Rezultatul va fi bineinteles 99. Programul scris in cod masina se va putea salva cu:

```
SAVE "progr-cod" CODE 30000,4
```

Pentru ca acest program in cod masina sa ruleze automat (ca si jocurile), introduceti urmatorul program **BASIC** care sa incarce programul scris in cod masina de pe banda si sa il si porneasca:

```
10 REM loader

20 LOAD "" CODE 30000

30 PRINT USR 30000
```

Salvati acest program cu:

```
SAVE "loader" LINE 1
```

Programul "loader", odata incarcat, va porni automat, va incarca programul in cod masina (nu uitati sa efectuati manevrele corespunzatoare la casetofon !) si il va rula.

In general este important sa rulam un program in cod masina, si nu sa aflam valoarea registrului BC; de aceea va veti putea intilni si cu urmatoarele forme:

```
RANDOMIZE USR adr
```

sau:

```
LET a=USR adr
```

sau inca:

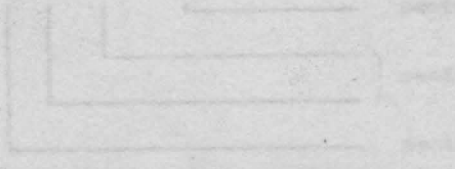
```
RUN USR adr
```

etc.

Rezumatul lectiei 12

- SAVE, LOAD, MERGE, VERIFY
- LPRINT, LLIST, COPY
- CLEAR
- USR

ALPHA Ltd. va multumeste pentru atentie !



LEI 100