Contents

Introducere **11**

0 Terminologie **13**

1 Ecranul **21**
1.1 `cuvânt cheie` . . . 21
1.2 `LIST` . . . 22
1.3 `PRINT` . . . 23
1.4 Exerciții . . . 26

2 Mai multă dinamică **27**
2.1 `RUN` . . . 27
2.2 `LET` . . . 28
2.3 Concatenarea șirurilor . . . 29
2.4 Exerciții . . . 30

3 Interacțiunea cu calculatorul **31**
3.1 `STOP` . . . 31
3.2 `CONTINUE` . . . 32
3.3 `INPUT` . . . 32
3.4 Exercițiu rezolvat . . . 33
3.5 Exerciții . . . 34

4 Culoare **35**
4.1 `GOTO` . . . 35
4.2 `BORDER` . . . 36
4.3 `PAPER` . . . 36
4.4 `INK` . . . 37
4.5 Exerciții . . . 37

5 Curățenie și variație **38**
5.1 CLS . . . 38
5.2 CLEAR . . . 39
5.3 NEW . . . 39
5.4 IF - THEN . . . 40
5.5 Exercițiu rezolvat . . . 42
5.6 Exerciții . . . 42

6 Matematici din plin **43**
6.1 REM . . . 43
6.2 PAUSE . . . 44
6.3 ABS . . . 44
6.4 INT . . . 44
6.5 SQR . . . 45
6.6 SGN . . . 45
6.7 EXP . . . 45
6.8 LN . . . 45
6.9 Exercițiu rezolvat . . . 46
6.10 Exerciții . . . 46

7 Mai multe dimensiuni **48**
7.1 DIM . . . 49
7.1.1 DIM pentru variabile numerice . . . 49
7.1.2 DIM pentru variabile de tip șir . . . 50
7.2 Exercițiu rezolvat . . . 51
7.3 Exercițiu . . . 52

8 Cicluri **53**
8.1 FOR - TO - STEP . . . 53
8.2 NEXT . . . 53
8.3 Exercițiu rezolvat . . . 57
8.3.1 FOR v=x TO y STEP z . . . 58
8.3.2 NEXT v . . . 59
8.4 Exerciții . . . 60

9 Întâmplare și interacțiune **61**
9.1 RND . . . 61
9.2 RANDOMIZE . . . 62
9.3 Exercițiu rezolvat . . . 62

9.4 INKEY$ . . . 63
9.5 Exercițiu rezolvat . . . 64
9.6 Exerciții . . . 65

10 Din nou culori și șiruri **66**
10.1 INVERSE . . . 66
10.2 BRIGHT . . . 67
10.3 FLASH . . . 68
10.4 OVER . . . 68
10.5 TO . . . 69
10.6 Exercițiu rezolvat . . . 70
10.7 VAL . . . 71
10.8 Formatul științific (exponențial) . . . 72
10.9 Exerciții . . . 73

11 Logică, printre altele **74**
11.1 NOT . . . 75
11.2 OR . . . 76
11.3 AND . . . 76
11.4 Exercițiu rezolvat . . . 79
11.5 LEN . . . 80
11.6 Exercițiu rezolvat . . . 80
11.7 VAL$ . . . 81
11.8 Exerciții . . . 81

12 Grafică **83**
12.1 PLOT . . . 84
12.2 DRAW . . . 84
12.2.1 DRAW *x*,*y* . . . 85
12.2.2 DRAW *x*, *y*, *u* . . . 85
12.3 CIRCLE . . . 86
12.4 Acțiunea culorilor . . . 86
12.5 POINT . . . 87
12.6 CHR$ . . . 88
12.7 CODE . . . 91
12.8 Exercițiu rezolvat . . . 91
12.9 Exerciții . . . 92

13 Subrutine și trigonometrie **93**
13.1 PI . . . 93

13.2 GOSUB . . . 94
13.3 RETURN . . . 94
13.4 SIN . . . 97
13.5 COS . . . 97
13.6 TAN . . . 97
13.7 ASN . . . 97
13.8 ACS . . . 98
13.9 ATN . . . 98
13.10Exerciţiu rezolvat . . . 98
13.11Exerciţii . . . 103

14 Manipulări de date şi sunet **104**
14.1 READ . . . 104
14.2 DATA . . . 105
14.3 RESTORE . . . 105
14.4 Relaţii de ordine pentru şiruri . . . 106
14.5 BEEP . . . 107
14.6 ATTR . . . 108
14.7 Exerciţiu rezolvat . . . 110
14.8 Exerciţii . . . 110

15 Memoria **111**
15.1 POKE . . . 114
15.2 PEEK . . . 114
15.3 Harta memoriei . . . 115
15.4 USR . . . 118
15.4.1 USR *adresa* . . . 118
15.4.2 USR *şir* . . . 118
15.5 Exerciţiu rezolvat . . . 118
15.6 Exerciţii . . . 119

16 Prin măruntaiele HC-ului **120**
16.1 BIN . . . 120
16.2 STR$ . . . 121
16.3 Exerciţiu rezolvat . . . 122
16.4 IN . . . 122
16.5 OUT . . . 123
16.6 Exerciţii . . . 125

17 Casetofonul **126**

17.1 SCREEN$ . . . 126
17.2 DEF FN . . . 127
17.3 FN . . . 127
17.4 SAVE . . . 129
17.4.1 SAVE *numeprg* . . . 129
17.4.2 SAVE *numeprg* LINE *etch* . . . 129
17.4.3 SAVE *numeprg* SCREEN$ . . . 130
17.4.4 SAVE *numeprg* CODE *adresa*, *lung* . . . 130
17.4.5 SAVE *numeprg* DATA *var* () . . . 130
17.5 LOAD . . . 130
17.5.1 LOAD *numeprg* . . . 130
17.5.2 LOAD *numeprg* SCREEN$ . . . 130
17.5.3 LOAD *numeprg* CODE *adresa*, *lung* . . . 130
17.5.4 LOAD *numeprg* CODE *adresa* . . . 131
17.5.5 LOAD *numeprg* CODE . . . 131
17.5.6 LOAD *numeprg* DATA *var*() . . . 131
17.6 MERGE . . . 131
17.7 VERIFY . . . 132
17.8 Exercițiu rezolvat . . . 132
17.9 Exerciții . . . 133

18 Periferice **134**
18.1 LLIST . . . 135
18.2 LPRINT . . . 135
18.3 COPY . . . 136
18.4 Drivere . . . 136
18.5 OPEN# . . . 138
18.6 CLOSE# . . . 138
18.7 Discul . . . 140
18.8 Exerciții . . . 141

A Erorile **142**

B Setul de caractere HC **143**

C Harta memoriei **149**

D Codurile culorilor **150**

E Baze de numerație **152**

F Reprezentări interne **154**

G Rezolvările exercițiilor **160**
G.1 Capitolul 1 . . . 160
G.2 Capitolul 2 . . . 160
G.3 Capitolul 3 . . . 161
G.4 Capitolul 4 . . . 161
G.5 Capitolul 5 . . . 162
G.6 Capitolul 6 . . . 162
G.7 Capitolul 7 . . . 165
G.8 Capitolul 8 . . . 166
G.9 Capitolul 9 . . . 167
G.10 Capitolul 10 . . . 168
G.11 Capitolul 11 . . . 170
G.12 Capitolul 12 . . . 171
G.13 Capitolul 13 . . . 175
G.14 Capitolul 14 . . . 178
G.15 Capitolul 15 . . . 180
G.16 Capitolul 16 . . . 182
G.17 Capitolul 17 . . . 184
G.18 Capitolul 18 . . . 185

H Variabilele sistem la nivel de bit. Jonglerii cu variabile sistem **186**
H.1 Variabilele sistemului BASIC . . . 186
H.2 Jonglerii cu variabile sistem . . . 189

Index **193**

Introducere

Aveți în față un HC. Dacă nu aveți aduceți-l acum. Dacă nu v-ați cumpărat, purcedeți. Dacă NU, aruncați cartea aceasta!

Nu se poate face programare la modul abstract, pe hârtie, ca problemele de fizică. Oricât ar părea de ciudat, informatica, prin metodele ei, este adesea o știință experimentală! Iar eu vă invit să descoperiți HC-ul experimentând.

Nu sunt multe lucruri asemenea programelor, pe care să le poți face și desface așa de ușor, pentru că nu sunt prea multe lucruri care să fie făcute din nimic! Nimic material, vreau să zic. Calculatoarele ne arată că există un ceva imaterial cu care putem opera și care se poate manifesta prin rezultate palpabile. Eu personal sunt pasionat de informatică fiindcă ea îmi arată că efectiv pot transforma lucrurile folosind „doar" informație. Un calculator alcătuit dintr-o grămadă de sârme și plastic, se transformă cu niște programe puse pe el într-o mașinărie colosală, de fiecare dată alta! Să nu credeți că astea sunt basme: există o întreagă teorie a mașinilor virtuale care se ocupă de astfel de probleme — cum pot face ceva să pară altceva, folosind doar informație.

Că în BASIC nu se poate scrie vreo aplicație profesională este un lucru care devine evident oricui știe și un limbaj mai serios. Însă HC-ul este un calculator drăguț, pentru care s-au scris o sumedenie de jocuri interesante. Mi se pare calculatorul ideal pentru un puști care începe să învețe să „butoneze". Asta pentru că partea mai atractivă (grafica) este bine dezvoltată. Și e și mult mai ieftin ca un PC.

Între persoanele implicate în activitatea didactică din informatică există o lungă polemică privitoare la BASIC. Problema principală nu este dacă limbajul este sau nu bun, ci dacă este sau nu dăunător. Nu puțini sunt cei care afirmă că BASIC este un limbaj care nu încurajează gândirea disciplinată și, ca atare, poate să ducă la deprinderi nefaste pentru cineva care vrea să lucreze ceva mai des informatică. Eu cred că BASIC HC este un limbaj suficient de bun pentru primul limbaj pe care îl învață cineva. În primul rând pentru că este foarte simplu, iar acest lucru nu este de loc de neglijat. A începe informatica cu BASIC înseamnă a începe urcușul muntelui cu panta cea mai lină. Nu toată lumea vrea să facă alpinism. Pe de altă parte, e suficient de distractiv ca să stimuleze perseverența și curiozitatea cuiva.

(Dacă sunteți nou în informatică, săriți vă rog peste următoarele două paragrafe)

De altfel, rog ca nimeni să nu se oprească aici. Să preia din HC ceea ce este bun. Iar eu cred că are multe lucruri „tari"; pentru cel care știe asamblare Z80, citirea programelor ce constituie interpretorul BASIC din ROM poate fi o lecție plină de învățăminte. Mie mi-a plăcut mai ales cum e scris și folosit evaluatorul — un adevărat emulator de virgulă flotantă — și partea de canale, care este un concept de o putere egală cu cea din cele mai bune sisteme de operare.

Dacă ne uităm la aplicațiile care se dezvoltă în ultima vreme pe PC-uri — cu jocuri de 150Mo, atunci programele de pe SPECTRUM, utilizând numai 48K de memorie, sunt niște

capodopere. Cum observa un informatician american: „oamenii au uitat să scrie programe scurte!"

Să vorbim acum despre această carte. Ea se vrea în același timp un manual (care să prezinte lucrurile într-o ordine progresivă, fără a se baza pe noțiuni pe care nu le-a explicat deja) dar unul complet (adică să spună tot ceea ce ține de BASIC HC). Aceste două cerințe sunt foarte greu de împăcat. Cred că totuși am reușit, folosind o șmecherie: de fiecare dată când o noțiune trebuia introdusă, dar necesita cunoștințe ulterioare, am amânat explicarea unora din proprietățile ei. În text sunt foarte multe trimiteri înainte (de genul „vezi Capitolul 18"). Ei bine, acestea pot fi complet ignorate de cititor la prima trecere. Le-am pus doar ca să se știe că voi reveni asupra unor aspecte, sau că ele vor fi deplin înțelese abia mai târziu. Citiți cartea în ordine, fără a vă lăsa distras de ele.

Anexe sunt multe, dar interesante. Acolo sunt prezentate și soluțiile tuturor exercițiilor, pe care am încercat să le fac cât mai amuzante, dar totodată și ilustrative pentru diverse tehnici de programare.

Cartea aceasta am scris-o în urmă cu opt ani, când eram în liceu. Atunci apariția ei ar fi fost categoric mult mai utilă (pentru că PC-uri vedeai mai rar la noi), dar din păcate și imposibilă (alte vremuri...). După ce mi-am făcut o socoteală am zis că poate nici acum nu-i prea târziu și am zis să încerc marea cu degetu'.

De un lucru îmi pare rău: pe vremea aceea nu știam cât de importantă este o bibliografie! Așa că nu mai știu de pe unde am preluat informațiile pe care vi le expun. O mare parte sunt originale, rezultând din experimentele mele, dar datorez mult manualelor calculatorului **SPECTRUM**. O sursă neprețuită a fost *The Complete Spectrum ROM Disassembly* de Ian Logan și Frank O'Hara, publicată de Melbourne House în 1983. Rezolvarea unuia din exerciții (de asta îmi aduc aminte) se datorește prietenului mei Cristi Frâncu (Exercițiul 2 din Capitolul 13). Mulțumesc, Cristi.

Iată că am ajuns la partea mulțumirilor. În primul rând acestea se cuvin adresate domnului profesor Sorin Tudor, care m-a îndemnat acum opt ani să scriu această carte. A fost un lucru foarte important pentru mine, pentru că mi-a arătat că pot să fac singur o grămadă de lucruri (cred că fiecare dintre noi poate...).

Nu pot să precizez influențe concrete, dar sunt sigur că au fost o mulțime în acea perioadă, din partea colegului meu Mihai Boicu. Îi mulțumesc și lui.

În fine, vă mulțumesc dumneavoastră pentru efortul financiar depus și încrederea acordată prin cumpărarea acestei cărți. Sper să nu vă pară (foarte) rău.

Mihai Budiu

București, 10 August 1995

Chapter 0

Terminologie

Un capitol plictisitor, dar un rău necesar cu privire la:

- programe;
- comenzi și funcții;
- setul de caractere;
- parametri;
- erori;
- tipuri de date;
- constante și variabile.

Pentru a indica unui calculator ce trebuie să facă, trebuie să-i exprimăm doleanțele noastre într-un anume fel. Succesiunea de operațiuni pe care el o va executa poartă denumirea de *program*. Programul este — din punct de vedere al omului — un text într-o anumită „limbă", pe care și calculatorul o înțelege. Aceasta este destul de sărăcuță în comparație cu limbile vorbite de oameni în mod normal, și de aceea numită *limbaj*. Noi o să studiem împreună limbajul pe care niște americani l-au numit BASIC. Tot așa cum limba română are o sumedenie de dialecte, așa și BASIC-ul cunoaște mai multe forme. Vom studia varianta pe care o pricepe calculatorul HC85.

„Basic" înseamnă în limba engleză „de bază". „BASIC" este în viziunea creatorilor săi prescurtarea de la „*Beginners' All-purpose Symbolic Instruction Code*", adică ceva de genul „Codul de instrucțiuni simbolice bune-la-toate pentru începători."

■

Așa cum în comunicarea dintre oameni folosim propoziții și fraze, în comunicarea cu calculatorul folosim *linii* de text care cuprind *instrucţiuni* ale limbajului în care facem comunicarea. Iată un exemplu de linie BASIC:

```
FOR i=1 TO 100 : PRINT i, i^2 : NEXT i
```

Nu ne preocupăm deocamdată de înțelesul ei. Pentru cei extrem de curioși, putem dezvălui că, în urma executării instrucțiunilor cuprinse în această linie, HC-ul trebuie să scrie pe ecran numerele de la 1 la 100 împreună cu pătratele lor. Încercați-l!

■

Calculatorul este un „sclav" perfect (când înțelege ce vrei de la el). Scopul său este să asculte comenzile primite și apoi să le execute. În BASIC îi putem impune două comportări diferite:

- pentru fiecare linie primită, el trebuie să îndeplinească imediat comenzile transmise (ca în exemplul de mai sus);
- trebuie să adune mai multe comenzi, pe care să le execute apoi într-o anumită ordine (un exemplu din categoria comunicației inter-umane: „la bani din dulap, du-te până la librăria cea mai apropiată și cumpără și pentru noi minunatul manual de BASIC pentru HC85!").

Astfel de comenzi sunt strânse laolaltă în ceea ce se numește un *program*. Iată un scurt exemplu:

```
 5 FOR k = 0 TO 21
10   PRINT AT k+5,k; PAPER 7; INK 7; "BASIC HC VA SALUTA"
17 NEXT k
19 OVER 1 : INVERSE 1 : INK 2
20 FOR i=0 TO 255 STEP 8
30   PLOT i, 0: DRAW PAPER 5; 0, 175
35   PAUSE 10
40 NEXT i
```

Ca să vedeți ce face acest program aveți o singură metodă: scrieți-l și apoi tastați RUN. Cum se întâmplă toate aceste grozăvii vom deslușI pe-ndelete de-a lungul celor 18 1/2 capitole care mai urmează.

Să vedem ce-am învățat: instrucțiunile pe care le dăm calculatorului spre executare se pot comporta în două moduri diferite. Și anume, unele dintre ele vor fi executate imediat ce au fost transmise calculatorului (la apăsarea tastei CR — *Carriage Return*, numită de asemenea ENTER; ea e plasată pe al treilea rând în dreapta), iar unele vor fi puse la păstrare pentru a fi executate mai târziu, în înlănțuire cu altele — formând un program.

Care este diferența dintre aceste două tipuri de instrucțiuni? Păi, liniile care intră în programul BASIC încep cu o *etichetă*, iar celelalte nu. Eticheta în BASIC HC85 este un număr întreg, cuprins între 1 și 9999. Priviți cele două exemple de mai sus și identificați etichetele sau lipsa lor! Ca verificare, vă spun că al doilea exemplu este format din 8 linii care au, în ordine, etichetele `5, 10, 17, 19, 20, 30, 35, 40`.

Nu am deslușit încă forma unei linii. O linie poate să conțină zero, una sau mai multe instrucțiuni BASICcare, atunci când se vor executa, o vor face în ordinea firească, de la stânga spre dreapta. În caz că o linie conține mai multe instrucțiuni, ele trebuie să fie separate de semnul „:" (două puncte). În al doilea exemplu, linia 40 conține o singură instrucțiune, pe când linia 30 are două.

■

Hai să vedem la ce folosește o etichetă.

- Pentru a identifica o linie. În programul BASIC nu pot exista două linii cu aceeași etichetă. Introducerea unei linii cu o etichetă egală cu a unei linii deja existente se soldează cu pierderea vechii linii și înlocuirea ei cu cea nouă. Pentru a elimina o linie dintr-un program BASIC, se va introduce doar eticheta ei (urmată de CR, bine-nîțeles). Folosind exemplul de mai sus, tastați 20 și apoi apăsați CR. Linia ar trebui să dispară.
- Pentru a indica ordinea de executare a liniilor. Liniile se așează în program în ordinea crescătoare a etichetelor lor, aceasta fiind și ordinea în care se execută. Nimeni nu ne obligă să le scriem în ordine crescătoare; dacă am uitat ceva, vom putea să introducem o linie nouă printre cele deja existente. Din această cauză este recomandabil să numerotăm liniile nu chiar consecutiv, ci — de exemplu — din 10 în 10. Rescrieți linia 20 și verificați că a fost pusă la locul ei, între 19 și 30.

Sau, mai deslușit, dacă avem programul:

```
10 FOR i=1 TO 10
20 NEXT i
```

putem apoi introduce

```
15 PRINT SQR i
```

ca să obținem

```
10 FOR i=1 TO 10
15 PRINT SQR i
20 NEXT i
```

■

Să trecem mai departe, examinând „cuvintele" ce formeaza limbajul BASIC. Ele se numesc *cuvinte cheie*. Putem distinge două mari clase:

Comenzile indică niște imperative adresate calculatorului. Fiecare linie începe cu o comandă. În scurtul program de mai sus, comenzi sunt `FOR`, `PRINT` și `NEXT`. Semnul „două puncte" este, de asemenea, urmat mereu de o nouă comandă (în afara cazului când este scris între ghilimele). Comenzile sunt câteodată însoțite de *parametri*, pentru a descrie mai precis acțiunea. În programul de mai sus, tot ce urmează în fiecare linie după comandă este un parametru al acesteia (în linia 15 parametru este `SQR i`; în linia 20 parametru este `i`). Vom vedea că fiecare comandă se așteaptă să fie urmată de parametri de un anumit tip, pe care îl vom indica când o vom studia.

Funcţiile indică un proces de calcul, care furnizează un rezultat. Funcțiile se folosesc întotdeauna pentru a calcula parametrii unei comenzi. Cu alte cuvinte, rezultatul pe care o funcție îl *întoarce* în urma calculului trebuie să fie folosit de „cineva" (o altă funcție sau o comandă), mai departe. În linia 15 avem funcția `SQR`. Unele funcții au nevoie de *argumente* pentru a genera un rezultat, altele nu. `SQR` din linia 15 are un argument, și anume `i`. Numărul și tipul argumentelor fiecărei funcții sunt fixate. De exemplu, funcția radical `SQR` are în mod evident nevoie de un argument al cărui radical să-l calculeze.

Interesant este faptul că, oriunde se poate afla un parametru (sau un argument), se poate afla și o *expresie* oricât de complicată care, în urma evaluării, generează un rezultat potrivit. (Puținele excepții de la această regulă vor fi semnalate pe parcurs.)

Lucrurile sunt destul de anoste, dar nu vă descurajați: începutul e mai greu, până ne formăm un vocabular comun. După aceea lucrurile merg mai repede și mai cursiv, iar termenii abstracți cu care am operat până acum devin limpezi.

■

Am spus că un program e un *text*, format din *linii* (atenție, o linie de program poate să se desfășoare pe mai multe linii de ecran sau de hârtie!). Textele, fie ele în română sau în BASIC, tot niște înșiruiri de *caractere* sunt (că nu orice înșiruire de caractere e un text, asta cred că e clar). Care sunt caracterele pe care le putem folosi pentru a ne exprima, nu e un lucru chiar evident: de pildă, limba română are ă, î, ș, ț, â, care în alte limbi nu există. De aceea, nu este lipsit de interes să vedem care sunt caracterele pe care BASIC-ul le poate folosi.

Putem împărți setul de caractere în câteva grupe:

Caractere de control (le vom discuta în Capitolul 12).

Litere mari şi mici `abcdefghijklmnopqrstuvwxyz ABCD...Z`

Cifre `0123456789`

Cuvinte cheie (acesta este un lucru specific HC-ului: un caracter nu se înfățișează neapărat ca un singur simbol grafic, ci uneori ca un cuvânt întreg); de exemplu, prin programele scrise mai sus am întâlnit:

```
FOR    PRINT    NEXT    SQR
```

Fiecare din aceste semne este în BASIC HC un singur caracter!

Semne speciale, cu două subgrupe:

semne speciale cu funcţionalitate BASIC:

```
. "  ; / , + - * ^ : ' $ = ( ) # <= >= < > <>
```

semne speciale decorative:

```
% & ! ? _ @ { | } ~  (c)  [ \ ]
```

Caractere grafice definite (arată ca un pătrat împărțit în 4, din care unele bucățele sunt colorate).

Caractere grafice definibile (le vom detalia in Capitolul 16).

Caracterul spaţiu (blanc).

Ce vreau să zic prin „funcționalitatea" unui caracter (special)? Anume că un caracter cu funcționalitate are un rol extrem de important pentru semnificația programului care îl conține. De exemplu, punctul „." se folosește în engleză (și în BASIC) acolo unde noi folosim virgula zecimală în scrierea numerelor! Atenție, deci: `23.45` și nu `23,45`!

Caracterul spațiu și caracterele de control au un statut aparte: ele sunt nesemnificative (adică pot să lipsească sau să fie oricât de multe) oriunde apar în program, cu excepția apariției între caracterele care formează o constantă. (Ce e o constantă o să explicăm un pic mai încolo.)

Ele (spațiul si caracterele de control) nu influențează nicicum execuția programului — cu sau fără ele un program face același lucru.

■

Cam asta-i cu caracterele.

Să introducem acum o convenție de notație cu ajutorul căreia vom explica *sintaxa* BASIC. Sintaxa, ca și la gramatică, este setul de reguli pe care le folosim pentru a scrie programe corecte. Există comenzi care se pot folosi în mai multe feluri, câteodată cu un parametru, câteodată fără. Când vom descrie sintaxa unei comenzi care are *parametri opționali* (adică parametri care pot să fie prezenți sau să lipsească), vom indica acest lucru punând între paranteze pătrate [ ] acei parametri. Cum [și] nu au vreo funcționalitate în BASIC, nu se vor ivi confuzii. Toate funcțiile, cu excepția uneia (`BIN`) nu au niciodată parametri opționali.

Convenția aceasta, de notare cu [ ], nu permite din păcate a se descrie foarte precis sintaxa diverselor instrucțiuni, dar este extrem de simplă. Vom suplini, dacă este nevoie, prin cuvinte lipsa ei de precizie.

Deci, când scriu că sintaxa comenzii `RUN` este:

SINTAXA: `RUN` [*eticheta*]

asta înseamnă că `RUN` se poate folosi în două feluri într-un program; fie în forma:

`RUN`

fie:

`RUN` *eticheta*

În acest exemplu *eticheta* este un parametru opțional al comenzii `RUN`.

■

Când vorbești cu un englez și i te adresezi în românește, sunt multe șanse să nu te înțeleagă. Probabil că după mai multe tentative de a comunica în acest fel cu el, o să dea din umeri. Ei bine, și calculatorul face cam la fel: când nu pricepe ce-i ceri, sau nu poate face ce-i ceri (nimeni nu-i perfect!), îți răspunde cu un *mesaj de eroare*. Când termini de tastat o linie BASIC și apeși CR, calculatorul se uită să vadă dacă linia ar putea fi o linie BASIC — adică dacă respectă sintaxa. Dacă nu, atunci nu acceptă linia și afișează un semn de întrebare clipitor în locul unde se află ceva neașteptat.

Dacă veți scrie ceva de genul:

```
FOR TO STOP
```

veți obține la apăsarea lui CR un astfel de semn:

```
FOR ? TO STOP
```

Pe de altă parte, se poate întâmpla ca linia să fie corectă din punct de vedere sintactic, dar imposibil de executat. O astfel de eroare poate fi depistată numai în timpul execuției programului de către calculator. Un exemplu tipic ar fi încercarea de a împărți ceva la zero. Dacă un lucru nu poate fi făcut de calculator, atunci executarea programului se oprește în chiar locul acela, iar utilizatorului i se arată un *mesaj de eroare*.

Mesajele încep cu un caracter care identifică cu precizie eroarea. Acesta este unul dintre caracterele 0-9 sau A-R. După acestea urmează alte informații despre eroarea obținută. Lista textelor generate în cazul erorilor și corespondența cu literele este dată în Anexa A; zgârciți

cum suntem la tastat, noi vom referi erorile numai prin caracterul lor de identificare, pentru a scurta textul.

Mesajul erorii se termină cu numărul liniei și numărul comenzii din linie la care s-a obținut eroarea. (O linie nu poate avea mai mult de 127 de comenzi cu parametrii lor, dar poate avea și comenzi „vide".)

De exemplu, încercarea de a executa linia

```
PRINT 0::::PRINT 1/0
```

va da eroarea

```
6 Number too big, 0:5
```

adică împărțire prin zero în linia 0 (o linie fără etichetă), comanda a cincea. (Ce face `PRINT` vom discuta la momentul oportun, important e că aici „/" a fost folosit pe post de împărțire, încercându-se împărțirea lui 1 la 0.)

Există un mesaj care nu corespunde unei erori în sensul rău al cuvântului, ci dimpotrivă; este vorba de mesajul `0 OK`, care indică utilizatorului că ceea ce el dăduse calculatorului de făcut s-a terminat. Se poate spune că este o eroare de „ieșire din program". Ea se deosebește de celelalte erori prin faptul că NU afectează locul repornirii programului cu `CONTINUE` (vezi Capitolul 3).

■

Acum trebuie să despicăm alt fir în patru, discutând despre tipurile de date.

De prin clasele mici ni se tot spune că nu putem aduna mere cu pere. Aceasta înseamnă că nu putem face aceleași operațiuni cu toate lucrurile ce se găsesc pe lumea asta. Și informaticienii s-au văzut siliți să clasifice valorile cu care operează programele după *tipul* lor. În termeni abstracți, un *tip de date* este definit de operațiile care se pot face cu acele date. Noi o să ne mulțumim, pentru moment, cu atât, lăsând o discuție mai amplă în seama unor alte manuale. Pentru a lucra în BASIC, ajunge să știm că *datele* pe care el le manipulează au una din două forme (sau tipuri):

- *numere*;
- *şiruri de caractere*.

■

O noțiune crucială în informatică este cea de *variabilă*. Ea se definește în opoziție cu cea de *constantă*. Constantele sunt niște date a căror valoare nu se schimbă (și nici nu se poate schimba) în cursul execuției programului. (Definiția asta nu este suficient de precisă, pentru că, după ea, programul însuși este o constantă!) Totuși nu o vom rafina, ci vom da doar exemple.

Putem avea deci patru combinații de obiecte care „conțin" date și tipurile lor:

1. constante numerice (a căror valoare e un număr);
2. constante „șir de caractere" (a căror valoare este un șir de caractere);
3. variabile numerice;

4. variabile „șir de caractere" (sau mai scurt „variabile șir").

Iată cum arată fiecare:

1. O constantă numerică apare în program cam cum apare pe hârtie: o înșiruire de cifre, care poate avea un semn (– sau +), un punct zecimal sau un exponent (despre exponenți prin Capitolul 10). Iată niște exemple concrete:

   ```
   10        1000        0.0002        -12345.5678       2e-5
   ```

   (cel din urmă conține și exponent).

   De exemplu, în linia

   ```
   FOR i = 1 TO 10 : PRINT i : NEXT i
   ```

   apar două constante numerice: 1 și 10.

2. O constantă de tip șir (de caractere) este indicată în programul BASIC prin delimitarea ei între ghilimele. Orice caracter (cu excepția lui CR) poate face parte dintr-o constantă de tip șir. Ghilimelele de la început și sfârșit nu fac parte din constantă (din valoarea ei, de fapt). Dacă în constantă trebuie să apară și ghilimele, atunci ele se vor scrie de două ori. Iată și niște exemple:

constantă	valoare
`"constanta sir"`	`constanta sir`
`"HC - 85, bun !"`	`HC - 85, bun !`
`"zic 'salut'"`	`zic 'salut'`
`"niste "" ghilimele"`	`niste " ghilimele`
`""`	(șirul nul, fără nici un caracter)
`"to be = "" a fi"""`	`to be = "a fi"`

   În ultimul exemplu, cele trei ghilimele de la sfârșit trebuie interpretate după cum urmează: ultima pereche indică sfârșitul constantei, celelalte două sunt de fapt o singura „ghilimea", dar reprezentată dublat în interiorul constantei.

■

Variabilele sunt niște obiecte informatice identificate prin numele lor, care este cu totul altceva decît valoarea lor. E ca și cum aș zice „obiectul de la mine din buzunar", sintagmă care poate să însemne câteodată batista, altădată cheile, sau mai știu eu ce (mai rar bani). „Buzunar" este numele „variabilei", iar valoarea ei este fie batista, fie cheile etc. Variabilele se diferențiază în tipuri după tipul valorilor pe care le pot avea. În BASIC avem deci variabile de tip numeric și variabile de tip șir de caractere. Pentru a evita confuziile, numele pe care o variabilă îl are arată și tipul (valorii) ei, după cum urmează:

3. Variabilele de tip numeric au numele format din litere și cifre, dar începând cu o literă. Numele nu are (teoretic) o limitare în lungime. Literele mari sunt totuna cu cele mici, iar eventualele spații și caractere de control sunt ignorate (acesta este un alt lucru cu totul specific BASIC-ului HC). Exemple de variabile numerice:

```
a
abc
A  b    C
Aceasta variabila are un nume mult prea lung pentru a fi utila
var12b
```

În aceste exemple, al doilea și al treilea nume reprezintă aceeași variabilă!

Există și specii mai bizare de variabile numerice, și anume cele `FOR-NEXT` și cele dimensionate, dar le vom analiza în Capitolele 7, respectiv 8. Deocamdată nici nu ne sunt utile.

4. În fine, variabilele de tip șir au numele format dintr-o singură literă urmată de caracterul *dolar* ($). Literele mari sunt echivalente cu cele mici. În general, în BASIC semnul dolar indică faptul că se așteptă o valoare de tip șir. El vine de la prescurtarea cuvântului englezesc pentru „șir" — „string". Pentru că există exact 26 de litere diferite, deducem că pot exista simultan cel mult 26 de variabile șir (cele numerice sunt mult mai multe!). Până acum, însă, în programele pe care le-am scris în BASIC, nu am avut vreodată nevoie de mai multe.

   Valoarea unei variabile șir poate fi — teoretic — orice șir de caractere cu lungimea între 0 (șirul nul) și 65536 de caractere, dar ea este de fapt limitată mult mai jos, de memoria disponibilă a calculatorului. Spre deosebire de constantele de tip șir, valoarea unei variabile șir poate conține și caracterul CR.

   Exemple de nume de variabile șir:

   ```
   a$          b$          c$          A$
   ```

   ultima e totuna cu prima.

   Mai există și o specie de variabile șir numite „indexate", dar le discutăm în Capitolul 8. Până atunci, uitați că există.

Of, ce de teorie! Aici se încheie prezentarea principalelor obiecte informatice cu care lucrează BASIC-ul. Din capitolul următor trecem la treabă și învățăm instrucțiunile.

Dacă ați parcurs tot acest capitol, este prea târziu ca să vă mai descurajați; de aici înainte lucrurile vor începe să fie din ce în ce mai interesante. Pe deasupra, cred că partea pe care ați terminat-o este cea mai grea din întregul manual!

Chapter 1

Ecranul

În acest capitol vom discuta în amănunt:

- `LIST` pentru a examina programele introduse;
- `scroll` – alunecarea ecranului;
- `PRINT` pentru scriere pe ecran;
- forma ecranului din punct de vedere al textului.

Vom respecta următoarea formă de *prezentare* a cuvintelor cheie din BASIC HC:

1.1 `cuvânt cheie`

ENGLEZĂ: cuvântul englezesc din care provine;
SINTAXĂ: sintaxa folosirii sale; reamintim că parametrii între [și] sunt opționali; ceea ce e scris cu *litere mici cursive* reprezintă un nume generic (a cărui semnificație o specifică în parte chiar el; de exemplu: *etichetă* va fi o valoare numerică ce reprezintă o etichetă); alte caractere trebuie reproduse întocmai;
CATEGORIE: comandă sau funcție — ce este cuvântul cheie;
DESCRIERE:

- explicații detaliate;
- erori posibile;
- condiții speciale de folosire.

■

Și acum să vedem primele instrucțiuni:

1.2 LIST

ENGLEZĂ: *to list* = a face o listă
SINTAXĂ: `LIST [#`*cale*`] [`*etch*`]`
CATEGORIE: comandă
DESCRIERE:

Despre folosirea parametrului #*cale* vom discuta abia în Capitolul 18.

Efectul comenzii `LIST`: pe ecranul monitorului atașat la calculator apar *listate* (barbarism englez) — afișate în ordine — toate liniile programului ce au eticheta mai mare sau egală cu parametrul *etch* specificat. Dacă `LIST` nu e însoțită de nici un parametru, atunci listarea începe de la prima linie a programului.

Cum se efectuează listarea?

Liniile sunt afișate una după alta, fiecare începând în stânga ecranului cu eticheta. Pentru că, în general, programele sunt ceva mai lungi decât încape pe un ecran, este interesant de văzut ce face calculatorul în acest caz.

Întotdeauna când are ceva de scris pe ecran (fie și textul unui program listat), calculatorul scrie până umple primele 22 de linii ale ecranului. Când și ultima pătrățică s-a umplut, calculatorul ne întreabă dacă suntem gata cu cititul, ca să treacă mai departe. Întrebarea este pusă astfel: pe ultima linie a ecranului calculatorul scrie `scroll?`. Acest cuvânt vine de la SCReen ROLL, adică „alunecare (rotire) a ecranului". În acest moment se așteaptă apăsarea unei taste de către utilizator.

Răspunsurile lui pot fi după cum urmează:

- apăsând una din tastele n, N, ␣ (spațiu), STOP (adică SS + a) sau BREAK (adică CS + 0) (vezi Capitolul 3), programul se oprește, afișând eroarea D.
- apăsând o altă tastă (exceptând SHIFT-urile CS sau SS singure), scrierea continuă sub ultimul rând scris anterior (deasupra lui `scroll?`, care dealtfel dispare), rândurile deja scrise urcând pe ecran. După ce prima linie care fusese scrisă pe ecran după `scroll?` ajunge la rândul ei în vârful ecranului, calculatorul va întreba din nou `scroll?`.

Revenind acum la `LIST`:

Dacă valoarea *etch* indicată nu este o etichetă posibilă între limitele 0 și 9999, survine eroarea B.

Dacă valoarea parametrului *etch* nu este un număr întreg, atunci este rotunjită la cel mai apropiat întreg.

Un alt efect important al lui `LIST` *etch* este mutarea cursorului > al *liniei curente* la linia cu eticheta *etch*. Linia curentă este cea care este adusă jos pentru modificare (*editare*) la apăsarea tastei EDIT (CS + 1).

1.3 PRINT

ENGLEZĂ: *to print* = a tipări
SINTAXĂ: `PRINT [#`*cale*`] [`*valoare*`] [;] [,] ['] [AT` *x*`,` *y*`] [TAB` *x*`]` **[culori]**
CATEGORIE: comandă
DESCRIERE:

Despre utilitatea „parametrilor" (nu sunt chiar parametri în sensul dat de noi cuvântului) *#cale* și **culori** vom discuta în Capitolele 18, respectiv 4 și 10.

`PRINT` este o instrucțiune cu care realizează modificarea imaginii (textului) de pe ecran. Denumirea „a tipări" se trage din vremurile ancestrale ale anilor cincizeci, când calculatoarele comunicau cu omul nu prin tastatură și ecran ca acum, ci prin cartele perforate și mașini electrice de scris. Atunci acțiunea de scriere din partea calculatorului era chiar o tipărire.

Principala acțiune a lui `PRINT` este de a scrie pe ecran parametrul său *valoare*. Cine este *valoare*? O expresie care are sens în BASIC și care generează un rezultat, de tip numeric sau șir. `PRINT` este una dintre puținele comenzi care tratează fără discriminare parametrii săi, în sensul că nu le impune un anumit tip. Cum arată o expresie încă nu putem spune dar, pe măsură ce vom învăța noi funcții, vom putea să alcătuim expresii din ce în ce mai complexe. Cel mai simplu tip de expresie este o constantă și are ca rezultat chiar valoarea constantei respective. Hai să vedem cum lucrează `PRINT`, pe niște exemple. Tastați:

```
PRINT "afisam un numar":PRINT 50
```

Tot expresie este și cea formată numai din numele unei variabile. O astfel de expresie are ca rezultat chiar valoarea pe care o are acea variabilă în momentul evaluării expresiei. Încă nu știm cum pot căpăta valori variabilele (vom afla în Capitolul 2), dar presupunând ca `XY=24.34` și `a$="texte"`, atunci

```
PRINT xy: PRINT a$: PRINT xY
```

va avea un efect de genul:

```
24.34
texte
24.34
```

În realitate, pentru a fi foarte riguroși, trebuie să spunem că `PRINT` convertește valoarea expresiei date din reprezentarea ei din interiorul calculatorului (acesta e un lucru complicat, la care vă sfătuiesc să vă gândiți după ce veți trece de Capitolul 15) într-o formă care să se afișeze pe ecran, și apoi face această afișare. Forma în care se afișează o valoare pe ecran poate să fie destul de complicată, după cum vom vedea studiind scrierea științifică a valorilor numerice, în Capitolul 12. Deocamdată ne e suficient să știm că `PRINT` scrie valoarea expresiei (expresiilor) care îi urmează (fiindu-i parametru), fie ea șir sau număr.

Celelalte semne care apar în sintaxa lui `PRINT` controlează poziția pe ecran la care se tipăresc feluritele expresii.

Cele trei semne speciale ' (apostrof) ; (punct și virgulă) și , (virgulă) mai sunt numite și *separatori* pentru că ele permit înlănțuirea mai multor valori în cadrul aceleiași instrucțiuni

PRINT: fiecare două valori consecutive trebue să fie separate prin cel puțin unul dintre semnele speciale. Ele diferă prin modul în care aranjează în pagină valorile succesive.

Pentru a putea să vedem cum se modifică paginarea, trebuie să vedem cum arată de fapt pagina (*ecranul*).

Ecranul se împarte în două mari porțiuni:

- prima este o centură pe margine care se numește BORDER;
- restul este centrul dreptunghiular care formează ecranul grafic.

BORDER-ul este o zonă puțin interesantă. În interiorul său nu se poate scrie nimic. Singura operațiune ce o putem face asupra ei este de a-i schimba culoarea în totalitate.

Ecranul grafic este format din mici punctulețe care se numesc *pixeli* (din englezescul *PICture CELL* = celulă de imagine). Orice imagine este compusă din astfel de punctulețe diferit colorate. Pixelii încap câte 256 pe orizontală și 192 pe verticală. Despre cum putem lucra cu fiecare din ei vom discuta abia în Capitolul 12. Pixelii sunt însă grupați câte 64, în pătrățele de 8×8. Avem 32 de astfel de pătrățele de-a lungul unei orizontale a ecranului și 24 de-a lungul unei verticale. Aceste pătrățele formează așa numita *grilă de joasă rezoluție* a ecranului. (*Rezoluția* este o măsură a numărului de semne care se pot face pe ecran; cu cât e mai mare, cu atât numărul de semne care se pot desena e mai mare). Deocamdată ne vom ocupa numai de aceste pătrățele. Orice simbol poate fi scris (cu PRINT) numai într-unul din ele.

Ecranul grafic, la rândul său, este împărțit în două bucăți, pe care le vom numi „partea de sus" și „partea de jos". De obicei partea de jos are numai două rânduri, dar poate să creasca. Partea de jos este folosită pentru introducerea datelor (acolo se editează liniile BASIC). Care este rațiunea unei atari împărțiri va fi explicat abia în ultimul capitol. Până atunci să notăm că, în mod normal, numai primele 22 din cele 24 de rânduri sunt accesibile lui PRINT.

Să vedem cum acționează separatorii. Apostroful trece pe rând nou. Dacă o instrucțiune PRINT nu se termină cu un separator, atunci se consideră implicit că se termină cu apostrof. Așa că următoarele linii vor avea efecte echivalente:

```
PRINT "un mesaj" ' "al doilea" ' "al treilea"
```

și

```
PRINT "un mesaj":PRINT "al doilea":PRINT "al treilea"
```

și anume:

```
un mesaj
al doilea
al treilea
```

Punct și virgulă se folosește în scopul tipăririi compacte, fără nici un spațiu între diferitele valori:

```
PRINT "cincizeci=";50;"saizeci=";:PRINT "60"
```

cu efectul

```
cincizeci=50saizeci=60
```

Virgula are un efect mai complicat. Valoarea tipărită după o virgulă va fi scrisă fie la începutul rândului, fie exact la jumătatea lui (în coloana 16), după cum urmează:

```
PRINT 1,2,3,4,5,6,"acum ar trebui sa vina sapte",8
```

care dă:

```
1               2
3               4
5               6
acum ar trebui sa vina sapte
8
```

Grila de joasă rezoluție își are pătrățelele numerotate după cum urmează: liniile sunt numerotate de la 0 la 23, iar coloanele de la 0 la 31. Poziția unui pătrat este complet definită dacă se dau coordonatele lui, adică linia și apoi coloana. Pătratul cu coordonatele 0, 0 se află în colțul din stânga sus al ecranului.

`TAB` se folosește pentru a crea tabele. El este urmat de o valoare numerică (care poate fi rezultatul evaluării unei expresii) ce indică din ce coloană se va continua tipărirea. Specificațiile `TAB` (și `AT`) trebuie separate și ele cu ajutorul separatorilor.

Experimentați:

```
PRINT TAB 2;"2";TAB 5;5;TAB 10;10;TAB 1;"doi"
```

care va da:

```
  2   5     10
 doi
```

După cum vedeți, `TAB` nu se întoarce înapoi; în cazul în care coloana la care s-a ajuns cu tipărirea este mai mare decât cea specificată la `TAB`, atunci acesta sare pe rând nou. `TAB` tipărește spații până la coloana specificată, ca și virgula, de altfel.

Dacă valoarea numerică ce urmează după `TAB` nu este un întreg, atunci ea se rotunjește. Dacă nu este un număr pozitiv, atunci se ia în modul. Dacă este mai mare ca 31, atunci se consideră restul împărțirii sale la 32.

`AT` este urmat de doi parametri numerici, care dau coordonatele la care se va scrie următorul simbol, în grila de joasă rezoluție. `AT` y,x trebuie să aibă pe x între 0 și 31 și pe y între 0 și 23. Valorile lor sunt rotunjite și luate în modul. Dacă una dintre valori nu „pică" între limitele expuse, instrucțiunea `PRINT AT` va da eroarea B.

Dacă valoarea lui y este între 0 și 23, dar mai mare decât (23 minus numărul de linii din „partea de jos a ecranului"), se obține eroarea 5.

Experimentați felurite amestecuri:

```
PRINT AT 2,2;2; AT 5,5; 5; AT 21
,21; 21; TAB 10;4,98,5,"titi"
```

Acum putem explica complet `scroll?`. Acest mesaj se afișează când tipărirea atinge partea de jos a ecranului.

În final, după atâtea detalii nu ne rămâne decât să vă adresăm un îndemn la noi experimente și la rezolvarea exercițiilor care urmează:

1.4 Exerciţii

1. Se presupun cunoscute valorile variabilelor `a, b, c, d, e`. Scrieți un programel care să afișeze valorile lor începând exact din colțul stânga sus al ecranului, cu câte două spații între ele.

2. Indicați care dintre următoarele suite de caractere nu poate fi numele unei variabile în BASIC HC:

   (a) `F 24`
   (b) `x2$`
   (c) `a:bc`
   (d) `xwz13`
   (e) `Q$`
   (f) `1gh3`
   (g) `p1=p`

Chapter 2

Mai multă dinamică

Acesta este un capitol tare scurt, dar bogat în învățăminte. Iată:

- `RUN` pentru a porni execuția unui program;
- semnele operațiilor aritmetice;
- `LET` pentru a da valori variabilelor;
- câte ceva despre operații cu șiruri.

2.1 RUN

ENGLEZĂ: *to run* = a alerga (calculatoarele nu merg, ci fug)
SINTAXĂ: `RUN` [*etch*]
CATEGORIE: comandă
DESCRIERE:

În caz că *etch* lipsește, valoarea sa (implicită) este 0. Dacă valoarea ei nu este un întreg, se rotunjește.

Principalul efect al executării comenzii `RUN` este pornirea în execuție a programului BASIC din calculator, începând cu prima linie ce are eticheta mai mare sau egală cu *etch*.

`RUN` are și o sumedenie de alte efecte importante, care nu trebuie trecute cu vederea (nu-i musai să le pricepeți acum, numa' să știți de ele):

- șterge toate variabilele;
- șterge ecranul (vezi `CLS`, Capitolul 5)

- face `RESTORE 0` (vezi Capitolul 14);
- șterge stiva subrutinelor (Capitolele 13, 15).

■

Semnele diverselor *operaţii aritmetice* în BASIC sunt (folosite între expresii de tip numeric produc, din valorile expresiilor, noi valori după reguli — sperăm — evidente):

Semn	Nume	Ce face
+	plus	adunare
-	minus	scădere
*	asterisc	înmulțire
/	*slash*	împărțire
^	săgeată	ridicare la putere

Dacă vreți să extrageți radical, folosiți regula care spune că radical de ordinul y din x este `x^(1/y)`.

Ca și în matematică, operațiunile nu se efectuează simplu de la stânga spre dreapta, ci fiecare are o anumită *precedenţă* și *asociativitate*. Și anume, precedența (o calitate care arată care operații se fac întâi) crește ca în matematică (de la + spre ridicare la putere). La precedențe egale operațiunile se fac de la stânga spre dreapta (asociativitate la stânga), cu excepția ridicării la putere, care este asociativă la dreapta. Experimentați diverse calcule:

```
PRINT 2, 2^(1/2), 2^(1/3), 2^(1/4), 2^(1/5)
```

pentru radicalii din 2.

N-am spus-o, dar cred că ați intuit că, iarăși ca-n matematică, se pot folosi parantezele pentru a modifica ordinea operațiilor. Se folosesc însă numai paranteze rotunde, oricâte una-ntr-alta. Exemplu:

```
PRINT 2^(3*(5+(1-4)))
```

S-ar putea să fiți dezamăgiți constatând că ridicarea la putere nu merge nicicum cu baze negative, nici măcar pentru exponenți întregi. O să vă consolez (poate), spunându-vă că matematica nici nu definește funcția putere pentru baze negative, în general. Va trebui să implementați singuri această operație (însă după Capitolul 6, în care mai învățăm câteva funcții utile).

2.2 LET

ENGLEZĂ: *to let* = a lăsa
SINTAXĂ: `LET` *var* = *expr*
CATEGORIE: comandă
DESCRIERE:

var este numele unei variabile (numerică sau șir). *expr* este o expresie care în urma evaluării generează un rezultat de același tip cu *var*.

`LET` evaluează expresia din partea dreaptă a semnului egal, apoi dă variabilei din stânga rezultatul drept valoare. Această operație se numește *atribuire*. Dacă variabila nu exista anterior, ea este creată; dacă exista, vechea ei valoare este pierdută.

Acesta este un lucru important: atâta vreme cât o variabilă nu a fost creată (cu `LET`, `INPUT` (Capitolul 3), `FOR` (Capitolul 8) sau cu `DIM` (Capitolul 7)), ea nu există, și ca atare tentativele de a folosi valoarea ei se vor solda cu eroarea 2. Încercați:

```
PRINT a
```

și veți avea eroarea:

```
2 Variable not found, 0:1
```

apoi

```
LET a=2: PRINT a
```

O expresie de genul `LET b=b+1` nu este o absurditate, pentru că nu este vorba de o egalitate, ci de o operațiune de creștere a valorii cu o unitate (*incrementare*).

Să vedem un mic exemplu:

```
  5 LET b=23 : PRINT b
  7 LET b=b+5 : PRINT b
  9 LET a=5: LET b= a * b : PRINT b
RUN
```

cu rezultatul:

```
23
28
140
```

2.3 Concatenarea şirurilor

Una din operațiile care sunt definite pe tipul șirurilor de caractere este *concatenarea* (înlănțuirea). Simbolul ei este același cu al adunării numerelor, semnul plus +. (Despre „plus" se spune că este *supraîncărcat*, pentru că face lucruri diferite cu tipuri diferite.) Remarcați că nu există posibilitate de confuzie, deoarece tipul rezultatului este dat de tipul operanzilor. Nu există operația de adunare între un șir și un număr (eroare de sintaxă). Încercați!

Cum lucrează + cu șiruri veți înțelege executând:

```
10 LET a$="gal"
20 LET b$="ben"
30 LET c$=a$ + b$
40 LET d$=b$ + a$
50 PRINT AT 0,0; "A$=" ; a$ , "B$=" ; b$ ' "A$+B$=" ; c$
55 PRINT "B$+A$="; d$
60 PRINT "A$+B$ este diferit de B$+A$ !"
```

cu efectul

```
A$=gal             B$=ben
A$+B$=galben
B$+A$=bengal
A$+B$ este diferit de B$+A$ !
```

2.4 Exerciţii

1. Realizați un program care calculează și afișează primele 5 puteri (de la 0 la 4) ale numărului 3. Tipărirea acestora se va face precizând pe coloana 0 puterea și pe coloana 16 valoarea.

2. Indicați greșelile din următorul „program”:

```
 50 LET A$="5"
 60 LET a=A$
 70 PRINT A$ - 3
 80 LET a=a+5*b
 90 LET LET=46
100 LIST -LET
```

Chapter 3

Interacţiunea cu calculatorul

Vom discuta despre:

- tasta [BREAK] pentru întreruperea programelor;
- `STOP` pentru oprirea programelor;
- `CONTINUE` pentru reluarea lor;
- `INPUT` pentru comunicarea interactivă cu utilizatorul.

■

[BREAK] este o tastă compusă din Caps Shift ([CS]) și blanc [ ]. La apăsarea acestora, dacă programul se află între două instrucțiuni, execuția programului se oprește cu una din erorile D sau L. Diferența dintre cele două mesaje este tranșată odată cu explicația instrucțiunii `CONTINUE`, în chiar acest capitol.

3.1 STOP

ENGLEZĂ: *to stop* = a (se) opri
SINTAXĂ: `STOP`
CATEGORIE: comandă
DESCRIERE:

Așa cum arată și numele ei, execuția acestei instrucțiuni se soldează cu oprirea execuției programului cu mesajul de eroare 9.

3.2 CONTINUE

ENGLEZĂ: *to continue* = a continua
SINTAXĂ: `CONTINUE`
CATEGORIE: comandă
DESCRIERE:

`CONTINUE` nu prea are sens ca parte dintr-un program. Scopul ei este de a relua executarea unui program după întâlnirea unei erori.

Dacă eroarea a fost 9 sau L, atunci execuția programului se va continua de la instrucțiunea exact următoare celei care a cauzat eroarea. Dacă eroarea a fost alta (inclusiv D), atunci `CONTINUE` va cauza reluarea instrucțiunii care a cauzat eroarea.

Dacă între producerea erorii, (moment în care programatorul de obicei se apucă să inspecteze și să modifice programul în scopul depistării/depanării greșelilor) și cel al executării instrucțiunii `CONTINUE` se petrec anumite modificări majore (de exemplu este scoasă linia în care s-a obținut eroarea), atunci `CONTINUE` ar putea să se soldeze cu eroarea N.

Atenție: dacă eroarea s-a obținut chiar în cursul executării unei instrucțiuni direct executabile (adică nu a unei linii de prin program), atunci încercarea de a relua cu `CONTINUE` va fi sortită eșecului, căci noua linie direct executabilă va fi nu cea care a dat eroarea, ci chiar cea cu instrucțiunea `CONTINUE`! Calculatorul va intra într-o buclă infinită, continuând să execute instrucțiunea `CONTINUE`! Îl puteți, bine-nțeles, opri cu BREAK.

(Tastați `PRINT 1/0` și după obținerea (în linia 0) a erorii `6 Number too big, 0:1` tastați `CONTINUE`. Acum linia 0 este chiar cea cu `CONTINUE`, deci programul s-a blocat! Îl puteți opri cu BREAK).

3.3 INPUT

ENGLEZĂ: *input* = intrare; ceea ce intră
SINTAXĂ: `INPUT [#`*cale*`] [`*valoare*`] [;] ['] [,] [LINE` *varsir*`] [`*var*`]`
`[AT` *y,x*`] [TAB` *x*`] [`**culori**`]`
CATEGORIE: comandă
DESCRIERE:

La întâlnirea acestei instrucțiuni, calculatorul va folosi partea de jos a ecranului pentru a afișa feluritele valori și pentru a cere de la utilizator introducerea unor alte valori pentru variabilele *var*. În caz că numărul de linii din partea de jos a ecranului nu ajunge, ele se vor înmulți mâncând din liniile din partea de sus, sau ridicându-le. După terminarea executării instrucțiunii `INPUT` partea de jos a ecranului este ștearsă și readusă la două linii.

În sintaxa de mai sus:

- *var* este numele unei variabile, numerică sau șir;
- *varsir* este numele unei variabile șir;
- *valoare* este o expresie ce se evaluează (de orice tip.)

Funcționarea lui `INPUT` seamănă foarte mult cu aceea a lui `PRINT`, așa că nu vom detalia rolul separatorilor și al lui `TAB`, care lucrează chiar la fel. Despre culori vom vorbi în capitolul

următor și în Capitolul 10. Să lămurim prin exemple ceea ce a mai rămas.

- `INPUT` a cere o valoare numerică de atribuit lui a ;
- `INPUT a$` cere o valoare șir de atribuit lui a$; utilizatorul va primi cursorul între ghilimele (pe care însă le poate șterge).

  La aceste două tipuri de `INPUT` utilizatorul poate să tasteze o expresie oricât de complicată, care va fi evaluată pentru a calcula valoarea de atribuit. Introducând primul caracter `STOP` ([SS]+[a]), instrucțiunea `INPUT` se va opri cu eroarea H.
- `INPUT LINE a$`, spre deosebire de `INPUT a$`, nu mai oferă ghilimelele „la vedere" utilizatorului, așa că acesta nu le mai poate șterge spre a introduce o expresie de evaluat. Toate caracterele pe care acesta le tastează vor deveni valoarea variabilei a$! Un `INPUT LINE` se oprește apăsând în timpul execuției sale [CS] + [6].

3.4 Exerciţiu rezolvat

Scrieți un program care să poată fi folosit pe post de calculator de buzunar, cel puțin cu operațiile `+ - * / ^`.

Rezolvare

```
 5 INPUT "expresia de calculat :";a
 8 PRINT A:INPUT "Daca ati citit tastati CR"; LINE a$
 9 RUN
```

Programul se bazează pe faptul că la `INPUT` putem introduce o expresie care să fie evaluată. Al doilea `INPUT LINE` asigură continuarea programului numai după apăsarea tastei [CR] (valoarea introdusă pentru a$ nu are nici o importanță), căci `RUN` șterge ecranul!

■

Ce trebuie să fac pentru a tipări la cu ajutorul lui `INPUT` valoarea unei variabile? De exemplu, întreb pe cineva cum îl cheamă:

```
INPUT "Cum va cheama ? ";n$
```

și apoi vreau să-l întreb pe n$ câți ani are. Nu pot să fac așa:

```
INPUT "cati ani aveti, ";n$;" ? ";ani
```

pentru că aceasta, în loc să scrie numele care este valoarea lui n$, ar cere o nouă valoare pentru n$. Soluția este să transform variabila într-o expresie, astfel:

```
INPUT "cati ani aveti, ";(n$);" ? ";ani
```

sau

```
INPUT "cati ani aveti, " + n$ + " ? ";ani
```

Să ne uităm puțin cum lucrează `INPUT AT` sau, echivalent, ce se întâmplă când are nevoie de mai multe rânduri. Am spus deja că urcă cele două rânduri care de obicei alcătuiesc partea de jos a ecranului, atâta cât e nevoie (dar această parte nu se mărește niciodată la mai mult de 22 de rânduri cu totul; încercarea de a trece acestă limită se va solda cu eroarea 5). Rândurile de jos tot urcă peste cele din partea de sus a ecranului, ștergându-le, până când întâlnesc *cursorul* lui `PRINT`, după care încep s-o împingă în sus. (Cursorul lui `PRINT` este locul unde `PRINT` ar trebui să scrie următorul caracter pe ecran.) Vorba lungă, amețeala omului. Hai să vedem diferența:

```
 1 PRINT "hai sa scriem ceva lunguiet pe ecran ca sa avem ce sterge"
 2 PRINT "hai sa scriem ceva lunguiet pe ecran ca sa avem ce sterge"
 3 PRINT "hai sa scriem ceva lunguiet pe ecran ca sa avem ce sterge"
 4 PRINT "hai sa scriem ceva lunguiet pe ecran ca sa avem ce sterge"
10 PRINT AT 0,0;: INPUT ''''''''''''''''
```

(Liniile 2, 3, 4 sunt linia 1 editată și cu eticheta schimbată.) Acest prgrămel va scrie ceva pe ecran, după care va urca partea de jos a ecranului, ștergând-o pe cea de sus, pentru că am mutat cursorul lui `PRINT` la 0,0.

Acum rulați din nou programul, punând în linia 10 `PRINT AT 21,0;`

3.5 Exerciţii

1. Scrieți un program care să ceară cu ajutorul lui `INPUT` o cifră și să tipărească numele ei în litere (nu-i ușor!).

2. Descrieți ordinea parcurgerii următorului program:

```
 25 LET A=4
 35 LET a=A*20
 40 RUN a
 50 PRINT "mare greseala !"
 70 STOP : RUN
 80 PRINT 90
 90 RUN a
100 STOP
```

Chapter 4

Culoare

Vom învăța să parcurgem instrucțiunile nu neapărat în ordinea firească și, de asemenea, vom adăuga un pic de culoare acestei vieți monocrome prin:

- `GOTO` pentru salturi;
- `BORDER` pentru colorarea marginii;
- `PAPER` pentru colorarea hârtiei;
- `INK` pentru cerneală.

4.1 GOTO

ENGLEZĂ: *go to* = du-te la
SINTAXĂ: `GOTO` *etch*
CATEGORIE: comandă
DESCRIERE:

etch, în urma evaluării și rotunjirii la cel mai apropiat întreg, trebuie să fie un număr între 0 și 9999, adică o etichetă. Plasarea sa în afara acestor limite se soldează cu eroarea B.

Efectul instrucțiunii `GOTO` este trecerea la executarea liniei cu eticheta specificată, sau la următoarea linie mai mare ca *etch*, dacă linia *etch* nu există. Cu alte cuvinte, este o instrucțiune de *salt*. Linia la care se sare va fi executată începând de la prima ei instrucțiune.

Spre deosebire de `RUN`, `GOTO` nu are alte *efecte laterale* (consecințe mai puțin importante — cum era la `RUN` ștergerea ecranului), deci este mai convenabil de folosit pentru a face salturi prin program.

Exemplu:

```
 1 GOTO 10
 5 PRINT "Pe aici nu se trece"
10 PRINT "Se ajunge direct aici"
```

Instrucțiunea `GOTO` este adesea numită „oaia neagră a informaticii", pentru că folosirea ei cu dărnicie duce cu ușurință la scrierea unor programe pe care nici creatorul lor nu le mai poate pricepe. În BASIC HC din păcate folosirea instrucțiunii `GOTO` este inevitabilă. Deocamdată BASIC avem, BASIC programăm.

4.2 BORDER

ENGLEZĂ: *border* = bordură, margine
SINTAXĂ: `BORDER` *culoare*
CATEGORIE: comandă
DESCRIERE:

culoare este un număr care, odată rotunjit, trebuie să cadă între 0 și 7. El reprezintă codul unei culori, după lista din anexa D. O valoare în afara intervalului menționat produce eroarea K.

Rezultatul instrucțiunii este schimbarea culorii întregului `BORDER`. Culoarea `BORDER`-ului este implicit și culoarea `PAPER`-ului pentru rândurile din partea de jos a ecranului (vezi un pic mai jos `PAPER`).

Exemplu:

```
BORDER 1
```

face bordura albastru închis.

■

Dacă mai țineți minte din Capitolul 1, ecranul grafic este împărțit în 768 (32×24) de pătrățele a câte 8 ori 8 pixeli. Reamintim că simbolurile afișabile se pot tipări numai în unul din aceste pătrățele. În fiecare din acestea se pot afla la un moment dat puncte în doar două culori. Una din ele se numește culoarea fondului (PAPER), cealaltă a cernelii (INK). Când `PRINT` scrie un caracter pe ecran, conturul său este format din puncte INK, iar fondul său din puncte PAPER.

4.3 PAPER

ENGLEZĂ: *paper* = hârtie
SINTAXĂ: `PAPER` *culoare*
sau în cadrul lui `PRINT` sau `INPUT` sau o instrucțiune grafică (Capitolul 12)
CATEGORIE: comandă
DESCRIERE:

Schimbă culoarea fondului în cea specificată. Comportarea sa o vom detalia odată cu cea a instrucțiunii `INK`, cu care este foarte asemănătoare.

4.4 INK

ENGLEZĂ: *ink* = cerneală
SINTAXĂ: `INK` *culoare*
sau în cadrul lui `PRINT`, `INPUT` sau o instrucțiune grafică (Capitolul 12)
CATEGORIE: comandă
DESCRIERE:

Schimbă culoarea cernelii în cea specificată.

culoare, atât la `PAPER` cât și la `INK`, este un întreg cuprins (după rotunjire) între 0 și 9. Altfel se produce eroarea K.

Valorile 0-7 sunt coduri de culori ca la `BORDER`. Valorile 8 și 9 au niște semnificații speciale:

- 8 înseamnă *transparență*. Un caracter tipărit cu `INK 8` va păstra pentru cerneala proprie exact cerneala care se găsea în locul în care a fost tipărit pe ecran.
- 9 înseamnă *contrast*. Calculatorul va alege pentru caractere în `INK 9` o cerneală albă (7) sau neagră (0) care să contrasteze cât mai puternic cu fondul pe care se tipăresc.

Folosite pe post de comenzi de sine stătătoare, `INK` și `PAPER` schimbă culoarea tuturor caracterelor care se vor tipări după executarea lor.

Folosite în cadrul lui `PRINT` sau `INPUT` (sintaxa este asemenea lui `TAB`), ele au un efect local, în sensul că schimbă numai culoarea caracterelor ce se tipăresc prin acea instrucțiune. Un exemplu edificator:

```
 5 PAPER 6: INK 9
10 PRINT "hartia e galbena"
15 PRINT PAPER 5; "acum e albastra"
20 PRINT "si iar e galbena"
```

Executați de asemenea niște `PRINT TAB` cu diverse culori, pentru a vedea cum apar spațiile pe care le tipărește `TAB`.

4.5 Exerciţii

1. Scrieți un program care se deseneze o mică color.

   Indicație: trebuie să facem opt dungi verticale, fiecare a câte 32/8=4 pătrate. Pentru a tipări exact 22 de dungi orizontale, trebuie să executăm un *ciclu* de 22 de ori. Vom folosi o variabilă care-și schimbă valoarea la fiecare trecere prin ciclu. Ciclul se va încheia prin instrucțiunea `GOTO variabila`. După 22 de treceri, ciclul trebuie să se spargă (**Notă**: soluția e ingenioasă, dar e un exemplu tipic de folosire neortodoxă a lui `GOTO`. Din păcate, până nu parcurgem și capitolul următor, cu `IF-THEN`, nu putem rezolva altfel această problema — decât poate scriind 22 de linii identice...).

2. Scrieți un program care cere coordonatele și culoarea unui pătrățel pe care apoi îl colorează.

Chapter 5

Curăţenie şi variaţie

- `CLS` pentru a șterge ecranul;
- `CLEAR` pentru a elibera memoria;
- `NEW` pentru a o lua de la început;
- `IF - THEN` pentru alternative.

5.1 CLS

ENGLEZĂ: *CLear Screen* = curăță ecranul
SINTAXĂ: `CLS`
CATEGORIE: comandă
DESCRIERE:

Șterge ecranul, aducând toate punctele la culoarea PAPER curentă. INK-ul din toate pătrățelele este cel fixat (deși toate conțin spații, acest lucru este important pentru eventuale tipăriri cu `INK 8`). Ștergerea ecranului implică mutarea *cursorului de tipărire* — cel care indică unde se va tipări următorul caracter) și cursorului grafic (discutăm în Capitolul 12) la coordonatele 0,0.

5.2 CLEAR

ENGLEZĂ: *to clear* = a curăța
SINTAXĂ: `CLEAR [`*ramtop*`]`
CATEGORIE: comandă
DESCRIERE:

Parametrul *ramtop* este o adresă de memorie (discutăm în Capitolul 15) și, ca atare, trebuie să fie cuprins între 0 și 65535. Dacă lipsește, valoarea sa este considerată a fi 0. Dacă după rotunjire nu se află între aceste limite, veți obține eroarea B.

`CLEAR` are o sumedenie de efecte, care în general curăță tot felul de informații de prin memorie. Anume:

- face CLS;
- șterge toate variabilele (ca și `RUN`);
- face `RESTORE 0` (vom vedea în Capitolul 14);
- șterge stiva GOSUB (vom vedea în Capitolul 15).

Când parametrul lui `CLEAR` există și este diferit de 0, `CLEAR` mai face ceva: stabilește adresa maximă de memorie până la care poate ajunge programul BASIC, la valoarea specificată (*ramtop*). În mod normal ea este aproape de valoarea maximă (65368). Utilitatea acestei instrucțiuni se vede când avem nevoie de memorie pentru scopuri diferite de ale BASIC-ului — de exemplu pentru programe cod mașină, care se pun în memorie deasupra acestei adrese.

`CLEAR` mai este folosită înainte de a *salva* programele pe bandă sau disc, pentru că în mod normal se salvează și variabilele. Dacă le ștergem, programul poate ocupa substanțial mai puțin loc.

Dacă se încearcă coborârea *ramtop* sub limita maximă deja atinsă de BASIC, se obține eroarea M.

Dacă programul BASIC crește prea mult (prin lungimea sa, prin variabilele folosite, sau datorită coborârii RAMTOP-ului), veți putea întâlni următoarele erori:

- eroarea 4, când acțiunea în curs de execuție nu se poate înfăptui (de pildă crearea unei variabile);
- eroarea G, când linia tastată nu mai are loc în program;
- prin neacceptarea unei linii corecte la apăsarea tastei CR (la apăsarea lui CR o linie își mărește de obicei lungimea cu diferite caractere de control — vezi și anexa F).

Cu aceste ultime 3 erori s-ar putea să nu vă întâlniți niciodată, pentru că se produc numai în condiții critice, pe care nu e probabil să le atingeți (sunt produse numai de programe lungi).

5.3 NEW

ENGLEZĂ: *new* = nou
SINTAXĂ: `NEW`
CATEGORIE: comandă
DESCRIERE:

`NEW` este o instrucțiune puternic distructivă, pe care o veți folosi numai când veți dori să scrieți un nou program BASIC. Ea șterge toată memoria până la `RAMTOP` — programul BASIC și tot ce ține de el, plus ecranul. Culorile sunt aduse la starea inițială (`PAPER 7, INK 0, BORDER 7`). Calculatorul pare ca nou, dar atenție!, ceea ce se găsește în memorie deasupra `RAMTOP`-ului rămâne neschimbat!

5.4 IF - THEN

ENGLEZĂ: *if* = dacă, *then* = atunci
SINTAXĂ: `IF` *condiţie* `THEN` **instrucţiuni**
CATEGORIE: comandă
DESCRIERE:

Până acum am discutat doar despre operații aritmetice. Cu această instrucțiune vom vedea în ce mod putem folosi o altă categorie de operațiuni, pe care le putem numi *logice*. Logica apare prin parametrul instrucțiunii `IF - THEN` pe care l-am numit *condiţie*. Acesta este de fapt o expresie care poate fi interpretată ca fiind *adevărată* sau *falsă*. În alte limbaje există chiar un tip de date, cum în BASIC sunt întregii și șirurile de caractere, care se numește *boolean* și ale cărui valori sunt doar două: *adevărat* și *fals*. BASIC-ul nu are un astfel de tip, dar folosește tipul numeric pentru valori de adevăr. Corespondența este următoarea:

Valorile de adevăr

fals	0
adevărat	orice număr diferit de 0 (în particular și 1)

Acum putem vedea cum lucrează `IF - THEN`:

1. întâi evaluează expresia dintre `IF` și `THEN` care trebuie să genereze un rezultat numeric;
2. dacă rezultatul este *adevărat* (diferit de 0), atunci se execută instrucțiunile de după `THEN` (notate cu **instrucţiuni** în descrierea sintaxei);
3. dacă rezultatul este *fals*, se trece la linia următoare.

Să vedem niște exemple:

```
 5 INPUT a
10 IF a THEN PRINT "ati tastat ceva diferit de 0" : GOTO 5
15 PRINT "ati tastat 0"
```

Programul anterior se va opri la prima introducere a unui zero.

Condițiile *logice* pe care le folosește `IF-THEN` se pot exprima foarte adesea prin comparații felurite. Există niște semne care permit compararea a două valori numerice. Iată-le:

Comparaţii

=	testează egalitatea dintre valorile între care este plasat
>	răspunde adevărat (1) dacă numărul care îl precede este mai mare decât cel care îl urmează
<	mai mic
>=	mai mare sau egal
<=	mai mic sau egal
<>	diferit

Să vedem un alt exemplu foarte instructiv:

```
 5 LET max=8
10 LET i=1
20 PRINT i, i ^ 2
30 IF i <= max THEN LET i = i + 1: GOTO 20
```

Acesta este un program foarte simplu care afișează pătratele numerelor de la 1 la 9. Modificând valoarea lui `max` se poate obține o altă limită superioară. Mecanismul este simplu și fundamental: `i` crește într-una până când condiția `i<=max` devine falsă, adică `i` îl depășește pe `max`. Altfel este tipărită o nouă valoare.

Relațiile de inegalitate pe care le-am enumerat se pot extinde și la date de tip șir. Abia în Capitolul 14 vom spune ce înseamnă că un șir este „mai mic" decât altul. Până atunci putem defini două relații al căror sens este evident, și anume:

=	pentru egalitatea a două șiruri;
<>	pentru ne-egalitatea a două șiruri.

Două șiruri sunt egale dacă și numai dacă au aceleași caractere dispuse în aceeași ordine. Altfel sunt diferite. Exemplu (fragment de program):

```
1000 INPUT "o luam de la capat ? (da/nu) "; a$
1010 IF a$ = "nu" THEN STOP
1020 IF a$ <> "da" THEN GOTO 1000
1030 RUN
```

Să reținem: operațiile de comparare (= > < >= <= <>) dau ca rezultat un număr, care este 1 pentru adevărat și 0 pentru fals.

5.5 Exerciţiu rezolvat

Desenați o spirală.

Rezolvare

Întâi să vedem cum s-ar trasa o linie orizontală între `x1` și `x2`:

```
1000 LET y=10 : LET x1=3 : LET x2=20
1010 LET x=x1
1020 PRINT AT y,x;PAPER 0;" "
1030 LET x=x+1 : IF x <= x2 THEN GOTO 1020
```

Dacă ați înțeles, probabil că puteți trasa și linii verticale. Programul nostru de spirală va avea patru bucle ca cea de mai sus, pentru a trasa cele patru brațe ale spiralei. După o trecere prin cele patru bucle se va relua totul de la prima dintre ele, cu limitele schimbate. Iată (liniile goale conțin după etichetă câte un blanc și sunt puse pentru a face programul mai ușor de citit):

```
 1 PAPER 7 : CLS : PAPER 0
 5 LET x1 = 0: LET y1 = 0 : LET x2 = 31 : LET y2 = 21
10 LET x = x1 : LET y = y1
12
15 PRINT AT y,x;" "
20 IF x < x2 THEN LET x = x + 1: GOTO 15
25 LET y1 = y1 + 2
27
30 PRINT AT y,x;" "
35 IF y < y2 THEN LET y = y + 1: GOTO 30
40 LET x2 = x2 - 2
42
45 PRINT AT y,x;" "
50 IF x > x1 THEN LET x = x - 1: GOTO 45
55 LET y2 = y2 - 2
57
60 PRINT AT y,x;" "
65 IF y > y1 THEN LET y = y - 1:GOTO 60
70 LET x1 = x1 + 2
72
80 IF y2 >= y1 + 2 THEN GOTO 15
```

5.6 Exerciţii

1. Să se scrie un program folosind instrucțiunile cunoscute, care să facă suma a N (N se cere prin program) numere cerute de la utilizator și să o afișeze.

2. Scrieți un program care pentru N numere (ca mai sus) cerute cu `INPUT`, afișează câte din ele sunt nule, câte-s pozitive și câte negative.

Chapter 6

Matematici din plin

Un capitol bogat în instrucțiuni și exerciții, dar scurt în întindere. Multe funcții.

- `REM` pentru comentarea programelor;
- `PAUSE` pentru a încetini ritmul;
- `ABS` pentru modul;
- `INT` ca parte întreagă;
- `SQR` ca radical;
- `SGN` pentru semn;
- `EXP` pentru exponențiale;
- `LN` pentru logaritmi.

6.1 REM

ENGLEZĂ: *REMark* = observație
SINTAXĂ: `REM` absolut orice
CATEGORIE: comandă
DESCRIERE:

`REM` este o instrucțiune care ... nu face nimic! Aceasta n-o face mai puțin utilă! Ea are un singur scop: pentru a face mai inteligibil listingul unui program, în interiorul său se pot insera *comentarii*. După `REM` poate urma orice suită de simboluri, chiar și comenzi. Tot ceea ce urmează până la sfârșitul liniei este ignorat. Recomandăm utilizarea ei intensă. Un exemplu:

```
5 IF b=a THEN GOTO 55 : REM la 55 testam semnele
```

6.2 PAUSE

ENGLEZĂ: *pause* = pauză
SINTAXĂ: `PAUSE` *timp*
CATEGORIE: comandă
DESCRIERE:

timp este un număr întreg, între 0 și 65535. Dacă după rotunjire nu pică în acest interval, va surveni eroarea B.

Așa cum arată și numele ei, `PAUSE` timp va face o pauză de *timp*/50 secunde în execuția programului. `PAUSE 200` înseamnă 4 secunde. De reținut totuși că, dacă în timpul pauzei se apasă o tastă, așteptarea este abandonată prin trecerea la instrucțiunea următoare. `PAUSE 0` are un efect special: așteaptă apăsarea unei taste (dar nu CS sau SS singure).

`PAUSE` de fapt numără *întreruperile* pe care le face un ceas intern. Ceasul este sincronizat și cu circuitele care formează imaginea pe ecran. Din cauza aceasta, folosind secvența:

```
5 PAUSE 1: BORDER 1: BORDER 4: BORDER 6: GOTO 5
```

veți obține pe `BORDER` niște dungi staționare. Pentru a înțelege tot ce se întâmplă ar fi nevoie de ceva mai multe cunoștințe despre funcționarea televizorului. Amânăm o explicație.

6.3 ABS

ENGLEZĂ: *ABSolute value* = valoare absolută
SINTAXĂ: `ABS` *expresie*
CATEGORIE: funcție
DESCRIERE:

`ABS` este o funcție care întoarce *valoarea absolută* (*modulul*) argumentului său *expresie*. Prioritatea de evaluare este mai mare ca a expresiilor aritmetice:

```
ABS -5-8*4 = ABS(-5)-8*4 = 5 - 32 = -27
ABS (-5-8*4) = ABS (-37) = 37
```

Această observație este valabilă pentru toate funcțiile.

6.4 INT

ENGLEZĂ: *INTeger* = întreg
SINTAXĂ: `INT` *expresie*
CATEGORIE: funcție
DESCRIERE:

`INT` este funcția *parte întreagă*, adică întoarce cel mai mare număr întreg mai mic sau egal cu argumentul său, *expresie*.

```
INT 3.14 = 3      INT 5 = 5      INT -5 = -5      INT -3.14 = -4
```

Pentru a *rotunji* numărul `X` se calculează `INT (X + 0.5)`.

6.5 SQR

ENGLEZĂ: *SQuare Root* = rădăcină pătrată
SINTAXĂ: `SQR` *expresie*
CATEGORIE: funcție
DESCRIERE:

Pentru *expresie* negativă urmează mesajul A. `SQR` calculează rădăcina pătrată din argumentul său, adică *expresie*`^(1/2)`.

6.6 SGN

ENGLEZĂ: *SiGN* = semn
SINTAXĂ: `SGN` *expresie*
CATEGORIE: funcție
DESCRIERE:

`SGN` este *funcţia semn*. Ea întoarce:

$$\texttt{SGN}\ expresie = \begin{cases} -1 & dacă \quad expresie\ \texttt{<}\ 0 \\ 0 & dacă \quad expresie\ \texttt{=}\ 0 \\ 1 & dacă \quad expresie\ \texttt{>}\ 0 \end{cases}$$

6.7 EXP

ENGLEZĂ: *EXPonential* = exponențială
SINTAXĂ: `EXP` *expresie*
CATEGORIE: funcție
DESCRIERE:

`EXP` *expresie* întoarce *e*`^`*expresie*, (adică *funţia exponenţială* unde *e* este numărul lui Euler, nu o variabilă. Valoarea lui aproximativă este *e*`^1 = EXP 1` = 2.7182818. Aceasta funcție are numeroase aplicații matematice. În acest manual n-o veți mai întâlni niciodată.

6.8 LN

ENGLEZĂ: *Logarithm of Neper* = logaritm neperian (natural)
SINTAXĂ: `LN` *expresie*
CATEGORIE: funcție
DESCRIERE:

Pentru *expresie* negativă — eroare A. `LN` *expresie* întoarce *logaritmul natural* al valorii *expresie*, adică puterea la care trebuie ridicat *e* (= `EXP 1`) pentru a obține *expresie*. Dacă vreți logaritmi în baza oarecare Q, folosiți regula :

$$\log_Q x = \frac{\ln x}{\ln Q}$$

■

Cu cele 6 funcții de mai sus putem deja să facem o mulțime de lucruri noi. Vom indica doar câteva formule mai des folosite :

- *câtul* împărțirii lui A la B este:
  `Q = INT (A / B)`
- *restul* împărțirii lui A la B este:
  `R = A - B * Q = A - B * INT (A / B)`
- valoarea de adevăr a propoziției „N se împarte exact la M":
  `(N = M * INT (N / M))`
- rădăcina de ordin impar N dintr-un număr oarecare A:
  `SGN a * (ABS a)^(1 / N)`

6.9 Exerciţiu rezolvat

Calculați c.m.m.d.c. și c.m.m.m.c. a două numere.

Rezolvare

Dacă vă mai amintiți algoritmul lui *Euclid*, îl veți recunoaște în următorul program:

```
 10 INPUT "numerele : ";n1,n2
 20 IF n1 <> INT n1 THEN GOTO 10: REM n1 trebuie sa fie intreg
 21 IF n2 <> INT n2 THEN GOTO 10
 25 LET b=n2 : LET a=n1
 27 REM *** incepe algoritmul ***
 30   LET q = INT ( a/b ) : REM catul
 40   LET r = a - b * q   : REM restul
 50   IF r=0 THEN GOTO 100 : REM s-a gasit c.m.m.d.c.!
 60   LET a = b           : REM catul devine deampartit
 70   LET b = r           : REM restul devine cat
 80 GOTO 30
100 PRINT "c.m.m.d.c. (" ; n1 ; "," ; n2 ; ")="; b
110 PRINT "c.m.m.m.c. (" ; n1 ; "," ; n2 ; ")="; n1*n2 / b
```

6.10 Exerciţii

De data asta am scornit o grămadă, unele chiar interesante:

1. Indicați valoarea tipărită de următorul program :

```
 5 CLEAR
10 LET a=14
20 LET q=INT (SQR ((a-12)*(a-6)))
30 LET a=a+q
```

```
40 REM se cere o noua valoare pentru a : INPUT a
50 IF a THEN LET ak = a - 3
60 IF ak > 0 THEN IF ak <> 2 THEN LET A = a + 2
70 PRINT ABS( INT (a-2*3))
```

2. Scrieți un program care să umple ecranul cu pătrățele colorate în ordinea 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, ... după codul culorilor.

3. Scrieți un program care să coloreze într-o culoare toate pătrățelele cu ambele coordonate pare și în altă culoare pe celelalte.

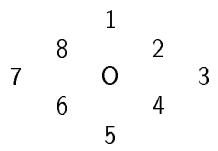
4. Rescrieți programul de la Exercițiul 1, Capitolul 4, (mira color) folosind noile instrucțiuni.

5. Scrieți un program care calculează primii N (care se cere prin program) termeni ai șirului lui *Fibonacci*. (Șirul acesta se obține astfel: primii doi termeni sunt 1 și 1, iar un termen ulterior se obține prin adunarea celor doi termeni care-i sunt anteriori. Iată: 1, 1, 2, 3(=1+2), 5(=2+3), 8(=3+5), 13 etc.).

6. Scrieți un program care determină dacă un număr dat N este *prim*.

   Indicaţie: testați resturile împărțirii lui *n* la toate numerele întregi $\leq \sqrt{n}$.

7. Scrieți un program care consideră ecranul o masă de biliard pe care se *mişcă* o bilă (litera O). Programul va cere direcția de lovire (ca în figură) și apoi va mișca încetinit bila până la oprirea ei.



Indicaţie: se va folosi `PAUSE` cu o valoare care crește pe măsură ce bila avansează. A mișca bila înseamnă a o șterge și a o desena într-o altă poziție.

Chapter 7

Mai multe dimensiuni

In acest capitol tratăm o singură problemă, dar destul de spinoasă:

- matrici (variabile multidimensionale);
- `DIM` pentru a crea variabile multidimensionale.

Dacă avem un șir de numere notat cu X, să zicem, în general referim primul său element drept X_1, al doilea drept X_2 ș.a.m.d. Un astfel de șir, cu un număr finit de termeni, este numit *vector*.

Să presupunem că ecranul este umplut cu cifre. Atunci fiecare pereche de coordonate identifică o cifră — cea care se află în acel loc pe ecran. Un *tablou* de acest fel se numește *matrice*. Vom folosi termenul de „matrice" pentru toate valorile cu mai multe dimensiuni. Iată două exemple:

Un vector X

x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1	2	3	5	8	13

O matrice M

	$coloana\ 1$	$coloana\ 2$	$coloana\ 3$	$coloana\ 4$	$coloana\ 5$	$coloana\ 6$
$linia\ 1$	0	1	2	3	4	5
$linia\ 2$	0	1	3	5	7	9
$linia\ 3$	0	1	4	8	12	16
$linia\ 4$	0	1	5	12	20	28
$linia\ 5$	0	1	6	17	32	48

Tot așa cum putem vorbi despre elementele vectorului X, putem vorbi despre elementele matricei M. În vorbirea curentă le putem indica în multiple moduri: M1,1 sau M(1,1) sau M[1,1] sau $M_{1,1}$ sau M[1][1] etc. BASIC alege a doua formă, cea cu paranteze rotunde.

Exemplele se pot extinde la mai multe dimensiuni. Cazul cu trei dimensiuni se poate reprezenta printr-o imagine spațială.

BASIC HC permite operarea cu astfel de obiecte numai sub formă de variabile. Tipul lor este dat de tipul elementelor lor; putem distinge matrici de numere sau de caractere (detalii imediat). *Indicii* (valorile care exprimă coordonatele unui element) unei matrici sunt întotdeauna numere întregi, începând de la 1. Numărul de dimensiuni poate cuprins fi între 1 și 255 (suficient de mare ca să nu fie restrictiv).

7.1 DIM

ENGLEZĂ: *DIMension* = dimensiune
SINTAXĂ: `DIM` *v*`[$]`$(d_1, d_2, \ldots, d_n)$
CATEGORIE: comandă
DESCRIERE:

v este numele variabilei-matrice. El nu poate fi mai lung de o literă. Deci o variabilă ca `ar2` nu poate fi matrice. `$` specifică dacă variabila este de tip șir sau numeric. $d_1, d_2, \ldots, d_n$ sunt numere pozitive, întregi. n este și dimensiunea matricii. Valorile lui $d_1, \ldots, d_n$ specifică mărimea fiecărei dimensiuni. Dacă un d_i nu e pozitiv, întreg, sau depășește 65536, ori dacă $n > 255$, survine eroarea B.

`DIM` rezervă spațiu în memorie și crează o variabilă multidimensională cu dimensiunile specificate. Pentru a crea variabile în care să memorăm cele două tablouri din exemplul de mai sus va trebui să executăm:

```
DIM x(7) : DIM m(6,5)
```

Acțiunile lui `DIM` pentru variabile numerice și șir sunt destul de diferite ca să merite o tratare separată.

7.1.1 DIM pentru variabile numerice

Să discutăm pe un exemplu. `DIM m(6,5)` rezervă spațiu în memorie pentru o matrice cu $6 \times 5 = 30$ de elemente, numerotate `m(1,1)`, `m(1,2)`, ..., `m(6,5)`. Variabila numerică simplă `m` și variabila multidimensională `m(6,5)` pot exista simultan!

Alte efecte:

- dacă un tablou cu numele de `m` exista anterior, el este șters, oricare i-ar fi fost dimensiunile;
- toate elementele tabloului capătă automat valoarea 0.

Elementele matricei se referă, după cum am anticipat specificând numele matricei urmat de coordonatele elementului între paranteze. Atenție, `m(1,2)` nu e totuna cu `m(2,1)` (se vede și pe figură).

Veți obține eroarea 3 dacă:

- numărul de indici la invocarea unui element nu este exact cel de la dimensionare; de pildă `LET c = m(3)` sau `LET m(1,1,2) = 0`
- valoarea unui indice nu se găsește între limitele indicate la dimensionare:
  `PRINT m(-1,2)` sau `INPUT m(8,8)`

7.1.2 DIM pentru variabile de tip şir

Pentru a rezerva spațiu pentru $d_1 \times d_2 \times \ldots \times d_n$ variabile șir (sau, cu alte cuvinte, o matrice de șiruri cu dimensiunile $d_1, d_2, \ldots, d_n$), trebuie specificate $n+1$ dimensiuni! Ultima va indica numărul maxim de caractere pe care îl va avea fiecare șir.

Pentru a înțelege de ce lucrurile stau așa, trebuie să ne lansăm în niște explicații mai ample, dar care au o mare importanță. Să purcedem!

În primul rând trebuie să observăm că fiecare șir de caractere se comportă ca un vector de caractere, dar care nu a fost anterior dimensionat. Aceasta înseamnă că avem acces la literele care-l compun ca la elementele unui vector. Astfel, `a$(1)` este primul caracter din șirul `a$` (dacă există). `a$(n)` este al `n`-lea caracter. (Dacă `a$` nu are `n` caractere veți obține eroarea 3.) Să vedem:

```
 10 LET a$="abcdefghijk"
 20 PRINT a$(1)
 30 PRINT a$(10),a$
```

cu efectul pe ecran

```
a
j               abcdefghijk
```

Mai mult decât atât: putem opera cu fiecare caracter din șir ca și cum ar fi o variabilă independentă:

```
 3 LET a$="computer"
 5 LET a$(1)="C"
 7 PRINT a$
 8 LET a$(1)="masina"
 9 PRINT a$
```

cu efectul

```
Computer
momputer
```

Programul anterior ne semnalează un fapt important: `a$(1)` este un șir format dintr-un singur caracter, deci dacă-i atribuim o valoare mai lungă, va prelua din ea numai o parte. Să mai vedem un exemplu:

```
 3 LET a$="verde"
 5 LET a$=a$(1) : PRINT a$
 7 LET a$=a$+a$(1) : PRINT a$
```

pe ecran:

```
v
vv
```

■

Revenind la DIM: `DIM a$(3,4,10)` crează o matrice cu $3 \times 4 = 12$ șiruri numerotate de la `a$(1,1)`, `a$(1,2)` la `a$(3,4)`, fiecare cu câte 10 caractere. Pentru că, așa cum am văzut, chiar și variabila simplă a$ poate fi considerată ca fiind un vector, dar cu lungime variabilă, variabila multidimensională `a$(3,4,10)` și variabila simplă a$ nu pot exista simultan. De aceea DIM cu parametru de tip șir va șterge orice alt șir cu același nume (reamintim că situația nu este aceeași în cazul variabilelor numerice). Tabloul creat are toate elementele inițializate cu " " (spațiu). Astfel se crează 3×4 șiruri de câte 10 spații.

Cu șirurile create putem lucra normal — `a$(2,2)` este al doilea șir, având 10 caractere — sau cu caracterele lor individuale: `a$(2,2,3)` este al treilea caracter al șirului de mai-nainte. Șirurile `a$(x,y)` vor avea întotdeauna 10 caractere. Atribuirile vor lucra *procustean*. (Procust a fost un tip care avea un pat după a cărui dimensiune ajusta pe cei pe care-i invita să doarmă-n el: dacă erau prea lungi, îi scurta de cap, dacă erau prea scurți, îi mai lungea. Un pic sadic.) Cea mai bună metodă pentru a pricepe este experimentul:

```
10 DIM a$(3,2,4)
20 LET a$(1,1) = "eu" : LET a$(2,1) = "tu" : LET a$(3,1) = "el"
30 LET a$(1,2) = "noi": LET a$(2,2) = "voi": LET a$(3,2) = "ei,toti"
40 PRINT a$(1,1);a$(1,2);a$(3,2)
50 LET a$(3,1,2) = "acasa"
60 PRINT a$(3,1);a$(2,2)
```

pe ecran:

```
eu  noi ei,t
ea  voi
```

Folosirea unor indici incorecți duce la eroarea 3.

7.2 Exerciţiu rezolvat

Scrieți un program care calculează maximul dintr-un șir de numere.

Rezolvare

Metoda este următoarea: presupunem că maximul este chiar primul număr. Îl comparăm cu al doilea, al treilea, etc., până dăm de unul mai mare. Atunci acesta va fi mai mare decât toate de până la el. Zicem că el e maximul și continuăm să-l comparăm cu cele ce-au rămas, după aceeași procedură.

```
 5 INPUT "lungimea sirului ";n: IF n < 2 THEN STOP
10 DIM a(n)
15 REM *** citim elementele
20 LET i = 1
25   INPUT "Dati elementul ";(i); a(i)
30   PRINT "a(" ; i ; ")=" ; a(i)
35   IF i < n THEN LET i = i + 1: GOTO 25
40 REM *** incepem cautarea maximului
```

```
 50 LET max = a(1) : REM presupunem ca e primul
 55 LET i=2        : REM incepem compararea de la al doilea
 60   IF a(i) > max THEN LET max = a(i)
 70   IF i<n THEN LET i = i + 1: GOTO 60
100 PRINT "Maxim este ";max
```

7.3 Exerciţiu

1. Scrieți un program care memorează N nume de persoane și ocupațiile lor. Folosiți-l pentru a afișa după dorință ori lista persoanelor, ori lista persoanelor ce au o anumită ocupație (cerută de program de la utilizator).

Chapter 8

Cicluri

Ca și capitolul anterior, cel de față tratează un singur concept, dar pe-ndelete. Vom putea trece la programe mai naturale, evitând șmecheriile pe care le-am făcut până acum ca să putem scrie ceva „mergător" cu puținele instrucțiuni pe care știam. Deși cele de până acum nu au fost modele de programare, totuși cred că v-au pus la lucru într-o oarecare măsură imaginația.

- `FOR - TO - STEP` pentru a face bucle;
- `NEXT` pentru reluarea execuției unei bucle.

8.1 FOR - TO - STEP

ENGLEZĂ: *for* = pentru, *to* = la, *step* = pas
SINTAXĂ: `FOR` *v* = *init* `TO` *fin* [`STEP` *pas*]
CATEGORIE: comandă
DESCRIERE:

8.2 NEXT

ENGLEZĂ: *next* = următorul, următoarea
SINTAXĂ: `NEXT` *v*
CATEGORIE: comandă
DESCRIERE:

Aceste instrucțiuni se folosesc în mod normal numai împreună, deși sunt distincte. Între ele se pot afla oricâte alte instrucțiuni, sau linii. Instrucțiunea `FOR` trebuie însă să fie executată înaintea `NEXT`-ului corespunzător. Scopul acestui cuplu este de a realiza o *buclă* (un *ciclu*).

Asta înseamnă că instrucțiunile care se află scrise între `FOR` și `NEXT` se vor repeta de un anumit număr de ori. Noi am mai scris până acum bucle, dar cu oarece chinuială, folosind o variabilă ce se incrementa și un `IF` pentru a decide atingerea limitei. Pentru cazul în care numărul de repetiții este cunoscut la intrarea în buclă, `FOR - NEXT` se dovedesc mai eficiente.

v este o variabilă numită *contor*, care trebuie să aibă numele format dintr-o singură literă. Valoarea ei se modifică la fiecare trecere prin buclă.

init este o expresie numerică, folosită pentru a genera valoarea inițială a variabilei *v* (valoarea pe care o primește *v* la executarea instrucțiunii `FOR`).

fin este valoarea limită pe care o poate lua *v*. La depășirea acestei limite, bucla nu mai este reluată.

pas este valoarea care se adaugă la *v* la fiecare nouă trecere prin buclă. Când `STEP` lipsește, valoarea lui *pas* este considerată 1.

Să exemplificăm:

```
10 DIM a(10)
20 FOR i=1 TO 10
30   INPUT "elem " ; (i) ; " " ; a(i)
40   PRINT i
50 NEXT i
```

În acest program `i` trece prin toate numerele întregi de la 1 la 10. În interiorul buclei se cere elementul unui vector cu indicele `i`. Deci acest program inițializează *interactiv* (adică luând datele de la utilizator, interacționând cu el) un vector cu 10 elemente.

Dacă înaintea executării unei instrucțiuni `FOR` exista o variabilă numerică având același nume cu *v*, atunci ea este distrusă.

Pasul poate fi subunitar:

```
 FOR i=10 to 13 STEP 0.5 : PRINT i;" ";: NEXT i
```

rezultat:

```
10 10.5 11 11.5 12 12.5 13
```

sau negativ (însă atunci trebuie ca $init \geq fin$):

```
 FOR k=20 to 10 STEP -2 : PRINT k,: NEXT k
```

care dă:

```
20              18
16              14
12              10
```

Mai avem de lămurit câteva detalii cu privire la buclele `FOR - NEXT`.

Întâi, care este valoarea variabilei de control la ieșirea din buclă? Este prima valoare care nu mai respectă condiția de parcurgere a buclei, și nu ultima care o mai respectă. Iată:

```
5 FOR i=1 TO 5 : PRINT i; : NEXT i
6 PRINT 'i
```

care dă

```
12345
6
```

Ce se întâmplă dacă bucla nu se poate parcurge niciodată? Asta se întâmplă numai în două cazuri:

- $init > fin$ și $pas \leq 0$, sau
- $init < fin$ și $pas > 0$.

În aceste condiții bucla nu se parcurge niciodată și se sare la prima instrucțiune de după `NEXT`-ul corespunzător.

Iată erorile pe care le pot genera buclele `FOR - NEXT`:

- eroarea 2: când se întâlnește `NEXT`-ul fără a fi trecut prin `FOR`-ul respectiv și nici nu există vreo variabilă normală cu acel nume;
- eroarea 1: când s-a întâlnit `NEXT`, `FOR`-ul nu a existat, dar variabila cu același nume există;
- eroarea I: când se întâlnește un `FOR` imposibil de executat (vezi paragraful anterior), iar `NEXT`-ul respectiv nu poate fi găsit.

Pentru a părăsi o buclă înainte de sfârșitul ei firesc, putem imagina două metode:

```
5 FOR i=1 TO 10
6   INPUT "continuam ? ";a$
7   IF a$="da" THEN LET i=11
8 NEXT i
```

sau

```
5 FOR i=1 TO 10
6   INPUT "continuam ? ";a$
7   IF a$="da" THEN GOTO 9
8 NEXT i
```

Prima metodă îl aduce pe `i` în afara limitei finale de ciclare, deci la întâlnirea instrucțiunii `NEXT` bucla nu mai este reluată. A doua metodă iese pur și simplu din buclă, fără a o mai parcurge până la `NEXT`.

În mod normal fiecărui `FOR` trebuie să-i corespundă un `NEXT` și numai unul. Aceasta este o recomandare, nu o obligație. Dar, structuri ca următoarea:

```
 10 FOR i=1 to 100
 ...
 50   IF cond THEN GOTO 100
 51   REM cond este falsa
 ....
 90 NEXT i
 95 GOTO 200
100   REM cond este adevarata
 ...
190 NEXT i
```

sunt mult prea îmbârligate pentru a permite o înțelegere ușoară. Plus că alte limbaje nici nu admit structuri de acest fel.

Într-un program se pot folosi, evident, mai multe bucle. Ele trebuie să fie ori complet distincte (adică FOR-ul uneia să nu se afle în interiorul celeilalte), în care caz pot folosi aceeași variabilă, ori complet una-ntr-alta (adică atât FOR-ul cât și NEXT-ul uneia să se afle în interiorul celeilalte). Cu alte cuvinte, avem:

Corect:

```
  5  FOR i=1 TO 10
 10      FOR j=2 TO 2.5 STEP 0.1
 ...
 90      NEXT j
100  NEXT i
```

și:

```
  5  FOR i=15 TO 10 STEP -1
 ...
 50  NEXT i
 ...
100  FOR j=0 TO 0.2 STEP 0.1
 ...
200  NEXT j
```

Greșit:

```
 10 FOR i = 1 TO 10
 20   FOR j = 2 TO 14 STEP 3
 ...
 50 NEXT i
 60   NEXT j
```

Ciudățenia este că programe ca cel greșit de mai de sus funcționează. Pentru a vedea și ce fac ele, citiți nota explicativă de la sfârșitul acestui capitol.

Să vedem un exemplu cu două bucle, una-ntr-alta:

```
 10 FOR l=1 TO 7
 20   FOR k=1 TO l
```

```
30     PRINT k;
40   NEXT k
50   PRINT
60 NEXT l
```

cu efectul:

```
1
12
123
1234
12345
123456
1234567
```

Instrucțiunea `PRINT` din linia 50 trece pe rând nou după fiecare șir de tipărit.

8.3 Exerciţiu rezolvat

Ordonați crescător un șir de numere (*sortaţi-l*).

Rezolvarea 1

Vom parcurge șirul. De fiecare dată când vom întâlni două numere vecine care nu se află în ordinea așteptată, le vom schimba între ele. Dacă la o trecere nu am făcut nici o schimbare înseamnă că șirul este ordonat. Altfel o luăm de la capăt. Aceasta metodă de sortare se numește *metoda bulelor*, pentru că la fiecare trecere numerele mari „urcă" spre locurile lor. Adăugând la programul de mai jos linia

```
45 FOR i=1 TO n:PRINT a(I);" ";:NEXT i
```

veți remarca acest efect.

```
 1 INPUT "Nr de elemente ";N : DIM a(N)
 5 REM *** cerem elementele
10 FOR i=1 TO N
13   INPUT "Elementul ";(i);" este ";a(i)
20 NEXT i
25
27 k=0 : REM k=1 va indica daca s-au facut schimbari
30 FOR i=1 TO n-1
32   REM comparam elementul i cu elementul i+1
33   IF a(i) <= a(i+1) THEN GOTO 40
34    REM elementele nu sunt in ordine, trebuie schimbate.
35    LET a = a(i)
36    LET a(i) = a(i+1)
37    LET a(i+1)=a
38    LET k=1
40 NEXT i
```

```
50 IF k <> 0 THEN GOTO 27 : REM mai trecem odata
60 FOR i=1 TO n : PRINT a(i);" ";: NEXT i
```

Rezolvarea 2

Vom proceda în modul următor: căutăm maximul din șir și îl mutăm pe ultimul loc. (Asta știm să facem.) După aceea, considerăm șirul format numai din primii $n-1$ termeni și facem același lucru. Această metodă de sortare se numește *prin selecţie*. Iată:

```
 1 INPUT "Nr de elemente ";N : DIM a(N)
 5 REM *** cerem elementele
10 FOR i=1 TO N
13   INPUT "Elementul ";(i);" este ";a(i)
20 NEXT i
25
30 FOR i=0 TO n-2 : REM cautam de n-1 ori maximul
33   LET max=a(1) : LET pozitie = 1
35   FOR j=2 TO n - i
40      IF a(j) <= max THEN GOTO 50
45      REM am gasit un nou maxim
48      LET max = a(j) : LET pozitie = j
50   NEXT j
55   REM mutam maximul la locul lui
58   LET a = a(pozitie) : LET a(pozitie) = a(n - i) : LET a(n - i) = a
60 NEXT i
65 FOR i=1 TO n : PRINT a(i);" ";: NEXI i
```

Pentru a vedea cum evoluează șirul, copiați linia 65 in linia 59.

■

Pentru a înțelege perfect comportarea instrucțiunilor `FOR` și `NEXT` vom descrie exact operațiunile pe care le face calculatorul la întâlnirea fiecăreia dintre ele.

8.3.1 `FOR v=x TO y STEP z`

1. Șterge din memorie orice altă variabilă numerică care nu e matrice și are același nume `v`.
2. Crează o nouă variabilă specială numerică de tip `FOR - NEXT`; informațiile conținute în această variabilă cuprind (vezi și anexa F):
   (a) numele și tipul (numerică, `FOR - NEXT`) variabilei;
   (b) valoarea curentă — la început `x`;
   (c) valoarea finală `y`;
   (d) valoarea pasului `z`;
   (e) linia în care se află instrucțiunea `FOR`;
   (f) numărul instrucțiunii `FOR` din linie.

   Aceste înregistrări se fac fără nici o verificare. Apoi:
3. Se verifică dacă bucla se poate parcurge. Avem două cazuri favorabile:

(a) `x <= y` și `z >= 0` (da, se poate și `STEP 0` — buclă infinită);

(b) `x > y` și `z < 0`.

4. În caz că bucla se poate parcurge se trece la instrucțiunea următoare.

5. Când bucla nu se poate parcurge, se caută în program începând de la următoarea instrucțiune, o comandă `NEXT` cu aceeași variabilă `v`.

   (a) dacă `NEXT v` este găsit, se sare la prima instrucțiune de după el;

   (b) dacă `NEXT v` nu există până la sfârșitul programului — eroare I.

8.3.2 NEXT v

1. Caută în spațiul variabilelor o variabilă numerică care nu e matrice și are același nume `v`.

2. Dacă nu o găsește dă eroare 2.

3. Dacă o găsește, dar variabila nu e de tip `FOR` (cu înregistrările indicate mai sus) — eroare 1.

4. Adună valoarea pasului la valoarea curentă.

5. Verifică dacă bucla s-a încheiat. Două cazuri favorabile:

   (a) `valoarea curenta > y` și `z >= 0` ;

   (b) `valoarea curenta < y` și `z < 0`.

6. Dacă bucla s-a încheiat, continuă cu instrucțiunea următoare.

7. Dacă bucla nu s-a încheiat, citește câmpurile (e), (f) care conțin poziția instrucțiunii `FOR`, pentru a găsi începutul buclei și sare acolo (imediat după `FOR`).

Ținând cont de acestea, vom vedea că programul următor:

```
1 FOR i = 1 TO 2
2 NEXT i : PRINT i;" "; : GOTO 2
```

merge și afișează `3 4 5 6 7` etc.

Încercați să deduceți ce face programul:

```
5 FOR i = 10 TO 1 STEP -1
6   FOR j = 5 TO i
7     PRINT i;" ";j;" ";
8 NEXT i
9   NEXT j
```

8.4 Exerciţii

1. Scrieți un program care să deseneze pătrate concentrice de culori diferite, în centrul ecranului. Dați mai multe soluții.

2. Care este valoarea tipărită de următorul program ?

```
 10 INPUT a
 20 FOR A = 5 TO 20 STEP 7
 30   REM : LET A = A + 5
 40 NEXT A
 50 LET W = SQR (A - 1)
 60 FOR z = 12 TO 10 STEP a
 70   LET w = w + 2
 80 NEXT z
 90 DIM w(3)
100 LET W = w + W(1) + W(2) * W(3)
110 INPUT r : LET r = - (ABS (SGN (ASB r))) :
    IF r = 0 THEN LET r = r - 1
120 LET W = w / r
130 PRINT "W=";W
```

3. Scrieți un progrămel care să permită ridicarea la puteri întregi a numerelor de orice semn.

Chapter 9

Întâmplare şi interacţiune

După atâtea lucruri serioase, acest capitol prezintă instrucțiuni care se pot folosi cu succes pentru a scrie jocuri (nu numai, desigur). Aplicațiile pe care le propunem caută să fie cât mai distractive.

- `RND` pentru a introduce întâmplarea în calculator;
- `RANDOMIZE` pentru a o ajuta;
- `INKEY$` pentru a citi tastatura.

9.1 RND

ENGLEZĂ: *RaNDom* = la întâmplare
SINTAXĂ: `RND`
CATEGORIE: funcție
DESCRIERE:

De fiecare dată când este chemată, `RND` generează un număr *aleator* (aleator vine din latinescul „alea", adică zaruri), cuprins între 0 și 1. Rezultatul poate fi 0, dar niciodată 1. Priviți-l la lucru:

```
5 PRINT RND, : GOTO 5
```

`RND` este de fapt o funcție pseudo-aleatoare. Ea generează numerele după o regulă foarte complicată și de aceea par aleatoare. Nu numai că funcția nu este aleatoare — este chiar periodică! Fiecare număr se calculează numai în funcție de cel generat anterior. Aceasta înseamnă că dacă un număr se repetă, atunci și toți succesorii lui se vor repeta. Perioada cu care se repetă numerele este 65536. Problema generării de numere aleatoare este mult mai

spinoasă decât vă imaginați. Încercați să descrieți „algoritmul" după care spuneți un număr la întâmplare!

La ce ne folosesc numerele aleatoare? La o grămadă de lucruri. De fiecare dată când vrem să simulăm un fenomen care are o aparență întâmplătoare, ele ne pot ajuta. Să vedem un exemplu — aruncarea unui zar:

```
 5 LET zar = INT (RND * 6) + 1
10 PRINT "s-a aruncat "; zar : PAUSE 0 : GOTO 5
```

`RND * 6` este un număr între 0 și 6, niciodată 6. `INT ( RND * 6)` este un număr întreg între 0 și 5. Dacă adăugăm 1 obținem un număr întreg între 1 și 6 — exact ca un zar!

În general, dacă vrem să obținem numere aleatoare în intervalul [a,b), fără b, scriem `(RND * (b - a)) + a`. Pentru numere întregi aplicăm apoi `INT`.

9.2 RANDOMIZE

ENGLEZĂ: *to randomize* = a introduce hazard, întâmplare
SINTAXĂ: `RANDOMIZE [`*num*`]`
CATEGORIE: comandă
DESCRIERE:

num este — după rotunjire — un întreg între 0 și 65535, altfel se obține eroare B. Dacă lipsește, este considerat a fi 0.

`RANDOMIZE` crează o *sămânţă* (*seed* în engleză) pornind de la care se va calcula următorul număr aleator. După `RANDOMIZE` urmat de o valoare nenulă, primul număr aleator obținut de la `RND` va fi întotdeauna același!

`RANDOMIZE` urmat de valoarea 0 (sau echivalent, de nici o valoare) are un efect special: folosește pentru SEED valoarea curentă a timpului! Folosind `RANDOMIZE 0` nu veți obține la două rulări ale aceluiași program aceleași efecte, pentru că timpul este altul! (Timpul folosit ca valoare pentru SEED este timpul de la ultima pornire a calculatorului.)

Verificați diferența rulând de mai multe ori următoarele exemple:

```
 5 RANDOMIZE 0 : PRINT RND : GOTO 5

 5 RANDOMIZE 0
10 PRINT RND : GOTO 10

 5 RANDOMIZE 1
10 PRINT RND : GOTO 10

 5 RANDOMIZE 1 : PRINT RND : GOTO 5
```

9.3 Exerciţiu rezolvat

Scrieți un program care distribuie într-o ordine aleatoare caracterele unui șir.

Rezolvare

Vom proceda după cum urmează:

1. generăm un număr aleator între 1 și lungimea șirului;
2. schimbăm caracterul respectiv cu cel de pe ultima poziție;
3. considerăm șirul ca fiind mai scurt (ca să nu rearanjăm ultimul caracter) și reluăm procedura.

```
 1 RANDOMIZE
 5 LET n=26 : DIM a$(n) : LET a$ = "abcdefghijklmnopqrstuvwxyz"
10 FOR i = 1 TO n-1
20   LET q = INT (RND * (n-i)) + 1
25    REM *** schimbam elementele
30    LET b$ = a$(q)
35    LET a$(n - i + 1) = b$
40    LET a$(q) = b$
50 NEXT i
```

Așa putem amesteca literele la un joc de SCRABBLE, sau cărțile la un joc de cărți; puteți imagina și alte aplicații.

9.4 INKEY$

ENGLEZĂ: *in* = înauntru, *key* = tastă
SINTAXĂ: `INKEY$ [#` *cale*`]`
CATEGORIE: funcție
DESCRIERE:

Despre folosirea cu parametrul #*cale* vom discuta abia în Capitolul 18.

`INKEY$` este o funcție fără argumente, care dă ca rezultat un șir (prima de acest fel pe care o învățăm). Ea este foarte folosită în cadrul jocurilor, care sunt aproape de neconceput fără ea (cele scrise în BASIC, bine-înțeles). Efectul său: `INKEY$` citește și returnează tasta apăsată tocmai în momentul executării instrucțiunii. Dacă nici o tastă nu este apăsată, întoarce șirul nul `""`. Deducem că `INKEY$` poate returna un șir cu lungimea de cel mult un caracter. Dacă mai mult de o tastă a fost apăsată, iar combinația nu este una normală (CS + *ceva* sau SS + *ceva*) atunci `INKEY$` returnează tot șirul nul.

Pentru `INKEY$` contează *cursorul de la editare* (deși în timpul execuției programului el nu este vizibil), pentru că dacă am rămas în CAPS LOCK (modul `C`), atunci și `INKEY$` va întoarce majuscule! (Cursorul de editare este cel care arată în momentul în care scrieți programul, cum se va interpreta următoarea tastă apăsată. El este o literă clipitoare — `L`, `C`, `G`, `E` sau `K`.)

Experimentați:

```
5 PRINT INKEY$; : GOTO 5
```

sau

```
5 PRINT AT 0,0;INKEY$, : GOTO 5
```

■

Atât INKEY$ cât și INPUT sunt instrucțiuni care permit programului să țină cont de intervenția din exterior a utilizatorului în timpul execuției. Pe lângă ele mai există o singură instrucțiune de acest tip, IN, pe care o vom discuta în Capitolul 16. (Într-o oarecare măsură și PAUSE este o astfel de instrucțiune.) Să vedem care sunt diferențele dintre INPUT și INKEY$:

INPUT	INKEY$
e *comandă*	e *funcţie*
așteaptă introducerea informațiilor; programul nu continuă până ce aceasta nu s-a făcut	nu așteaptă apăsarea unei taste, ci trece mai departe
pentru a valida rezultatul trebuie apăsat CR	
poate citi oricâte caractere	poate citi cel mult un caracter
poate evalua expresiile introduse (fără LINE)	întoarce exact tasta apăsată

Iată o secvență de instrucțiuni care permite continuarea execuției numai la apăsarea tastei N (majusculă), lucru care nu poate fi realizat nici cu INPUT, nici cu PAUSE:

```
100 IF INKEY$ = "N" THEN GOTO 111
110 GOTO 100
```

sau

```
100 PAUSE 0 : IF INKEY$ <> "N" THEN GOTO 100
```

9.5 Exerciţiu rezolvat

Scrieți un program care să miște un pătrățel pe ecran cu ajutorul a patru taste:

stânga	5
dreapta	8
sus	7
jos	6

(am ales chiar tastele cu cursorul)

Rezolvare

Două lucruri mai subtile se găsesc în acest program:

1. nu trebuie să permitem deplasarea pătrățelului dincolo de margine (asta fac liniile 20, 25, 30, 35);

2. odată deplasat pătrățelul, va trebui să-l ștergem din vechea sa poziție. La asta folosesc variabilele `XA, YA`, care țin minte unde se afla el mai-nainte.

```
 5 LET x=0 : LET y=0 : REM coordonatele initiale
 6 LET xa = x : LET ya = y : REM coordonatele anterioare
 7
10 PRINT AT y,x; PAPER 0; " " ; AT ya,xa ; PAPER 7 ; " "
12 FOR i=1 TO 5: NEXT i : REM bucla de intarziere
15 LET xa = x : LET ya = y
16
20   IF x = 31 THEN GOTO 26
22   IF INKEY$ = "8" THEN LET x = x + 1: GOTO 10
23
25   IF x = 0 THEN GOTO 30
27   IF INKEY$ = "5" THEN LET x = x - 1: GOTO 10
29
30   IF y = 21 THEN GOTO 36
32   IF INKEY$ = "6" THEN LET y = y + 1 : GOTO 10
33
35   IF y = 0 THEN GOTO 40
37   IF INKEY$ = "7" THEN LET y = y - 1 : GOTO 10
40 GOTO 20
```

Programul ciclează în mod normal între liniile 20–40. Numai dacă s-a apăsat o tastă și coordonatele s-au modificat se trece prin linia 10 și pătrățelul se deplasează. Rulați-l și fără linia 12.

9.6 Exerciţii

1. Scrieți un program care mișcă aleator un „monstru" pe ecran. Atenție la ieșirea din ecran.

2. Folosind programul anterior și exercițiul rezolvat, scrieți un joculeț care să se desfășoare astfel: monstrul se mișcă aleator iar eu încerc să mă suprapun peste el, utilizând patru taste. Țineți și un fel de scor (de exemplu numărul de mutări).

Chapter 10

Din nou culori şi şiruri

De mult eram datori cu instrucțiunile de culoare. Am prezentat, ce-i drept, `PAPER` și `INK`, dar abia acum le înfățișăm pe toate. Apoi vom învăța și alte lucruri interesante.

- `INVERSE` pentru a inversa culorile;
- `BRIGHT` pentru puțină strălucire;
- `FLASH` pentru a clipi;
- `OVER` pentru suprapuneri de efect;
- `TO` pentru lucrul cu subșiruri;
- `VAL` pentru a transforma șirurile de caractere în expresii;
- despre formatul exponențial de scriere a numerelor.

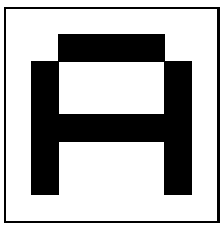
10.1 INVERSE

ENGLEZĂ: *inverse* = invers
SINTAXĂ: `INVERSE` *mod*
sau într-o instrucțiune `PRINT` sau `INPUT` sau grafică (Capitolul 12).
CATEGORIE: comandă
DESCRIERE:

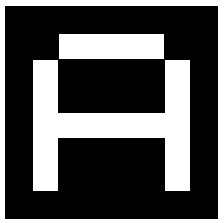
mod se rotunjește la cel mai apropiat întreg. Trebuie apoi să fie 0 sau 1, altfel se obține eroarea K.

Modul de folosire al lui `INVERSE` este același cu al lui `INK` și `PAPER`: fie drept comandă cu efect global, fie într-o instrucțiune `PRINT` sau `INPUT` cu efect local.

`INVERSE 0` este starea normală de tipărire. `INVERSE 1` este starea inversată. Efectul lui `INVERSE 1` este de a tipări pixelii care trebuie tipăriți în `INK` ca pixeli în culoarea `PAPER` și invers. Atenție: nu culoarea se schimba, ci caracterul tipărit. Dacă litera A în `INVERSE 0` arată așa:



atunci A în `INVERSE 1` arată așa: (Am încadrat litera într-un chenar ca să se vadă cele 8×8 pătrățele pe care le ocupă.)



Testați:

```
LET a$ = "TEST" : PRINT INVERSE 0 ; a$, INVERSE 1 ; a$,
```

10.2 BRIGHT

ENGLEZĂ: *bright* = strălucitor
SINTAXĂ: `BRIGHT` *nr*
sau în cadrul lui `PRINT` sau `INPUT` sau o instrucțiune grafică (Capitolul 12)
CATEGORIE: comandă
DESCRIERE:

În urma rotunjirii *nr* trebuie să fie 0, 1 sau 8 — altfel se generează eroarea K. `BRIGHT 0` este starea normală. Caracterele tipărite cu `BRIGHT 1` vor fi mai strălucitoare (atât `PAPER` cât și `INK`-ul lor). `BRIGHT 8`, ca și `PAPER 8` înseamnă transparent: un caracter tipărit cu `BRIGHT 8` va păstra strălucirea pătrățelului de pe ecran în care este scris.

Vedeți diferența:

Notă: unele HC-uri mai vechi nu pot afișa străluciri diferite.

```
LET a$ = "TEST" : PRINT BRIGHT 0 ; a$, BRIGHT 1 ; a$ ,
```

10.3 FLASH

ENGLEZĂ: **flashing** = clipitor
SINTAXĂ: `FLASH` *nr*
sau în cadrul lui `PRINT` sau `INPUT` sau o instrucțiune grafică (Capitolul 12)
CATEGORIE: comandă
DESCRIERE:

În urma rotunjirii, *nr* trebuie să fie 0, 1 sau 8 — sau se generează eroare K. `FLASH 0` este starea normală; caracterele tipărite cu `FLASH 1` vor alterna cam de două ori pe secundă culoarea `PAPER`-ului și a `INK`-ului, schimbându-le între ele.

`FLASH 8` înseamnă transparență: păstrarea `FLASH`-ului pătrățelului în care se face tipărirea.

```
FLASH 1:CLS
```

va face un ecran clipitor.

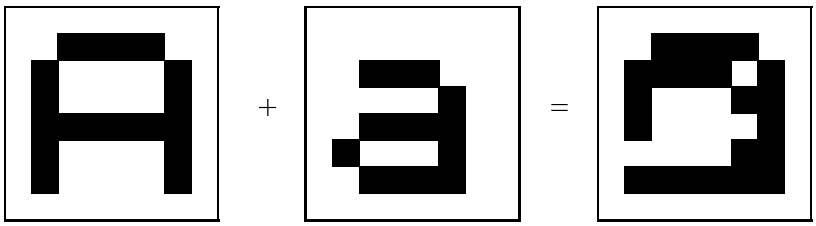
10.4 OVER

ENGLEZĂ: *over* = deasupra
SINTAXĂ: `OVER` *mod*
sau în cadrul lui `PRINT` sau `INPUT` sau o instrucțiune grafică (Capitolul 12)
CATEGORIE: comandă
DESCRIERE:

Aceasta este singura instrucțiune care permite să se tipărească ceva pe ecran fără a șterge ceea ce era în acel loc mai înainte. *mod* trebuie odată rotunjit să fie 0 sau 1, altfel survine eroarea K. `OVER 0` este starea normală de tipărire. Tipărirea în `OVER 1` acționează după cum urmează: fiecare din punctele caracterului care se tipărește se suprapune cu punctul care se afla pe ecran în acel loc, ca în tabel (notăm cu 0 punctele în PAPER și cu 1 punctele în INK):

Punctul inițial	Punctul tipărit	Rezultat
0	0	0
1	0	1
0	1	1
1	1	0

În limbajul logicii o astfel de „suprapunere" se numește *sau exclusiv*. Să vedem și un desen mărit, de două litere suprapuse în `OVER 1`:



Un scurt studiu asupra particularităților lui `OVER 1` ne va arăta că, tipărind de două ori în același loc un caracter oarecare, vom obține ceea ce se afla acolo înainte neschimbat!

Încercați:

```
PRINT AT 0,0;"A": OVER 1 :
PRINT AT 0,0 ;"a" : PAUSE 0 :
PRINT AT 0,0;"a"
```

10.5 TO

Aceasta nu este o instrucțiune în sensul dat în capitolul 0. Seamănă oarecum cu o funcție sau cu un operator. Ea se folosește pentru a extrage un subșir dintr-o valoare de tip șir. Forma ei de folosire este:

sir([*n*] TO [*m*])

Această expresie înseamnă:

- un șir format din caracterele celui inițial luate între cel cu numărul *n* și cel cu numărul *m*;
- dacă *n* lipsește: șirul de la primul până la caracterul *m*;
- dacă *m* lipsește: șirul de la caracterul *n* până la sfârșit;
- dacă lipsesc amândouă: șirul în întregime (vom vedea prin ce diferă `a$` de `a$( TO )`);
- dacă $n > m$: `""` (șirul nul);
- dacă $n = m$: `a$(`*n* `TO` *m*`)` este `a$(`*n*`)`.

Eroarea 3 se obține pentru limite negative sau superioare lungimii șirului (dar $m < n$).
Să ilustrăm:

```
LET a$="abcdef"
```

avem următoarele egalități (care nu constituie un program BASIC):

```
a$( TO 3) = a$(1 TO 3) = "abc"
a$(2 TO ) = a$(2 TO 6) = "bcdef"
a$( TO ) = a$( 1 TO 6) = a$(1 TO ) = a$( TO 6) = a$ = "abcdef"
a$(3 TO 3) = a$(3) = "c"
a$(8 TO 7) = ""
a$(7 TO 8) - eroare 3
a$(0 TO 3) - eroare 3
```

Putem folosi TO și pentru a extrage un subșir al unei variabile dimensionate (matrice); există două moduri de a scrie un subșir:

```
DIM a$(5,5,10)
a$(2,2, 4 TO 6) = a$(2,2) (4 TO 6)
```

10.6 Exerciţiu rezolvat

Scrieți un program care să rotească un lanț de caractere (cele care ies prin stânga să intre în dreapta).

Rezolvare

Foarte simplu, folosind TO:

```
 5 LET a$="-+-  -+-  -+-  -+-  -+-  -+-  -."
 7   REM a$ are 32 de caractere
10 PRINT AT 0,0;a$
15 LET a$ = a$(32) + a$( TO 31)
20 PAUSE 5: GOTO 10
```

■

Vom scrie acum un program (care nu-i bun la nimic altceva decât la ilustrarea lui TO) care va mișca pe ecran un semn în toate cele opt direcții, cu ajutorul a opt taste diferite. Dacă va ieși din ecran, va apărea în partea opusă. Vom plasa semnul într-un șir care ocupă tot ecranul, pe care-l vom tipări mereu. Vom modifica doar locul caracterului în cadrul șirului. Iată pentru început un tabel, care arată cu cât variază poziția semnului pentru fiecare direcție de mișcare:

Direcţia	Tasta	Variaţia
sus	1	−32
sus-dreapta	2	−31
dreapta	3	+1
dreapta-jos	4	+33
jos	5	+32
stânga-jos	6	+31
stânga	7	−1
stânga-sus	8	−33

```
  5 DIM a$(32 * 22): REM 704 = 32 * 22
  6 LET a$(1) = "#"
 10 IF INKEY$="1" THEN LET a$ = a$(33 TO ) + a$( TO 32)
 20 IF INKEY$="2" THEN LET a$ = a$(32 TO ) + a$( TO 31)
 30 IF INKEY$="3" THEN LET a$ = a$(704) + a$( TO 703)
 40 IF INKEY$="4" THEN LET a$ = a$(672 TO ) + a$( TO 671)
 50 IF INKEY$="5" THEN LET a$ = a$(673 TO ) + a$( TO 672)
 60 IF INKEY$="6" THEN LET a$ = a$(674 TO ) + a$( TO 673)
 70 IF INKEY$="7" THEN LET a$ = a$(2 TO ) + a$(1)
 80 IF INKEY$="8" THEN LET a$ = a$(34 TO ) + a$( TO 33)
 90 PRINT AT 0,0;a$
100 GOTO 10
```

Dacă doriți să sesizați procesul de tipărire al acestui șir enorm, transformați linia 90 în:

```
  90 PRINT PAPER RND*8; INK 9 ; AT 0,0; a$
```

■

Să mai spunem câte ceva despre `TO`:

Tot așa cum putem considera `b$(3)` ca variabilă de sine-stătătoare (când există variabila șir `b$`), căreia putem să-i atribuim diverse valori, tot așa putem opera cu părți de variabilă șir extrase cu `TO`. Atribuirea se face tot „procustean", în sensul că atunci când unui subșir dintr-o variabilă șir i se atribuie o valoare șir, aceasta este adusă la lungimea subșirului, fie prin trunchiere, fie prin umplere cu spații. Un exemplu:

```
  5 LET a$ = "Calculator personal"
 10 LET a$(11 TO ) = "ul HC 85 este bun"
 20 PRINT a$
 30 LET a$( TO ) = "!": PRINT a$+"!"
```

cu efectul pe ecran

```
Calculatorul HC 85
!                 !
```

Aceasta ne arată că din șirul `"ul HC 85 este bun"` s-a păstrat numai atât cât încăpea în `a$` de la 11 la sfârșit. De asemenea, `LET a$( TO ) = ceva` păstrează lungimea lui `a$`, spre deosebire de `LET a$ = ceva`, care îi dă lui `a$` lungimea lui `ceva`.

10.7 VAL

ENGLEZĂ: *VALue* = valoare
SINTAXĂ: `VAL` *sir*
CATEGORIE: funcție
DESCRIERE:

`VAL` este una dintre cele mai puternice instrucțiuni BASIC. Puține limbaje au o funcție atât de interesantă, dar nu ne vom bate capul aici discutând de ce. Ea primește ca argument un șir de caractere și returnează un număr.

`VAL` își calculează rezultatul considerând șirul primit ca pe un șir de caractere ce descrie o expresie cu rezultat număr, pe care o evaluează! Dacă șirul nu are forma unei expresii, vom obține eroarea C.

Exemple:

Comanda	**Efect**
`PRINT VAL "234"`	234
`PRINT VAL "2+5+7"`	14
`LET a=23:PRINT VAL "INT a-2.4"`	20.6
`PRINT VAL "SGN (2) + VAL ""-1"""`	0
`PRINT VAL "a$"`	eroare C
`LET a$ = "VAL a$":PRINT VAL a$`	blochează calculatorul într-o buclă infinită, care se va termina odată cu memoria, afișând o eroare

Atenție: mai sus, între ghilimele, avem caracterele: `INT, VAL, SGN` și nu șirurile formate din aceste litere!

10.8 Formatul ştiinţific (exponenţial)

Pentru numerele foarte mari sau foarte mici, calculatorul adoptă o metodă specială de afișare a valorii. Aceasta folosește doar 8 cifre semnificative. Numărul este scris sub forma:

$$[-]valoare_1.valoare_2E[-][+]valoare_3$$

$valoare_i$ sunt numere întregi.

Primul semn este al numărului, al doilea al *exponentului*. (Exponentul este numărul $valoare_3$, cel scris după `E`.) Valoarea tipărită în forma de mai sus are următoarea semnificație:

$[-]valoare_1.valoare_2 \times 10^{[-]valoare_3}$

Cu alte cuvinte, exponentul este puterea lui 10 cu care trebuie să înmulțim numărul zecimal cu partea întreagă $valoare_1$ și partea zecimală $valoare_2$ pentru a obține valoarea corectă.

Numerele pot fi introduse (sub formă de constante) în format științific. Se poate folosi și e (literă mică) pentru exponent.

Să vedem niște exemple:

instrucțiune	rezultat
`PRINT 1.234e+3`	`1234`
`PRINT 1.2345e+3`	`1234.5`
`PRINT 1234e-3`	`1.234`
`PRINT 3993967295`	`3.9939673E+9`
`PRINT 0.0000000001`	`1E-10`

Intern se memorează fără nici o pierdere numere până la $4\,294\,967\,295$. Acesta este $2^{32}-1$. Nu pot tipări niciodată mai mult de 8 cifre semnificative (14 semne cu tot cu exponent). Că intern se memorează mai mult, se vede încercând:

```
PRINT 4294967295, 4294967295 - 4.29496729E10
```

care scrie 5.

Numărul maxim ce se poate reprezenta în calculator (cu o oarecare lipsă de precizie) este aproximativ 4E+38. Dacă în cursul calculelor se depășește această valoare, se obține mesajul de eroare 6.

10.9 Exerciţii

1. Ce va tipări programul următor ?

   ```
   5 PRINT 1e11 - 1e11 + 1, 1e11 + 1 - 1e11
   ```

   Ce e bizar în asta ?

2. Cum va arăta ecranul după execuția următorului program ?

   ```
    5 CLS
    6 LIST
    7 OVER 1 : PRINT AT 0,0;
   10 FOR I=1 TO 200: PRINT i;: NEXT I
   20 PRINT AT 0,0;
   50 LIST: OVER 1: PRINT AT 0,0;
   70 FLASH 1
   80 DIM a$(64)
  100 FOR i=1 TO 11: PRINT a$:NEXT i
   ```

3. Scrieți un program care să miște pe linia 21 un tun stânga–dreapta, cu tastele [x], respectiv [z] și să tragă în sus la apăsarea tastei blanc [ ].

Chapter 11

Logică, printre altele

Vom extinde gama operațiunilor logice pe care le putem efectua, prin niște instrucțiuni bizare, vom învăța despre ordinea de evaluare și vom mai învăța niște funcții care lucrează cu șiruri.

- funcții infixate;
- `NOT` pentru a calcula inversa unei valori de adevăr;
- `OR` pentru a găsi cel puțin o valoare adevărată;
- `AND` pentru a vedea dacă toate valorile sunt adevărate;
- prioritatea operatorilor și funcțiilor BASIC;
- `LEN` pentru a calcula lungimea șirurilor;
- `VAL$` — sora lui `VAL` pentru evaluarea șirurilor.

Până acum am folosit pe larg tot felul de operațiuni pentru a alcătui expresii. Am folosit pentru aceasta semne cum ar fi `+`, `-`, `*`, la un moment dat chiar `=`, `<`, `<>`. Am mai folosit și funcții. Semnele din primele două categorii le-am mai numit și operatori. De fapt ele sunt tot funcții, dar sunt deghizate prin modul lor de folosire: ele nu se scriu înaintea argumentelor lor, ci între acestea! De aceea ele se mai numesc și funcții *infixate*, pentru a le deosebi de funcțiile *prefixate*, care se scriu înaintea argumentelor lor. Mai sunt „ciudate" și pentru că au câte două argumente. Funcții prefixate de câte două argumente nu am văzut până acum, dar le vom întâlni puțin mai încolo. Acum că am eliminat deosebirile care aruncau în categorii atât de deosebite funcțiile și operatorii, putem să trecem mai departe.

11.1 NOT

ENGLEZĂ: *not* = nu
SINTAXĂ: NOT $expr$
CATEGORIE: funcție
DESCRIERE:

NOT este o funcție prefixată tipică. Din punct de vedere logic, ea operează cu o condiție ($expr$) căreia îi inversează valoarea de adevăr. Ea calculează ceva de genul:

$expr$	NOT $expr$
adevărat	fals
fals	adevărat

Ne vom aminti că BASIC-ul folosește tipul de date numeric pentru a reprezenta valori de adevăr. Dacă vă amintiți, corespondența era:

0	fals
$\neq 0$	adevărat

Se pune problema ce face NOT cu numărul căruia i se aplică (căci număr este)? Funcționarea „generalizată" la numere a lui NOT este:

$expr$	NOT $expr$
0	1
$\neq 0$	0

Instrucțiunea

```
IF NOT (a = 0) THEN GOTO 50
```

este totuna cu

```
IF a <> 0 THEN GOTO 50
```

sau cu

```
IF a THEN GOTO 50
```

și, întâmplător, cu

```
IF (NOT a) = 0 THEN GOTO 50
```

11.2 OR

ENGLEZĂ: *or* = sau
SINTAXĂ: $expr_1$ `OR` $expr_2$
CATEGORIE: funcție infixată
DESCRIERE:

La nivel logic `OR` lucrează după cum arată și tabelul:

$expr_1$	$expr_2$	$expr_1$ `OR` $expr_2$
fals	fals	fals
adevărat	fals	adevărat
fals	adevărat	adevărat
adevărat	adevărat	adevărat

adică rezultatul este adevărat dacă cel puțin una dintre expresii este adevărată și este fals dacă amîndouă sunt false.

```
10 IF (a=1) OR (a=3) THEN STOP
```

Linia de mai sus va opri programul dacă `a=1` sau `a=3`.

Revenind la expresii aritmetice, comportarea lui `OR` este mai bizară; vom vedea că poate întoarce nu numai 1 sau 0. Aplicată la valori care sunt doar 0 și 1, se comportă însă după cum ne așteptăm.

$$expr_1 \texttt{ OR } expr_2 = \begin{cases} 1 & dacă\ expr_2 \neq 0 \\ expr_1 & dacă\ expr_2 = 0 \end{cases}$$

11.3 AND

ENGLEZĂ: *and* = și
SINTAXĂ: $expr_1$ `AND` $expr_2$
CATEGORIE: funcție infixată
DESCRIERE:

La nivel logic, `AND` lucrează după cum arată și tabelul:

$expr_1$	$expr_2$	$expr_1$ `OR` $expr_2$
fals	fals	fals
adevărat	fals	fals
fals	adevărat	fals
adevărat	adevărat	adevărat

adică rezultatul este fals dacă cel puțin una dintre expresii este falsă și este adevărat dacă amîndouă sunt adevărate.

Folosită cu expresii aritmetice, `AND` se comportă în două feluri; în ambele cazuri $expr_2$ este întotdeauna un număr:

1. dacă $expr_1$ este tot un număr ;

$$expr_1 \texttt{ AND } expr_2 = \begin{cases} 0 & dacă\ expr_2 = 0 \\ expr_1 & dacă\ expr_2 \neq 0 \end{cases}$$

2. dacă $expr_1$ este un șir (sic!):

$$expr_1 \texttt{ AND } expr_2 = \begin{cases} "" & dacă\ expr_2 = 0 \\ expr_1 & dacă\ expr_2 \neq 0 \end{cases}$$

Rezultatul este deci un șir!

Iată doar un exemplu de folosire a lui AND cu un argument șir (deși probabil că ar trebui rescris cu dolari în loc de lei):

```
40 INPUT "un numar intreg pozitiv nenul ";a
50 PRINT "mai aveti " ; a ; "le"+("u" AND a=1)+("i" AND a>1)
```

Dintre expresiile a=1 și a>1, doar una va fi adevărată, deci una din expresiile cu AND va da rezultatul "", iar cealaltă șirul respectiv. După caz va apărea cuvântul leu sau lei .

Să mai vedem niște exemple cu operatori logici:

```
10 IF a=b AND b=c AND NOT c=0 THEN PAUSE 0: NEW
```

acest scurt program se va autodistruge numai dacă a=b, b=c, c<>0. Sau:

```
5 IF (a=0 OR a=1) AND (b=0 OR b=1) AND (NOT a=b) THEN CLS
```

Analizând această linie, deducem că ecranul se va șterge în doar două cazuri: a=1 și b=0, sau a=0 și b=1.

Faptul că a este cuprins între b și c se poate testa în felurite moduri:

```
10 IF a > b AND a < c THEN ....
```

sau

```
10 IF NOT (a <= b OR a >= c) THEN ....
```

O ultimă observație în legătură cu AND: deși cele două linii care urmează par identice ca efect, ele au o semnificație diferită:

```
IF cond1 AND cond2 THEN instr
IF cond1 THEN IF cond2 THEN instr
```

cu toate că instr se va executa dacă și numai dacă cond1 și cond2 sunt ambele adevărate (nenule). Ca să înțelegeți diferența, să considerăm următorul caz:

```
cond1 este c=0
cond2 este a/c = 2
```

Pentru c=0 prima formă va da eroarea 6, pentru că AND evaluează ambele argumente, deci face și împărțirea la 0, pe când a doua formă va funcționa corect — prima condiție fiind falsă, a doua nu se mai evaluează.

■

Pentru că am învățat deja o grămadă de funcții și operatori, este cazul să vedem în ce ordine se efectuează sau, cu alte cuvinte care este precedența (numită și *prioritatea*) fiecăreia, pentru a scrie programe corecte. Precedența, cu cât este mai mare, cu atât funcția respectivă se va aplica mai repede. Iată lista precedențelor:

Operator, funcție	Prioritate
OR	2
AND	3
NOT	4
= > < >= <= <>	5
- + (cu număr sau șir)	6
* /	8
- (pentru negare, unar)	9
^	10
toate celelalte funcții	11
calcul indici de tabel	12

Astfel, linia următoare:

```
10 LET a$ = a$ + "le"+"u" AND a=1 + "i" AND a>1
```

înseamnă

```
10 LET a$ = ( a$+"le"+"u") AND (a = (1 + "i")) AND (a > 1)
```

ceea ce este evident incorect, căci se adună un număr cu un șir. Pentru a schimba ordinea operațiilor se folosesc parantezele, după cum am învățat.

■

Vom vedea acum o utilizare a unui fapt pe care îl cunoaștem de mai multă vreme. După cum am spus, toate operațiunile „logice", — comparații (=, <, <>), sau funcții logice de alte valori (NOT, AND) — generează un rezultat numeric. Vom vedea cum putem folosi acest rezultat pentru a scrie expresii mai concise. Ne vom folosi în special de faptul că o comparație dă 1 pentru adevărat și 0 pentru fals.

```
5 INPUT a : PRINT a=1 : GOTO 5
```

va tipări „valoarea de adevăr" (0 sau 1) a propoziției a=1.

Să vedem cum rezolvăm următoarea problemă: în funcție de valoarea numărului a, care poate fi 1, 2, 3 sau 4, trebuie să-i atribuim lui b una din valorile 3, 5, 7, respectiv 11. Soluția este:

```
LET b=3 * (a=1) + 5 * (a=2) + 7 * (a=3) + 8 * (a=4)
```

Cum „merge" linia de mai sus? Simplu, una singură din egalitățile a=1, a=2, a=3, a=4 va fi adevărată. Paranteza respectivă va avea ca rezultat 1, celelalte 0. Atunci, dintre numerele 3, 5, 7, 11, numai unul va fi înmulțit cu valoarea nenulă 1, și acesta va deveni valoarea lui b. Să observăm că, dacă a nu este 1, 2, 3 sau 4, b capătă valoarea 0, căci toate parantezele sunt nule.

Testele de genul a=1 se pun întotdeauna între paranteze, din cauza priorității foarte mici a operațiunii =. De exemplu:

```
PRINT a=5 + a=6
```

e totuna cu

```
PRINT (a = (5 + a)) = 6
```

care e întotdeauna 0. (De ce ?)

De asemenea, să remarcăm că

```
IF a=b=c THEN ...
```

înseamnă de fapt

```
IF (a=b) = c THEN ...
```

care este cu totul altceva decât

```
IF (a=b) AND (b=c) THEN
```

Prima formă este adevărată dacă c este chiar valoarea de adevăr a propoziției a=b.

Vom vedea o altă aplicare a acestei tehnici în următorul

11.4 Exerciţiu rezolvat

Scrieți un program care mișcă un pătrățel pe ecran, prin intermediul a patru taste.

Rezolvare

Enunțul este același ca la exercițiul rezolvat din Capitolul 9 și o parte din metodele de acolo le vom păstra. Mă refer în special la folosirea variabilelor xa, ya.

```
 5 LET x=0 : LET y=0: LET xa=x: LET ya=y
10    PRINT AT yi,xi; " "
12    LET xi=x : LET yi=y
15    PRINT AT y,x; PAPER 0;" "
17    FOR i=1 TO 5: NEXT i
20    LET x=x + (INKEY$="8" AND x < 31) - (INKEY$="5" AND x > 0)
30    LET y=y + (INKEY$="6" AND y < 21) - (INKEY$="7" AND y > 0)
40 GOTO 10
```

Comentarii

Spre deosebire de soluția dată atunci, se trece prin instrucțiunile de scriere din liniile 10–15 chiar dacă nu s-a apăsat vreo tastă. Acest lucru va cauza clipirea pătrățelului, datorată ștergerii sale în linia 10 și redesenării sale în linia 15.

Să vedem cum funcționează acele paranteze bizare. Pentru că seamănă foarte mult între ele, vom discuta doar despre una dintre ele. De pildă: `(INKEY$="5" AND x>0)`. Avem aici două teste: `INKEY$="5"`, care dă 0 dacă nu s-a apăsat tasta 5 sau 1 dacă ea a fost apăsată și `x>0`, care dă 1 dacă pătrățelul nu a atins marginea din stânga sau 0 altfel. Făcând `AND` între ele, vom obține 1 doar dacă cineva încearcă să mute pătrățelul la stânga (apăsând 5) și acest lucru mai este posibil. Întreaga paranteză dă 1, care este scăzut din valoarea curentă a lui `x`, cauzând astfel mișcarea către stânga.

Ciclul din linia 17 este o *buclă de întârziere*. Ea nu se putea realiza cu `PAUSE`, căci apăsarea unei taste întrerupe pauza. Dacă nu doriți ca pătrățelul să clipească, lângă bucla de întârziere puteți adăuga un `PAUSE 0`.

Dacă vreți ca pătrățelul să lase urme, scoateți liniile 10 și 12.

11.5 LEN

ENGLEZĂ: *LENgth* = lungime
SINTAXĂ: `LEN` *sir*
CATEGORIE: funcție
DESCRIERE:

`LEN` este o funcție foarte utilă. Argumentul ei este un șir, iar răspunsul — un număr, care reprezintă chiar lungimea, socotită în caractere, a șirului. Datorită ciudățeniei acestui BASIC, un șir poate avea doar 10 caractere chiar dacă, scris pe ecran, nu încape într-un singur rând (pentru că un caracter poate să fie format din mai multe simboluri tipăribile).

`PRINT LEN "variabila sir"` va tipări 13.

`PRINT LEN "A>=B"` va tipări fie 3, dacă >= este un singur caracter, fie 4, dacă este format din două caractere, > și =

.

`PRINT LEN ""` va tipări 0.

11.6 Exerciţiu rezolvat

Numărați de câte ori apare într-un șir un anume caracter.

Rezolvare

Cred că programul este suficient de simplu pentru a nu avea nevoie de comentarii:

```
 5 INPUT "sirul :"; LINE a$: IF LEN a$=0 THEN GOTO 5
 7 INPUT "caracterul :"; LINE b$: IF LEN b$=0 THEN GOTO 7
 9 LET b$=b$(1) : LET k=0 : REM k este rezultatul
10 FOR i=1 TO LEN a$
```

```
15   IF a$(i) = b$ THEN LET k = k + 1: FLASH 1
20   PRINT a$(i); : FLASH 0
25 NEXT i
30 PRINT '"caracterul """ ; b$; """ figureaza de " ; k ;" ori"
```

11.7 VAL$

ENGLEZĂ: *VALue* = valoare, *String* = şir
SINTAXĂ: `VAL$` *şir*
CATEGORIE: funcţie
DESCRIERE:

`VAL$` este sora lui `VAL`. Ca şi `VAL`, primeşte un şir ca argument dar, aşa cum ne arată şi dolarul din coadă, dă ca rezultat tot un şir. `VAL$` face acelaşi lucru ca `VAL`: presupune că şirul argument este o expresie BASIC, pe care o evaluează (mai exact, caracterele şirului formează o expresie BASIC). Diferenţa faţă de `VAL` este că `VAL$` se aşteaptă ca rezultatul expresiei să fie un şir, şi nu un număr. Dacă şirul nu este o expresie corectă, se obţine eroarea C.

Un exemplu:

```
3 LET cu=2
5 LET a$="cu"
7 LET b$="a$+a$"
8 PRINT b$;" - ";VAL$ b$
```

care are ca efect

```
a$+a$ - cucu
```

O linie `10 PRINT VAL$ a$` ar da eroarea C, pentru că putem interpreta conţinutul lui a$ ca nume al unei variabile numerice (cu), care nu dă un rezultat de tip şir.

Atenţie: `VAL` şi `VAL$` evaluează *expresii* şi nu *comenzi* BASIC. O instrucţiune de tipul

```
 LET a$="PAPER 5" : PRINT VAL$ a$
```

nu va face hârtia albastră, ci va genera eroare C.

11.8 Exerciţii

1. Se defineşte operatorul logic *sau exclusiv* — notat *xor* cu următorul tabel de adevăr:

$cond_1$	$cond_2$	$cond_1$ `XOR` $cond_2$
fals	fals	fals
adevărat	fals	adevărat
fals	adevărat	adevărat
adevărat	adevărat	fals

Creați această funcție combinând operatori cunoscuți. Încercați să vă gândiți care ar fi forma ei generalizată (așa cum reiese din programul vostru).

2. Scrieți un program care numără de câte ori un șir se găsește în alt șir.

3. Folosind instrucțiunea `POKE 23692, 255` înainte de fiecare tipărire întrebarea `scroll?` nu mai survine și alunecarea este continuă. (Despre `POKE` în Capitolul 15.)

   Scrieți un joc de „raliu" astfel:

   - „mașina" (litera `V` în desen) se află pe linia 21;
   - jucătorul o poate mișca stânga-dreapta cu două taste alese de el;
   - șoseaua „urcă" pe ecran, generată pe linia 21 în mod sinuos;
   - jocul se oprește când mașina iese din șosea;
   - scorul e proporțional cu timpul de joc;
   - urma mașinii nu trebuie neapărat să fie ștearsă (deși nu e greu).

   Iată cum ar putea să arate ecranul la un moment dat:

```
|          #    V #    scor 223  |
|           #  V   #              |
|           #  V   #              |
|           # V    #              |
|          #  V   #               |
|         #   V  #                |
|        #    V #                 |
|       #    V #                  |
|      #    V #                   |
|      #    V #                   |
|       #    V #                  |
|        #    V #                 |
|        #      V      CRASH !    |
```

Chapter 12

Grafică

Vom învăța să facem desene mai finuțe și, totodată câteva lucruri noi despre setul de caractere.

- grila de înaltă rezoluție;
- `PLOT` pentru a face puncte;
- `DRAW` pentru a trasa linii și arce de cerc;
- `CIRCLE` pentru desenat cercuri;
- acțiunea culorilor la instrucțiunile grafice;
- `POINT` pentru a inspecta ecranul;
- `CHR$` pentru caractere;
- caractere „de control";
- `CODE` pentru decodificarea caracterelor.

Încă din capitolul 1 am spus că ecranul este format din pixeli, în număr de 256×192. Am mai văzut că aceștia erau grupați câte 8×8 formând o rețea de pătrate pe care am numit-o „grila de joasă rezoluție". În fiecare din aceste pătrate am văzut că putem scrie un caracter. Dar putem face mai mult decât atât: putem controla individual fiecare din cei 256×192 de pixeli. O restricție există totuși; o vom desluși în Capitolele 14 (`ATTR`) și 15: într-unul din pătratele grilei de joasă rezoluție nu pot exista simultan mai mult de două culori distincte. Odată fixate aceste două culori pentru un pătrățel, putem vorbi despre punctele din interiorul său ca fiind de două tipuri:

- aprinse sau în culoarea INK;
- stinse sau în culoarea PAPER.

Grila de înaltă rezoluție are anumite ciudățenii. Întâi, cele două rânduri, care de obicei formează „partea de jos" a ecranului nu fac parte din grila de înaltă rezoluție. De aceea grila are pe verticală numai 192 – 16 = 176 de puncte.

După cum ne așteptăm, coordonatele punctelor variază:

- pe orizontală între 0 și 255;
- pe verticală între 0 și 175.

Există în definirea grilei de înaltă rezoluție două asimetrii (comparativ cu cea de joasă rezoluție):

- originea sistemului de coordonate nu este în colțul din stânga-sus, ci în cel din stânga-jos, chiar deasupra celor două linii „interzise". Coordonatele sale sunt 0, 0;
- coordonatele punctelor în grila de înaltă rezoluție se specifică în ordine inversă: întotdeauna întâi coordonata *x* — pe orizontală — și apoi coordonata *y* — pe verticală. Astfel colțurile grilei au coordonatele:

(0,0)	stânga-jos
(0,175)	stânga-sus
(255,175)	dreapta-sus
(255, 0)	dreapta-jos

12.1 PLOT

ENGLEZĂ: *to plot* = a desena; a trasa
SINTAXĂ: `PLOT` [**culori**] *x*, *y*
CATEGORIE: comandă
DESCRIERE:

Despre folosirea culorilor în instrucțiunile grafice vom vorbi după ce le vom prezenta pe toate. *x* și *y* trebuie să fie două expresii care generează întregi între limitele 0-255 pentru *x* și 0-175 pentru *y*. De fapt, valorile lor sunt rotunjite și luate în modul. Dacă nu se respectă aceste condiții, survine eroarea B.

Deși depinde destul de mult de valorile lui `OVER` și `INVERSE`, cum vom vedea în partea care descrie folosirea culorilor, putem spune că `PLOT` „face" punctul cu coordonatele specificate *x, y*.

Încercați:

```
5 PLOT RND * 255, RND * 175 : GOTO 5
```

12.2 DRAW

ENGLEZĂ: *to draw* = a trasa
SINTAXĂ: `DRAW` [**culori**] *x*,*y* [,*u*]
CATEGORIE: comandă
DESCRIERE:

Discutăm separat DRAW cu 2 și cu 3 parametri.

12.2.1 DRAW *x,y*

desenează un segment cu lungimea orizontală (proiecția pe orizontală) *x* și cu lungimea proiecției verticale *y*. Segmentul se trasează începând de la ultimul punct făcut anterior pe ecran (cu un PLOT, ultimul capăt de la un DRAW sau ultimul punct al unui CIRCLE). Dacă nici o altă instrucțiune grafică nu s-a mai executat de la ultimul CLS, trasarea începe din origine. Din această cauză spunem că DRAW lucrează în *coordonate relative* (adică face un segment până la coordonatele *x, y*, dar considerând că originea este în ultimul punct trasat). Dacă unul din punctele segmentului iese din ecran, se obține eroarea B.

Iată un program care trasează un pătrat:

```
PLOT 5,5 : DRAW 100,0 : DRAW 0,100 : DRAW -100,0 : DRAW 0,-100
```

Sesizați rolul parametrilor negativi la DRAW: apropie capătul respectiv de origine.

12.2.2 DRAW *x, y, u*

desenează un arc de cerc cu capetele chiar în capetele segmentului pe care l-ar fi trasat același DRAW fără ultimul parametru. Valoarea *u* specifică lungimea arcului de cerc, în *radiani*. (Un radian are un arc de lungime egală cu raza cercului. Din formula circumferinței rezultă că 3.141592..., adică π radiani, înseamnă un arc de 180 de grade — un semicerc.) Semnul lui *u* precizează care din cele două arce posibile se trasează: + înseamnă cel în sens trigonometric, – cel în sens orar.

Dacă un punct al arcului iese din ecran, se obține eroarea B.

Iată cum trasăm 100 de arce aleatoare între două puncte (un fel de glob):

```
5 FOR i = 1 TO 100
6   LET u = RND * 3.1415 * 2 - 3.1415
7   PLOT 128,50 : DRAW 0,100, u
8 NEXT i
```

Avantajul trasării în coordonate relative este acela că, schimbând doar poziția primului punct, mutăm întreaga figură, cum ilustrează și următorul exemplu:

```
5 PLOT 0,87
7 FOR i = 1 TO 16
8  DRAW 8,8 : DRAW 8,-8 : DRAW -8,-8 : DRAW -8,8 : DRAW 15,0
9 NEXT i
```

Algoritmul care trasează arce de cerc le aproximează de fapt cu segmente mititele. Pentru a trasa un arc, el este împărțit în multe arce mici care se trasează ca segmente. Dacă arcul de trasat este foarte mare (ceea ce ar trebui să corespundă de fapt la o mulțime de cercuri suprapuse), împărțind arcul în subarce mai mici se obțin totuși arce de dimensiuni suficient de mari ca aproximarea lor cu segmente să fie grosolană. (Numărul de segmente nu crește nelimitat.) De aceea, trasarea unor arce cu lungime mare în radiani dă naștere de fapt la niște foarte frumoase arabescuri. Experimentați:

```
PLOT 30,30 : DRAW 10,10,12000
```

sau

```
PLOT 120,20 : DRAW 10,10,9890
```

12.3 CIRCLE

ENGLEZĂ: *circle* = cerc
SINTAXĂ: `CIRCLE` [**culori**] *x*, *y*, *r*
CATEGORIE: comandă
DESCRIERE:

`CIRCLE` *x, y, r* trasează un cerc cu coordonatele absolute ale centrului *x, y* și cu raza *r* (pixeli). Pentru $r \leq 1$ face doar `PLOT` *x, y*. Dacă unul din punctele cercului iese din ecran avem eroare B. Trasarea cercului începe și se termină într-un punct de coordonate aproximativ $x + r,\ y$, deci `CIRCLE` schimbă poziția ultimului punct, față de care `DRAW` lucrează!

12.4 Acţiunea culorilor

1. Dacă nu sunt inserate directive de culoare în cadrul instrucțiunii:

   Instrucțiunile grafice (`PLOT, DRAW, CIRCLE`) trasează punctele în culoarea `INK` globală curentă. Toate punctele `INK` aflate în pătrățelul 8×8 din care face parte punctul își schimbă culoarea în noua culoare `INK` (nu pot exista două culori `INK` în același pătrățel 8×8). Iată un exemplu:

   ```
   PRINT AT 0,0; INK 1;"A": PAUSE 0: INK 3: PLOT 0,175
   ```

   Instrucțiunile **nu** schimbă PAPER-ul, BRIGHT-ul și FLASH-ul în pătrățele din care fac puncte (ele sunt implicit `PAPER 8, BRIGHT 8, FLASH 8`).

   `OVER` și `INVERSE` au efecte foarte ciudate asupra punctelor trasate de instrucțiunile grafice, după cum urmează:

   - `OVER 0, INVERSE 0` înseamnă trasarea punctelor în culoarea `INK`;
   - `OVER 0, INVERSE 1` duce la ștergerea punctelor trasate (sunt aduse la culoarea `PAPER`);
   - `OVER 1, INVERSE 0` produce inversarea punctelor trasate (din `INK` în `PAPER` și viceversa);
   - `OVER 1, INVERSE 1` lasă punctele neschimbate, însă modifică poziția ultimului `PLOT` și culorile din pătrățelul respectiv (`INK`-ul).

2. Instrucțiunile de culoare (`INK, PAPER, BRIGHT, FLASH, INVERSE, OVER`) pot fi inserate în cadrul celor grafice, separându-le cu ; sau cu , (nu și cu '). Atunci ele vor avea doar un efect local (valabil doar pentru instrucțiunea respectivă).

În plus, inserarea lui PAPER, FLASH sau BRIGHT face ca punctele trasate să schimbe acești parametri pentru întreg pătrățelul din care fac parte. INVERSE și OVER acționează în același fel. Putem rezuma acțiunea lor:

INVERSE	OVER	acțiune
0	0	punct = 1
0	1	punct = punct vechi XOR 1
1	0	punct = 0
1	1	punct = punct vechi

Am notat:

- cu 1 un punct care este în INK;
- cu 0 un punct care este în PAPER;
- cu XOR operația de SAU EXCLUSIV (definită în Capitolul 11, Exercițiul 1).

Dacă într-un pătrățel avem aceleași valori pentru INK și PAPER, nu înseamnă că în calculator nu se știe care puncte sunt INK și care PAPER. Ne putem convinge, schimbând ulterior culorile cu un PLOT:

```
PRINT PAPER 5; INK 5; AT 21,0;"a": PAUSE 0: PLOT PAPER 6, INK 0; 0,0
```

12.5 POINT

ENGLEZĂ: *point* = punct
SINTAXĂ: POINT (*x*,*y*)
CATEGORIE: funcție
DESCRIERE:

x este între 0 și 255, *y* între 0 și 175; altfel obținem eroare B. POINT este prima funcție prefixată cu două argumente, care trebuie puse între paranteze. POINT este într-un anume sens funcția opusă comenzii PLOT. Ea întoarce un număr, 0 sau 1, astfel:

- 0 dacă punctul (*x*,*y*) este în culoarea INK a pătrățelului lui;
- 1 dacă punctul (*x*,*y*) este în culoarea PAPER din acel pătrățel.

Puteți folosi POINT pentru a testa, de pildă, trecerea unui desen mobil printr-un punct.

■

Trebuie să atragem atenția că trasarea segmentelor e aproximativă:

```
DRAW 70,40 : DRAW INVERSE 1, -70, -40
```

nu șterge segmentul cu totul! Pentru asta trebuie scris:

```
DRAW 70, 40: DRAW OVER 1, INVERSE 1, -70, -40: DRAW INVERSE 1, 70, 40
```

12.6 CHR$

ENGLEZĂ: *CHaRacter* = caracter
SINTAXĂ: `CHR$` *cod*
CATEGORIE: funcție
DESCRIERE:

cod trebuie să fie cuprins, după rotunjire, între 0 și 255; în caz contrar se obține eroarea B. `CHR$` întoarce caracterul cu codul *cod*.

HC85 folosește un set de caractere format din 256 de elemente. O clasificare a lor este dată în Capitolul 0. În plus, fiecare caracter are un cod numeric (formă în care este de altfel și memorat, căci memoria unui calculator conține de fapt doar numere). Pentru multe din caracterele sale HC folosește un cod standardizat, numit ASCII. ASCII înseamnă „American Standard Code for Information Interchange", adică „Codul American Standard pentru schimb de informații". HC însă folosește și coduri care nu sunt ASCII pentru unele din caracterele sale (de altfel codul ASCII descrie numai 128 de caractere).

După cum este de imaginat, codurile sunt cuprinse între 0 și 255. Anexa B pezintă pe coloana 1 codul în baza 10, pe coloana 2 codul în baza 16 și pe coloana 3 caracterul corespunzător. Celelalte coloane nu ne interesează pentru moment; (ele arată codurile numerice ale instrucțiunilor limbajului de asamblare Z80).

Astfel `PRINT CHR$ 100` va tipări litera `d` (mic). Până aici nimic deosebit. Mai interesante sunt însă caracterele pe care în Capitolul 0 le-am numit „caractere de control". Ele au codurile de la 0 la 31. Înainte de a le discuta, să „rugăm" calculatorul să ne afișeze celelalte caractere:

```
 5 FOR i = 32 TO 255
 7  PRINT i;"  ";CHR$ i,
 9 NEXT i
```

Caracterele de control se mai numesc și *neprintabile*, pentru că, folosite în instrucțiunea `PRINT`, nu au un efect obișnuit.

După cum vedeți (în Anexa B), cea mai mare parte poartă mențiunea „nefolosit". Ele sunt practic echivalente cu caracterul cu codul 0 și tipărirea lor produce un semn de întrebare. Nici o tastă nu poate genera (la `INKEY$`) un asemenea caracter. Să le discutăm pe celelalte, luate separat:

caracterul cu codul 6 numit „`PRINT` virgulă". Tipărirea acestui caracter are exact efectul pe care l-ar fi avut întâlnirea în acest punct a unei virgule în instrucțiunea `PRINT`. Încercați:

```
PRINT "a"; CHR$ 6; "b"
```

caracterul cu codul 7 numit și „EDIT". Tipărit nu are nici un efect. Poate fi însă returnat de `INKEY$`, când se apasă [EDIT] ([CS] + [1]);

caracterul codificat 8 – „cursor stânga". Tipărirea sa mută pe ecran cursorul de la `PRINT` cu o pătrățică spre stânga. El este de asemenea generat de tasta „săgeată stânga" ([CS] + [5], ⇐). Încercați:

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

caracterul cu cod 9 – „cursor dreapta". Tipărirea lui mută cursorul spre dreapta. Scrierea lui este echivalentă cu instrucțiunea

```
PRINT PAPER 8;INK 8;FLASH 8;BRIGHT 8;INVERSE 0;OVER 1;" "
```

De asemenea, este generat de tasta „săgeată dreapta" (CS + 8);

caracterul codat 10 – „cursor jos" este generat de tasta „săgeată jos" (CS + 6). Tipărit, nu are efect;

caracterul codat 11 – „cursor sus" este generat de tasta „săgeată sus" (CS + 7). Tipărit, nu are efect;

caracterul cu codul 12 „DELETE" este generat de tasta cu același nume (CS + 0). Tipărit, nu are efect;

caracterul cu codul 13 „ENTER" este generat de tasta CR. Tipărit, se comportă ca un apostrof: trece pe rând nou;

caracterul cu codul 14 „urmează număr" nu este generat de nimic. Tipărit face ca următoarele cinci caractere care sunt scrise să nu apară! În anexa F se deslușește utilitatea lui.

caracterele 16 - 21 au o utilizare înrudită, de aceea le tratăm laolaltă.

Ele se numesc „control INK", „control PAPER", „control FLASH", „control BRIGHT", „control INVERSE" și „control OVER". Tipărirea unuia face ca următorul caracter tipărit să fie considerat drept „parametru" al caracterului de control. Cele două caractere (cel de control și parametrul său) au ca efect cumulat același efect pe care îl au instrucțiunile de culoare inserate în `PRINT`-ul respectiv: schimbă culorile locale. Un exemplu este lămuritor:

```
PRINT CHR$ 17;CHR$ 2;"albastru"
```

care e totuna cu

```
PRINT PAPER 2;"albastru"
```

Utilitatea unor asemenea caractere este că pot face parte dintr-un șir (chiar din valoarea unei variabile șir) sau pot fi inserate în listingul programului prin felurite combinații de taste

CS + 4 inserează `CHR$ 20 + CHR$ 1`
CS + 3 inserează `CHR$ 20 + CHR$ 0`

În modul extins apăsarea tastelor numerice (cu și fără CS) introduce alte combinații de caractere de control cu parametrii lor în text. Încercați. În acest mod, puteți face listingul mai inteligibil, marcând felurite părți din el.

caracterul cu codul 22 „control AT", lucrează asemenea celor 6 caractere de mai sus, numai că interpretează următoarele 2 caractere drept parametri. Primul este *y*-ul, al doilea *x*-ul. Astfel:

```
PRINT CHR$ 22 + CHR$ 3 + CHR$ 5 + "text"
```

e echivalent cu

```
PRINT AT 3, 5; "text"
```

în fine, caracterul codat 23 sau „control TAB", lucrează asemănător lui „control AT". Ciudat este că folosește tot doi parametri, deși `TAB` are nevoie de unul singur. Al doilea este ignorat. De fapt,

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

este totuna cu

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

care este totuna cu

```
PRINT TAB a
```

pentru că argumentul lui `TAB` se ia *modulo* 32 (adică restul împărțirii sale la 32).

Dacă unul din caracterele 16-21 nu are ca „parametru" un caracter între limitele cerute de valorile posibile ale culorii, se obține eroarea K. `CHR$~22` poate da aceleași erori ca `PRINT AT` cu valori incorecte.

Astfel,

```
PRINT CHR$ 16 + CHR$ 16
```

eroare K, fiind echivalent cu

```
PRINT INK 16
```

După cum spuneam, aceste caractere își dovedesc utilitatea la folosirea lor în variabile șir. Iată niște exemple:

```
LET a$ = CHR$ 22 + CHR$ 5 + CHR$ 5 + "text fixat pe ecran": PRINT a$
```

sau:

```
 5 LET a$=""
10 FOR i = 1 TO 7
12  LET a$ = a$ + CHR$ 17 + CHR$ i + " ": REM 17 e codul "control PAPER"
13 NEXT i
15 LET a$ = a$ + a$ + a$ + a$ : REM 32 de spatii
20 OVER 1 : PRINT AT 0,0;"* EFECTE DE ROTATIE A CULORILOR."
25 FOR i=1 TO 100
30   PRINT AT 0,0 ; a$
35   LET a$ = a$(4 TO )+ a$( TO 3)
40 NEXT i
```

Tehnica de mai sus e simplă: am un șir de spații, fiecare în altă culoare. Mut mereu la coadă primul spațiu și caracterele ce-i definesc culoarea. Folosesc tipărirea de spații în `OVER 1`, care lasă ecranul neschimbat, dar schimbă culorile.

12.7 CODE

ENGLEZĂ: *code* = cod
SINTAXĂ: `CODE` *şir*
CATEGORIE: funcţie
DESCRIERE:

`CODE` este funcţia opusă lui `CHR$`. Întoarce codul primului caracter din şirul argument. `CODE ""` este 0.

Putem folosi `CODE` pentru a vedea dacă un caracter se află între anumite limite, după cum arată şi următorul

12.8 Exerciţiu rezolvat

Scrieţi o secvenţă de program care citeşte un număr întreg de maximum 4 cifre, fără a folosi `INPUT`.

Rezolvare

```
 1 OVER 0 : INK 9
 5 LET a$ = "" : REM sirul rezultat
10 PRINT AT 5,5; "dati numarul :";
15   LET b$ = INKEY$
17   IF (CODE b$ < CODE "0") OR (CODE b$ > CODE "9") THEN GOTO 30
18   REM daca s-a ajuns aici, s-a tastat o cifra
20   IF LEN a$ < 4 THEN LET a$ = a$ + b$ : PRINT b$; :
       FOR i=1 TO 5: NEXT i
25
30   IF b$ = CHR$ 13 THEN IF LEN a$ <> 0 THEN GOTO 50 : REM 13=CR
35   IF b$ = CHR$ 12 THEN IF LEN a$ > 0 THEN
       LET a$=a$( TO LEN a$ - 1): PRINT CHR$ 8;" ";CHR$ 8;
40 GOTO 15
50 LET a = VAL a$ : REM a este numarul
```

Comentarii

- Linia 17 testează dacă s-a apăsat vreo cifră (se bazează pe faptul că cele 10 cifre au codurile alăturate şi dispuse în aceeaşi ordine cu cifrele).
- Linia 20 adaugă la număr noua cifră, dacă acesta nu are deja 4 cifre.
- Linia 30 vede dacă nu s-a apăsat CR. Dacă da şi numărul are cel puţin o cifră, iese din bucla de citire formată de liniile 15 - 40.
- Linia 35 vede dacă nu s-a apăsat DELETE pentru a şterge ultimul caracter. Dacă s-a apăsat, atunci în caz că şirul are cel puţin un caracter, îl extrage pe ultimul. Îl şterge şi de pe ecran, mutând cursorul la stânga (`CHR$ 8`), scriind un spaţiu şi din nou stânga.
- Linia 50 extrage valoarea numărului, în variabila a.

12.9 Exerciţii

1. Scrieți un program care să miște aleator pe ecran un păianjen, care să „țească" o pânză în urma sa.
2. Scrieți un program pentru conversie grade $\leftrightarrow$ radiani ($\pi = 3.141592$).
3. Scrieți un program care mișcă pe ecran un desen, compus din maximum 5 segmente (găsiți o metodă pentru a-l ajuta pe utilizator să introducă desenul) cu ajutorul a patru taste.
4. Scrieți un program pentru făcut desene — un punct mișcat cu opt taste lasă urme.

Chapter 13

Subrutine şi trigonometrie

Vom învăța câte ceva despre funcțiile trigonometrice și despre proceduri (subrutine).

- `PI`, pentru numărul omonim;
- `GOSUB`, pentru a executa o subrutină;
- `RETURN`, pentru a încheia o subrutină;
- funcții trigonometrice:
  - `SIN`, pentru sinus;
  - `COS`, pentru cosinus;
  - `TAN`, pentru tangentă;
  - `ASN`, pentru arcsinus;
  - `ACS`, pentru arccosinus;
  - `ATN`, pentru arctangentă.

13.1 PI

ENGLEZĂ: vine din litera grecească ce poartă același nume — π
SINTAXĂ: `PI`
CATEGORIE: funcție
DESCRIERE:

`PI` este o funcție fără argumente; de fiecare dată când ea este invocată, întoarce valoarea aproximativă a numărului omonim celebru cu care matematicienii notează raportul dintre lungimea unui cerc și cea a diametrului său. În BASIC, `PI` se folosește frecvent, pentru că unghiurile sunt exprimate în radiani și nu în grade și după cum știm, la π radiani corespund 180 de grade. 1 grad = `PI` / 180 radiani.

```
DRAW 100,0,-PI
```

va face un semicerc cât mai aproape de ideal.

13.2 GOSUB

ENGLEZĂ: *GOto SUBroutine* = du-te la subrutină
SINTAXĂ: `GOSUB` *etch*
CATEGORIE: comandă
DESCRIERE:

etch este valoarea unei etichete, altfel survine eroare B. Vom descrie aici ideea care se găsește în spatele acestei instrucțiuni; modul ei de funcționare îl vom expune abia după prezentarea instrucțiunii următoare (`RETURN`), cu care se folosește întotdeauna împreună.

Foarte adesea, în programe se ivește necesitatea folosirii unei aceleiași secvențe de instrucțiuni. De pildă, un program care scrie, scrie, scrie pe ecran pagini întregi, va apela adesea la o secvență de linii precum următoarea:

```
50 PRINT AT 21,0;"Daca ati terminat, tastati ceva .."
60 IF INKEY$ = "" THEN GOTO 60
70 IF INKEY$ <> "" THEN GOTO 70
```

Această secvență așteaptă apăsarea unei taste și apoi eliberarea ei. Pentru a simplifica lucrurile, ar fi mult mai convenabil să putem scrie aceste linii o singură dată și, printr-un procedeu oarecare, să le invocăm de câte ori avem nevoie. O astfel de secvență de instrucțiuni se numește *subrutină* sau *procedură*. Cum funcționează o să vedem imediat.

13.3 RETURN

ENGLEZĂ: *return* = întoarcere
SINTAXĂ: `RETURN`
CATEGORIE: comandă
DESCRIERE:

`RETURN` este întotdeauna ultima instrucțiune a unei subrutine, indicând sfârșitul execuției ei. Și acum să vedem cum se construiește o subrutină și cum se invocă:

Secvența de linii cu utilizare frecventă se scrie o singură dată, de preferință spre sfârșitul programului. Ultima instrucțiune a secvenței este `RETURN`. Când se ivește necesitatea executării subrutinei, în textul programului se inserează instrucțiunea `GOSUB` *etch*, unde *etch* este prima etichetă din subrutină. Totul funcționează astfel:

Când se întâlnește instrucțiunea `GOSUB` *etch*, se memorează undeva (pe așa numita *stivă GOSUB*) locul unde a fost întâlnită instrucțiunea (linia și numărul instrucțiunii). Apoi se face un `GOTO` *etch*. Instrucțiunile subrutinei se execută ca o secvență normală de linii. La executarea instrucțiunii `RETURN` se citește de pe stiva `GOSUB` locul de unde s-a executat instrucțiunea `GOSUB` și se revine la instrucțiunea imediat următoare.

Această schemă este foarte flexibilă tocmai pentru că `RETURN` este un fel de `GOTO` automat: procedura revine la locul din care a fost chemată și nu întotdeauna în același loc.

Pentru că o subrutină poate chema la rândul ei o alta, întoarcerea din cea din urmă trebuie să se facă în prima, și din prima în program. Deci locurile de unde s-au chemat procedurile se extrag din stivă în ordine inversă chemării: ultima apelată — prima ieșită (în engleză *last in first out*, adică prescurtat *LIFO*). Acest mod de „păstrare" a informației este caracteristic unui obiect informatic care se numește *stivă*. Tot așa e și o stivă de cărți (așezate orizontal, una peste alta): poți lua numai cartea de deasupra, ultima pusă.

Să vedem niște exemple:

```
   5 PRINT "Chemam prima oara procedura 1"
  10 GOSUB 1000
  15 PRINT "Chemam a doua oara procedura 1"
  20 GOSUB 1000
  30 PRINT "Chemam procedura 2"
  40 GOSUB 2000
  45 PRINT "suntem in programul principal"
  50 STOP
 999
1000   PRINT "** suntem in procedura 1 **"
1010   PAUSE 100
1020 RETURN
1999
2000   PRINT "**** suntem in procedura 2 ****"
2010   PRINT "procedura 2 cheama procedura 1"
2020   GOSUB 1000
2025   PRINT "**** am revenit in 2 ****"
2030 RETURN
```

efectul:

```
Chemam prima oara procedura 1
** suntem in procedura 1 **
Chemam a doua oara procedura 1
** suntem in procedura 1 **
Chemam procedura 2
**** suntem in procedura 2 ****
procedura 2 cheama procedura 1
** suntem in procedura 1 **
**** am revenit in 2 ****
suntem in programul principal
```

Despre erori:

- se obține eroarea 7 dacă se întâlnește un `RETURN` și nici un `GOSUB` corespunzător nu a fost executat (nu mai e nici o înregistrare pe stivă).
- dacă stiva se umple (întreaga memorie în care s-ar putea extinde este folosită) puteți obține eroarea 4. Stiva GOSUB „crește" de la sfârșitul memoriei spre început, programul BASIC (și variabilele sale) de la început spre sfârșit. Deci programul și stiva au împreună o anumită cantitate de memorie pe care o folosesc. Dacă cele două se întâlnesc, atunci se obține eroarea 4.

O procedură se poate apela pe ea însăși. Atunci ea se numește *recursivă*. Prin felurite mecanisme de programare, trebuie să se asigure că această auto-apelare nu se va petrece la infinit. Iată un exemplu:

```
    5 LET apeluri = 10: GOSUB 1000
   10 STOP
 1000 PRINT apeluri;" ";
 1010 LET apeluri = apeluri - 1
 1020 IF apeluri <> 0 THEN GOSUB 1000
 1030 RETURN
```

cu efectul:

```
10 9 8 7 6 5 4 3 2 1
```

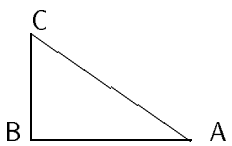
Încercați să înțelegeți cum merge procedura anterioară. Deși procedurile recursive pot părea la prima vedere o ciudățenie inutilă, ele sunt un obiect informatic foarte util. Din păcate, limbajul BASIC, datorită slăbiciunii cu care folosește acest concept (nu are „variabile locale" și nici „parametri" pentru proceduri) nu poate beneficia din plin de forța acestui mecanism.

Recomandăm atribuirea unui nume fiecărei proceduri și folosirea sa în locul etichetei de început. Aceasta face programul mai ușor de scris și de depanat (și după cum se deduce citind Anexa F, chiar mai scurt!). Exemplu:

```
    1 LET tasta = 1000
    ....
   50 GOSUB tasta
    ....
  999 STOP
 1000 REM procedura citeste tasta
    ....
 1100 RETURN
```

■

Următorul grup de instrucțiuni face parte din grupul funcțiilor *trigonometrice*. Aceste funcții au drept argumente măsuri de unghiuri și dau ca rezultate numere remarcabile, caracteristice unghiului argument. În BASIC unghiurile se dau întotdeauna în radiani (vezi Capitolul 12). Cele mai importante funcții trigonometrice sunt: sinus, cosinus, tangenta și cotangenta. Există mai multe moduri de defini sinusul (și celelalte funcții trigonometrice) unui unghi (toate echivalente); vom da aici una care ni se pare mai simplă. Să construim un triunghi dreptunghic ABC, cu unghiul B $= 90^o =$ PI/2 rad (rad e prescurtarea de la radiani). Prin definiție, avem:



sinus	$\sin A = BC/AC$	cateta opusă supra ipotenuză;
cosinus	$\cos A = AB/AC$	cateta alăturată supra ipotenuză;
tangentă	$\tan A = BC/AB$	cateta opusă supra cateta alăturată.

(Românii notează tangenta cu TG, englezii cu TAN.)

Cotangenta nu există în BASIC HC, așa că nu am definit-o. Ca să nu se zică despre noi că nu știm, adăugăm că ea este definită ca COT A = 1/TAN A.

Între aceste funcții avem tot felul de relații interesante, dar care deocamdată nu ne folosesc. Să observăm că funcțiile trigonometrice sunt definite și pentru unghiuri de peste 90 de grade, ceea ce un triunghi dreptunghic nu poate avea.

Nu intrăm în alte subtilități trigonometrice, le lăsăm pe seama unui curs de matematică (deși pe unele le vom folosi...).

13.4 SIN

ENGLEZĂ: *SINus* = sinus
SINTAXĂ: `SIN` *u*
CATEGORIE: funcție
DESCRIERE:

întoarce sinusul unghiului *u*, dat în radiani.

13.5 COS

ENGLEZĂ: *COSinus* = cosinus
SINTAXĂ: `COS` *u*
CATEGORIE: funcție
DESCRIERE:

întoarce cosinusul unghiului *u*, dat în radiani.

13.6 TAN

ENGLEZĂ: *TANgent* = tangentă
SINTAXĂ: `TAN` *u*
CATEGORIE: funcție
DESCRIERE:

Întoarce tangenta unghiului u, dat în radiani. TAN *u*=SIN *u*/COS *u*. Când COS *u*=0, tangenta tinde spre infinit. `TAN (PI/2)` înseamnă deci eroare 6.

13.7 ASN

ENGLEZĂ: *ArcSiNus* = arcsinus
SINTAXĂ: `ASN` *n*
CATEGORIE: funcție
DESCRIERE:

Întoarce *arcsinusul* numărului n, adică unghiul care are sinusul *n*. `ASN 1 = PI/2 = 1.5707963`. Pentru că toate unghiurile care diferă prin `2*PI` au același sinus, arcsinus trebuie să aleagă pe unul din ele. Îl alege întotdeauna pe cel între –PI/2 și PI/2.

Pentru că $-1 \leq \sin u \leq 1$ pentru orice unghi, deducem că *n* trebuie să fie între –1 și 1. Altfel survine eroarea A.

13.8 ACS

ENGLEZĂ: *ArcCoSinus* = arccosinus
SINTAXĂ: `ACS` *n*
CATEGORIE: funcție
DESCRIERE:

Întoarce *arccosinusul* numărului *n*, adică unghiul care are cosinusul *n*. `ACS 1 = 0`. Pentru că $-1 \leq \cos u \leq 1$ pentru orice unghi u, deducem că *n* trebuie să fie între –1 și 1. Altfel survine eroarea A.

13.9 ATN

ENGLEZĂ: *ArcTaNgent* = arctangenta
SINTAXĂ: `ATN` *n*
CATEGORIE: funcție
DESCRIERE:

Întoarce unghiul cu tangenta *n*.

13.10 Exerciţiu rezolvat

Scrieți un program care să funcționeze ca un ceas cu limbi.

Rezolvare

Folosim intens funcții trigonometrice. Ceasul nu este prea exact, dar arată bine. Mai încolo o să învățăm și cum să-l facem precis.

```
 1 REM ceas cu limbi
 2 PAPER 7 : INK 9 : CLS
 3 REM definim etichetele a 6 subrutine:
 4  LET secunda = 100 : LET minut = 200 : LET ora = 300
 5  LET unghi sec = 1000 : LET unghi min = 2000 : LET unghi ora = 3000
 6 LET sec = 0
 7 LET x centru = 117: LET y centru = 82 : REM centrul ceasului
 8 LET lg sec = 55 : LET lg min = 50 :
   LET lg ora = 40 : REM lungimi limbi
 9 LET alfa = 20 * PI / 180 :
   LET beta = 40 * PI / 180 : REM forma limbii
10 FOR i=1 TO 12 : REM cadran
11  PRINT AT 11 - 8 * COS (i*PI/6) ,
```

```
            14 + 8 * SIN (i*PI/6) - (i>9) ; i
 12 NEXT i
 13 CIRCLE x centru, y centru, 2
 20 INPUT "ORA curenta ";ora, "MINUTUL ";Min
 22 LET ora = INT ABS ora : LET ora = ora - 12 * INT (ora/12)
 23 LET min = INT ABS min : LET min = min - 60 * INT (min/60)
 30 GOSUB unghi sec : GOSUB unghi min : GOSUB unghi ora
 35  REM am calculat unghiurile limbilor la inceput
 45 INPUT "Apasati ENTER pentru a porni "; LINE a$
 49
 50   PRINT AT 0,0; ("0" AND ora < 10); ora; ":";
                    ("0" AND min < 10); min; ":";
                    ("0" AND sec < 10); sec
 60   GOSUB secunda : REM avansam o secunda
 65   IF sec <> 0 THEN PAUSE 25: GOTO 50
 67
 69   REM ** se schimba minutul
 70   GOSUB minut : REM avansam un minut
 75   IF min <> 0 THEN GOTO 50
 77
 79   REM ** se schimba ora (min = 0)
 80   GOSUB ora
 90 GOTO 50
 99
100 REM ****** subrutina care bate secunda
103  REM in us avem vechiul unghi al limbii secundelor!
110  INVERSE 1 : REM STERG limba la vechea pozitie!
120  PLOT xcentru, ycentru: DRAW lgsec * SIN us, lgsec * COS us
125  LET sec = sec + 1 : IF sec = 60 THEN sec = 0
130  GOSUB unghi sec : REM calculez noua pozitie
135  INVERSE 0 : REM desenez limba la noua pozitie
150  PLOT xcentru, ycentru: DRAW lgsec * SIN us, lgsec * COS us
160 RETURN
190
200 REM ******* subrutina care bate minutul
205  LET lungime = lg min : LET unghi = um : REM um = unghiul anterior
207  INVERSE 1 : REM sterg
210  GOSUB 5000 : REM desenez limba ciudata
220  LET min = min + 1 : IF min = 60 THEN min = 0
230  GOSUB unghi min : LET unghi = um
235  INVERSE 0 : REM desenez
240  GOSUB 5000
250 RETURN
290
300 REM ******* subrutina care bate ora
305  LET lungime = lg ora : LET unghi = uh : REM uh = unghiul anterior
307  INVERSE 1 : REM sterg
310  GOSUB 5000 : REM desenez limba ciudata
320  LET ora = ora + 1 : IF ora = 12 THEN ora = 0
330  GOSUB unghi ora : LET unghi = uh
335  INVERSE 0 : REM desenez
```

```
 340  GOSUB 5000
 350 RETURN
 999
1000 REM calculul unghiului secundei in radiani
1004 REM din variabila sec -> in variabila us
1005  LET us = sec * 360 / 60  : REM 360 grade la 60 sec
1010  LET us = us * PI / 180 : REM 180 grade la radian
1020 RETURN
1999
2000 REM calculul unghiului minutelor
2010 REM min -> um
2020  LET um = min * 360 / 60
2030  LET um = um * PI / 180
2040 RETURN
2999
3000 REM calculul unghiului orelor ora -> uh
3010  LET uh = ora * 360 / 12
3020  LET uh = uh * PI / 180
3030 RETURN
4999
5000 REM procedura care deseneaza o limba ciudata
5010 REM forma limbii este descrisa de alfa, beta si lungime
5020 REM orientarea este data de unghi
5030  PLOT x centru, y centru
5040  LET temp = lungime * ( 1 / TAN alfa + 1 / TAN beta)
5050  LET segm1 = temp / SIN alfa
5060  LET segm2 = temp / SIN beta
5100  DRAW segm1 * SIN (unghi - alfa), segm1 * COS (unghi - alfa)
5200  DRAW segm2 * SIN (unghi + beta), segm2 * COS (unghi + beta)
5300  DRAW segm2 * SIN (unghi + PI - beta), segm2 * COS (unghi + PI - beta)
5400  DRAW segm1 * SIN (unghi + PI + alfa), segm1 * COS (unghi + PI + alfa)
5500 RETURN
```

Pentru că ștergem limbile cu INVERSE 1 și pentru că adesea ele se suprapun, mutarea unora va lăsa niște urme neplăcute peste celelalte. Programul poate fi rescris cu puțină grijă ca să folosească OVER 1 pentru a desena. Reamintiți-vă că un obiect desenat de două ori în OVER 1 dispare cu totul. Din păcate nici acea variantă nu este cu totul scutită de deficiențe, pentru cîntersecțiile de limbi vor dispărea. O altă posibilitate ar fi ca la fiecare mutare a secundarului să redesenăm (fără a șterge) și celelalte limbi. Atunci totul ar merge frumos, însă nu foarte repede.

Înainte de a înțelege cum lucrează programul, trebuie să facem niște mici pregătiri cu iz de geometrie analitică.

Întâi: dacă se cunoaște un capăt al unui segment (x_1, y_1), lungimea și unghiul cu verticala (u), care este celălalt capăt (x_2, y_2)? Folosind definițiile funcțiilor trigonometrice, avem

$$\begin{aligned} x_2 &= x_1 + lungime \times \sin u \\ y_2 &= y_1 + lungime \times \cos u \end{aligned}$$

Aceasta este formula de bază a acestui program!

Apoi: ecuația parametrică a unui cerc. Un cerc este al doilea capăt al unui segment ce se rotește în jurul primului capăt fix. Lungimea segmentului e raza. „Se rotește" înseamnă că unghiul lui cu verticala variază între 0 și $2 \times \pi$. Cercul de centru (x_0, y_0) și rază r este dat de

$$\begin{aligned} x &= x_0 + r \times \sin t \\ y &= y_0 + r \times \cos t \\ t &\ \epsilon \ [0 \ldots 2\pi) \end{aligned}$$

Încercați:

```
2000 FOR t = 0 TO 2*PI STEP .05
2010       PLOT 128 + 50*COS t, 87 + 50*SIN t
2020 NEXT t
```

Comparați cu `CIRCLE 128,87,50`.

Doar de amuzament, să observăm că pentru a obține o elipsă ajunge să luăm una din raze diferită de cealaltă, de pildă:

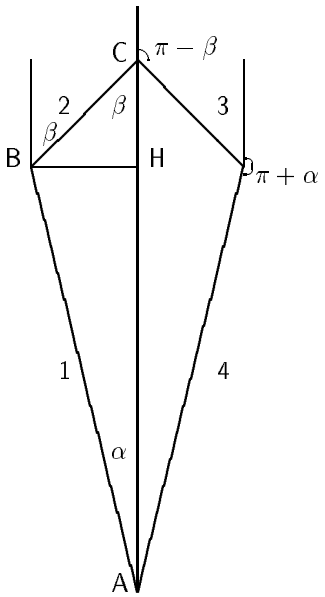
```
2010 PLOT 128 + 80*SIN t, 87 + 30*COS t
```

Folosind această ecuație parametrică am trasat cadranul ceasului (liniile 10-12). Aceasta este și formula care mișcă limba secundarului: când știu unghiul ei cu verticala (variabila us) pot să o trasez pentru că îi știu un capăt și lungimea. Desenarea se face în subrutina de la linia 100 (`secunda`). Trasarea se face de două ori, pentru că întâi șterg limba (liniile 110-120), apoi calculez noul unghi (125-130) și o desenez în noua poziție (135-150).

Aceeași metodă, din trei pași (șterg, calculez, desenez) este folosită și pentru celelalte două limbi. Acestea sunt trasate numai când se mută de la un loc la altul. (Așa cum arată, programul mută limba orelor numai din oră în oră. Ce ar trebui făcut ca să o mute mai des?) Diferența este că limbile orelor și minutelor au o formă mult mai complicată, dar asemănătoare. De aceea am scris subrutina de la linia 5000, care primește în variabilele `lungime` și `unghi` descrierea limbii de trasat și o desenează. Observați ca subrutina este de asemenea chemată de câte două ori, pentru a șterge și desena, de fiecare din procedurile pentru minute și secunde.

Procedurile pentru calculul unghiurilor nu sunt prea complicate. Ele se bazează pe regula de trei simplă pentru a calcula întâi câte grade , și apoi câți radiani corespund unui număr dat de secunde, respectiv minute și ore. Formula face dealtfel și obiectul unui exercițiu, printr-un alt capitol. Variabilele `us`, `um` și `uh` sunt folosite pentru a memora valorile unghiurilor secundarului, minutarului, respectiv orarului. Unghiul se exprimă în radiani și se calculează relativ la axa verticală.

Cel mai interesant este însă modul de trasare al unei limbi, realizat de procedura începând la linia 5000. Pentru a înțelege mai bine, trebuie să facem un desen. Pentru simplificare vom considera limba în poziție verticală.



Limbile sunt trasate de procedura de la linia 5000 în ordinea de pe desen (1 – linia 5100, 2 – 5200, 3 – 5300, 4 – 5400). Limbile 1 și 4 au lungimile (în pixeli) reprezentate de variabilela `segm1`. Limbile 2 și 3 au lungimea `segm2`. Vom vedea imediat cum am ajuns la formulele pentru ele. Să vedem întâi cum se trasează.

Presupunem întâi că unghiul lor cu axa verticală este 0, pentru simplificare (putem folosi atunci desenul de mai sus). Segmentul numărul (1) are lungimea `segm1` și unghiul față de verticală $-\alpha$ (minus pentru că se află la stânga axei verticale). Dar stim să trasăm un astfel de segment!

Segmentul (2) are un unghi de β cu axa verticală, de data asta luat cu plus. Pentru că `DRAW` lucrează în coordonate relative, el este foarte ușor de trasat. Capătul lui pornește de unde s-a oprit segmentul (1). Îi cunoaștem lungimea și unghiul. Totul este deci foarte simplu.

Segmentul (3) are unghiul față de linia verticală care merge în sus $180^o - \beta$ (după cum se vede și pe desen) sau $\pi - \beta$ în radiani. Lungimea o știm, deci...

În fine, segmentul (4) are un unghi de $\pi + \alpha$ în radiani față de verticală.

Dacă limba este rotită cu un unghi oarecare `unghi` față de verticală, la toate unghiurile descrise mai sus trebuie să adăugăm chiar această valoare. De aceea în program apare peste tot `COS(... + unghi)`.

A mai rămas de văzut cum am calculat lungimile `segm1` și `segm2`. Pentru aceasta am folosit ca valori: α, β și lungimea până la vârf a unei limbi, `lungime`. În triunghiul ABC luăm înălțimea BH. Cu notațiile din program avem relațiile:

$$\begin{aligned} AC &= \texttt{lungime} \\ \angle BAC &= \alpha = \texttt{alfa} \\ \angle BCA &= \beta = \texttt{beta} \\ BH &= \texttt{temp} \end{aligned}$$

Am scris pentru `temp` relațiile în triunghiurile dreptunghice ABH:

$$\texttt{temp}/AH = \tan\alpha$$

și BCH:

$$\texttt{temp}/CH = \tan\beta$$

Am observat apoi că $AH + CH =$ `lungime`, sau

$$\texttt{temp}/\tan\alpha + \texttt{temp}/\tan\beta = \texttt{lungime}$$

Din această relație rezultă valoarea lui `temp`, calculată în linia `5040`:

$$\texttt{temp} = \texttt{lungime} \times (1/\tan\alpha + 1/\tan\beta)$$

Mai departe, valorile `segm1` = AB și `segm2` = BC rezultă imediat din sinusurile unghiurilor `alfa` și respectiv `beta`, după cum se vede și în liniile `5050-5060`.

Modificând `alfa` și `beta` puteți obține forme noi pentru limbi.

Programul poate fi grăbit puțin scoțând din buclă calculul tangentelor, care de fapt nu se schimbă niciodată. Optimizați-l!

13.11 Exerciţii

1. Scrieți un program care „umple" un contur închis cu puncte de aceeași culoare (*fill* pe engleză).
2. Desenați un glob pământesc, presupus sferic.
3. Luați un segment. Mișcați capetele sale independent pe ecran, ca pe niște bile de biliard. Trasați-l.

Chapter 14

Manipulări de date şi sunet

Printre altele, o să învățăm să cântăm.

- `READ` pentru a inițializa variabile;
- `DATA` pentru liste de valori;
- `RESTORE` pentru alegerea listelor;
- relații de ordine pentru șiruri de caractere;
- `BEEP` pentru cântat;
- `ATTR` pentru a inspecta culorile ecranului.

14.1 READ

ENGLEZĂ: *to read* = a citi
SINTAXĂ: `READ` $var_1, \ldots, var_n$
CATEGORIE: comandă
DESCRIERE:

După `READ` se găsește întotdeauna o listă de variabile, separate prin virgulă. Variabilele pot fi de tip număr sau șir. Instrucțiunea `READ` menține un indicator ce arată care a fost ultima expresie citită din lista `DATA` (explicația vine un pic mai jos). `READ` citește valori din lista de după instrucțiunea `DATA` și le atribuie în ordine variabilelor din lista sa.

14.2 DATA

ENGLEZĂ: *data* = date
SINTAXĂ: `DATA` $expr_1, \ldots, expr_m$
CATEGORIE: comandă
DESCRIERE:

`DATA` este o instrucțiune pasivă, care nu face nimic. Ea se poate afla oriunde în program (doar după un `REM` nu), atât înaintea lui `READ` cât și după. Scopul ei este de a menține o listă de expresii, care vor fi citite eventual de `READ`, evaluate și atribuite variabilelor din lista `READ`. Expresiile pot avea ca rezultat orice tip.

14.3 RESTORE

ENGLEZĂ: *to restore* = a pune la loc
SINTAXĂ: `RESTORE [`*etch*`]`
CATEGORIE: comandă
DESCRIERE:

Dacă *etch* lipsește este considerat 0. *etch* este o etichetă și, ca atare, trebuie să fie între 0 și 9999 pentru a nu obține eroarea B. Tot ceea ce face `RESTORE` este să mute indicatorul care îi arată lui `READ` care a fost ultima expresie citită, la linia cu eticheta specificată.

Înainte de a da și alte detalii în legătură cu aceste instrucțiuni, să vedem la ce ar putea fi utile. Ele se folosesc cu mult succes în două cazuri: când sunt de făcut atribuiri masive sau în decizii multiple. De exemplu: un program care în funcție de un număr introdus de utilizator — să zicem între 1 și 25 — să atribuie variabilei a$ o valoare dintr-o listă de nume. Fără `DATA` am putea face cam așa:

```
 5 INPUT "numarul ";nr
10 GOSUB nr+19
15 GOTO 100
20 LET a$="Dan" : RETURN
21 LET a$="Ion" : RETURN
....
```

sau, mai puțin economic:

```
 5 DIM b$(25, 10)
10 LET b$(1) = "Dan"
11 LET b$(2) = "Ion"
...
50 INPUT nr : LET a$=b$(nr)
```

Cu `READ - DATA - RESTORE` putem rezolva mai elegant:

```
 5 INPUT nr
10 RESTORE nr+19 : READ a$
20 DATA "Dan"
21 DATA "Ion"
....
```

sau, dacă preferați:

```
 5 INPUT nr
10 RESTORE : FOR i=1 TO nr: READ a$: NEXT i
20 DATA "Dan", "Ion", .....
```

După cum vedeți, listele DATA sunt parcurse în ordinea apariției (în caz că nu se întâlnește RESTORE). Când una din liste se termină, se folosește cea care îi succede în program.

Dacă se încearcă citirea unei valori cu READ și toate listele DATA au fost epuizate, survine eroarea E.

Dacă în lista DATA urmează un șir și se încearcă citirea unui număr sau invers, survine eroarea C.

14.4 Relaţii de ordine pentru şiruri

Știm deja că putem compara două șiruri cu = sau cu <>. Am anticipat că putem să o facem și cu <, <=, >=, >. Un șir este mai mic decât altul dacă îl precede într-o ordonare *lexicografică* sau, mai simplu spus, dacă într-un dicționar se află scris înaintea lui. Oricine a căutat măcar odată într-un dicționar știe că, pentru a căuta, te bazezi pe ordinea alfabetică. Pe post de ordine alfabetică în BASIC stă ordinea caracterelor după codul lor. Și acum: șirul a este mai mic decât șirul b dacă primele lor n (poate $n = 0$) caractere coincid, iar caracterul $n + 1$ în șirul a este „mai mic" (adică are codul mai mic) decât caracterul $n + 1$ în șirul b.

Exemple:

- `"a" < "b"`
- `"A" < "a"` căci `CODE "A"=65, CODE "a"=97`
- `CHR$ 6 < "+"`
- `"AA" > "A"`
- `"aa" > "aA"`
- `"aa" > "A"`
- `"casa" < "masa"`
- `"sus" > " sus"`
- `" TO " > " TO "` unde primul șir este `CHR$ 204`, iar al doilea e format din patru semne.
- `"Zoologie" < "animal"` datorită codului majusculelor!

Un exemplu: dacă `LEN a$ = 1` atunci

```
IF CODE (a$ >= CODE "0") AND (CODE a$ <= CODE "9") THEN
```

se scrie mai scurt

```
 IF (a$ >= "0") AND (a$ <= "9") THEN
```

14.5 BEEP

ENGLEZĂ: *beep* = bip! (onomatopee)
SINTAXĂ: `BEEP` *timp*, *nota*
CATEGORIE: comandă
DESCRIERE:

`BEEP` generează prin difuzorul calculatorului un sunet descris de cele două numere *timp* și *nota*.

timp este durata sunetului, în secunde. Limitele admisibile sunt 0 (sunet nul, fără durată) și 10. Poate fi un număr fracționar.

nota descrie înălțimea sunetului. În mod normal este un întreg, dar nimic nu-l împiedică să fie fracționar. Limitele sunt –60 și 69.8. Valorile întregi corespund notelor dodecafonice din gama muzicală astfel:

Valoare	Notă	Valoare	Notă
–12	do de jos	1	do#
–11	do#	2	re
–10	re	3	re#
–9	re#	4	mi
–8	mi	5	fa
–7	fa	6	fa#
–6	fa#	7	sol
–5	sol	8	sol#
–4	sol#	9	la
–3	la	10	la#
–2	la#	11	si
–1	si	12	do de sus
0	do central		

■

Iată un program care interpretează un preludiu de J.S. Bach. Am folosit o metodă interesantă de codificare a notelor: în loc de numere, caractere. Pentru „do central" este convenabil caracterul `"I"` (i mare). Re devine `"K"`, mi devine `"M"`, etc.

(`nota = CODE c$ - CODE "I" = CODE c$ - 73`, unde `c$` este codul caracter al notei).

Atenție să nu greșiți când introduceți lista `DATA`, pentru că veți obține disonanțe!

```
9000 REM   subrutina muzicala
9010 REM     BACH Preludio
9015 REM Clavecinul bine temperat,
9016 REM   vol 1, preludiul 1,
9017 REM    BWV 846, do major
9020 LET b$ = "" : RESTORE 9100
9030 FOR i=1 TO 33 : REM 33 de masuri
9040    READ a$ : LET b$=b$+a$+a$ : REM fiecare fraza se repeta
9043 NEXT i
9045 READ a$ : LET b$=b$ + a$ : REM ultima masura
9046
```

```
9050 FOR j=1 TO LEN b$
9060   BEEP .12 + .01*(j > 520) + .01*(J > 525), CODE b$(j) - 73
9070 NEXT j
9080 BEEP .5, 0 : REM ultima nota prelungita
9099
9100 DATA "IMPUYPUY","IKRWZRWZ"
9110 DATA "HKPWZPWZ","IMPUYPUY"
9120 DATA "IMRY^RY^","IKORWORW"
9130 DATA "HKP\WP\W","HIMPUMPU"
9140 DATA "FIMPUMPU","?FKOUKOU"
9150 DATA "DHKPTKPT","DGMPVMPV"
9160 DATA "BFKRWKRW","BEKNTKNT"
9170 DATA "ADIPUIPU","ABFINFIN"
9180 DATA "?BFINFIN","8?DHNDHN"
9190 DATA "=ADIMDIM",";DGIMGIM"
9200 DATA "5BFIMFIM","7=FILFIL"
9210 DATA "9BHIKHIK","8BDHKDHK"
9220 DATA "8ADIMDIM","8<DINDIN"
9230 DATA "8<DHNDHN","8?DINDIN"
9240 DATA "8@FIOFIO","8ADIPDIP"
9250 DATA "8?DINDIN","8?DHNDHN"
9260 DATA "1=DGMDGM"
9270 DATA "1=BFINIFIFBFB?B1<PTWZWTWTPTKNMK"
```

În linia 9060 se produce o rărire a tempo-ului, spre final.

Firește, puteam scrie conținutul lui b$ dintr-o dată. Iată două motive pentru care n-am făcut-o: în primul rând pentru că este format din șiruri care se repetă, deci am fi scris mai mult. (De fapt tot mai poate fi scurtat, căci primele 33 de măsuri repetă notele 3, 4, 5 în notele 6, 7, 8.) În al doilea rând, să fi scris un șir atât de lung dintr-o dată ar fi făcut o eventuală greșeală greu de corectat.

Atunci se pune întrebarea: de ce să nu citim a$ și să-l cântăm direct? De ce-l mai adunăm la b$? Răspunsul e simplu: s-ar simți după fiecare 16 note pauza necesară instrucțiunii `READ`, ce-i drept scurtă, dar supărătoare. Așa cum stau lucrurile, execuția este uniformă.

Adăugând linia

```
9055 PLOT j/2.2 , (CODE b$(j) - 53) * 4
```

veți obține un fel de „grafic" al melodiei!

14.6 ATTR

ENGLEZĂ: *ATTRibutes* = atribute
SINTAXĂ: `ATTR` (*y*,*x*)
CATEGORIE: funcție
DESCRIERE:

y, *x* sunt linia, respectiv coloana unui pătrățel din grila de joasă rezoluție. *x* este cuprins între 0 și 31, iar y între 0 și 23, altfel executarea se soldează cu eroare B. `ATTR` (*y*, *x*) întoarce un număr

întreg între 0 și 255 reprezintând *atributele* coloristice ale pătrățelului respectiv. Numărul acesta codifică într-un mod ingenios, cele patru informații de culoare care caracterizează pătrățelul respectiv: `INK`, `PAPER`, `BRIGHT` și `FLASH`. (`OVER` și `INVERSE` nu sunt niște caracteristici ale pătrățelelor, ci doar niște moduri de trasate a punctelor și caracterelor.) Modul de codificare este astfel ales încât orice combinație să fie reprezentată unic. (**Atenţie**: valorile 8 și 9 pentru una din comenzile de tipărire înseamnă de fapt tot un mod de tipărire, nu un atribut al pătrățelului. Meditați la acest lucru!)

Valoarea atributelor se calculează astfel:

```
INK + 8*PAPER + 64*BRIGHT + 128*FLASH
```

Exemplu:

```
1 PRINT AT 2,1;INK 6;PAPER 2;BRIGHT 0;FLASH 1;" "
2 PRINT ATTR (2,1)
```

va tipări întâi un pătrățel clipitor care alternează roșu cu galben și apoi numărul 150 ($= 6 + 8 \times 2 + 64 \times 0 + 128 \times 1$).

Se pune problema inversă: de a determina culorile știind atributele. Dacă `n` este valoarea atributelor, atunci avem:

Atribut	Valoare
`INK`	`n - 8*INT (n/8)`
`PAPER`	`INT (n/8) - 8*INT (n/64)`
`BRIGHT`	`INT (n/64) - 2*INT (n/128)`
`FLASH`	`INT (n/128)`

De ce acest mod de codificare și de unde provin bizarele formule, explicăm imediat. Dacă lucrăm în baza 2 (vezi și Anexa E) cerneala și hârtia au 8 valori posibile, deci se pot exprima cu câte 3 cifre binare (biți). `BRIGHT` și `FLASH` pot avea doar două valori, deci un bit ajunge. În total 3+3+1+1=8 biți pentru un pătrățel 8×8. Acești 8 biți formează un număr în baza 2 astfel:

Biţii	Atributul
0,1,2	INK
3,4,5	PAPER
6	BRIGHT
7	FLASH

(Am numerotat biții de la 0 la 7, cu bitul 0 fiind cel mai puțin semnificativ = LSB – *less significant bit*.)

Se remarcă faptul că `ATTR` poate testa și cele două rânduri din parte de jos a ecranului (ele au de obicei `PAPER`-ul în culoarea `BORDER`-ului, care se poate astfel determina).

14.7 Exerciţiu rezolvat

Scrieți un program care să traseze graficul oricărei funcții.

Rezolvare

Comentariile le puteți găsi la Rezolvările exercițiilor în Anexa G, pentru exercițiul 1 de la acest capitol. Instrucțiunea cheie a programului este `VAL q$` din linia 210, care calculează valoarea expresiei introduse de utilizator ca un șir de caractere.

```
  1 REM Graphix
 10 INPUT "Introduceti f(x)=";LINE q$
 15  IF q$="" THEN STOP
 20 INPUT "Domeniu de reprezentare [" ; ls ; "," ; ld ;"]"
 22  IF ls >= ld THEN GOTO 20
 25 PRINT AT 0,0;"f: [" ; ls ; "," ; ld; "] -> R" ' "f(x)=" ; q$
 30 LET rap = 255 / (ld - ls) : REM raport pixeli / interval
 40 LET orig = -ls * rap  : REM coordonata pe ecran a originii
 50 BEEP .5,0: BEEP .5,4 : BEEP .8,0 : CLS
 60     REM *** trasam axele ***
 70 PLOT 0,87 : DRAW 255,0 : DRAW -4,2 : DRAW 0, -4 : DRAW 4,2
 80 PRINT AT 10,0;ls; AT 10,25;ld
 90 IF ls > 0 OR ld < 0 THEN GOTO 120
 95 REM axa verticala e pe ecran
100 PLOT orig, 0: DRAW 0,175
101  IF orig >= 2 AND orig <= 253 THEN DRAW 2, -4: DRAW -4,0 : DRAW 2,4
102  IF orig < 2 THEN DRAW 2,-4 : DRAW -2,0
103  IF orig > 253 THEN DRAW -2,-4 : DRAW 2,0
110
120 INPUT "La cite puncte trasam ? ";fin : LET fin = ABS fin
125  IF fin > 10 OR fin < 1e-1 THEN GOTO 120
130 INPUT "Raportul unitatii de lungime pe axe ";cv
135  IF cv < 1e-5 OR cv > 1e4 THEN GOTO 130
140 LET ymax = 87/rap/cv : LET ymax = INT (ymax * 100) / 100
150 PRINT AT 0,0; ymax; AT 21,0; -ymax
199     REM *** trasam graficul ***
200 FOR x=ls TO ld STEP fin/rap
205     LET xr = x * rap + orig : REM coordonata reala, pe ecran
210     LET y = VAL q$ * rap * cv + 87
220      IF y < 0 OR y > 175 THEN GOTO 240
230     PLOT xr,y
240 NEXT x
250 PAUSE 0: GOTO 10
```

14.8 Exerciţii

1. Adaptați programul de mai sus ca să ceară și intervalul pe axa Oy.
2. Scrieți un program care să scrie cu litere în relief (construite ca din cubulețe transparente).

Chapter 15

Memoria

Vom discuta despre organizarea memoriei și vom învăța câteva instrucțiuni BASIC care ne permit să o explorăm și modificăm. Este un capitol uriaș, dar pentru o bună manipulare a BASIC-ului nu este necesară cunoașterea tuturor detaliilor.

- structura unui calculator;
- limbaje mașină;
- interpretoare și compilatoare;
- organizarea memoriei (ROM, RAM);
- `POKE`, pentru a modifica conținutul memoriei;
- `PEEK`, pentru a inspecta conținutul memoriei;
  - memoria video;
  - variabilele sistemului BASIC;
- `USR`, care invocă programe în cod mașină și calculează adrese.

Prin anii '40 apăreau primele calculatoare electronice. (Pe vremea aceea erau în mare măsură electromecanice.) Nimeni nu citise undeva o rețetă: „un calculator trebuie să arate așa și așa". Un tip foarte inteligent, pe nume *John von Neumann*, a studiat în mod deosebit această problemă. El a avut numeroase idei, extrem de importante pentru evoluția acestei discipline. Printre altele, a sugerat abordarea construcției calculatoarelor prin împărțirea în probleme mai mici și mai simple. Sugestia cea mai cunoscută pe care a făcut-o este de a construi calculatoarele din trei părți distincte, care interacționează. Aceste părți sunt:

- unitatea centrală (UC);
- memoria;

- dispozitivele periferice.

Această *arhitectură* pe care o respectă majoritatea covârșitoare a calculatoarelor moderne se numește *arhitectura Von Neumann*. Ceasul electronic, calculatorul de buzunar, HC-ul și o grămadă de alte calculatoare, programabile sau nu, sunt bazate pe această arhitectură. Să vedem la ce e bună fiecare parte a sa.

Unitatea centrală (UC) este singura care face efectiv operațiuni. Este „creierul" calculatorului. Unitatea centrală „înțelege" un limbaj foarte rudimentar care se numește *limbaj mașină* sau *cod mașină*.

Memoria este o parte pasivă (care nu poate schimba nimic din informații) a calculatorului, folosită de UC pentru păstrarea datelor, programelor, rezultatelor.

Dispozitivele periferice permit UC să comunice cu exteriorul. Dispozitivele de *intrare* îi permit să achiziționeze informații din afară, pentru a percepe starea lumii exterioare calculatorului, iar dispozitivele de *ieșire* îi permit să comunice exteriorului rezultatele operațiunilor sale.

La HC:

- Unitatea centrală este *microprocesorul* Z80. („Micro" pentru că e mic, „procesor" pentru că procesează.)
- Despre memorie vom discuta pe larg mai târziu.
- Periferice
  - de intrare: tastatura;
  - de ieșire: ecranul;

  Perifericele pot fi extinse cu, casetofon, imprimantă, disc etc.

■

După cum am spus, UC pricepe doar limbajul mașină. Acesta este o succesiune de numere care reprezintă pentru el, codificate, instrucțiunile pe care le execută. Primele calculatoare se programau direct în cod-mașină, dar era destul de greu. Oamenii s-au gândit atunci să scrie niște programe care să-i permită calculatorului să fie programat într-o formă ceva mai prietenească; misiunea acestor programe fiind de a face accesibil un limbaje mai expresiv UC (vom vedea imediat cum). Astfel a apărut *limbajul de asamblare*, care rezultă dând fiecărei instrucțiuni cod-mașină câte un nume care să sugereze ceva. De pildă, în loc de 1100 1001 (în baza 2) se scrie `RET`, ceea ce înseamnă la nivel de cod mașină (pentru UC-ul HC-ului) un fel de `RETURN`.

Corespondența pentru UC Z80 între codul mașină și *mnemonica* din asamblare corespunzătoare (așa se mai numesc instrucțiunile din limbajul de asamblare) este dată în Anexa B, în coloanele 4, 5, 6. Pentru mai multe detalii recomandăm învățarea limbajului de asamblare Z80.

Exista deci un program numit *asamblor*, care lua textul scris în asamblare și îl traducea în cod mașină, pentru a putea fi executat de UC. Bine-înțeles, asamblorul era un program cod-mașină, executat de CPU însuși. Situația se prezintă cam așa:

Programatorul scrie un text de program în limbaj de asamblare, care este furnizat ca date de intrare asamblorului. Asamblorul generează un șir de numere, care este codul mașină corespunzător.

Programarea în limbaj de asamblare nu este nici ea prea comodă, pentru că instrucțiunile sunt prea simple. Pentru a scrie un program mai de doamne-ajută, înșiri mii de linii. Acesta nu este singurul dezavantaj, dar ajunge pentru a stimula informaticienii să inventeze limbaje mai complexe, cu instrucțiuni a căror semnificație este mai apropiată de necesități. Este mult mai normal să ai o instrucțiune care scrie 10 litere pe ecran decât un program de 200 de linii care face același lucru. Plus că limbajele evoluate introduc și alte concepte, cum ar fi variabile, proceduri ș.a.m.d. Pentru a putea excuta programe într-un limbaj evoluat (sau *limbaj de nivel înalt*) — BASIC-ul însuși este un astfel de limbaj, deși spre limita de jos — trebuie să existe în memorie un alt program, care să permită acest lucru, așa cum asamblorul traducea programul din limbajul de asamblare în cod (prescurtare pentru cod-mașină). Există două metode pentru asta:

- Prin traducerea fiecărei instrucțiuni din programul în limbajul *sursă* (limbajul evoluat) într-o secvența de instrucțiuni în cod mașină. (Lucrurile sunt puțin mai complicate, dar esența fenomenului asta e.) Procedeul se numește *compilare*. Asamblarea este un exemplu de compilare. *Compilatorul* este un program care efectuează această traducere. Ceea ce rezultă în urma traducerii este un program pe care îl putem executa.
- Al doilea procedeu constă în folosirea unui program (cod mașină) care să inspecteze textul programului scris în limbajul de nivel înalt și, în funcție de instrucțiunea întâlnită, să invoce felurite proceduri, care fac toată treaba. Acest tip de executare se numește *interpretare*, iar programul care „inspectează" instrucțiunile și execută în funcție de aceasta propriile sale proceduri se numește *interpretor*. BASIC-ul este un limbaj interpretat (există și variante de BASIC compilate, dar sunt niște hibrizi).

Un fel de interpretor am scris și noi: programul din capitolul precedent care interpretează (nu degeaba se folosește același cuvânt!) un preludiu de Bach. De ce? Pentru că privește textul unui „program muzical" (variabila `b$`) și, în funcție de conținutul ei, invocă o anumită instrucțiune `BEEP`. Este un interpretor sărăcuț, dar face întocmai ceea ce am spus.

Cum ar lucra un „compilator" muzical? Păi ar trebui să genereze un nou program care ar fi o succesiune de `BEEP`-uri, fiecare cu valoarea proprie, adică ceva de genul:

```
5 BEEP .12,0 : BEEP .12,4 : BEEP .12,4 : BEEP .12,12 : BEEP .12,16
 : BEEP .12,4 : BEEP .12,12 : BEEP .12,16
```

și așa mai departe. (Ar fi un compilator care traduce din limbajul muzical — în care un program este un șir de litere — în BASIC.)

■

Să privim mai îndeaproape organizarea memoriei HC-ului. Pentru început, observăm că memoria poate fi împărțită în două bucăți mari, care au proprietăți diferite.

Prima bucată e un *ROM* (*Read Only Memory* = memorie numai pentru citire), adică o memorie al cărei conținut nu se schimbă niciodată. Aici este scris din fabrică programul care interpretează BASIC-ul.

A doua bucată e un *RAM* (*Random Access Memory* = memorie cu acces aleator). Numele de RAM nu e prea fericit ales, dar este deja încetățenit. Conținutul acestei memorii poate fi schimbat de UC. De asemenea, la întreruperea alimentării cu curent electric, informația de

aici se pierde. Cum este ea folosită de interpretorul BASIC (și nu numai), constituie principalul subiect al acestui capitol.

Înainte de a trece mai departe, ar trebui să vedem ce este de fapt *memoria* și cum putem măsura o cantitate de memorie. O imagine destul de potrivită este a unui dulap cu o sumedenie de sertărașe. În fiecare din aceste sertărașe se află un număr. UC poate privi în oricare din ele și, pentru sertărașele RAM, poate schimba numărul dinăuntru. Numerele din sertărașe sunt scrise în baza 2 (vezi Anexa E). Pentru fiecare sunt disponibili 8 biți. Asta înseamnă că numărul de dinăuntru are o valoare între 0 și 255. Sertarele se numesc *locații de memorie*. (Comparația cu un sertar nu ține prea bine, pentru că o locație de memorie conține întotdeauna un număr, pe când un sertar poate să fie gol.) O cantitate de 8 biți se mai numește și *octet* sau în engleză *byte*.

Sertarele sunt așezate într-o ordine fixă (imaginați-vă un dulap lung de tot) și fiecare poate fi caracterizat de numărul său de ordine. Acest număr de ordine se numește *adresa* locației respective de memorie. Prima locație are adresa 0, celelalte 1, 2 și tot așa până la 65535 (în baza 10). Numerele din intervalul 0–65535 se bucură de proprietatea că pot fi exprimate folosind numai 16 biți, adică 2 octeți. Asta pentru că $2^{16} = 65536$.

În informatică se folosește valoarea de *kilooctet* pentru a măsura cantitatea de informație dintr-o memorie. Dar, atenție, un kilooctet nu are 1000 de octeți, ci 1024, adică 2^{10}. Asta pentru a putea exprima adresele unei memorii de un kilooctet pe exact 10 biți. Folosind această unitate putem spune că HC are $65536/1024 = 64$ kiloocteți, sau pe scurt 64K.

Ca să putem să pricepem mai ușor cum e organizată memoria vom învăța întâi care sunt instrucțiunile care operează direct la nivel de locație.

15.1 POKE

ENGLEZĂ: *to poke* = a înfige
SINTAXĂ: `POKE` *adresa*, *byte*
CATEGORIE: comandă
DESCRIERE:

adresa este un număr între 0 și 65535, *byte* unul între –256 și 255. Dacă *byte* este negativ, i se adaugă 256. Cele două numere se rotunjesc. În cazul că nu respectă limitele indicate survine eroarea B.

`POKE` scrie în octetul cu adresa specificată numărul indicat. Scrierea în ROM nu are efect.

15.2 PEEK

ENGLEZĂ: *to peek* = a privi pe furiș
SINTAXĂ: `PEEK` *adresa*
CATEGORIE: funcție
DESCRIERE:

adresa trebuie să fie între 0 și 65535 pentru a nu obține eroarea B.

`PEEK` este funcția opusă lui `POKE`. Ea întoarce numărul care este conținutul locației cu adresa specificată.

15.3 Harta memoriei

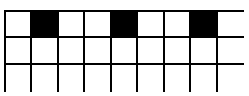
O schemă grafică a organizării memoriei se găsește în Anexa C.

Primii 16K sunt ocupați de ROM, având adresele între 0 și 16383. Pentru a descoperi cum lucrează BASIC-ul trebuie doar să descifrați programele (cod-mașină) care sunt scrise aici. Pentru acest scop recomand cu căldură excelenta carte „The Complete Spectrum ROM Disassembly" citată și în introducere.

Restul memoriei este RAM.

De la 16384 la 22527 se află o zonă numită *memoria video*. Aici calculatorul menține informații despre fiecare punct de pe ecran, dacă un pixel este „aprins" (*on*) sau „stins" (*off*). Fiecare pixel corespunde unui bit, care este 0 pentru *off* și 1 pentru *on*. Valoarea 0 înseamnă că punctul va avea culoarea `PAPER`. Folosindu-se de conținutul acestei memorii, anumite circuite electronice formează imaginea pentru televizor, pe care o vedeți pe ecran. Pentru că o locație are 8 biți, punctele sunt memorate câte 8 într-un cuvânt de memorie (folosim și termenul *cuvânt* pentru conținutul unei locații). Ordinea în care sunt așezate în memorie locațiile corespunzătoare punctelor consecutive nu este cea naturală și merită o privire mai apropiată.

Primele opt puncte de pe ecran (din colțul stânga-sus) sunt memorate în primul octet al memoriei ecran (sau video) — 16384. Primului punct îi corespunde bitul cel mai semnificativ, celui de-al doilea următorul etc:



← primele opt puncte de pe ecran;
conținutul octetului 16384 este: 01001001

Și ca să nu se ivească discuții, hai să verificăm:

```
      5 PLOT 1,175: PLOT 4,175: PLOT 7,175
     10 PRINT AT 1,0; PEEK 16384
RUN
```

Ar trebui să obțineți $2^6 + 2^3 + 2^0 = 73$ (biții 6, 3 și 0 sunt 1). Presupun că ați obținut acest rezultat și trec mai departe. Să aprindem punctul din colț, dar nu cu `PLOT`, ci direct, scriind în memorie. `NEW`.

Trebuie să facem ca bitul 7 al octetului 16384 să fie 1 (să-l *setăm*). Octetul are valoarea 0, căci după `NEW` toate punctele sunt `PAPER` — deci biții sunt toți 0; atunci, cu bitul 7 setat, octetul va căpăta valoarea $2^7 = 128$.

```
POKE 16384,128
```

A mers, nu? Să rotim acum punctul printre cele 8:

```
     10 FOR i=7 TO 0 STEP -1 : POKE 16384, 2^i : NEXT i
     20 GOTO 10
RUN
```

Ajunge, acum tastați BREAK.

■

Bun! Să vedem mai departe, în ce ordine vin celelalte pachețele de câte 8 puncte. Cu punctele aflate pe aceeași orizontală nu-i mare problemă, pentru că formează 256 puncte/8 = 32 de pachețele, care în memorie sunt consecutive.

Ca să ne chinuim mai puțin vom băga de seamă că ecranul e împărțit în trei felii orizontale suprapuse, care se comportă la fel din punct de vedere al memorării. Primele 8 rânduri de caractere fac parte din prima treime, rândurile 8–15 din a doua și ultimele 8 din a treia. O să privim în detaliu prima treime și vom ști cum stă treaba și cu celelalte.

După cum ne așteptăm, primul memorat este primul rând de pixeli de pe ecran (cel cu y=175). Urmează însă al nouălea șir de puncte sau, dacă preferați, primul șir de puncte din al doilea rând de caractere (y=163). Urmează al 9+8 = 17-lea șir (primul din al treilea rând de caractere) și tot așa, până la primul șir de pixeli din al optulea rând de caractere, ultimul din prima treime. Abia după aceea vine al doilea șir de pe ecran, urmat de al zecelea, optsprezecelea ș.a.m.d.

Să urmărim chiar pe ecran ordinea pentru cei

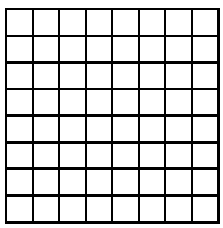
8 (rânduri de caractere) × 8 (felii de pixeli pe caracter) × 32 (caractere pe rând) = 2048 de octeți ai primei treimi:

```
10 FOR i=0 TO 2047 : PAUSE 50
20 POKE 16384+i,255 : PRINT AT 10,0; i
30 NEXT i
```

Apăsând o tastă veți grăbi puțin programul.

Cele 8 feliuțe din care este alcătuit un caracter pe ecran au și ele o regulă interesantă de dispunere. Distanța dintre două felii consecutive este de 256 de octeți, oricare ar fi caracterul de pe ecran.

Iată adresele rândurilor de pixeli ai primului caracter de pe ecran:



← 0 × 256 + 16384
← 1 × 256 + 16384
← 2 × 256 + 16384
← 3 × 256 + 16384
← 4 × 256 + 16384
← 5 × 256 + 16384
← 6 × 256 + 16384
← 7 × 256 + 16384

Se remarcă faptul că cele două rânduri din josul ecranul nu au o formă privilegiată de memorare.

■

Următoarea zonă a memoriei se întinde de la 22528 la 23295 și ocupă 768 de octeți, adică 32×24. Fiecare locație corespunde unui pătrățel din grila de joasă rezoluție (în ordinea firească) și conține chiar atributele acelui pătrățel.

Pentru a memora întreaga imagine de pe ecran sunt deci necesari

3(treimi) $\times$ 2048(octeți pe treime) + 768(culori) octeți, adică 6.75K.

Un alt mod de memorare a imaginii, care să permită fiecărui pixel să aibă o culoare independentă de a vecinilor săi, ar fi avut nevoie de 256 (puncte pe orizontală) $\times$ 192 (puncte pe verticală) $\times$ 1/2 (octet pe punct) = 24576 octeți = 24K!

(Am considerat că fiecare pixel poate avea 16 culori, adică se folosesc pentru el 4 biți — 1/2 de octet.) Iată deci că restricția impusă, de a nu avea mai mult de două culori diferite într-un pătrățel 8$\times$8 duce la importante economii de memorie!

■

Să continuăm explorarea memoriei.

Urmează *bufferul* imprimantei, între 23296 și 23551. Aici se memorează caracterele de transmis spre imprimantă (detalii în Capitolul 18).

De la 23552 la 23733 se găsește zona *variabilelor de sistem*. Aici interpretorul BASIC își memorează tot felul de date care îi sunt necesare. Fiecare variabilă are asociat în mod convențional un nume, dar nu este vorba despre numele unei variabile BASIC, ci un nume pe care i-l dăm spre a putea vorbi mai ușor despre ea. (Acestea sunt numele variabilelor folosite de programul în limbaj de asamblare care a fost pus în ROM — după compilare — și care este interpretorul de BASIC). Lista tuturor variabilelor de sistem se găsește în Anexa H.

Următoarele zone de memorie nu au adrese fixe. Limitele lor sunt arătate chiar de valorile unor variabile sistem. Facem următoarea convenție de notare: dacă PROG desemnează variabila de sistem cu acest nume (adică perechea de locații 23635, 23636), atunci valoarea variabilei PROG (adică `PEEK 23635 + 256 * PEEK 23636`) o vom nota punând între paranteze numele variabilei, astfel: (PROG).

Când valoarea unei variabile sistem este o adresă, atunci octetul cu adresa mai mică din variabilă (nu uitați că o adresă se memorează pe 2 octeți) va conține întotdeauna octetul mai puțin semnificativ al adresei. De aceea, adresa memorată la locația a este `PEEK a + 256 * PEEK (a+1)` și nu `256 * PEEK a + PEEK (a+1)`.

După variabilele sistem, de la 23734 până la (CHANS)–1 se află zona de informații a discului. Dacă un asemenea periferic nu e conectat, zona aceasta are lungimea 0.

De la (CHANS) la (PROG)–2 se găsesc informațiile despre canale și căile asociate. Detalii în Capitolul 18.

Între (PROG) și (VARS)–1 este zona de memorie a programului BASIC.

La (VARS) începe zona variabilelor programului BASIC.

Pentru forma de memorare a programului și variabilelor consultați și Anexa F.

La (E_LINE) se află zona în care se ține comanda sau linia în curs de editare.

Zonele care urmează sunt de mai mic interes. Să mai remarcăm poziția stivei GOSUB imediat sub (RAMTOP). (UDG) arată începutul unei zone interesante, în care se memorează forma grafică a celor 21 de caractere definibile de către utilizator. În setul de caractere ele au coduri între 144 și 164. Inițial formele lor sunt cele ale unor litere, dar pot fi schimbate scriind informații în acești octeți. Cum — vom vedea în capitolul următor.

Se poate ca memoria să nu fie în întregime funcțională. Instrucțiunea `NEW` testează memoria (scriind ceva și verificând dacă rezultatul este întocmai) și plasează valoarea lui P_RAMT chiar sub primul octet care nu merge cum trebuie. Aceasta variabilă indică deci sfârșitul RAM-ului funcțional, care poate să fie diferit de 65535! (Acesta ar fi un semn de defecțiune.)

15.4 USR

ENGLEZĂ: *User SubRoutine* = subrutină a utilizatorului
SINTAXĂ: `USR` *adresa* sau `USR` *şir*
CATEGORIE: funcţie
DESCRIERE:

`USR` funcţionează complet diferit în funcţie de tipul argumentului său. Să vedem deci separat:

15.4.1 USR *adresa*

adresa trebuie să fie cuprinsă între 0 şi 65535. `USR` *adresa* porneşte în execuţie programul cod-maşină de la adresa specificată. În caz că programul se întoarce după terminarea execuţiei la interpretorul BASIC, această funcţie returnează conţinutul registrului BC al microprocesorului. Ca să puteţi folosi această funcţie, trebuie să învăţaţi asamblare (sau cod-maşină).

Iată nişte exemple care invocă programe cod-maşină din componenţa interpretorului BASIC (din ROM).

`LET a = USR 3190` face scroll (valoarea lui a nu ne interesează).

`PRINT 65536 - USR 7962` tipăreşte câtă memorie mai este disponibilă pentru BASIC.

15.4.2 USR *şir*

şir trebuie să fie o literă majusculă sau minusculă situată între `"a"` şi `"u"`, sau unul din caracterele definibile. Funcţia `USR` întoarce adresa primului octet care memorează forma caracterului definibil. Dacă argumentul nu este între limitele indicate, survine eroarea A.

Pentru exemple de folosire vedeţi funcţia `BIN`, în capitolul următor.

În mod normal `PRINT USR "a"` scrie 65368 (= (UDG))

15.5 Exerciţiu rezolvat

Afişaţi un caracter pe ecran fără a folosi `PRINT`.

Rezolvare

Un caracter este memorat folosind 8 octeţi, care descriu exact forma pe care o are el la tipărire. Aceşti octeţi se află în mod normal în ROM, la adresa indicată de variabila sistem CHARS (23606). (Nimeni nu ne împiedică să mutăm CHARS astfel încât să puncteze pe undeva prin RAM şi să definim propriul nostru set de caractere. Citiţi şi Anexa H, partea referitoare la „jonglerii cu variabile sistem".) Dacă transferăm aceşti 8 octeţi din ROM direct în memoria ecran, putem obţine imaginea caracterului. (Aşa face dealtfel interpretorul BASIC din ROM, pentru a tipări un caracter.) Pentru alte precizări citiţi şi partea consacrată programării caracterelor definibile din capitolul următor, căci acestea se memorează ca şi caracterele normale tipăribile (cele cu coduri între 32 şi 127).

```
10 LET x=0 : REM coordonata orizontala
```

```
 20 IF x>31 THEN STOP
 30 LET adset = PEEK 23606 + 256*PEEK 23607
 35 REM adset este adresa unde se afla imaginea setului de caractere
 40 INPUT "caracterul ";LINE b$
 45 LET b = CODE b$ : IF b < 32 OR b > 127 THEN GOTO 40
 50 LET adchar = adset + 8*b : REM un caracter = 8 octeti
 60 FOR i=0 TO 7 : REM cele 8 randuri ale caracterului
 70   POKE 16384 + x + i*256 , PEEK (adchar + i)
 80 NEXT I
 90 LET x = x + 1
100 GOTO 40
```

15.6 Exerciţii

1. Scrieți un scurt program care indică poziția, lungimea și spațiul variabilelor unui program.
2. Folosind variabila de sistem CHARS, mimați un contor de casetofon cu trei cifre. (Folosiți indicațiile de la descrierea acestei variabile în Anexa H, la secțiunea „jonglerii".)

Chapter 16

Prin măruntaiele HC-ului

Un capitol ceva mai scurt. Învățăm să definim noi caractere, să convertim numerele în șiruri și să interacționăm cu perifericele.

- `BIN` pentru a introduce constante binare;
- programarea caracterelor definibile;
- `STR$`, care transformă numere în șiruri de cifre;
- `IN` pentru a citi porturi;
- `OUT` pentru a scrie la porturi;
- porturile folosite de HC.

16.1 BIN

ENGLEZĂ: *BINary* = în baza doi; cu două valori
SINTAXĂ: `BIN [0][1]`
CATEGORIE: funcție
DESCRIERE:

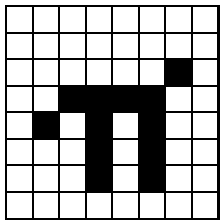
`BIN` este o funcție foarte specială, din două motive. Întâi, ea poate să nu aibă argument; atunci argumentul ei este considerat 0 (toate celelalte funcții au un număr fix de argumente). Apoi, argumentul lui `BIN` trebuie să fie o constantă binară; `BIN` nu poate avea drept argument o expresie!

Numărul de după `BIN` trebuie să fie o înșiruire de 0 până la 16 cifre binare, fără punct zecimal. `BIN` returnează valoarea acelui număr.

■

BIN este adesea folosită pentru a „programa" noi caractere (cele definibile).

Fiecare *caracter definibil* (prescurtat UDG=*User Defined Graphics*) își are imaginea memorată undeva în RAM, în mod normal imediat deasupra RAMTOP-ului. Imaginea se memorează pe 8 octeți așezați în locații succesive, conținând câte un rând de pixeli ai caracterului. Fiecare rând este memorat în octetul corespunzător, astfel: un punct aprins pe ecran corespunde unui bit 1 în octet, iar fiecare punct stins, unui bit 0. Iată corespondența dintre rânduri și locațiile de memorie pentru un semn ce înfățișează litera grecească π:



```
← BIN 00000000 = BIN 0 = BIN = 0
← BIN 00000000
← BIN 00000010 = BIN 10 = 2
← BIN 00111100 = BIN 111100 = 60
← BIN 01010100 = BIN 1010100 = 84
← BIN 00010100 = BIN 10100 = 20
← BIN 00010100
← BIN 00000000
```

Programul care transformă caracterul UDG P în π este:

```
 5 FOR i = 0 TO 7
10   READ a
20   POKE USR "p" + i, a
30 NEXT i
40 DATA 0,0,BIN 10, BIN 111100, BIN 1010100, BIN 10100, BIN 10100, 0
```

16.2 STR$

ENGLEZĂ: *STRing* = șir
SINTAXĂ: `STR$` *expr*
CATEGORIE: funcție
DESCRIERE:

`STR$` este funcția complementară lui `VAL`; argumentul ei este un număr, iar rezultatul este șirul de caractere pe care l-ar scrie `PRINT` pe ecran dacă ar tipări acest număr. Exemple:

```
STR$ 1e3       este "1000"
STR$ 100.000   este "100"
```

Combinând `STR$` cu `VAL` putem obține efecte interesante. De pildă, putem forța evaluarea funcției `BIN` cu un argument neconstant. Iată:

```
LET a=101001 : PRINT VAL ("BIN "+STR$ a)
```

16.3 Exerciţiu rezolvat

Scrieți un program care testează cunoștințele de tabla înmulțirii.

Rezolvare

Soluția banală care urmează cuprinde o cursă:

```
 10 LET a = INT (RND*10) + 1: LET b = INT (RND*10) + 1
 20 PRINT "Cat face ";a;" * ";b;" ?";TAB 20; : POKE 23692, -1
 30 INPUT c : PRINT c;" - ";
100 IF c = a*b THEN PRINT "Corect" : GOTO 10
110 PRINT "Gresit. Mai incercati.": GOTO 20
```

Iată de ce un astfel de program nu este corect: la întrebarea `Cat face 2 * 3 ?` pot să răspund la `INPUT` cu `2*3` — o expresie, care va da răspunsul corect. Pentru a evita posibilitatea introducerii unei expresii, vom folosi `INPUT` cu un șir pe care îl vom evalua. Modificăm programul astfel:

```
 30 INPUT c$ : LET c = VAL c$ : PRINT c$;" - ";
 40 IF c$ <> STR$ c THEN PRINT "Nu trisa !": GOTO 20
```

Nici măcar acum programul nu este perfect, pentru că un bun cunoscător al BASIC-ului va putea șterge ghilimelele afișate de `INPUT` și va putea tasta în schimb `STR$(2*3)`. Pentru a evita și aceasta perfidie, `INPUT` trebuie să fie schimbat cu `INPUT LINE`.

■

Așa cum există 65536 locații de memorie, numerotate de la 0 la 65535, există și 65356 de *porturi* între aceleași limite. Porturile servesc UC pentru a schimba informații cu dispozitivele periferice, tot așa cum locațiile se folosesc pentru a citi date din memorie. Comunicația cu exteriorul este realizată prin intermediul porturilor. Dacă un port este conectat la un periferic care așteaptă să citească informații de la calculator, atunci este un port *de ieşire*. Dacă prin acel port calculatorul primește informații de la periferic, portul se numește *de intrare*. Informația care se comunică unui/de la un port este un număr întreg între 0 și 255, deci un octet.

16.4 `IN`

ENGLEZĂ: *in* = înăuntru
SINTAXĂ: `IN` *port*
CATEGORIE: funcție
DESCRIERE:

port este adresa unui port, adică un număr cuprins între 0 și 65535. Altfel survine eroarea B. `IN` citește și returnează valoarea de la portul specificat.

16.5 OUT

ENGLEZĂ: *out* = în afară
SINTAXĂ: `OUT` *port*, *byte*
CATEGORIE: comandă
DESCRIERE:

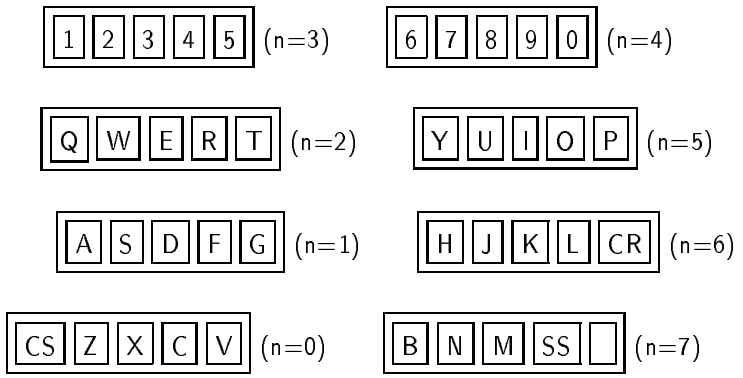
port este adresa unui port. *byte* este un număr între 0 și 255. Altfel obținem eroare B. `OUT` trimite valoarea *byte* la portul specificat.

Nu toate cele 65536 porturi sunt conectate la un periferic. Vom da aici lista celor care sunt utile. Să remarcăm că scrierea la un port nefolosit (cu `OUT`) nu are efect, iar citirea de la un port nefolosit dă un rezultat neprecizat (probabil 0).

■

Tastatura este citită prin intermediul porturilor. Există o sumă de programe care se ocupă cu decodificarea tastei apăsate și transformarea ei într-un caracter, ținând cont de tipul cursorului și toată bucătăria. `IN` permite doar verificarea faptului că o tastă este apăsată sau nu.

Întreaga tastatură este grupată în 8 semi-rânduri numerotate de la 0 la 7, ca în figură (n este numărul semi-rândului):



Fiecare semirând are asociat un port care depistează care din taste sunt apăsate. Cele 8 porturi sunt:

Port	n	Taste
65278	0	CS-V
65022	1	A-G
64510	2	Q-T
61486	3	1-5
61438	4	0-6
57342	5	P-Y
49150	6	CR-H
32766	7	blanc-B

Regula de calcul a adresei este `port(n) = 254 + 256 *(255 - 2^n)`.

Informația citită de la port este un număr care, dacă-l considerăm scris în baza 2, are în ultimii cinci biți o „oglindă" a tastelor din semi-rândul respectiv. Ordinea tastelor este următoarea: tasta cea mai din exterior corespunde bitului 0 (cel mai puțin semnificativ) tasta cea mai din interior bitului 4. Valoarea biților 5-7 nu este influențată de tastatură, dar nu e bine să mizați pe ea ca fiind constantă. Bitul 5 este influențat de casetofon (este folosit pentru comunicația cu casetofonul)!

În realitate putem citi mai multe semi-rânduri odată, dar nu putem spune pe care din ele e apăsată o tastă. Adresa unui port pentru această operațiune se obține după cum urmează: octetul mai puțin semnificativ e 254. În octetul mai semnificativ, toți biții sunt 1, cu excepția celor care corespund unor semi-rânduri pe care le dorim citite. De exemplu `IN BIN 1011110111111110` citește semi-rândurile 1 și 6 (biții 1 și 6 din cel mai semnificativ octet sunt 0). Testați următorul program:

```
 5 INPUT "Octetul mai semnificativ in binar ";a : PRINT a
10 LET adr = 254 + 256*VAL ("BIN "+STR$ a)
20   LET info = IN adr
30   LET a$ = "" : REM extragem in a$ ultimii 5 biti
40   FOR i=1 TO 5
50        LET r=a - 2*INT (a/2) : LET a = INT (a/2)
60        LET a$ = STR$ r + a$
70   NEXT i
80   PRINT AT 1,10;" taste apasate :";a$
90 GOTO 20
```

Apăsați apoi diferite taste, corespunzând sau nu semi-rândurilor alese.

Remarcați că `IN` ne permite să citim mai multe taste apăsate simultan sau să vedem dacă [SS] e apăsat singur, ceea ce `INKEY$` nu poate!

Din păcate, datorită construcției calculatorului, se întâmplă un fenomen neplăcut: dacă sunt apăsate trei taste astfel: două în același semi-rând, și una în alt semi-rând corespunzând uneia din celelalte două, atunci calculatorul are impresia că și a patra tastă, corespunzătoare celei de-a doua din primul semi-rând este apăsată! Acesta este un defect de construcție și nu poate fi nicicum evitat folosind programe.

De aceea, apăsând simultan [CS], [V] și [B], calculatorul crede că s-a apăsat și spațiu ([V] corespunde lui [B], spațiul lui [CS]). De aceea, această combinație de trei taste este echivalentă cu [BREAK] ([CS] + [ ]) și poate opri un program din execuție. Încercați! Acesta este un impediment serios pentru jocurile care se desfășoară între doi participanți, și au nevoie de multe taste, pentru că cei doi se influențează reciproc prin tastatură.

■

Alte porturi importante:

- Portul 254, folosit ca port de ieșire (este distinct de portul de intrare 254!): biții 0, 1, 2 controlează culoarea BORDER-ului (care e considerat periferic!), bitul 3 ieșirea spre casetofon (este folosit la salvarea programelor pe bandă), bitul 4 este dedicat difuzorului (este folosit de comanda `BEEP`). Din păcate, viteza BASIC-ului este prea mică pentru a putea produce semnale utile la biții 3 și 4. Puteți încerca `OUT 254,0` pentru a vedea efectul asupra BORDER-ului (dar nu și asupra celor două linii din josul ecranului).

- Portul 251 este folosit în lucrul cu imprimanta, atât pentru a trimite date, cât și pentru a le primi (imprimanta semnalizează când a terminat de lucrat).
- Porturile 254, 247 și 239 mai sunt folosite și pentru comunicarea cu alte periferice: *microdrive*, interfața serială RS232.

Pentru alte informații, rugăm să va adresați cărții tehnice a calculatorului. Pentru cât vom lucra noi, ceea ce știm este mai mult decât suficient.

16.6 Exerciţii

1. Scrieți un program pentru definirea caracterelor ă, î, ș, ț, â . Eventual folosiți formele predefinite ale caracterelor a, i, s, t, a din ROM.
2. Scrieți un program pentru conversii între cele trei baze importante de numerație (2, 10, 16). Lucrați numai cu numere întregi.
3. Scrieți un program pentru desenare, în care utilizatorul ghidează un punct pe ecran cu patru taste, dar are și posibilitatea de a combina câte două direcții simultan. Prevedeți o metodă pentru a putea șterge desenele greșite.
4. Este adevărată relația `ATTR (y,x) = PEEK (22528 + y*32 + x)` pentru orice `x` și `y` admisibili pozitivi? Justificați.

Chapter 17

Casetofonul

Învățăm să inspectăm ecranul, să definim noi funcții și să folosim casetofonul pentru a pune la păstrare capodoperele BASIC pe care le-am creat.

- `SCREEN$` pentru a citi un caracter de pe ecran;
- `DEF FN` pentru definirea de funcții;
- `FN` pentru folosirea funcțiilor definite;
- `SAVE` care pune programe pe bandă;
- `LOAD` care citește programe de pe bandă;
- `MERGE` – amestecă programe;
- `VERIFY` – verifică programe.

17.1 SCREEN$

ENGLEZĂ: *screen* = ecran
SINTAXĂ: `SCREEN$ (`*y,x*`)`
CATEGORIE: funcție
DESCRIERE:

`SCREEN$` este o funcție mai rar utilizată, care are drept parametri coordonatele unui pătrățel din grila de joasă rezoluție. Dacă *x* nu e între 0 și 31 și *y* între 0 și 23, veți obține eroarea B.

`SCREEN$` încearcă să recunoască caracterul tipărit pe ecran la coordonatele *y, x* și să întoarcă un șir format din acest caracter. Dacă nu recunoaște caracterul, întoarce `""`. `SCREEN$` recunoaște și caracterele scrise cu `INVERSE 1`.

```
PRINT AT 5,5;"STOP" : PRINT SCREEN$ (5,6)
```

va scrie pe ecran STOP și apoi T.

17.2 DEF FN

ENGLEZĂ: *to DEFine* = a defini, *FuNction* = funcție
SINTAXĂ: `DEF FN` *f*`[$]`$(p_1, \ldots, p_n)$ `=` *expr*
CATEGORIE: comandă
DESCRIERE:

`DEF FN` permite programatorului să-și definească propriile funcții, prin compunerea funcțiilor existente.

f este numele funcției care se definește, format dintr-o singură literă. Literele mici sunt echivalente cu majusculele.

Numele funcției trebuie să fie succedat de semnul `$`, dacă funcția va avea un rezultat de tip șir. `$` trebuie să lipsească dacă funcția dă un rezultat numeric.

$p_1, \ldots, p_n$ sunt *parametrii simbolici* ai funcției, care apar în expresia de definiție a funcției și al căror rol îl vom defini la prezentarea instrucțiunii `FN`. p_i sunt ca niște nume de variabile de o singură literă, conținând semnul `$` dacă este vorba de parametri de tip șir.

expr este o expresie BASIC, în care de obicei apar parametrii simbolici, expresie care arată cum se definește funcția.

Executarea instrucțiunii `DEF FN` nu are nici un efect, asemenea instrucțiunii `DATA`. Ea este folosită doar în momentul executării instrucțiunii `FN` corespunzătoare.

17.3 FN

ENGLEZĂ: *FuNction* = funcție
SINTAXĂ: `FN` *f*$(v_1, \ldots, v_n)$
CATEGORIE: funcție
DESCRIERE:

f este numele funcției invocate, care trebuie să fie definită undeva în program cu `DEF FN`.

$v_1, \ldots v_n$ sunt expresii care, evaluate, dau valorile parametrilor cu care se va apela funcția *f*.

Să vedem ce se întâmplă la executarea instrucțiunii `FN`.

1. Întâi, calculatorul caută în întreg programul o definiție de funcție (o instrucțiune `DEF FN`) pentru același nume *f* (și de același tip) cu funcția invocată. Dacă instrucțiunea `DEF FN` nu există, survine eroarea P.
2. Dacă definiția a fost găsită, se evaluează pe rând expresiile (numite *argumente*) v_i. Rezultatele evaluărilor devin valorile parametrilor simbolici p_i din instrucțiunea `DEF FN` (dar nu valorile unor variabile BASIC cu același nume!). Dacă expresiile nu generează rezultate de același tip cu al parametrilor simbolici corespunzători sau numărul parametrilor simbolici nu este egal cu al argumentelor, se obține eroarea Q.

3. După aceasta, se trece la evaluarea expresiei din instrucțiunea `DEF FN`. Oriunde apar nume de parametri simbolici, se folosește valoarea pe care tocmai au căpătat-o. Dacă apar numele altor variabile, atunci sunt folosite valorile variabilelor programului BASIC.

4. În fine, rezultatul evaluării acestei expresii este întors de apelul funcției `FN`.

Exemplele sunt mai mult decât utile:

```
5 DEF FN r(x) = INT (x+.5)
```

este definiția funcției „rotunjire". La executarea instrucțiunii:

```
PRINT FN r(PI)
```

se parcurg următorii pași:

1. definiția este localizată în linia 5;
2. se evaluează expresia `PI`. Parametrul `x` devine 3.1415926 (o eventuală variabilă `x` este nealterată);
3. se evaluează expresia `INT (3.1415926 + .5)`, care dă 3;
4. se tipărește 3.

Iată o definiție pentru funcția putere:

```
 1 DEF FN p(x,y) = (ABS x) ^ y
```

Funcția aleatoare (fără parametri !):

```
 2 DEF FN a() = RND
```

Să mai vedem un exemplu:

```
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 DEF FN r(a) = FN p( FN q() , 6)
50 PRINT FN p(2,3), FN q(), FN r(0)
```

Acest program va scrie numerele 16, 10 și 70. Să vedem cum le obține:

`FN p(2,3) = a + 2*3 = 10 + 2*3 = 16`

(se folosește valoarea curentă a variabilei `a`).

`FN q() = a + x*y = 10 + 0*0 = 10`

(folosește valorile curente ale tuturor celor trei variabile.)

`FN r(0) = FN p(10, 6) = a + 10*6 = 70`

(funcția `r` este, practic, independentă de argumentul său.)

Se pot defini și funcții *recursive* (care folosesc în expresia de definiție un apel la ele însele), dar trebuie făcute niște scamatorii. Curioșilor le recomandăm să studieze această problemă mai pe larg. Evaluarea lui `FN` nu folosește o stivă; valorile parametrilor sunt memorate chiar în linia BASIC care conține instrucțiunea `DEF FN` (vezi și Anexa F), de aceea se pot realiza numai *recursii la coadă*. (Nu intrăm în amănunte în legătură cu recursia la coadă.)

Iată un exemplu : definiția *factorialului* (n!=1 × 2 × 3 × . . . × n):

```
5 DEF FN f(n) = VAL ( ("1" AND n=0) +
                      ("n * FN f(n-1)" AND n>0) )
```

Încercați `PRINT FN f(5)`.

■

Instrucțiunile care urmează se folosesc pentru a schimba informații cu casetofonul. Acesta este un lucru util pentru transportarea programelor de la un calculator la altul și păstrarea lor după întreruperea alimentării cu curent electric. Se spune despre casetofon că este *un suport de memorie permanentă*.

17.4 SAVE

ENGLEZĂ: *to save* = a salva
SINTAXĂ: `SAVE` *numeprg* `[LINE` *etch*`] [SCREEN$] [CODE` *adr*`,` *lung*`] [DATA` *var*`()]`
CATEGORIE: comandă
DESCRIERE:

`SAVE` *salvează* (termen consacrat pentru păstrarea pe suporturi permanente de memorie) o parte din conținutul memoriei. Pentru informația salvată se mai folosește numele de *fișier*. `SAVE` poate avea o mulțime de forme, pe care le vom analiza separat. Numai unul dintre „parametrii" indicați ca opționali poate apărea într-o instrucțiune `SAVE`.

În toate cazurile, *numeprg* este un șir de 1 până la 10 caractere, care va fi numele cu care este identificată informația salvată. Dacă numele nu respectă această regulă survine eroarea F.

Înainte de a începe salvarea, pe ultima linie a ecranului se afișează mesajul `Start tape, then press any key`, care înseamnă „Porniți casetofonul și apoi apăsați orice tastă". După apăsarea unei taste, mesajul este șters și salvarea începe.

17.4.1 SAVE *numeprg*

Salvează pe bandă programul BASIC din memorie, împreună cu toate variabilele sale și valorile lor curente. Dacă nu este nevoie de variabile la o încărcare ulterioară (pentru că programul își face singur inițializările), se recomandă `CLEAR` înainte de `SAVE`.

17.4.2 SAVE *numeprg* LINE *etch*

etch este o etichetă. Această instrucțiune face același lucru ca `SAVE` *numeprg*, cu diferența că programul, atunci când va fi încărcat în calculator, va porni automat în execuție de la linia cu eticheta *etch*.

17.4.3 SAVE *numeprg* SCREEN$

salvează pe bandă conținutul memoriei video și informația de culoare. Este echivalentă cu SAVE *numeprg* CODE 16384,6912

17.4.4 SAVE *numeprg* CODE *adresa*, *lung*

salvează conținutul unei zone de memorie cu lungimea *lung* (în octeți), octet cu octet, începând de la *adresa*.

17.4.5 SAVE *numeprg* DATA *var* ()

unde *var* este numele unei variabile multidimensionale de orice tip. Această instrucțiune salvează conținutul matricei cu numele *var*.

17.5 LOAD

ENGLEZĂ: *to load* = a încărca
SINTAXĂ: LOAD *numeprg* [SCREEN$] [CODE [*adresa* [, *lung*]]] [DATA *var*()]
CATEGORIE: comandă
DESCRIERE:

LOAD este comanda pentru citirea unui program de pe bandă magnetică în memorie. Vom discuta pe rând formele posibile.

numeprg poate să fie un șir nul. În acest caz se încarcă de pe bandă primul program care are tipul specificat de „parametrii" opționali (nu sunt chiar niște parametri, dar nu inventăm acum nume noi).

17.5.1 LOAD *numeprg*

citește de pe bandă programul BASIC și variabilele sale, care au fost salvate sub același nume cu cel specificat. Programul prezent în memorie este pierdut.

17.5.2 LOAD *numeprg* SCREEN$

este echivalent cu LOAD *numeprg* CODE 16384, 6912

17.5.3 LOAD *numeprg* CODE *adresa*, *lung*

citește de pe bandă primul fișier salvat cu SAVE *n* CODE *x*, *y*, care are numele corespunzător și lungimea $y \leq$ *lung*. Citirea se face în memorie începând de la adresa specificată și nu de la cea folosită la salvare (*x*).

17.5.4 LOAD *numeprg* CODE *adresa*

acționează precum instrucțiunea precedentă, dar fără a mai constrânge în vreun fel lungimea programului salvat.

17.5.5 LOAD *numeprg* CODE

încarcă un program salvat cu SAVE ...CODE chiar la adresa de la care a fost salvat.

17.5.6 LOAD *numeprg* DATA *var*()

citește matricea de pe bandă, ștergând matricea existentă cu același nume. Poate surveni eroarea 4 în cazul lipsei de memorie.

■

Dacă citirea de pe bandă se face eronat (cel mai adesea calculatorul poate depista dacă sunt informații incorecte pe bandă), survine eroarea R.

Să aruncăm o privire asupra modului în care este organizat un fișier pe bandă. Salvarea se face în două părți. Prima parte este o bucățică numită în engleză *header* și poartă informații despre fișier: nume, tip și lungime. Header-ul are întotdeauna 17 octeți. După header urmează fișierul propriu-zis. Atât header-ul cât și fișierul încep pe bandă printr-o zonă cu un sunet uniform, folosit în sincronizare. În timpul citirii zonelor de sincronizare, BORDER-ul este în dungi roșii și albastre deschis (*red* și *cyan*). Pentru zonele de informație, dungile sunt albastre și galbene (*blue* și *yellow*).

17.6 MERGE

ENGLEZĂ: *to merge* = a amesteca
SINTAXĂ: MERGE *numeprg*
CATEGORIE: comandă
DESCRIERE:

MERGE există doar pentru fișiere de programe BASIC (nu și pentru CODE sau DATA). Efectul său este de a amesteca variabilele și programul de pe bandă cu cele din memorie. Liniile și variabilele care există în ambele programe se iau de pe bandă, iar vechea informație se pierde. Liniile de pe bandă sunt intercalate la locurile lor, printre cele existente deja.

Erori posibile:

- R, eroare de citire pe bandă;
- 4, memorie insuficientă.

Un program încărcat cu MERGE nu pornește automat în execuție, chiar dacă a fost salvat cu SAVE LINE *x*.

Ca și la LOAD, *numeprg* poate fi șirul nul și atunci este folosit primul fișier întâlnit.

17.7 VERIFY

ENGLEZĂ: *to verify* = a verifica
SINTAXĂ: `VERIFY` *numeprg* `[SCREEN$] [CODE [`*adr* `[,`*lg*`]]] [DATA` *var*`()]`
CATEGORIE: comandă
DESCRIERE:

`VERIFY` seamănă foarte mult cu `LOAD`, doar că nu încarcă programul (datele) de pe bandă, ci îl compară cu cel existent în memorie. Diferențele sunt anunțate prin eroarea R. Se recomandă întotdeauna verificarea unui program salvat.

■

La executarea comenzilor `LOAD`, `MERGE` sau `VERIFY` calculatorul afișează când apar header-ele întâlnite, în forma următoare:

`Program : numeprg`	pentru programe BASIC
`Bytes : numeprg`	pentru programe salvate cu `CODE` sau `SCREEN$`
`Number array : numeprg`	pentru o matrice numerică
`Character array : numeprg`	pentru o matrice de caractere

unde `numeprg` presupunem că este numele sub care a fost salvată informația. Acest mesaj se tipărește întotdeauna pe rând nou.

`LOAD`, `VERIFY` și `MERGE` dau variabilei sistem SCR_CT valoarea 3, așa încât niciodată nu vor produce apariția mesajului `scroll?`.

Iată de ce `VERIFY "" SCREEN$` va da aproape sigur eroare: în momentul întâlnirii antetului (header-ului) se va scrie pe ecran `Bytes : nume`, ceea ce va modifica imaginea existentă, care nu va mai corespunde cu cea salvată. Pentru a salva integral imagini (cu tot cu cele două rânduri de jos) și pentru a le putea verifica, se recomandă întâi mutarea lor în altă zonă de memorie, rezervată anterior cu `CLEAR` (dar anterior desenării, deoarece `CLEAR` șterge și ecranul).

17.8 Exerciţiu rezolvat

Imaginați o metodă prin care citirea header-ului să provoace nu scrierea mesajului

`Program: nume`

ci a unui text oarecare (nu prea lung).

Rezolvare

Programul care realizează această încărcare miraculoasă arată după cum urmează (să presupunem că numele programului salvat este un șir format din 10 caractere `CHR$ 226 = STOP`):

```
 1 PAPER 7: INK 7: OVER 1: CLS
 5 PRINT AT 1,0; "Program : ";
 7 FOR i=1 TO 10: PRINT CHR$ 226; : NEXT i
10 PRINT AT 1,0; "Textul acesta va aparea la citirea headerului"
15 INK 0: PRINT AT 0,0;
20 LOAD ""
```

Pe ecran sunt suprapuse în `OVER 1` două texte: cel ce trebuie să apară și cel afișat în mod normal de `LOAD` la întâlnirea header-ului, alb pe alb. Când header-ul tipărește a doua oară textul (de data aceasta în `INK 0`), datorită proprietăților lui `OVER`, va apărea de fapt celălalt text.

17.9 Exerciţii

1. Scrieți un program care să permită salvarea și verificarea conținutului zonei memoriei video.

   Indicaţie: pentru a salva integral imagini (inclusiv cele două rânduri de jos) și pentru a le putea verifica, se recomandă întâi mutarea lor în altă zonă de memorie, rezervată anterior cu `CLEAR`.

2. Imaginați o situație când `VERIFY "" SCREEN$` ar merge.

3. Este `SCREEN$` funcția „opusă" vreunei comenzi sau funcții (în sensul `IN-OUT` de pildă)?

Chapter 18

Periferice

- `LLIST` pentru listare la imprimantă;
- `LPRINT` pentru tipărire la imprimantă;
- `COPY` pentru desenare la imprimantă;
- drivere care deservesc periferice. Căi și canale.
- `OPEN#` pentru a deschide o cale;
- `CLOSE#` pentru a închide o cale;
- `PRINT#, INPUT#, LIST#, INKEY$#` pentru comunicații pe căi;
- Discul:
  - `LOAD*` pentru a încărca;
  - `SAVE*` pentru a salva;
  - `MERGE*` pentru a amesteca;
  - `VERIFY*` pentru a verifica;
  - `CAT` pentru a inspecta;
  - `ERASE` pentru a șterge;
  - `FORMAT` pentru a inițializa;
  - `MOVE` pentru a copia;
  - `CLEAR#` pentru a închide canale;
  - `CLS#` pentru a șterge ecranul.

Probabil că știți ce este o *imprimantă*: un fel de mașină de scris comandată de calculator. (Imprimantele tind să semene din ce în ce mai puțin cu o mașină de scris.) Pentru SPECTRUM exista o imprimantă foarte specială, care nu semăna cu nici o alta. Dacă aveți una obișnuită s-ar putea să mai aveți nevoie și de un program special (*driver*, după cum vom explica de îndată) pentru a o face să funcționeze.

Să vedem cum se poate folosi cu HC una:

18.1 LLIST

ENGLEZĂ: *Line printer* LIST = LIST la imprimantă
SINTAXĂ: `LLIST [#` *cale*`] [`*etch*`]`
CATEGORIE: comandă
DESCRIERE:

`LLIST` listează programul la imprimantă începând de la linia *etch* sau de la prima dacă *etch* lipsește. Listarea poate fi oprită cu BREAK. Despre #*cale* vorbim spre sfârșitul capitolului.

18.2 LPRINT

ENGLEZĂ: *Line printer* PRINT = PRINT la imprimantă
SINTAXĂ: `LPRINT [#` *cale*`] [ AT` *y*`,` *x*`] [TAB` *x*`] [`*expresie*`] ['] [;] [,]`
CATEGORIE: comandă
DESCRIERE:

Sintaxa este ca a lui `PRINT`, doar că nu mai apar culorile. Despre #*cale*, spre sfârșitul capitolului.

`LPRINT` tipărește la imprimantă valorile expresiilor date ca parametri. Există câteva mici diferențe față de `PRINT`. În primul rând, tipărirea nu are loc odată cu execuția instrucțiunii. Caracterele sunt strânse în buffer-ul imprimantei (vezi Capitolul 15 și Anexa C) și trimise grupat. Tipărirea pe hârtie se face atunci când buffer-ul este trimis spre imprimantă; asta se întâmplă când:

- buffer-ul este plin;
- o instrucțiune `LPRINT` nu se termină cu , sau cu ;
- o virgulă, un apostrof sau un `TAB` trebuie să treacă pe rând nou;
- la sfârșitul programului.

Instrucțiunea `LPRINT AT` ignoră coordonata *y*. Nu este echivalentă cu `LPRINT TAB` *x*, pentru că `AT` nu trece niciodată pe rând nou.

```
10 FOR n=5 TO 0 STEP -‘
20 LPRINT TAB n;"x"
30 NEXT n
```

scrie ceva de genul

```
    x
   x
  x
 x
x
```

pe când, schimbând linia 20 în

```
 20 LPRINT AT 21-n ,n ;"x"
```

vom obține:

```
xxxxx
```

18.3 COPY

ENGLEZĂ: *to copy* = a copia
SINTAXĂ: `COPY`
CATEGORIE: comandă
DESCRIERE:

Dacă imprimanta lucrează în mod grafic, atunci `COPY` pur și simplu face o copie după conținutul ecranului.

Instrucțiunile pentru lucrul cu imprimanta, folosite fără a avea o imprimantă conectată, nu au efect.

18.4 Drivere

Să presupunem că un program a efectuat niște calcule și trebuie să comunice utilizatorului rezultatele. Utilizatorul poate cere ca datele sa-i fie afișate pe ecran, sau la imprimantă sau poate trimise spre un alt calculator prin interfața de la rețea sau mai știu eu ce. De toate aceste operații se ocupă o parte a interpretorului BASIC numită *driver*. Driver este un cuvânt englezesc și înseamnă „șofer". Numele arată că driverul „conduce" informația spre perifericul respectiv. Acest program (căci un program este) primește un șir de caractere și trebuie să-l transforme într-o formă care să fie „priceută" de un periferic.

De ce sunt driverele importante? Pentru că interacțiunea programatorului cu un driver este aceeași, indiferent de natura perifericului pe care acel driver îl deservește. Cu alte cuvinte, cu instrucțiuni foarte asemănătoare se scre atât pe ecran cât și la imprimantă, folosind drivere diferite. Este treaba driverelor să transforme informația într-una înțeleasă de perifericul respectiv.

Există drivere pentru fiecare tip de periferic care se poate conecta la calculator. Dacă inventați un periferic nou, trebuie să scrieți și un driver pentru el. Pentru un același periferic pot exista mai multe drivere diferite. De pildă, pentru ecran putem imagina, pe lângă driverul existent, unul care scrie cu caractere mai înguste. Pentru cei care știu asamblare este un exercițiu interesant.

Există și drivere care funcționează în sens invers, dinspre periferic spre calculator. Ele convertesc datele achiziționate de la periferic într-o formă care are o semnificație pentru programele

care se execută pe calculator. Un exemplu tipic este driverul de tastatură, care, citind tastele de la porturile tastaturii, transformă informația în caractere, ținând cont de cursor.

La un calculator „adevărat" (HC este mai curând o jucărie), driverele fac parte dintr-un program mare care numit ***sistem de operare***. HC-ul, deși nu are sistem de operare, niște drivere tot are. La ele și la modul în care preiau informația o să ne uităm acum.

În nomenclatura HC (preluată de la SINCLAIR) driverele poartă denumirea de ***canale***. Canalele au nume de câte o literă. Există din construcție trei canale, dar la adăugarea unor periferice mai apar și altele.

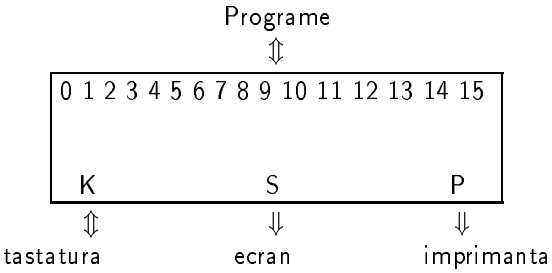
Canalul K (KEYBOARD) deservește tastatura. Surprinzător, este un canal bidirecțional, adică servește atât la citirea datelor (de la tastatură) cât și la scrierea lor.

Canalul S (SCREEN) deservește ecranul. Este un canal unidirecțional de ieșire.

Canalul P (PRINTER) pentru imprimantă. Este un canal unidirecțional.

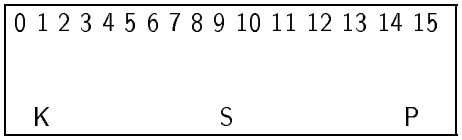
Calculatorul TIM-S, ***compatibil*** cu HC (adică toate programele care merg pe unul merg și pe celălalt), are și alte canale: G, A, M, B, X. Pentru detalii, trimitem la cartea tehnică a TIM-S ului.

Pentru a putea trimite/citi datele spre/de la un canal este prevăzut un mecanism suplimentar. Avem 16 ***căi***, numerotate de la 0 la 15. Imaginea pe care trebuie să ne-o facem este următoarea:



În „cutia" pe care am desenat-o mai sus, între căi (0–15) și canale (K, S, P) pot exista niște „conexiuni". Căile pot fi ***închise***, și atunci nu sunt conectate cu nici un canal sau ***deschise***, și atunci ele sunt conectate cu un canal. Vom vedea imediat cum se fac și cum se desfac aceste conexiuni. Ajunge să spunem că, dacă am conectat calea 8 cu canalul K, atunci toate datele pe care programul le va scrie pe calea 8 vor ajunge la canalul K și toate datele pe care programul le va citi de la calea 8 vor veni de la canalul K. Dacă o cale este conectată la canalul S sau P, atunci de pe ea nu se pot citi date, căci aceste canale sunt unidirecționale.

La pornirea calculatorului sunt făcute automat următoarele conexiuni:



adică avem legate:

căile 0 și 1	cu canalul K
calea 2	cu canalul S
calea 3	cu canalul P
căile 4-15	sunt închise

18.5 OPEN#

ENGLEZĂ: *to open* = a deschide, # = prescurtare pentru „nr"
SINTAXĂ: `OPEN#` *cale*, *canal*
CATEGORIE: comandă
DESCRIERE:

Această comandă face o conexiune cale-canal. *cale* specificată este legată de *canal* specificat. *cale* trebuie să fie un număr cuprins între 0 și 15. Dacă este mai mare ca 255 se obține eroarea B. Dacă este peste 15 se obține eroarea O. Canalul este indicat printr-o literă, majusculă sau minusculă, dintre cele trei: `"k"`, `"s"` sau `"p"`.

18.6 CLOSE#

ENGLEZĂ: *to close* = a închide
SINTAXĂ: `CLOSE#` *cale*
CATEGORIE: comandă
DESCRIERE:

Închide calea specificată. **Atenţie**: tentativa de a închide o cale nedeschisă se soldează cu catastrofe!

Încercarea de a închide una din căile 0-3 se soldează cu redeschiderea lor automată pe canalul inițial.

■

Să vedem acum ce instrucțiuni pot transfera date prin canale.

Există numai patru instrucțiuni BASIC care pot face acest lucru; iată-le: `PRINT` (`LPRINT`), `INPUT`, `LIST` (`LLIST`) și `INKEY$#`.

Fiecare din instrucțiunile de mai sus lucrează în mod implicit pe o cale. Pentru fiecare dintre ele putem schimba calea, adăugând în instrucțiune directiva `#` *cale*. Tot ce urmează după parametrul de indicare a căii va fi vehiculat pe calea indicată. Încercarea de a vehicula date pe o cale inexistentă sau închisă se soldează cu mesajul O.

- `PRINT` este echivalent cu `PRINT #2` (`PRINT` lucrează în mod normal pe calea 2);
- `LPRINT` este echivalent cu `LPRINT #3` (sau cu `PRINT #3`);
- `INPUT` este implicit `INPUT #0`;
- `LIST` este `LIST #2`;

- `LLIST` este `LLIST #3`;
- `INKEY$` în forma ei de funcție BASIC, nu lucrează folosind căile! Există o instrucțiune care întâmplător are același nume, `INKEY$#`, care lucrează pe o cale specificată și citește un caracter de pe acea cale. `INKEY$#` nu merge pe căi asociate canalului K. Ea are sens numai după conectarea unor noi drivere și deschiderea canalelelor corespunzătoare. Un exemplu este lucrul cu discul, la care vom reveni.

Tentativa de a citi date de pe o cale asociată unui canal care nu are intrare sau de a scrie date pe un canal care nu are ieșire se soldează cu eroarea J.

■

A rămas nelămurită întrebarea: cum de este canalul K și canal de ieșire? Simplu: canalului tastaturii îi este asociată ca periferic de ieșire nimic altceva decât partea de jos a ecranului! Iată că începem să înțelegem de ce `INPUT` scrie acolo textele — pentru că folosește calea 0, asociată de obicei canalului K.

■

Putem face lucruri interesante, dirijând datele pe altă cale decât cea obișnuită.

Astfel `PRINT #0` va tipări în partea de jos a ecranului.

`INPUT #2` va tipări pe ecran (însă nu va putea cere valori pentru expresii pe ecran, căci canalul S este unidirecțional).

Încercați `PRINT #0; AT 1,0; "Linia 23": PAUSE 0`

Putem să facem un program să tipărească totul la imprimantă în loc să scrie pe ecran, fără a schimba o singură linie din program: pentru că `PRINT` e totuna cu `PRINT #2`, trebuie doar să mutăm calea 2 la alt canal. În cazul nostru la canalul K:

```
OPEN#2, "p"
RUN
```

E mai greu să schimbăm canalul asociat căii 1, pentru că fiecare `INPUT` face automat `OPEN #1, "k"`.

Folosind asamblare (cod-mașină) puteți scrie noi drivere pe care să le adăugați celor existente. Puteți crea noi drivere pentru ecran: care să gestioneze un sistem de ferestre, să scrie cu alte seturi de caractere ș.a.m.d. Programele voastre BASIC vor putea rămâne aceleași, indiferent de canalele care vor fi folosite: doar o instrucțiune `OPEN# 2, xx` va face toate `PRINT`-urile să lucreze cu noile canale `xx`. Iată cât de flexibil este acest sistem!

Notă: mai există încă trei căi, care însă nu-i sunt accesibile BASIC-ului; ele sunt folosite de interpretorul BASIC pentru treburile-i interne:

- calea 253, asociată canalului `"K"`: folosită la editarea (scrierea) programelor și la afișarea mesajelor de eroare;
- calea 254, asociată canalului `"S"`: folosită pentru a genera listingurile automate (un listing automat este acela care apare în timp ce scrieți un program după ce introduceți o linie);
- calea 255, care folosește canalul `"R"` (WORKSPACE) — un canal care nu e accesibil BASIC-ului. Este folosită pentru transferul datelor între zone de memorie.

Variabila sistem STRMS (23568) reține datele despre canalul asociat fiecărei căi. În zona de memorie care începe la (CHANS) se află datele despre fiecare canal (numele și adresele driverelor care asigură intrarea și ieșirea pe acel canal; dacă una din ele nu este suportată de acel canal, acolo se găsește adresa unei rutine de eroare).

18.7 Discul

Acest periferic odată introdus extinde BASIC-ul cu noi instrucțiuni.

Driverele de disc sunt programe foarte deștepte, care fac din disc un suport de memorie extern foarte atractiv. Ele apar odată cu montarea interfeței de disc.

Pentru că instrucțiunile de disc nu fac parte din BASIC-ul standard, le vom prezenta pe scurt:

Extensii ale comenzilor pentru casetofon

- `LOAD *` *specificator*;
- `SAVE *` *specificator*;
- `MERGE *` *specificator*;
- `VERIFY *` *specificator*;

unde *specificator* are forma:

aparat; *nr_aparat*; *nume_fişier*

- *aparat* este `"d"` pentru disc și `"m"` pentru *microdrive* (unitate automată de bandă) (un periferic nu prea răspândit);
- *nr_aparat* este numărul unității (0–2 pentru disc, 0–7 pentru microdrive);
- *nume_fişier* este ca la instrucțiunile BASIC de lucru cu casetofonul (cu `CODE` etc.).

Comenzi noi

- `CAT` *nr* produce o listă a fișierelor de pe unitatea de disc *nr*;
- `ERASE` *specificator* șterge fișierul specificat;
- `FORMAT` *specificator* reinițializează discul (îl șterge complet);
- `MOVE` *specificator1* `TO` *specificator2* copiază fișierul dat de prima specificație într-un fișier cu a doua specificație.

Comenzi de căi pentru disc

- `OPEN #` *cale*, *specificator* asociază acea cale unui fișier; atunci:
  - `PRINT #` *cale*, `LIST #` *cale* vor scrie în acel fișier;
  - `INPUT #` *cale*; *var* va citi valoarea variabilei din fișier;
  - `INKEY$#` *cale* va citi un caracter din fișier.

Comenzi „ad-hoc”

- `MOVE` $cale_1$ `TO` $cale_2$ — toate informațiile trimise pe prima cale vor ajunge pe a doua;
- `CLEAR#` închide toate căile deschise;
- `CLS#` șterge ecranul și aduce culorile la cele normale.

18.8 Exerciţii

(Cam puține și cam subțirele.)

1. Scoateți la imprimantă graficul funcției sinus.
2. Experimentați lucrul cu canalele (nu are rezolvare în carte).

Appendix A

Erorile

Cod	Text	Traducere
0	OK	totul e în regulă
1	NEXT without FOR	NEXT fără FOR
2	Variable not found	variabila nu a fost găsită
3	Subscript wrong	indice (de matrice) greşit
4	Out of memory	s-a terminat memoria
5	Out of screen	în afara ecranului
6	Number too big	număr prea mare
7	RETURN without GOSUB	RETURN fără GOSUB
8	End of file	sfârşit de fişier
9	STOP statement	instrucţiune STOP
A	Invalid argument	argument nepotrivit
B	Integer out of range	număr întreg în afara limitelor (admise)
C	Nonsense in BASIC	nu are sens în BASIC (eroare de sintaxă)
D	BREAK - CONT repeats	BREAK, CONTINUE repetă (instrucţiunea)
E	Out of DATA	s-a terminat lista DATA
F	Invalid file name	nume incorect de fişier
G	No room for line	nu mai e spaţiu pentru linia (nouă)
H	STOP in INPUT	STOP la INPUT
I	FOR without NEXT	FOR fără NEXT
J	Invalid I/O device	dispozitiv de intrare/ieşire incorect
K	Invalid colour	culoare incorectă
L	BREAK into program	program întrerupt cu BREAK
M	RAMTOP no good	valoare greşită pentru RAMTOP
N	Statement lost	instrucţiunea s-a pierdut
O	Invalid stream	calea aleasă este greşită
P	FN without DEF	FN fără DEF (FN
Q	Parameter error	parametri eronaţi
R	Tape loading error	eroare de citire de pe bandă

Modul de afişare al erorilor este `cod MESAJ, xxxx:yyyy`, unde

- xxxx este eticheta liniei unde a survenit eroarea;
- yyyy este numărul comenzii în linie, la a cărei execuţie a survenit eroarea.

Appendix B

Setul de caractere HC

Dec	Hex	Caracter	Z80		
				Cu prefix CB	Cu prefix ED
0	00	nefolosit	NOP	RLC B	
1	01	nefolosit	LD BC,nn	RLC C	
2	02	nefolosit	LD (BC),A	RLC D	
3	03	nefolosit	INC BC	RLC E	
4	04	nefolosit	INC B	RLC H	
5	05	nefolosit	DEC B	RLC L	
6	06	PRINT virgulă	LD B,n	RLC (HL)	
7	07	EDIT	RLCA	RLC A	
8	08	⇐	EX AF,AF'	RRC B	
9	09	⇒	ADD HL,BC	RRC C	
10	0A	⇓	LD A,(BC)	RRC D	
11	0B	⇑	DEC BC	RRC E	
12	0C	DELETE	INC C	RRC H	
13	0D	ENTER	DEC C	RRC L	
14	0E	urm. număr	LD C,n	RRC (HL)	
15	0F	nefolosit	RRCA	RRC A	
16	10	Ctrl. INK	DJNZ dis	RL B	
17	11	Ctrl. PAPER	LD DE,nn	RL C	
18	12	Ctrl. FLASH	LD (DE),A	RL D	
19	13	Ctrl. BRIGHT	INC DE	RL E	
20	14	Ctrl. INVERSE	INC D	RL H	
21	15	Ctrl. OVER	DEC D	RL L	
22	16	Ctrl. AT	LD D.n	RL (HL)	
23	17	Ctrl. TAB	RLA	RL A	
24	18	nefolosit	JR dis	RR B	
25	19	nefolosit	ADD HL,DE	RR C	
26	1A	nefolosit	LD A,(DE)	RR D	
27	1B	nefolosit	DEC DE	RR E	
28	1C	nefolosit	INC E	RR H	
29	1D	nefolosit	DEC E	RR L	
30	1E	nefolosit	LD E,n	RR (HL)	
31	1F	nefolosit	RRA	RR A	

Dec	Hex	Caracter	Z80		
				Cu prefix CB	Cu prefix ED
32	20	blanc	JR NZ,dis	SLA B	
33	21	!	LD HL,nn	SLA C	
34	22	"	LD (nn),HL	SLA D	
35	23	#	INC HL	SLA E	
36	24	$	INC H	SLA H	
37	25	%	DEC H	SLA L	
38	26	&	LD H,n	SLA (HL)	
39	27	'	DAA	SLA A	
40	28	(	JR Z,dis	SRA B	
41	29	)	ADD HL,HL	SRA C	
42	2A	*	LD HL,(nn)	SRA D	
43	2B	+	DEC HL	SRA E	
44	2C	,	INC L	SRA H	
45	2D	-	DEC L	SRA L	
46	2E	.	LD L,n	SRA (HL)	
47	2F	/	CPL	SRA A	
48	30	0	JR NC,dis	sls B	
49	31	1	LD SP,(nn)	sls C	
50	32	2	LD (nn),A	sls D	
51	33	3	INC SP	sls E	
52	34	4	INC (HL)	sls H	
53	35	5	DEC (HL)	sls L	
54	36	6	LD (HL),n	sls (HL)	
55	37	7	SCF	sls A	
56	38	8	JR C,dis	SRL B	
57	39	9	ADD HL,SP	SRL C	
58	3A	:	LD A,(nn)	SRL D	
59	3B	;	DEC SP	SRL E	
60	3C	<	INC A	SRL H	
61	3D	=	DEC A	SRL L	
62	3E	>	LD A,n	SRL (HL)	
63	3F	?	CCF	SRL A	
64	40	@	LD B,B	BIT 0,B	IN B,(C)
65	41	A	LD B,C	BIT 0,C	OUT (C),B
66	42	B	LD B,D	BIT 0,D	SBC HL,BC
67	43	C	LD B,E	BIT 0,E	LD (nn),BC
68	44	D	LD B,H	BIT 0,H	NEG
69	45	E	LD B,L	BIT 0,L	RETN
70	46	F	LD B,(HL)	BIT 0,(HL)	IM 0
71	47	G	LD B,A	BIT 0,A	LD I,A
72	48	H	LD C,B	BIT 1,B	IN C,(C)
73	49	I	LD C,C	BIT 1,C	OUT (C),C
74	4A	J	LD C,D	BIT 1,D	ADC HL,BC
75	4B	K	LD C,E	BIT 1,E	LD BC,(nn)
76	4C	L	LD C,H	BIT 1,H	
77	4D	M	LD C,L	BIT 1,L	RETI
78	4E	N	LD C,(HL)	BIT 1,(HL)	
79	4F	O	LD C,A	BIT 1,A	LD R,A
80	50	P	LD D,B	BIT 2,B	IN D,(C)

Dec	Hex	Caracter	Z80		
				Cu prefix CB	Cu prefix ED
81	51	Q	LD D,C	BIT 2,C	OUT (C),D
82	52	R	LD D,D	BIT 2,D	SBC HL,DE
83	53	S	LD D,E	BIT 2,E	LD (nn),DE
84	54	T	LD D,H	BIT 2,H	
85	55	U	LD D,L	BIT 2,L	
86	56	V	LD D,(HL)	BIT 2,(HL)	IM 1
87	57	W	LD D,A	BIT 2,A	LD A,I
88	58	X	LD E,B	BIT 3,B	IN E,(C)
89	59	Y	LD E,C	BIT 3,C	OUT (C),E
90	5A	Z	LD E,D	BIT 3,D	ADC HL,DE
91	5B	[	LD E,E	BIT 3,E	LD DE,(nn)
92	5C	\	LD E,H	BIT 3,H	
93	5D	]	LD E,L	BIT 3,L	
94	5E	^	LD E,(HL)	BIT 3,(HL)	IM 2
95	5F	_	LD E,A	BIT 3,A	LD A,R
96	60	*$*	LD H,B	BIT 4,B	IN H,(C)
97	61	a	LD H,C	BIT 4,C	OUT (C),H
98	62	b	LD H,D	BIT 4,D	SBC HL,HL
99	63	c	LD H,E	BIT 4,E	LD (nn),HL
100	64	d	LD H,H	BIT 4,H	
101	65	e	LD H,L	BIT 4,L	
102	66	f	LD H,(HL)	BIT 4,(HL)	
103	67	g	LD H,A	BIT 4,A	RRD
104	68	h	LD L,B	BIT 5,B	IN L,(C)
105	69	i	LD L,C	BIT 5,C	OUT (C),L
106	6A	j	LD L,D	BIT 5,D	ADC HL,HL
107	6B	k	LD L,E	BIT 5,E	LD HL,(nn)
108	6C	l	LD L,H	BIT 5,H	
109	6D	m	LD L,L	BIT 5,L	
110	6E	n	LD L,(HL)	BIT 5,(HL)	
111	6F	o	LD L,A	BIT 5,A	RLD
112	70	p	LD (HL),B	BIT 6,B	IN F,(C)
113	71	q	LD (HL),C	BIT 6,C	
114	72	r	LD (HL),D	BIT 6,D	
115	73	s	LD (HL),E	BIT 6,E	SBC HL,SP
116	74	t	LD (HL),H	BIT 6,H	LD (nn),SP
117	75	u	LD (HL),L	BIT 6,L	
118	76	v	HALT	BIT 6,(HL)	
119	77	w	LD (HL),A	BIT 6,A	
120	78	x	LD A,B	BIT 7,B	
121	79	y	LD A,C	BIT 7,C	IN A,(C)
122	7A	z	LD A,D	BIT 7,D	OUT (C),A
123	7B	{	LD A,E	BIT 7,E	ADC HL,SP
124	7C	—	LD A,H	BIT 7,H	LD SP,(nn)
125	7D	}	LD A,L	BIT 7,L	
126	7E	~	LD A,(HL)	BIT 7,(HL)	
127	7F	(C)	LD A,A	BIT 7,A	

Dec	Hex	Caracter	Z80		
				Cu prefix CB	Cu prefix ED
128	80	semigrafic 0	ADD A,B	RES 0,B	
129	81	semigrafic 1	ADD A,C	RES 0,C	
130	82	semigrafic 2	ADD A,D	RES 0,D	
131	83	semigrafic 3	ADD A,E	RES 0,E	
132	84	semigrafic 4	ADD A,H	RES 0,H	
133	85	semigrafic 5	ADD A,L	RES 0,L	
134	86	semigrafic 6	ADD A,(HL)	RES 0,(HL)	
135	87	semigrafic 7	ADD A,D	RES 0,A	
136	88	semigrafic 8	ADC A,B	RES 1,B	
137	89	semigrafic 9	ADC A,C	RES 1,C	
138	8A	semigrafic 10	ADC A,D	RES 1,D	
139	8B	semigrafic 11	ADC A,E	RES 1,E	
140	8C	semigrafic 12	ADC A,H	RES 1,H	
141	8D	semigrafic 13	ADC A,L	RES 1,L	
142	8E	semigrafic 14	ADC A,(HL)	RES 1,(HL)	
143	8F	semigrafic 15	ADC A,A	RES 1,A	
144	90	UDG a	SUB B	RES 2,B	
145	91	UDG b	SUB C	RES 2,C	
146	92	UDG c	SUB D	RES 2,D	
147	93	UDG d	SUB E	RES 2,E	
148	94	UDG e	SUB H	RES 2,H	
149	95	UDG f	SUB L	RES 2,L	
150	96	UDG g	SUB (HL)	RES 2,(HL)	
151	97	UDG h	SUB A	RES 2,A	
152	98	UDG i	SBC A,B	RES 3,B	
153	99	UDG j	SBC A,C	RES 3,C	
154	9A	UDG k	SBC A,D	RES 3,D	
155	9B	UDG l	SBC A,E	RES 3,E	
156	9C	UDG m	SBC A,H	RES 3,H	
157	9D	UDG n	SBC A,L	RES 3,L	
158	9E	UDG o	SBC A,(HL)	RES 3,(HL)	
159	9F	UDG p	SBC A,A	RES 3,A	
160	A0	UDG q	AND B	RES 4,B	LDI
161	A1	UDG r	AND C	RES 4,C	CPI
162	A2	UDG s	AND D	RES 4,D	INI
163	A3	UDG t	AND E	RES 4,E	OUTI
164	A4	UDG u	AND H	RES 4,H	
165	A5	RND	AND L	RES 4,L	
166	A6	INKEY$	AND (HL)	RES 4,(HL)	
167	A7	PI	AND A	RES 4,A	
168	A8	FN	XOR B	RES 5,B	LDD
169	A9	POINT	XOR C	RES 5,C	CPD
170	AA	SCREEN$	XOR D	RES 5,D	IND
171	AB	ATTR	XOR E	RES 5,E	OUTD
172	AC	AT	XOR H	RES 5,H	
173	AD	TAB	XOR L	RES 5,L	
174	AE	VAL$	XOR (HL)	RES 5,(HL)	
175	AF	CODE	XOR A	RES 5,A	

Dec	Hex	Caracter	Z80		
				Cu prefix CB	Cu prefix ED
176	B0	VAL	OR B	RES 6,B	LDIR
177	B1	LEN	OR C	RES 6,C	CPIR
178	B2	SIN	OR D	RES 6,D	INIR
179	B3	COS	OR E	RES 6,E	OTIR
180	B4	TAN	OR H	RES 6,H	
181	B5	ASN	OR L	RES 6,L	
182	B6	ACS	OR (HL)	RES 6,(HL)	
183	B7	ATN	OR A	RES 6,A	
184	B8	LN	CP B	RES 7,B	LDDR
185	B9	EXP	CP C	RES 7,C	CPDR
186	BA	INT	CP D	RES 7,D	INDR
187	BB	SQR	CP E	RES 7,E	OTDR
188	BC	SGN	CP H	RES 7,H	
189	BD	ABS	CP L	RES 7,L	
190	BE	PEEK	CP (HL)	RES 7,(HL)	
191	BF	IN	CP A	RES 7,A	
192	C0	USR	RET NZ	SET 0,B	
193	C1	STR$	POP BC	SET 0,C	
194	C2	CHR$	JP NZ,nn	SET 0,D	
195	C3	NDT	JP nn	SET 0,E	
196	C4	BIN	CALL NZ,nn	SET 0,H	
197	C5	OR	PUSH BC	SET 0,L	
198	C6	AND	ADD A,n	SET 0,(HL)	
199	C7	<=	RST 0	SET 0,A	
200	C8	>=	RET Z	SET 1,B	
201	C9	<>	RET	SET 1,C	
202	CA	LINE	JP Z,nn	SET 1,D	
203	CB	THEN	.prefix.	SET 1,E	
204	CC	TO	CALL Z,nn	SET 1,H	
205	CD	STEP	CALL nn	SET 1,L	
206	CE	DEF FN	ADC A,n	SET 1,(HL)	
207	CF	CAT	RST 8	SET 1,A	
208	D0	FORMAT	RET NC	SET 2,B	
209	D1	MOVE	POP DE	SET 2,C	
210	D2	ERASE	JP NC,nn	SET 2,D	
211	D3	OPEN#	OUT (n),A	SET 2,E	
212	D4	CLOSE#	CALL NC,nn	SET 2,H	
213	D5	MERGE	PUSH DE	SET 2,L	
214	D6	VERIFY	SUB n	SET 2,(HL)	
215	D7	BEEP	RST 16	SET 2,A	
216	D8	CIRCLE	RET c	SET 3,B	
217	D9	INK	EXX	SET 3,C	
218	DA	PAPER	JP C,nn	SET 3,D	
219	DB	FLASH	IN A,(n)	SET 3,E	

Dec	Hex	Caracter	Z80		
				Cu prefix CB	Cu prefix ED
220	DC	BRIGHT	CALL C,nn	SET 3,H	
221	DD	INVERSE	.prefix.	SET 3,L	
222	DE	OVER	SBC A,n	SET 3,(HL)	
223	DF	OUT	RST 24	SET 3,A	
224	E0	LPRINT	RET PO	SET 4,B	
225	E1	LLIST	POP HL	SET 4,C	
226	E2	STOP	JP PO,nn	SET 4,D	
227	E3	READ	EX (SP),HL	SET 4,E	
228	E4	DATA	CALL PO,nn	SET 4,H	
229	E5	RESTORE	PUSH HL	SET 4,L	
230	E6	NEW	AND n	SET 4,(HL)	
231	E7	BORDER	RST 32	SET 4,A	
232	E8	CONTINUE	RET PE	SET 5,B	
233	E9	DIM	JP (HL)	SET 5,C	
234	EA	REM	JP PE,nn	SET 5,D	
235	EB	FOR	EX DE,HL	SET 5,E	
236	EC	GOTO	CALL PE,nn	SET 5,H	
237	ED	GOSUB	.prefix.	SET 5,L	
238	EE	INPUT	XOR n	SET 5,(HL)	
239	EF	LOAD	RST 40	SET 5,A	
240	F0	LIST	RET P	SET 6,B	
241	F1	LET	POP AF	SET 6,C	
242	F2	PAUSE	JP P,nn	SET 6,D	
243	F3	NEXT	DI	SET 6,E	
244	F4	POKE	CALL P,nn	SET 6,H	
245	F5	PRINT	PUSH AF	SET 6,L	
246	F6	PLOT	OR n	SET 6,(HL)	
247	F7	RUN	RST 48	SET 6,A	
248	F8	SAVE	RET M	SET 7,B	
249	F9	RANDOMIZE	LD SP,HL	SET 7,C	
250	FA	IF	JP M,nn	SET 7,D	
251	FB	CLS	E1	SET 7,E	
252	FC	DRAW	CALL M,nn	SET 7,H	
253	FD	CLEAR	.prefix IY.	SET 7,L	
254	FE	RETURN	CP n	SET 7,(HL)	
255	FF	COPY	RST 56	SET 7,A	

Funcţiile care se comportă la fel sunt grupate împreună în setul de caractere:

Codul	Caracteristici	Codul	Caracteristici
165-167	fără parametri	192	număr sau şir $\rightarrow$ număr
168	definibilă	193-194	număr $\rightarrow$ şir
169-171	de două variabile, prefixate	195	număr $\rightarrow$ număr, prioritate 4
172-173	folosite numai cu PRINT şi INPUT	196	cu parametru opţional, constant
174	şir $\rightarrow$ şir	197-201	funcţii binare infixate
175-177	şir $\rightarrow$ număr	202-205	părţi opţionale de comenzi
178-191	număr $\rightarrow$ număr, prioritate 11	206-255	comenzi

Appendix C

Harta memoriei

Adresa	Conţinut
00000	ROM Interpretorul BASIC
16384	memoria video (pixelii)
22528	atribute
23296	buffer-ul imprimantei
23552	variabilele sistemului BASIC
23734	informaţii despre disc/microdrive
CHANS	informaţii despre canale
	128 (80h)
PROG	programul BASIC
VARS	variabile BASIC
	128 (80h)
E_LINE	comanda sau linia în curs de interpretare
	13 (0Ch)
	128 (80h)
WORKSP	date pentru INPUT
	13 (0Ch)
	spaţiu de lucru temporar
STKBOT	stiva evaluatorului de expresii BASIC
STKEND	liber
sp	stiva codului maşină (Z80)
	stiva GOSUB
	3E h
RAMTOP	?
UDG	caractere grafice definibile
P_RAMT	?
65536	prima adresă în afara memoriei

Appendix D

Codurile culorilor

La HC culorile fundamentale sunt codificate după cum urmează:

Cod	Culoare	Engleză
0	negru	black
1	albastru	blue
2	roşu	red
3	mov	magenta
4	verde	green
5	albastru deschis	cyan
6	galben	yellow
7	alb	white
8	contrast	
9	transparent	

- `BORDER` poate avea una din valorile 0-7.
- `PAPER` şi `INK` au una din valorile 0–9. Valorile 8 şi 9 au următoarele semnificaţii:
  - tipărind un caracter cu `INK 8` (`PAPER 8`), el va păstra drept culoare a cernelii (hârtiei) chiar culoarea existentă anterior tipăririi sale în acel loc pe ecran. Iată un exemplu:

```
25 PRINT AT 0,0;
30 FOR i = 0 TO 31
35  PRINT INK i - 4*INT(i/4) ; "=";
40 NEXT I
45 PAUSE 0
50 PRINT AT 0,0; INK 8;"REMARCATI CERNEALA CARACTERELOR"
```

  `PLOT`, `DRAW` şi `CIRCLE` sunt implicit `PAPER 8` (adică trasarea unui punct într-un pătrăţel nu va modifica PAPER-ul acestuia dacă nu se specifică explicit). Exemplu:

```
55 PRINT AT 0,0; PAPER 1, : PAUSE 0
65 PAPER 2: PLOT 0,715 : PAUSE 0
75 PLOT PAPER 2, 0, 175
```

– tipărind un caracter cu `INK 9` (`PAPER 9`) el va apărea fie alb, fie negru, astfel încât să iasă cât mai bine în evidenţă pe fondul (cerneala) pe care se scrie. Din următorul exemplu deducem că vom avea alb pentru fond 0-3 şi negru pentru fond 4-7:

```
22 FOR i=0 TO 7: PRINT PAPER i; INK 9 ; "**";: NEXT i
```

- `FLASH` poate fi 0 (standard), 1 (clipitor) sau 8 (se păstrează FLASH-ul din locul respectiv de pe ecran). `PLOT`, `DRAW` şi `CIRCLE` sunt implicit `FLASH 8`.
- `BRIGHT` poate fi 0 (standard), 1 (strălucitor; pe HC85 `BRIGHT 1` nu are nici un efect vizual!) sau 8 (transparent).
- `INVERSE` şi `OVER` pot fi 0 sau 1 (Capitolul 10).

Atragem atenţia asupra diferenţelor dintre instrucţiunile de culoare:

- `BORDER` e întotdeauna comandă (nu are sens `PRINT BORDER 2;`);
- `INK, PAPER, BRIGHT, FLASH` sunt atribute, care caracterizează un pătrat de pe ecran;
- `OVER` şi `INVERSE` sunt moduri de scriere/trasare; ele nu sunt caracteristicile imaginii; de aceea nu are sens `OVER 8` sau `INVERSE 8`.

Să mai remarcăm aşezarea culorilor. Televizorul formează culorile folosind numai 3 *culori fundamentale*: ALBASTRU, ROŞU şi VERDE. Celelalte culori se obţin din combinaţii ale acestora. Codurile culorilor la HC scrise în baza 2 (vezi şi Anexa E) au pe bitul 0 componenţa în albastru, pe bitul 1 componenţa în roşu şi pe bitul 2 componenţa în verde. Astfel:

Culoare	**Cod**	**Binar**	**Componenente**
negru	0	000	—
albastru	1	001	albastru
roşu	2	010	roşu
mov	3	011	albastru, roşu
verde	4	100	verde
bleu	5	101	albastru, verde
galben	6	110	roşu, verde
alb	7	111	albastru, roşu, verde

Appendix E

Baze de numeraţie

Pentru a exprima valoarea unui număr am preluat de la arabi sistemul de scriere poziţională în baza 10 (cu 10 cifre). Fie un număr în baza 10 de forma $A_n \ldots A_2 A_1 A_0$ unde A_i sunt cifrele sale. Atunci valoarea lui se mai poate scrie $A_n \times 10^n + \ldots + A_2 \times 10^2 + A_1 \times 10^1 + A_0 \times 10^0$.

În mod analog, putem exprima valoarea unui număr într-o *bază de numeraţie* arbitrară (bazele sunt numere întregi nenule), folosind un număr corespunzător de cifre distincte. De pildă, baza 2. Numerele sunt scrise numai cu 0 şi 1 (numite *cifre binare*). Valoarea se calculează ca în formula de mai sus, doar că punem 2 în loc de 10. Vom calcula efectiv valoarea unui număr în baza 2, exprimând-o în baza 10.

```
10110010 = 1 * 2^7 +
           0 * 2^6 +
           1 * 2^5 +
           1 * 2^4 +
           0 * 2^3 +
           0 * 2^2 +
           1 * 2^1 +
           0 * 2^0
          ----------
= 128 + 0 + 32 + 16 + 0 + 0 + 2 + 0 = 178.
```

Pentru baze mai mari ca 10 avem un deficit de cifre. De aceea construim baza 16 cu 6 noi cifre ad-hoc, simbolizate de literele A-F. Avem corespondenţele:

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

Observăm că, odată cu creşterea bazei, numerele îşi micşorează lungimea. Cu toate acestea, în informatică, cea mai folosită bază de numeraţie este 2. Aceasta pentru că e, evident, mai simplu

de construit un circuit care are doar două poziţii posibile (de pildă aprins-stins) decât unul care are mai multe. Asociind dispozitivelor electronice *bistabile* (cu două stări) cifrele binare, putem construi cu uşurinţă dispozitive pentru operaţii cu numere binare. O cifră binară se numeşte în engleză *bit* (*BInary digiT*; *binary* = binar, în baza 2; *digit* = cifră (nu deget!)).

Folosind n biţi putem exprima numere întregi pozitive în intervalul $0,\ 2^n - 1$.

Să descriem algoritmul de transformare din baza 10 în baza 2 şi algoritmii de calcul în baza 2.

Iată, fără justificări matematice, cum se procedează pentru a transforma un număr din baza 10 în baza 2. Fie N numărul de transformat. Împărţim N la 2, obţinând câtul M şi restul R. R este 0 sau 1 şi este chiar ultima cifră a numărului în baza 2. Apoi împărţim M la 2 şi obţinem un nou rest, care este penultima cifră a rezultatului. Continuăm până obţinem câtul 0. Resturile obţinute, citite de la coadă spre cap, formează chiar rezultatul căutat. Exemplu:

178 în baza 2 =?

178	: 2 =	89	rest 0
89	: 2 =	44	rest 1
44	: 2 =	22	rest 0
22	: 2 =	11	rest 0
11	: 2 =	5	rest 1
5	: 2 =	2	rest 1
2	: 2 =	1	rest 0
1	: 2 =	0	rest 1

Numărul este 10110010 (citind resturile de jos în sus).

Iată acum tablele adunării şi înmulţirii în baza 2:

0	+	0	=	0
0	+	1	=	1
1	+	0	=	1
1	+	1	=	10

0	*	0	=	0
0	*	1	=	0
1	*	0	=	0
1	*	1	=	1

Să adunăm 1011010 cu 10111; să înmulţim 1001 cu 110:

```
1011010+
  10111
-------
      1
    10
    1
   1
 10
 0
1
-------
1110001
```

```
1001*
 110
----
   0
 1001
1001
-------
110110
```

Să dublezi un număr în baza doi este echivalent cu a-i adăuga un zero la coadă. Restul împărţirii unui număr binar la doi este ultima lui cifră.

Appendix F

Reprezentări interne

Locaţiile de memorie le calculatorului sunt colecţii de câte 8 bistabile. Pe un octet putem reprezenta în memorie numere cu valori cuprinse între 0 şi 255 ($2^8 - 1$). Pentru a reprezenta valori ale diverselor tipuri de date, variabile şi programe BASIC se folosesc diferite metode. Le vom explora pe rând.

Variabilele sistem a căror valoare este o adresă (un număr pe 16 biţi), se memorează astfel: reprezentarea foloseşte doi octeţi consecutivi; cel care are adresa mai mică reţine (surprinzător) partea mai puţin semnificativă. (Zic surprinzător pentru că noi scriem întâi cifrele mai semnificative.) Partea mai puţin semnificativă se numeşte *Less Significant Byte*, şi o vom prescurta LSB. Partea mai semnificativă se numeşte în engleză *Most Significant Byte* — MSB.

De exemplu, pentru a afla adresa de la care începe programul BASIC trebuie să citim valoarea variabilei sistem PROG, situată la adresa 23635 (vezi şi Anexa H). Atunci putem tipări adresa programului BASIC astfel:

```
PRINT PEEK 23635 + 256*PEEK 23636
```

Pentru că o vom folosi de mai multe ori, construim o funcţie PEEK extinsă, care citeşte doi octeţi consecutivi.

```
  5 DEF FN p(a)=PEEK a + 256*PEEK (a+1)
```

Iată un program pentru exemplu (ataşaţi linia de mai sus):

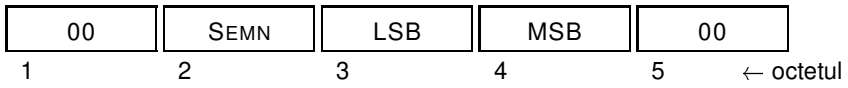
```
 15 PRINT "ADRESA BASIC-ULUI ";FN P(23635)
 25 PRINT "ADRESA VARIABILELOR ";FN p(23627)
 35 PRINT "RAMTOP ";FN p(23730)
 45 PRINT "SFARSITUL RAMULUI ";FN p(23732)
 55 PRINT "Timpul de cand merge BASICul (sec) ";
 65 PRINT (PEEK 23672 + 256*FN p(23673)) / 50
```

Ultima linie afişează valoarea ceasului de timp real (vezi Anexa H), care se calculează astfel: `PEEK 23672 + 256*PEEK 23673 + 65536*PEEK 23674` — o valoare memorată pe 3 octeţi!

Există două metode pentru a reprezenta valorile datelor de tip numeric. Ambele folosesc 5 octeţi.

Numere întregi în intervalul –65535, +65535:

Aceste numere sunt numite *mici întregi* (*small integers*). Cei cinci octeţi au următoarele valori:



- Semn este:
  - 00 pentru numere pozitive;
  - 255 (FF hexazecimal) pentru numere negative.
- Fie N numărul de reprezentat. Definim K astfel:
  - K = N dacă N $\geq$ 0;
  - K = 65536 + N dacă N $<$ 0.
- LSB este cel mai puţin semnificativ octet al lui K.
- MSB este cel mai semnificativ octet al lui K.

Exemple de reprezentări:

Număr	**Cei cinci octeţi**					**Observaţii**
0	0	0	0	0	0	0 e pozitiv (semn=0)
1	0	0	1	0	0	
23650	0	0	98	92	0	98 + 256 $\times$ 92 = 23650
65535	0	0	255	255	0	255 + 256 $\times$ 255 = 65535
–1	0	255	255	255	0	65536 – 1 = 65535
–23650	0	255	158	163	0	158+ 256 $\times$ 163 = 65536 – 23650

Numere întregi înafara intervalului de mai sus şi numere neîntregi:

Algoritmul de conversie este următorul (acest mod de reprezentare se mai numeşte şi în *virgulă flotantă – Floating Point*, abreviat FP):

1. numărul se transformă în baza 2; fie N valoarea sa;
2. N se separă în două valori, astfel:
   - semnul S (0 pentru pozitiv, 1 pentru negativ);
   - modulul M;
3. M este adus la forma $M = K * 2^E$ unde K este cuprins între 0.5 şi 1. (Această operaţie se numeşte *normalizare*. Ea nu face altceva decât să pună virgula înainte de prima cifră semnificativă nenulă a numărului.)
   - K se numeşte *mantisă*;
   - E se numeşte *exponent*;
4. numărul K are întotdeauna primul bit 1, aşa că în locul lui putem reprezenta valoarea semnului S;
5. Cei cinci octeţi primesc următoarele valori:

6. Exponentul este reprezentat adăugându-i 128, pentru a putea reprezenta şi valori negative. Dacă E+128 nu încape într-un octet, numărul e prea mare pentru a putea fi reprezentat.
   - Când E $>$ 127 se obţine eroarea `6 Number too big.` (*overflow*).
   - Când E < -128 numărul devine 0 (*underflow*).

Să vedem nişte exemple:

- Numărul 0.25:
  1. în baza 2 : 0.25 = .0100
  2. semnul S = 0; modulul M = .01
  3. M = (0.1 în baza 2) * 2^{-1}, deci
     - K = .10000000 .... pe 32 de biţi
     - E = –1
  4. primul octet este E+128 = 127
  5. al doilea octet are primul bit S = 0 iar restul sunt biţii luaţi de la K
  6. Reprezentarea pe 5 octeţi este:

127	0	0	0	0

- Numărul 1000000 (un milion):
  1. N = 1111 0100 0010 0100 0000 (verificaţi)
  2. S = 0; M = N
  3. M = (.11110100 00100100 în baza 2)$\times 2^{20}$
  4. Reprezentarea este

148	116	36	0	0

  (BIN 01110100 = 116, BIN 00100100 = 36)
- Numărul –0.25:
  Paşii ca la calculul lui 1/4, doar semnul e S=1

127	128	0	0	0

- Numărul 0.1:
  1. N = .0001100110011001100.... (periodic!)
  2. S = 0, M = N
  3. E = –3, K = 11001100 11001100 11001100 11001100 (32 de cifre)
  4. reprezentarea:

125	76	204	204	204

  (avem BIN 01001100 = 76, BIN 11001100 = 204)

■

Să vedem cum sunt reprezentate intern liniile unui program BASIC.

MSB	LSB	LSB	MSB	...	13
ETICHETA		LUNGIMEA		TEXTUL	ENTER

Reprezentarea etichetei unei linii BASIC face excepţie de la regula de reprezentare pe doi octeţi a numerelor: întâi vine MSB.

Urmează, reprezentată pe doi octeţi, o valoare care arată lungimea restului liniei (de la următorul octet până la octetul final ENTER).

Textul liniei este o succesiune de coduri de caractere, în ordinea în care sunt plasate în textul liniei. Sunt necesare două precizări:

- în textul unei linii pot apărea şi caractere de control, cu parametrii lor. De exemplu, dacă, scriind o linie, am inserat un caracter INVERSE VIDEO (apăsând CS + 4) în locul corespunzător din memorie vom avea doi octeţi cu valorile 20, respectiv 1 (vezi şi Capitolul 12);
- când în textul unui program apare un număr (o constantă numerică), în linie se inserează şase octeţi, cu următoarele semnificaţii: un octet cu codul 14, care indică reprezentarea valorii unui număr şi apoi pe 5 octeţi chiar valoarea numărului cum am descris-o mai sus. Inserarea acestor informaţii se face în momentul în care cel care scrie linia apasă CR: dacă linia este corectă sintactic, i se inserează aceste valori şi apoi este mutată din zona liniei în curs de editare în zona programului BASIC. Valorile inserate măresc viteza BASIC-ului, pentru că în momentul execuţiei nu se mai ţine cont de caracterele care formează textul numărului, ci doar de forma convertită. Conversia nu mai trebuie făcută. (Unii şmecheri schimbă cu bună ştiinţă caracterele care formează numărul pentru a induce în eroare privitorii.) Când o linie este editată, toate reprezentările acestea sunt scoase automat.

În fine, ultimul caracter al liniei este întotdeauna CR (`CHR$ 13`).

Bazat pe aceste consideraţii, iată un program care permite explorarea zonei de memorie a programului BASIC:

```
 1 DEF FN p(a) = PEEK a + 256*PEEK (a+1)
 3 LET PROG = 23635 : LET VARS = 23627
 5 LET PROG AD = FN p(PROG) : LET VARS AD = FN p(VARS)
 7 LET nr = 0 : REM indicator de valoare FP
10 LET adr = PROG AD
15   PRINT "Linia "; 256*PEEK adr + PEEK (adr + 1);
17   PRINT " are lungimea ";FN p(adr + 2): LET adr = adr + 4
20     IF PEEK adr = 14 AND nr = 0 THEN LET nr = 6
25     PRINT adr; TAB 10; PEEK adr; TAB 20;
27     PRINT ((CHR$ PEEK adr) AND PEEK adr > 31) AND nr = 0;
28     PRINT ("*FP*") AND nr > 0
29     LET adr = adr + 1: IF nr > 0 THEN LET nr = nr - 1
30     IF PEEK adr = 13 AND nr = 0 THEN LET adr = adr + 1:
          GOTO 15 + 30*(adr = VARS AD)
40   GOTO 20
```

Explicaţii:

- `nr` este o variabilă a cărei valoare diferită de 0 arată că adresa pe care tocmai o explorăm conţine un octet din reprezentarea FP a unui număr. Când se întîlneşte un `CHR$ 14` (care nu face parte el însuşi dintr-o formă FP), atunci `nr` devine 6, semn că următorii 6 octeţi nu sunt coduri de caractere. `nr` scade apoi mereu spre 0.
- `adr` este adresa pe care tocmai o explorăm.
- Linia 25 scrie adresa curentă şi conţinutul ei. Linia 27 scrie caracterul care se găseşte acolo, dacă este printabil.
- Programul nu ţine cont de faptul că există caractere de control care pot avea parametri. Îmbunătăţiţi-l!
- Linia 30 trece la descrierea unei noi linii, dacă s-a întâlnit CR, sau termină execuţia, dacă s-a atins zona destinată variabilelor.

Executând programul anterior aţi, remarcat, probabil că a apărut o zonă care memorează un număr FP şi în linia 1, după `a`-ul dintre parantezele lui `DEF FN` (deşi acesta nu este un număr). Explicaţia este dată de modul de lucru al lui `DEF FN`: la momentul evaluării lui `FN` — de exemplu în linia

5, unde argumentul este 23635 — valoarea obţinută pentru parametrul simbolic `a` al funcţiei definite este pusă chiar în locul rezervat în linia cu `DEF FN` în cei şase octeţi.

Acest mod de lucru permite doar recurenţe de ordinul întâi pentru funcţii care lucrează cu numere şi nu permite deloc recurenţe pentru funcţii care lucrează cu şiruri (căci şirurile sunt indicate în cei cinci octeţi printr-un *pointer* la valoarea reală). (Un pointer la un obiect este adresa unde se găseşte acel obiect.)

O remarcă interesantă: pentru a memora etichetele unei linii BASIC avem la dispoziţie doi octeţi, dar valorile admise sunt doar între 0 şi 9999. Cu `POKE` putem forţa etichetele şi la valori non-standard. Editorul, însă, nu permite lucrul cu linii care capătă etichete ciudate.

Folosind asemenea linii le veţi da multă bătaie de cap celor care încearcă să vă „fure” programele. O protecţie foarte interesantă, care se obţine, este cea anti-`MERGE`. Transformând eticheta unei linii dinspre sfârşitul programului într-un număr foarte mare, `MERGE` va confunda această linie cu o variabilă şi rezultatul va fi dezastruos. Astfel de programe nu pot fi încărcate cu `MERGE`! (Şi, deci, nu pot fi încărcate fără a porni în execuţie.)

■

Toate formele de memorare ale variabilelor au ceva comun:

- Literele sunt transformate în litere mici;
- Primul octet are următoarea formă:

```
|7|6|5|4|3|2|1|0|  bit
 ---------------
| cod | litera  |
```

- Primii trei biţi dau tipul variabilei, astfel:

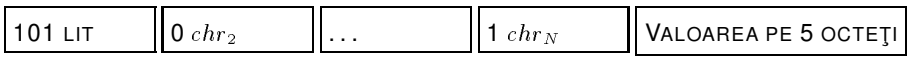
Cod	Cod binar	Variabila tip
2	010	şir
3	011	numerică, cu nume de o literă
4	100	matrice numerică
5	101	numerică, nume de mai multe caractere
6	110	matrice de şiruri
7	111	`FOR - NEXT`

- Următorii trei biţi sunt ultimii 5 biţi din codul literei cu care începe numele (căci orice nume de variabilă începe cu o literă!).

Iată cum se memorează fiecare tip:

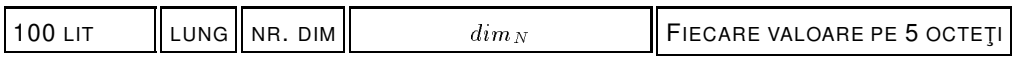
Variabilă numerică, nume de o literă:

Variabilă numerică, nume lung:



Ultimul caracter din nume are primul bit 1.

Matrice de numere:



- $lung$ e lungimea până la sfârşit, pe doi octeţi;

- $nr.dim$ e numărul de dimensiuni, pe un octet;
- dim_K sunt valorile fiecărei dimensiuni, pe doi octeţi.

Variabilă FOR – NEXT:

111 LIT	VAL. CURENTA	LIMITA	PAS	LINIA FOR	NR. INSTR.

- Linia `FOR` e eticheta liniei cu `FOR`-ul, pe doi octeţi;
- NR. instr. e numărul instrucţiunii din linie care e `FOR`-ul, pe un octet;
- cele trei valori sunt pe 5 octeţi.

Variabilă şir simplă:

010 LIT	LUNGIME (2 O)	CARACTERE (POT LIPSI) (1 OCTET FIECARE)

Variabilă şir matrice:

110 LIT	LUNG. 2O	NR. DIM.	dim_K	...	FIECARE CHR. 1O

- `nr dim` e numărul de dimensiuni reprezentat, pe 1 octet;
- `dimK` e valoarea dimensiunii K, pe doi octeţi.

Appendix G

Rezolvările exerciţiilor

G.1 Capitolul 1

1. Programul este următorul

```
10 PRINT AT 0,0;a;"  ";b;"  ";c;"  ";d;"  ";e;"  ";f
```

2. Nu pot fi nume de variabile
   b) variabilă şir cu nume prea lung
   c) caracterul „:”
   d) începe cu o cifră
   g) semnul „=”

G.2 Capitolul 2

1. Cam aşa:

```
10 PRINT AT 0,0; 0, 3^0, 1, 3^1, 2, 3^2, 3, 3^3, 4, 3^4
```

2. Greşelile sunt următoarele:
   - în linia 60, unei variabile numerice i se atribuie valoarea unei variabile şir;
   - în linia 70 se foloseşte operaţia „–” care nu este definită pentru şiruri;
   - în linia 80 putem aprecia drept greşeală folosirea variabilei `b`, care nu a fost definită (iniţializată, adică nu i s-a dat vreo valoare) anterior;
   - dacă variabila `LET` este pozitivă, atunci instrucţiunea din linia

     ```
     100 LIST -LET
     ```

     va da eroare.
     În linia 90 nu este nici eroare, pentru că avem de a face cu variabila scrisă cu semnele `L`, `E` şi `T`, iar nu cu comanda `LET`, care este un singur caracter.

G.3 Capitolul 3

1. Pentru a putea executa numai o instrucţiune din mai multe posibile (trebuie să existe instrucţiuni pentru scrierea numelui fiecărei cifre) este necesar să putem selecta cumva una din ele care să se execute. Acest lucru îl putem face numai cu `RUN`, calculând eticheta de destinaţie în funcţie de cifra de tipărit. Iată:

```
 5 INPUT "cifra ";cfr
10 RUN cfr + 20
20 PRINT "zero": STOP
21 PRINT "unu": STOP
22 PRINT "doi": STOP
23 PRINT "trei": STOP
24 PRINT "patru": STOP
25 PRINT "cinci": STOP
26 PRINT "sase": STOP
27 PRINT "sapte": STOP
28 PRINT "opt": STOP
29 PRINT "noua": STOP
```

   `STOP`-urile sunt necesare pentru a scrie un singur text. Dacă se doreşte executarea repetată a programului, `STOP`-urile trebuie schimbate cu:

```
INPUT "Tastati CR": RUN
```

   `INPUT "Tastati CR"` este necesar pentru că `RUN` şterge ecranul.

2. Se vor executa în ordine liniile:
   25 a=4
   35 a=80
   40
   80 a nu mai există (`RUN` şterge variabilele!)
   90 eroare 2 (a nu există)

G.4 Capitolul 4

1. Programul este următorul:

```
  5 LET a=10 : PRINT AT 0,0;
  7 LET a$ = "    "
220 PRINT PAPER 0;a$;PAPER 1;a$;PAPER 2;a$;PAPER 3;a$;PAPER 4;a$;
          PAPER 5;a$;PAPER 6;a$;PAPER 7;a$: LET a = a + 10 : GOTO a
```

   Iată cum funcţionează: la prima tură `a=10`. Apoi `a=20`, `GOTO a`, apoi `a=30`, `GOTO a` şi tot aşa, până ce `a=230`, când `GOTO a` opreşte programul — exact 22 de treceri, câte rânduri are ecranul.

2. Iată un exemplu :

```
 5 INPUT "culoarea ";ppr
10 INPUT "coordonata x=";x'"coordonata y=";y
20 PRINT AT y,x; PAPER ppr; " " : GOTO 5
```

G.5 Capitolul 5

1. Programul este destul de simplu. Să remarcăm că el nu memorează nicăieri numerele a căror sumă o face (nici nu putea, folosind numai instrucţiunile cunoscute). Variabila `S` este suma, iar `N` numărul de numere.

```
 5 INPUT "Cate numere ? ..."; N : LET s = 0
10 IF n > 30 THEN PRINT "prea multe." : GOTO 5
15 IF n < 2 THEN PRINT "Sa fim seriosi!" : GOTO 5
18
20 CLS : LET c = 1 : REM contor
30   PRINT AT 0,0;"Numarul nr. ";c;" =";
35   INPUT nr : PRINT nr
40   LET S = S + NR : LET C = C + 1
50 IF C <= N THEN GOTO 30
60 PRINT "Suma celor ";n;" numere este ";s
70 INPUT "O noua suma ? (da/nu) ";a$
80 IF a$ = "da" THEN RUN
```

2. Programul seamănă cu cel anterior:

```
 5 INPUT "Cate numere ? ..."; N
10 LET P = 0: LET M = 0: LET O = 0
20
30 LET c=1 : REM contor
40   INPUT "nr al ";(c);"-lea :";nr
50   IF nr < 0 THEN LET M = M + 1
55   IF nr = 0 THEN LET O = O + 1
60   IF nr > 0 THEN LET P = P + 1
70   LET c = c + 1
80 IF c <= N THEN GOTO 40
90 PRINT "sunt ";M;" numere negative"' TAB 5; O; "nule si "
100 PRINT TAB 5; P; " pozitive"
110   INPUT "o noua rulare ? (d/n) ";a$
120   IF a$ = "d" THEN RUN
130   IF a$ = "n" THEN STOP
140 GOTO 110
```

G.6 Capitolul 6

1. Să urmărim valorile tuturor variabilelor de-a lungul programului:

```
 5   nu sunt variabile
10   a=14
20   a=14      q=4
30   a=18      q=4
40   a=18      q=4   (REM este un comentariu!)
50   a=18      q=4       ak=15
60   a=20      q=4       ak=15
```

   Deci se tipăreşte 14.

2. Să calculăm întâi câte pătrăţele are ecranul: 32 $\times$ 24 = 768. Cum pe cele două rânduri de jos nu putem scrie în mod obişnuit, rămân 768 – 64 = 704.

   Ordinea culorilor ne sugerează un algoritm interesant: valorile 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2 etc. sunt de fapt resturile împărţirii la 8 a unui număr care creşte mereu. Vom folosi o singură

variabilă *contor*, care va indica prin valoarea sa câte pătrăţele s-au tipărit. Restul împărţirii valorii acestui contor la 8 va da chiar culoarea. Deci:

```
 5 PRINT AT 0,0;
 7 LET c = 0
10   PAPER c - 8*INT (c/8)
15   PRINT " "; : LET c = c + 1
20 IF c < 704 THEN GOTO 10
25 PAUSE 0
```

Ultima instrucţiune amână apariţia mesajului `0 OK` până după apăsarea unei taste. Am pus < şi nu <= în linia 20, pentru că începem de la 0.

3. Pentru că majoritatea pătrăţelelor nu au ambele coordonate pare, le vom colora folosind `CLS`. Vom folosi roşu (2) pe galben (6) pentru desen. Programul foloseşte două bucle, una-ntr-alta. Prima buclă trasează cele 11 linii, iar a doua trasează pătrăţelele fiecărei linii.

```
 5 BORDER 6 : PAPER 6 : CLS : PAPER 2
10 LET x = 0 : LET y = 0 : REM contoare
15     PRINT AT y, x;" "
20     LET x = x + 2
30   IF x < 31 THEN GOTO 15
35   LET x = 0 : REM incepem un nou rand
40   LET y = y + 2
50 IF y < 21 THEN GOTO 15
```

4. Iată noul program:

```
 1 REM   mira
 5 PRINT AT 0,0; :LET t = 0
10   PRINT PAPER t - 8 * INT(t/8);"    ";
20   LET t = t + 1
30 IF t < 32*22/4 THEN GOTO 10
```

Trebuie să remarcăm că nu am folosit transcrierea soluţiei prezentate la exerciţiul respectiv, ci am dat o rezolvare care se bazează pe cea a primului exerciţiu.

5. Pentru a memora ultimii doi termeni ai şirului se foloseşte metoda din exerciţiul rezolvat cu c.m.m.d.c., adică două variabile ce reprezintă ultimii doi termeni şi care îşi schimbă valoarea la calcularea unui nou termen astfel: a doua trece în prima, iar noul termen trece în a doua.

```
 1 REM    sirul lui Fibonacci
 5 INPUT "Cati termeni se afiseaza (4-100) ? ";t
10 LET t = INT ABS t : IF t < 4 THEN GOTO 5
11 IF t > 100 THEN GOTO 5
15 LET a = 1 : LET b = 1
20 PRINT "Primul termen este 1"
30 LET k = 2 : REM contor
40   PRINT "termenul ";k;" este ";c,
50   LET d = b : REM punem momentan la pastrare b
60   LET b = a + b : LET a = d
70   LET k = k + 1
80 IF k <= t THEN GOTO 40
```

6. O variantă de program este:

```
 1 REM   numere prime
 5 INPUT "nr de testat ";nr
10 LET nr = INT nr
```

```
 20 LET s = SQR nr : REM limita de testare
 30 IF s = INT s THEN GOTO 100 : REM patrat perfect
 40 LET k = 2 : REM contor
 50   LET b = nr / k
 60   IF b = INT b THEN GOTO 100
 70 IF k < s THEN LET k = k + 1 : GOTO 50
 90
 95 PRINT "Numarul este prim" : STOP
100 PRINT "Numarul nu este prim; se divide cu "; k : STOP
```

Scrieţi, folosind această metodă, un program care determină toţi divizorii unui număr.

7. Probleme pune numai „reflexia” bilei la atingerea marginii mesei. Să studiem un caz particular: presupunem că bila se mişcă pe verticală. Atunci coordonata ei verticală (y) va varia la fiecare pas cu 1, cea orizontală (x) rămânând neschimbată. Dacă bila urcă, la fiecare pas se scade 1 din y-ul ei. Dacă o ia în jos, atunci se adaugă 1. Când ajunge la margine, trebuie ca direcţia de mişcare să se schimbe, ceea ce este echivalent cu a schimba semnul „incrementului” (valorii care se adaugă). Lucrurile stau similar pentru mişcarea pe orizontală. Mişcarea pe oblică este suprapunerea a două mişcări independente, una pe verticală şi alta pe orizontală.

   De aici sugestia unui algoritm general de mişcare: folosim două variabile, `xi` şi `yi`, care reprezintă incrementţii pentru fiecare direcţie. Pentru direcţia 2 (stânga, sus) ei ar fi `xi=1` şi `yi=-1` (căci în sus y scade). Bila se mişcă până când atinge o margine (sau se opreşte). Atunci, incrementul corespunzător îşi schimbă semnul.

   Iată o propunere de program. Notăm cu `X`, `Y` coordonatele curente ale bilei, cu `pasi` numărul de paşi pe care l-a făcut bila (proporţional cu PAUSE), cu `xi`, `yi` incrementţii şi cu `xa`, `ya` coordonatele cu un pas mai înainte (coordonatele anterioare):

```
  5 REM    biliard
  7 BORDER 0 : PAPER 7 : INK 0 : CLS
 10 LET x = 11 : LET y = 16 : LET pa = 0
 15 PRINT AT y,x ; "O"
 18 INPUT "Directia (1-8)"; d : LET d = INT d
 19 IF D < 1 THEN GOTO 18
 20 IF D > 8 THEN GOTO 18
 25 GOTO D + 100
100 REM    se initializeaza incrementii
101 LET xi = 0 : LET yi = -1 : GOTO 120
102 LET xi = 1 : LET yi = -1 : GOTO 120
103 LET xi = 1 : LET yi = 0 : GOTO 120
104 LET xi = 1 : LET yi = 1 : GOTO 120
105 LET xi = 0 : LET yi = 1 : GOTO 120
106 LET xi = -1 : LET yi = 1 : GOTO 120
107 LET xi = -1 : LET yi = 0 : GOTO 120
108 LET xi = -1 : LET yi = -1
109
110 REM    pornire de la margine
120 IF x = 0 THEN LET xi = ABS xi
123 IF x = 31 THEN LET xi = -ABS xi
125 IF y = 0 THEN LET yi = ABS yi
128 IF y = 21 THEN LET yi = -ABS yi
130 REM    bucla principala
199
200   LET xa = x : LET ya = y : REM vechile coordonate
210   LET x = x + xi : LET y = y + yi : REM noile coordonate
220   PRINT AT y,x; "O" ;AT ya, xa; " "
230   LET pa = pa + 1
```

```
240   PAUSE INT (pa/2)
245
250   REM    test de reflexie
260   IF x = 0 THEN LET xi = - xi
270   IF x = 31 THEN LET xi = - xi
280   IF y = 0 THEN LET yi = - yi
290   IF y = 21 THEN LET yi = - yi
300 IF pa < 40 THEN GOTO 200
310 GOTO 18
```

G.7 Capitolul 7

1. Listingul programului este prezentat mai jos. Să subliniem punctele mai delicate ale rezolvării:

   - în linia 140 se face asupra numărului `a` următoarea operaţie: `LET a = INT ABS a` pentru a scurta testele asupra lui. `ABS` elimină semnul, `INT` îl aduce la o valoare întreagă, urmând a mai fi verificată doar apartenenţa la intervalul 1–3, ceea ce se face în liniile 140 şi 145;
   - în linia 160, acel `a$` de 320 de caractere este folosit pentru a şterge doar primele 10 linii ale ecranului;
   - în linia 245 este exemplificată o metodă interesantă de a opri execuţia programului până la apăsarea tastei CR;
   - cel mai subtil punct al programului este linia 260 — acel `DIM a$(32)`. Variabila `a$` este pregătită să preia de la utilizator o denumire de ocupaţie. Pentru a determina care dintre elementele matricei `c$` este egal cu `a$`, trebuie efectuată o comparaţie. Dar două şiruri nu pot fi egale dacă nu au aceeaşi lungime. Pe de altă parte toate elementele lui `c$` au lungimea fixată prin linia 110, la 32 de caractere. `a$` este forţat prin `DIM` să aibă şi el această lungime.
   - variabila `A` din linia 275 iniţializată cu 0 este un contor al numărului de persoane ce au ocupaţia căutată; acest contor e folosit numai în cazul in care are valoarea finală 0, pentru a indica printr-un mesaj că nu există nici o persoană care să corespundă cerinţelor (linia 292).

   De remarcat aşa numitul *meniu*, care oferă posibilitatea de a alege o opţiune din mai multe posibile (liniile 120-149).

```
100 BORDER 4 : INPUT "Cate persoane ? :";n
105 IF n < 1 THEN GOTO 100
106 LET n = INT n : IF n > 100 THEN GOTO 100
110 DIM n$(n, 15) : DIM o$(n, 32) : REM nume, ocupatii
113
115 PRINT AT 0,10; PAPER 3; n; " persoane"
120 PRINT ''TAB 5;"Selectati optiunea :"
130 PRINT '"1. Introduceti o persoana" ''
          "2. Listati persoanele si ocupatiile lor" ''
          "3. Listati persoanele cu o" ' "anumita ocupatie"
140 INPUT a : LET a = INT ABS a : IF a < 1 THEN GOTO 140
145 IF a > 3 THEN GOTO 140
148 GOTO 100 + 50*a
149
150 REM   introd. pers.
160 DIM a$(320) : PRINT AT 3,0; a$
```

```
165 PRINT AT 3,0;" Dati numarul la care doriti sa introduceti persoana";
170 INPUT p: IF p < 1 THEN GOTO 170
171 IF p > n THEN GOTO 170
175 PRINT p
180 PRINT "Numele :";
182 INPUT p$: IF p$ = "" THEN GOTO 182
185 PRINT p$: LET n$(p) = p$
188 PRINT "Ocupatia :";
190 INPUT p$: IF p$ = "" THEN GOTO 190
195 PRINT p$: LET o$(p) = p$
198 PRINT AT 21,0;"Apasati o tasta ...": PAUSE 0: CLS: GOTO 115
199
200 REM list. pers.
210 CLS : PRINT INK 9; TAB 6; PAPER 1; "Ocupatii si persoane" : LET p=1
220   PRINT TAB 5; INK 2; PAPER 6;"PERSOANA " ; p ; " SE NUMESTE" '
          PAPER 5; n$(p); TAB 0; PAPER 4; TAB 9;
          "SI ARE OCUPATIA "; TAB 0; PAPER 5; o$(p)
230   PRINT : PAUSE 50 : LET p = p + 1
240 IF p <= n THEN GOTO 220
245 INPUT "Apasati CR ";LINE a$: CLS : GOTO 115
249
250 REM list. pers. cu o oc.
260 CLS : DIM a$(32)
270 INPUT "Ocupatia :";a$
273 PRINT "PERSOANE CU OCUPATIA ";a$''
275 LET p = 1 : LET A = 0
280   IF o$(p) = a$ THEN PRINT p; ". "; n$(p) : LET a = a + 1
290   LET p = p + 1
291 IF p <= n THEN GOTO 280
292 IF A=0 THEN PRINT PAPER 0; INK 7;
          "Nu sunt persoane cu aceasta ocupatie !"
295 INPUT "Apasati CR ";LINE a$: CLS : GOTO 115
```

G.8 Capitolul 8

1. Programul pe care îl prezentăm constituie probabil cea mai rapidă şi eficientă metodă de a rezolva problema. Se foloseşte instrucţiunea `TAB` pentru tipărirea de blancuri. Ţinând cont că `TAB` scrie spaţii şi nu trece pur şi simplu la coloana respectivă, obţinem un mod rapid de a tipări blancuri, pentru că `TAB` se execută într-un timp scurt. Mai departe, logica programului nu e prea complicată: bucla cu contor `i` trece prin toate culorile, iar bucla `j` contorizează rândurile din care este format pătratul pentru fiecare culoare, cu două mai puţin decât pentru culoarea precedentă (de aici expresia `15 - 2*i`). Primul pătrat (cel negru) are latura de 15, următorul 13 etc. Laturile verticale ale pătratului de culoare `I` sunt la `7+I` şi `23-I`. Foarte importantă este specificarea `PAPER 8` cu care începe `PRINT`, pentru că altfel primul `TAB (i+7)` ar şterge pătratele desenate anterior. (Rulaţi programul şi fără acel `PAPER 8`). În fine, instrucţiunea `PRINT AT i+4, i;` fixează colţul din stânga sus al fiecărui pătrat.

```
 1 REM    patrate concentrice
10 FOR i = 0 TO 7
15 PRINT AT i+4, i;
20   FOR j = 0 TO 15 - 2*i
30     PRINT PAPER 8; TAB i+7; PAPER i; TAB 23 - i
35   NEXT j
40 NEXT i
```

2. Să urmărim valorile variabilelor pe diferitele linii:

10	a=oarecare
20-40	(în buclă) a=5, a=12, a=19, a=26
50	w=5, a=26
60-80	bucla nu se parcurge : z=12. w=5, a=26
90	w(1)=w(2)=w(3)=0
100	w=5
110	întâi r=oarecare, apoi r=0 dacă iniţial era 0 sau .... r=-1 dacă iniţial r$\neq$0; apoi dacă r=0, atunci r = –1, ... altfel r = –1 concluzie : w=5, r=–1
120-130	w=–5

3. Iată soluţia (`n^p` este modul de `n` la puterea `p`, cu semnul plus când `p` e par şi cu semnul lui `n` altfel):

```
10 INPUT "Numarul de ridicat la putere ";n
20 INPUT "Puterea (intreaga) ";p
30 IF p <> INT p THEN GOTO 20
40 PRINT n;"^";p;"=";
50 IF p/2 = INT (p/2) THEN LET n = ABS n
60 PRINT SGN n * (ABS n) ^ p
```

De remarcat metoda de test a divizibilităţii: `A` se divide cu `B` numai dacă `A/B` e întreg, adică `A/B = INT (A/B)`.

G.9 Capitolul 9

1. Vom folosi pentru „monstru” litera `M`. Variabilele `x1` şi `y1` memorează poziţia anterioară a monstrului. Prezintă interes mai ales liniile 40-50, în care se relizează calculul aleator al noilor coordonate ale monstrului.

   Să urmărim procedeul: coordonata variază cu `INT (RND*3)-1`. `INT (RND*3)` poate fi 0, 1 sau 2. Scăzând 1 vom obţine –1, 0 sau 1. Cele două coordonate variază independent. În liniile 60-90 se aduce monstrul înapoi pe ecran, în caz că luat-o razna. Linia 95 realizează oprirea monstrului din periplu, la apăsarea unei taste. Variabila `i` este folosită pentru a reţine culoarea monstrului, care, având însuşiri cameleonice, şi-o schimbă la fiecare pas. Instrucţiunea `INK 9` de la început ne asigură că orătania va fi întotdeauna vizibilă. Remarcaţi ajustarea lui `i` în linia 98 pentru a-l cicla prin toate valorile posibile. Acestea fiind zise, iată şi programul:

```
 10 INK 9 : LET i=0 : PAUSE 5 : RANDOMIZE : LET x=10 : LET y=10
 20   PRINT PAPER i; AT y,x;"M"
 30   LET x1 = x : LET y1 = y
 40   LET x = x + INT (RND*3) - 1
 50   LET y = y + INT (RND*3) - 1
 60   IF x < 0 THEN LET x = 0
 70   IF x > 31 THEN LET x = 31
 80   IF y < 0 THEN LET y = 0
 90   IF x > 21 THEN LET x = 21
 95   IF INKEY$ <> "" THEN STOP
 98   LET i = i + 1 : IF i > 7 THEN LET i = 0
100 PRINT AT y1, x1;" ": GOTO 20
```

2. Programul este un mixaj al celui anterior cu exerciţiul rezolvat, la care se adaugă liniile 20-23 pentru testarea coincidenţei poziţiilor. Am complicat puţin rezolvarea faţă de cerinţe, cerând trecerea de 8 ori peste monstru, acesta schimbându-şi la fiecare trecere culoarea (variabila `d`) — pasămite pierde putere şi se înnegreşte de supărare. Avem şi timp, reprezentat de variabila `i`. `xj, yj` sunt coordonatele jucătorului, `xa, ya` coordonatele sale anterioare. Taste 5, 6, 7, 8.

```
 10 LET d=7 : LET xj = 0: LET yj = 0: INK 9 : LET i = 0 : PAUSE 5
 15 RANDOMIZE : LET x = 10 : LET y = 10
 20   IF x<>xj THEN GOTO 25
 23   IF y=yj THEN LET d = d - 1: IF d=0 THEN STOP
 25   PRINT PAPER d; AT y,x; "M"
 30   LET x1 = x: LET y1 = y
 40   LET x = x + INT (RND*3) - 1
 50   LET y = y + INT (RND*3) - 1
 60   IF x < 0 THEN LET x = 0
 70   IF x > 31 THEN LET x = 31
 80   IF y < 0 THEN LET y = 0
 90   IF x > 20 THEN LET x = 20
 98   LET i = i + 1: PRINT AT 21,0; "timp :"; i
 99   LET xa = xj: LET ya = yj
100   IF INKEY$="" THEN GOTO 150
105   IF INKEY$="8" THEN IF xj < 31 THEN LET xj = xj + 1
110   IF INKEY$="5" THEN IF xj > 0  THEN LET xj = xj - 1
120   IF INKEY$="6" THEN IF yj < 20 THEN LET yj = yj + 1
130   IF INKEY$="7" THEN IF yj > 0  THEN LET yj = yj - 1
140   PRINT AT ya, xa;" "
150   PRINT AT yj, xj; "$"
200   PRINT AT y1, x1;" "
210 GOTO 2
```

G.10 Capitolul 10

1. Programul va tipări numerele 1 şi 0. Iată de ce: `1e11-1e11 = 0` (au aceeaşi reprezentare în memorie), deci `1e11-1e11+1 = 1` apoi `1e11+1` se va aproxima tot prin `1e11`, căci nu există suficientă precizie pentru a memora ultima cifră a numărului `10000000001`. De aceea `1e11+1-1e11=0`!

   Adunarea nu este asociativă!

   (Acest lucru este adevărat la toate calculatoarele care lucrează numeric.)

2. Neavând nici o specificaţie de `PAPER, INK, BRIGHT, BORDER` sau `INVERSE`, aceste culori şi moduri de scriere vor rămâne aşa cum erau înainte de executarea programului. Înainte de a urmări desfăşurarea programului, să facem o observaţie care ne va folosi adesea: tipărind un spaţiu în `OVER 1`, imaginea rămâne neschimbată. Acest blanc poate însă cauza schimbarea culorilor! Putem foarte repede să schimbăm culorile unui desen sofisticat, fără a trebui să-l redesenăm. Iată o secvenţă care face fondul galben, lăsând cernelurile neschimbate:

   ```
   105 DIM a$(32*22)
   110 PAPER 6
   115 INK 8 : REM  transparenta
   120 PRINT OVER 1; AT 0,0; a$
   ```

   Şi acum, să urmărim programul nostru pe linii:

5 ecranul e „curat”;
6 listingul e pe ecran (începe fără `scroll?`);
7 cursorul lui `PRINT` se mută în origine; `OVER 1`;
10 deasupra listingului sunt scrise numerele de la 1 la 200 (fără `scroll?`);
20 cursorul la origine;
50 o nouă listare care o şterge pe cea veche; rămân numerele; cursor sus;
70 `FLASH 1`;
80 `a$` are 64 de spaţii (2 rânduri);
100 11*2=22 de rânduri de spaţii în `FLASH`;

Deci ecranul e în `FLASH`, cu cele 200 de numere scrise.

3. Iată rezolvarea, pe care o s-o comentăm pe-ndelete:

```
 1 REM tunulet umblaret
 5 OVER 0: INK 9: CLS
10 LET x=0 : LET xg=0 : LET yg=21
20   PRINT AT 21,x;" || "
25   IF INKEY$="" THEN GOTO 50
30   IF INKEY$="x" THEN IF x < 28 THEN LET x = x + 1
40   IF INKEY$="z" THEN IF x > 0 THEN LET x = x - 1
45   IF INKEY$=" " THEN IF yg = 21 THEN LET yg = 20: LET xg = x + 1
50   IF yg = 21 THEN GOTO 80
60    PRINT AT yg, xg;"\/"; AT yg+1, xg;"  "
70    LET yg = yg - 1: IF yg = -1 THEN LET yg = 21: PRINT AT 0,xg;"  "
80 GOTO 20
```

Întâi, variabilele folosite:

`x` = coordonata orizontală a tunului (cea verticală e mereu 21);

`xg` şi `yg` = coordonatele eventualului obuz (g de la glonţ);

Am desenat tunul din patru caractere: un spaţiu, tunul `||` şi încă un spaţiu. Această formă ne oferă un mare avantaj: după mutare, tunul nu mai trebuie şters din vechea poziţie! Aceasta pentru că, mutându-l întotdeauna cu cel mult un pătrăţel, spaţiile marginale vor fi tipărite peste locul unde se afla tunul anterior, ştergându-l! Există şi un dezavantaj: cele două spaţii împiedică tunul să ajungă la marginile ecranului (acest lucru se poate evita adăugând nişte teste suplimentare). Variabila `x` reprezintă coordonata spaţiului din stânga al tunului, putând varia între 0 şi 32–4=28.

O altă restricţie (nu foarte nenaturală) este că pe ecran nu se poate afla la un moment dat mai mult de un glonţ (altfel poziţiile tuturor proiectilelor ar trebui memorate într-un vector, iar la viteza BASIC-ului mişcarea lor nu ar fi prea cursivă). Glontele l-am construit din două caractere, numitele „*slash*” şi „*backslash*” (\). Când nu există nici un glonte pe ecran, considerăm că se află în tun, iar coordonata sa verticală este `yg=21`.

Un lucru foarte important, pe care nu trebuie sa-l scăpăm din vedere, este că glontele, odată plecat, se mişcă independent de tun.

Întâi iniţializăm variabilele: `x=0` (tunul pleacă din stânga), `xg=0` (iniţializare redundantă, adică nenecesară) şi `yg=21` (tun „încărcat”).

Linia 20 este prima linie a ciclului şi realizează tipărirea belicoasei instalaţii.

În linia 25 se verifică apăsarea vreunei taste; dacă nu sunt taste apăsate, se sare la linia 50, de unde începe partea care se ocupă cu mişcarea proiectilului.

Linia 30 mută tunul la dreapta, la apăsarea tastei x (dacă se mai poate).

Linia 40 îl mută la stânga.

În linia 45, la apasărea unui spaţiu (`INKEY$=" "`), dacă nu există glonte pe ecran (`yg=21`), se porneşte glontele (`LET yg=20`) memorând totodată şi coordonata de la care a fost tras (`xg=x+1`, din cauza spaţiului din stânga tunului).

În linia 50 programul închide bucla, dacă nu e nici un glonţ de mişcat.

Linia 60 desenează obuzul, ştergându-l din vechea sa poziţie (cu un rând mai jos). Interesant este că, dacă acesta tocmai a fost tras, se şterge nu vechea sa imagine, ci chiar tunul, ceea ce însă nu supără prea tare pentru că tunul va fi redesenat curând (un test suplimentar ar elimina şi această situaţie).

Linia 70 urcă glonţul (`LET yg = yg-1`) şi testează dacă nu cumva acesta a ieşit din „spaţiul de luptă” — ecranul (adică `yg` a devenit –1) — caz în care şterge ultima apariţie a glontelui şi încarcă tunul din nou.

În fine, linia 80 reia ciclul.

G.11 Capitolul 11

1. Calculând `a XOR b` vedem că este totuna cu `a OR b` cu excepţia cazului când `a=b=1`. Acest caz se exprimă prin `a AND b = 1`. Deci `a XOR b` se defineşte prin:

   ```
   (a OR b) AND (NOT (a AND b))
   ```

   Programul următor afişează şi tabelul de adevăr (valoarea funcţiei pentru toate combinaţiile posibile de valori argument).

   ```
    5 FOR a=0 TO 1
    7   FOR b=0 TO 1
   10     PRINT a;" XOR ";b;" =";
   20     LET c=(a OR b) AND (NOT (a AND b))
   30     PRINT c
   40   NEXT b
   50 NEXT a
   ```

2. Iată întâi rezolvarea, după care vom face unele precizări:

   ```
   10 LET a=0 : INPUT "Sirul in care se cauta :";a$
   15 PRINT PAPER 6; a$; PAPER 7;" cuprinde pe ";
   20 INPUT "Sirul care se cauta :"; b$
   25 PRINT PAPER 5; b$; PAPER 7;" ?"'''
   30 IF B$="" OR LEN B$>LEN A$ THEN STOP
   40 FOR I=1 TO LEN A$-LEN B$+1
   50   IF A$(i TO i+LEN b$-1) = b$ THEN LET a = a+1: PRINT PAPER 5; B$;:
             LET i = i + LEN B$ - 1: GOTO 60
   55   PRINT a$(i);
   60 NEXT i
   70 PRINT '''B$; PAPER 3;" apare de ";a;" ori."
   ```

   Corectitudinea acestui program este discutabilă, în funcţie de sensul pe care îl dăm cuvântului „cuprinde” din enunţ. Putem considera că şirul `"aba"` se cuprinde de două ori în şirul `"ababa"` , sau o singură dată fără să se „autointersecteze”. În forma scrisă mai sus, programul găseşte doar şiruri care nu se autointersectează, însă poate fi modificat cu uşurinţă pentru a găsi şi astfel de apariţii, înlocuind linia 50 cu:

   ```
   50 IF a$(i TO i+LEN b$-1) = b$ THEN LET a = a + 1
   ```

   Am dat programul în această formă pentru că am exemplificat „marcarea” şirurilor găsite, scriindu-le cu o altă culoare, lucru mai dificil în cazul autointersectării.

   Remarcaţi căutarea lui `b$` începând de la primul până la al `(LEN a$ - LEN b$ + 1)`-lea caracter, precum şi instrucţiunea de test cu `TO`. `a` este contorul apariţiilor.

3. Acest program introduce unele elemente interesante, cum ar fi alegerea tastelor de către jucător şi o metodă originală de animaţie folosind `scroll`. Iată, comentate, punctele delicate:

- cele două taste vor fi memorate într-o variabilă şir cu lungimea de două caractere (`a$`), căreia la început îi dăm valoarea arbitrară `"po"`.
- variabilele `S, a, b` reprezintă, respectiv, scorul (timpul de joc), coordonata din stânga şoselei şi coordonata maşinii, care, pentru a se afla pe şoseaua cu lăţime de 8 caractere, trebuie să satisfacă relaţia $b < a < b + 8$.
- Tasta „dreapta" este memorată în `a$(1)`. Lăsăm cititorului testarea alegerii unor taste incorecte (de pildă, aceeaşi tastă pentru ambele direcţii).
- O şmecherie care ne uşurează munca este afişarea scorului curent pe prima linie a ecranului, astfel încât `scroll`-ul să elimine necesitatea ştergerii vechii valori.
- Testul ieşirii din şosea este evident. Recunoaştem, de asemenea, metoda unduirii aleatoare a şoselei şi a variaţiei coordonatei maşinii.
- Interesant este faptul că nu trebuie testat dacă maşina a ieşit din ecran, pentru că înainte ea ar fi trebuit să iasă din şosea. Linia 65 produce o accelerare cu timpul. Cele două apostrofuri din linia 20 au menirea de a avansa ecranul.

```
 1 REM   raliu
 2 LET a$="po": INPUT "stanga ";b$: LET a$(2) = b$
 4 INPUT "dreapta ";b$: LET a$(1) = b$
10 LET s=0 : LET b=11 : LET a=15
20   PRINT AT 21,b;"#"; AT 21, b+8;"#"; AT 21,a;"V" '' AT 0,24;"SCOR ";S
25   POKE 23692, 255
30   IF a <= b OR a >= b+8 THEN PRINT AT 21,21; "CRASH !": STOP
40   LET b = b + INT (RND * 3) - 1
43   IF b < 0 THEN LET b = 0
47   IF b > 23 THEN LET b = 23
50   LET a = a + (INKEY$ = a$(1)) - (INKEY$ = a$(2))
65   FOR i = 1 TO (1500 - s)/300 : NEXT i
70   LET s = s+1
80 GOTO 20
```

G.12 Capitolul 12

1. Vom folosi un păianjen cu numai 6 picioare, şi alea scurte, reprezentat de caracterul `*`.

   Vom proceda altfel decât deobicei, pentru că nu vom memora poziţia veche pe care arahnoida a ocupat-o, ci deplasarea pe care o face pentru a ajunge în poziţia cea nouă. `a` şi `b` vor fi respectiv adaosurile la coordonatele `x` şi `y`. Acest mod de reprezentare ne permite să trasăm mai uşor segmentele pânzei. Pentru a asigura imposibilitatea ieşirii din ecran atunci când păianjenul este la margine, vom face ca la programul de biliard, forţând incrementii la valori cu un anume semn. Asta fac liniile 23-27.

   Pentru a nu şterge şi pânza când orătania se mişcă, o vom şterge şi desena folosind `OVER 1`.

   Interesantă este şi metoda de calcul a coordonatelor unui punct din grila de înaltă rezoluţie care corespunde unui pătrăţel de coordonate `x` şi `y` în cea de joasă rezoluţie. Pe orizontală vom avea `x*8 + 4`, pentru că fiecare pătrat are latura de 8, iar centrul e la 4 pixeli de latura din stânga. Pe verticală e un pic mai greu, pentru că originile nu corespund pe ecran. De aceea, `y` trebuie întâi inversat în `21-y` şi abia apoi aplicat acelaşi procedeu. Pânza este formată din segmente cu dimensiunile `8*a` şi respectiv `-8*b` (minus din cauza inversării lui `y`).

   Un ultim amănunt: pentru că pânza nu se trasează în `OVER 1`, ordinea operaţiilor este următoarea: se şterge păianjenul, se desenează pânza spre noua poziţie şi abia apoi se mută păianjenul. Cum totul merge repede, nu se vede că pânza o ia de fapt înainte!

```
 1 REM    spiders
10 LET x=10 : LET y=10 : PRINT AT y,x;"*"
```

```
20   LET a=INT (RND*3)-1
21   LET b=INT (RND*3)-1
23   IF x=0 THEN LET a = ABS a
24   IF x=31 THEN LET a = - ABS a
26   IF y=0 THEN LET b = ABS b
27   IF y=21 THEN LET b = - ABS b
29   PRINT OVER 1; AT y,x; "*"
30   PLOT x*8 + 4, (21-y)*8 + 4 : DRAW 8*a, -8*b
35   PRINT OVER 1; AT y+b, x+a; "*"
40   LET x = x + a: LET y = y + b
50 GOTO 20
```

2. Pentru a testa formula propusă facem şi o tablă de conversie grade-radiani:

```
10 PRINT AT 0,0;"grade","radiani"
20 FOR i=0 TO 90 : PRINT i, i/180 * 3.141592 : NEXT i
```

   Realizaţi şi conversia inversă — radiani-grade — din 0.05 în 0.05 radiani.

3. Pentru a face mai util programul, vom mări numărul de segmente al desenului până la 24. Vom proceda la rezolvare, astfel:

   - vom afla numărul de vârfuri ale desenului; îl punem în variabila `N` (liniile 10-30);
   - vom rezerva spaţiu pentru a memora celelalte `n-1` vârfuri (primul schimbându-şi coordonatele la mişcarea desenului) în doi vectori, `x(n-1)`, `y(n-1)` (liniile 40-45);
   - vom permite utilizatorului să fixeze vârful de pornire cu ajutorul a cinci taste (5, 6, 7, 8 pentru deplasări şi 0 pentru a-l fixa) (50-90);
   - cu aceleaşi taste vom afla şi coordonatele celorlalte vârfuri, memorându-le în vectorii corespunzători; vom trasa segmentele (95-180);
   - permitem mişcarea întregului desen cu tastele 5, 6, 7, 8 (200-240).

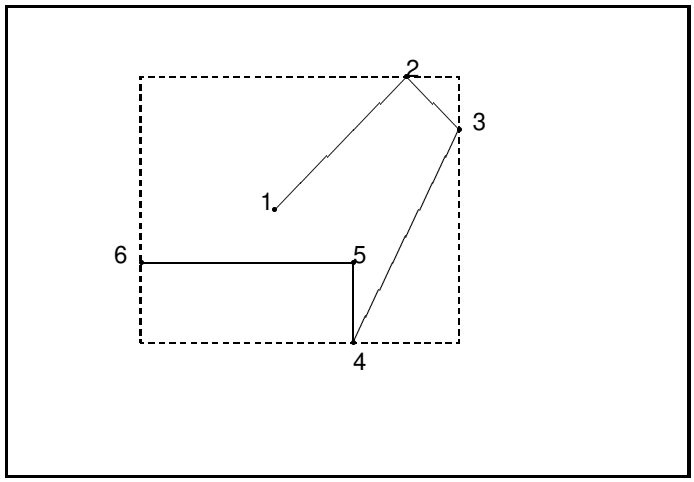
   Listingul:

```
10 REM  introducerea desenului
15 BORDER 5 : OVER 0
20 INPUT " Numar de varfuri :";n
30 LET n = INT n: IF n<2 OR n>25 THEN GOTO 20
40 DIM x(n-1) : DIM y(n-1)
45 PRINT TAB 10;n;" varfuri"
50 LET x=0 : LET y=0
55   PLOT x,y : PAUSE 0: PLOT INVERSE 1; x,y
60   LET x=x + (INKEY$="8" AND x<255) - (INKEY$="5" AND x>0)
70   LET y=y + (INKEY$="7" AND y<175) - (INKEY$="6" AND y>0)
80 IF INKEY$ <> "0" THEN GOTO 55
90 LET x0 = 0: LET y0 = 0: PLOT x0,y0
95 LET x1 = x: LET y1 = y
97 LET minx = x0 : LET MAXx = x0 : LET miny = y0 :LET MAXy = y0
100 FOR i = 1 TO n-1
105   PRINT AT 0,5; "varful curent :"; i+1; " din ";n
110       OVER 1: PLOT x,y : DRAW x1-x, y1-y : PAUSE 0:
                  PLOT x,y : DRAW x1-x, y1-y
120       LET x1=x1 + (INKEY$="8" AND x1 < 255) - (INKEY$="5" AND x1 > 0)
125       LET y1=y1 + (INKEY$="7" AND y1 < 175) - (INKEY$="6" AND y1 > 0)
130   IF INKEY$ <> "0" THEN GOTO 110
140   LET x(i) = x1 - x : LET y(i) = y1 - y
145   OVER 0: PLOT x,y : DRAW x1-x, y1-y
```

```
150   LET x=x1 : LET y=y1
155   IF x1 < minx THEN LET minx = x1
156   IF x1 > MAXx THEN LET MAXx = x1
157   IF y1 < miny THEN LET miny = y1
158   IF y1 > MAXy THEN LET MAXy = y1
160 NEXT i
170
180 OVER 1: LET x=x0 : LET y=y0
200 CLS : REM miscarea desenului
210   PLOT x,y : FOR i=1 TO n-1 : DRAW x(i), y(i) : NEXT i: PAUSE 0
215   PLOT x,y : FOR i=1 TO n-1 : DRAW x(i), y(i) : NEXT i
220   LET x=x + (INKEY$="8" AND x < 255 - MAXX + x0)
            - (INKEY$="5" AND x > x0 - minx)
230   LET y=y + (INKEY$="7" AND y < 175 - MAXY + y0)
            - (INKEY$="6" AND y1 > y0 - miny)
240 GOTO 210
```

O primă problemă care trebuie lămurită este modul de memorare a poziţiei vârfurilor — în coordonate absolute sau relative la vârful anterior? Am optat pentru a doua soluţie, pentru că `DRAW` lucrează în coordonate relative, deci scurtăm calculele de făcut la momentul trasării, care trebuie să fie cât mai rapidă (şi aşa mişcarea desenului va avea mai mult un caracter de redesenare).

Cel mai greu de rezolvat se dovedeşte chestiunea ieşirii desenului din ecran: având mai multe vârfuri, oricare din ele ar putea ieşi în cursul mişcării. Totuşi, am ieşit la liman destul de simplu. Să facem o schiţă a unui eventual desen, pentru a ne clarifica ideile (suntem în faza introducerii desenului):



- 1, 2, 3, 4, 5, 6 sunt vârfurile, în ordinea introducerii lor;
- cu linie punctată este figurat dreptunghiul minim de aceeaşi orientare cu ecranul în care încape tot desenul;
- cu linie continuă — desenul;
- cu linie continuă îngroşată — marginile ecranului grafic;
- `minx`, `MAXX`, `miny`, `MAXY` desemnează valorile acestor variabile din program.

Constatăm că un segment nu poate ieşi din ecran decât dacă cel puţin unul dintre vârfurile sale iese, pentru că ecranul este o figură convexă. Notăm cu `minx` cea mai mică dintre coordonatele `x` ale tuturor vârfurilor (pe desen e vârful 6), cu `MAXX` cea mai mare dintre coordonatele `x` ale tuturor vârfurilor (pe desen pentru 3); la fel `miny` şi `MAXY`. Fie acum coodonatele primului vârf, `x0` şi `y0`. Dacă în cursul mişcării (partea a doua a programului) acest vârf ajunge la coordonatele `x`,`y`, atunci coordonata orizontală a vârfului, care iniţial era la `minx`, va fi acum `minx+x-x0`. Condiţia ca acest vârf să nu iasă prin stânga (deci ca nici un vârf să nu iasă prin stânga, acesta fiind cel mai din stânga vârf) este `minx+x-x0 > 0` sau, cum apare în linia 220, `x > x0-minx`.

Analog, vârful care iniţial era la `MAXX` se va afla la `MAXX + x - x0`. Condiţia să nu iasă prin dreapta este `x < 255 - MAXX + x0`. Avem relaţii similare pentru `y`.

Ar mai fi de explicat prin program:

- metoda de a mişca primul punct: linia 55 îl desenează, aşteaptă apăsarea unei taste şi îl şterge, scriindu-l în `INVERSE 1`;
- metoda de a desena segmente „elastice” — linia 110;
- în linia 150 se ţin în `x` şi `y` coordonatele absolute punctului tocmai trasat, pentru a putea calcula poziţia relativă a următorului punct faţă de acesta. `x1` şi `y1` sunt coordonatele absolute ale vârfului curent;
- liniile 155-158 actualizează valorile coodonatelor extreme în cazul fixării unui nou punct;
- în fine, liniile 200-240 ajută la mutarea desenului, modificând coordonatele `x` şi `y` ale vârfului iniţial. Remarcaţi ştergerea şi trasarea în `OVER 1`.

4. Spunem în enunţ „opt taste”, pentru cele patru direcţii fundamentale şi cele patru diagonale. După cum am mai spus, mişcarea pe diagonală o vom concepe ca sumă a două mişcari pe axe.

   Pentru a ilustra şi alte tehnici am introdus şi posibilitatea schimbării modului de trasare: punctul lasă/şterge urme, fără a complica prin aceasta programul. Am ales sugestiv tastele în jurul lui [S], fiecare pentru direcţia corespunzătoare:

   [Q] [W] [E]
   [A] [D]
   [Z] [X] [C]

   Folosim butonul 0 pentru schimbarea de mod scrie/şterge.

   Am sacrificat prima linie a ecranului pentru a afişa permanent coordonatele punctului şi modul de trasare. (În Capitolul 18 vom învăţa să folosim linia 23 pentru acest scop.) Valoarea maximă pentru `y` devine 167.

   Pentru a scrie/şterge vom folosi o variabilă `M`, care va fi 0 pentru scris şi 1 pentru şters; vezi folosirea ei în linia 45.

   Am mai introdus următorul efect: linia 40 va face ca pătrăţelul în care se află punctul să clipească alternativ în alb şi galben, pentru a ajuta reperarea acestuia pe ecran. În această linie punctul doar îşi semnalează poziţia clipind; trasarea efectivă se face în 45.

   Remarcaţi şi schimbarea valorii lui `m` în linia 70. Schimbarea este însoţită de un efect coloristic pe `BORDER`.

   În fine, se vede cum grupe de câte trei taste folosesc la deplasarea punctului pe câte o direcţie. Tastele care corespund diagonalelor au acţiune pe mai multe direcţii simultan (liniile 50 şi 55). Explicaţi ce se întâmplă dacă punctul ajunge la o margine şi îl mişcăm în continuare în diagonală.

```
 1 REM    editor grafic
10 LET m = 0: LET x = 128: LET y = 87
```

```
20 BORDER 6: PRINT AT 0,0; PAPER 6; TAB 31;" "
30 PAPER 8
40   OVER 1: PLOT PAPER 6,x,y: PLOT PAPER 7,x,y: IF INKEY$="" THEN GOTO 40
45   PLOT OVER 0; INVERSE m; x,y
50   LET x = x + ((INKEY$="d" OR INKEY$="d" OR INKEY$="c") AND x<255)
             - ((INKEY$="q" OR INKEY$="a" OR INKEY$="z") AND x > 0)
55   LET y = y + ((INKEY$="q" OR INKEY$="w" OR INKEY$="e") AND y<167)
             - ((INKEY$="z" OR INKEY$="x" OR INKEY$="c") AND y > 0)
60   PRINT OVER 0; AT 0,0; "x=" ; x ; TAB 8; "y=" ; y ; TAB 18;
65   PRINT ("scrie" AND m=0) + ("sterge" AND m=1)
70   IF INKEY$="0" THEN LET m=NOT m: FOR i=1 TO 5: PAUSE 1: BORDER 5:
          BORDER 4: BORDER 3: BORDER 2: BORDER 1: BORDER 0: BORDER 6:
          NEXT i: GOTO 60
80 GOTO 40
```

G.13 Capitolul 13

1. Algoritmul de *fill*(= a umple, în engleză) este tipic pentru un algoritm recursiv. Iată esenţa sa: Pentru a aplica algoritmul într-un punct (a haşura conturul ce cuprinde acel punct) întâi colorăm acel punct, apoi testăm în ordine cei patru vecini ai săi (de pildă, întâi cel din dreapta, apoi cel de deasupra, apoi cel din stânga, apoi cel de dedesubt). Dacă punctele testate sunt „albe” le aplicăm acelaşi algoritm. Dacă am testat toţi vecinii, am terminat cu algoritmul pentru acest punct şi revenim la cel anterior. Pentru a aplica algoritmul unui vecin şi apoi a reveni la punctul anterior se foloseşte `GOSUB-RETURN`. Iată o subrutină care face acest lucru (o mare consumatoare de memorie şi timp):

   ```
   9000 IF POINT(x,y) = 1 THEN RETURN : REM punct aprins
   9005 PLOT x,y
   9010 IF x < 255 THEN LET x = x + 1: GOSUB 9000 : LET x = x - 1
   9020 IF y < 175 THEN LET y = y + 1: GOSUB 9000 : LET y = y - 1
   9030 IF x THEN LET x = x - 1: GOSUB 9000 : LET x = x + 1
   9040 IF y THEN LET y = y - 1: GOSUB 9000 : LET y = y + 1
   9050 RETURN
   ```

   Să-l încercăm:

   ```
    CIRCLE 128,87,15 : LET x = 128: LET y = 87: GOSUB 9000
   ```

   sau, mai complicat:

   ```
     5 PLOT 128, 87
    15 FOR i= 1 TO 25 STEP 4
    25   DRAW 2*i,0 : DRAW 0,2*i: DRAW -2*i-4,0: DRAW 0, -2*i-4
    35 NEXT i
    45 DRAW 3,3
    55 LET x=128 : LET y = 87: GOSUB 9000
  8999 STOP
   ```

2. Din nou, o problemă cu multă geometrie:

   ```
     1 REM   sferoid
   100 LET r = 86: LET u=PI/4: LET v=-PI/3: LET nm=6: LET np=6
   102 LET su=SIN u: LET cu=COS u: LET sv=SIN v: LET cv=COS v
   108
   109 REM  trasam meridianele
   ```

```
 110 FOR a=0 TO PI-.1 STEP PI/nm
 115  LET ca=COS a: LET sa=SIN a
 120  FOR i=-PI/2 TO PI/2 STEP .04: LET ci=COS i: LET si=SIN i
 125    LET x=x*ca*ci : LET y=r*(si*cu - su*ci*sa): GOSUB 9000
 160  NEXT i
 170 NEXT a
 199
 200 REM trasam paralelele
 290 FOR a=PI/np TO PI/2 STEP PI/np : LET sa=SIN a: LET ca=COS a
 300  FOR i=0 TO PI STEP .04: LET si=SIN i: LET ci=COS i
 310    LET k=r*sa*su*si
 320    LET x=r*sa*ci : LET y=r*ca*cu+k : GOSUB 9000 :
                        LET y=y-k-k : GOSUB 9000
 330  NEXT i
 340 NEXT a
 900 STOP
8999 REM rotatie in jurul Oz
9000 LET x1=x*cv - y*sv: LET y1=x*sv + y*cv:
     PLOT 128+x1, 88+y1: PLOT 128-x1, 88-y1 : RETURN
9100 PLOT 128+x, 88+y: PLOT 128-x, 88-y : RETURN
```

În acest program se ascund concepte extrem de complexe, cum ar fi cel al reprezentării grafice a figurilor tridimensionale. Este un subiect palpitant, dar din păcate prea amplu pentru a fi tratat aici. Exerciţiul acesta este accesibil mai curând celor care au cunoştinţe în domeniu din alte surse decât acest modest material. Explicaţiile pe care le vom da vor fi succinte. Ideea centrală este următoarea: dacă avem o figură în trei dimensiuni formată din puncte cu coordonatele $(x,\ y,\ z)$, atunci o metodă de a reprezenta aceste puncte este de a le proiecta simplu pe planul xOy, adică de a le reprezenta prin punctele cu coordonate $(x,\ y)$. Asta am şi făcut în cazul sferei, numai că am complicat puţin problema, rotind sfera. Poziţia sferei am descris-o prin poziţia axului ei (faţă de care ea este simetrică), iar poziţia axului am descris-o prin două unghiuri, `u` cu planul ecranului şi `v` cu planul vertical perpendicular pe ecran.

Alte variabile: `r`=raza în pixeli, `nm`=numărul de cercuri meridiane trasate, `np`=numărul de cercuri paralele (inclusiv punctele de la poli).

Buclele în `i` trasează de fiecare dată câte un cerc; întâi cercuri meridiane, apoi cercuri paralele. Buclele în `a` selectează care din aceste cercuri se trasează. Pe linii:

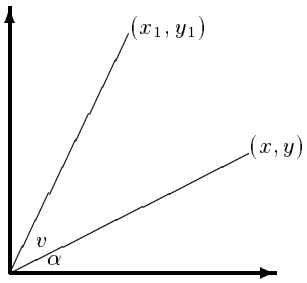
- 110 contorizează meridianele;
- 120 pregăteşte punctele fiecărui meridian;
- 125 calculează meridianele ca elipse rotite;
- 290 contorizează paralelele;
- 300 variază prin punctele fiecărei paralele;
- 320 trasează punctele fiecărei paralele şi a simetricei ei.

Toate aceste linii calculează punctele sferei ca şi cum ar fi rotite numai în jurul axei Ox. Procedura de la linia 9000 realizează şi rotirea în jurul axei Oz, rotind doar proiecţiile punctelor (cu alte cuvinte, realizează înclinarea axei).

9000 înclină axa şi trasează două puncte: cel calculat şi simetricul faţă de centrul sferei.

Procedura de la 9100 nu se execută niciodată; ea este identică celei de la 9000, cu excepţia faptului că nu mai înclină axa; introducând un `REM` la începutul liniei 9000, puteţi să rulaţi programul şi în această variantă.

Iată şi un desen care ar trebui să explice formulele de rotaţie în jurul unei axe:



Vrem să rotim punctul (x, y) în jurul originii (sau, dacă preferaţi, în jurul axei Oz) cu un unghi v. Trebuie să calculăm poziţia finală (x_1, y_1). Fie l lungimea segmentului din origine până la (x, y). Putem scrie relaţiile (evidente):

$$\begin{aligned} x &= l\cos\alpha \\ y &= l\sin\alpha \\ x_1 &= l\cos\alpha + v \\ y_1 &= l\sin\alpha + v \end{aligned}$$

Dezvoltăm $\cos(\alpha + v)$ şi $\sin(\alpha + v)$ ca la şcoală:

$$\begin{aligned} l\cos(\alpha + v) &= (l\cos\alpha)\cos v - (l\sin\alpha)\sin v \\ l\sin(\alpha + v) &= (l\sin\alpha)\cos v + (l\cos\alpha)\sin v \end{aligned}$$

Observăm că parantezele de mai sus din partea dreaptă reprezintă chiar valorile lui x şi y. Deci:

$$\begin{aligned} x_1 &= x\cos v - y\sin v \\ y_1 &= x\sin v + y\cos v \end{aligned}$$

Această formulă apare limpede în procedura de la linia 9000. Prin metode similare se pot deduce şi celelalte. Lăsăm ca exerciţiu (nerezolvat) demonstrarea lor.

3. Un program amuzant şi simplu:

```
10 LET x=100 : LET y=50
15 LET z=50: LET w=80
18 LET a=-2.5 : LET b=-2
19 LET c=-1 : LET d=2.5
20 FOR i=1 TO 300
22   PLOT x,y: DRAW z-x,w-y
25   IF x<5 OR x>250 THEN LET a=-a
30   IF z<5 OR z>250 THEN LET c=-c
35   IF y<5 OR y>170 THEN LET b=-b
40   IF w<5 OR w>250 THEN LET d=-d
50   LET x=x+a : LET y=y+b
60   LET z=z+c : LET w=w+d
70 NEXT i
80 CLS : GOTO 20
```

Capetele au coordonatele (x,y) şi (z,w), care iau iniţial valori arbitrare (liniile 10-15) dar nu prea aproape de margine. a, b, c, d sunt valorile cu care cresc x, y, z, w. Iau şi ele valori arbitrare (dar mai mici ca 5).

Linia 20 asigură trasarea a câte 300 de segmente.

Linia 22 le trasează.

Liniile 25-40 reflectă capetele când ajung la o distanţă mai mică de 5 faţă de margine.

În 50-60 se calculează următorul segment.

Linia 80 face puţină curăţenie şi trece la un nou desen.

G.14 Capitolul 14

1. Iată noul program:

```
 10 INPUT "f(x)="; LINE q$
 20 INPUT "f:["; xl; "," ; xh; "]->["; yl; ","; yh; "]"
 30 LET rx = 255/(xh-xl) : LET ox = -xl*rx
 40 LET ry = 175/(yh-yl) : LET oy = -yl*ry
 55 IF xl > 0 OR xh < 0 THEN GOTO 80
 60  PLOT ox, 0: DRAW 0,175
 62  IF ox >=2 AND ox <=253 THEN DRAW 2,-4: DRAW -4,0: DRAW 2,4
 65  IF ox < 2 THEN DRAW 2,-4 : DRAW -2,0
 68  IF ox > 253 THEN DRAW 2,4: DRAW 2,0
 80 IF yl > 0 OR yh < 0 THEN GOTO 100
 82  PLOT 0, oy: DRAW 255,0
 85  IF oy >=2 AND oy <=173 THEN DRAW -4,2: DRAW 0,-4: DRAW 4,2
 91  IF oy < 2 THEN DRAW -4,2 : DRAW 0,2
 95  IF oy > 17253 THEN DRAW -4,-2 : DRAW 0,-2
 99
100 FOR x = xl TO xh STEP 1/rx
105   LET y = VAL q$
110   LET xr = x*rx + ox
115   LET yr = y*ry + oy
120   IF yr >=0 AND yr <= 175 THEN PLOT xr,yr
130 NEXT x
```

Programul a ieşit ceva mai scurt decât cel din exerciţiul rezolvat, pentru că am eliminat testele care nu prezintă interes din punct de vedere al noutăţii şi ceva grafică (scrierea limitelor de trasare). Să-l comentăm pe acesta:

Pe linii:

10 Se cere expresia funcţiei.
20 Se defineşte suprafaţa pe care se trasează (coordonate reale).
30 Folosind regula de trei simplă, se determină coordonata (orizontală, în sistemul ecranului) a axei Oy. (La un interval de lungime 255 corespunde un interval de lungime `xh-xl`; `ox` se află la o distanţă egală cu `-xl` de marginea stângă a intervalului [`xl, xh`], deci la `-xl * 255 / (xh-xl)` de limita stângă a intervalului [0, 255].)
40 Ca mai sus, dar pentru axa Ox.
55 Dacă intervalul [`xl, xh`] nu include punctul 0, atunci axa Oy nu apare pe ecran şi se sare la linia 80.
60 Se trasează Oy.
62 Dacă Oy nu e prea aproape de marginile ecranului, atunci e loc pentru a trasa şi o săgeată în vârful ei.
65 Dacă Oy e prea în stânga, trasăm doar jumătatea din dreapta a săgeţii.
68 Când Oy e în dreapta, trasăm partea din stânga a săgeţii.
80-95 Echivalentul liniilor 55-68 pentru cealaltă axă.
100 Pentru punctele din intervalul [`xl, xh`] luate astfel încât să fie în număr de 256 (se vor trasa `(xh-xl)/STEP = (xh-xl)/rx = 255` la care se adaugă şi punctul din dreapta), vom trasa funcţia.
105 `y` este valoarea funcţiei în punctul `x`; *linia cheie a programului*.
110 Folosind din nou regula de trei simplă, calculăm unde pică `x` pe ecran (justificaţi formula obţinută).
115 Iată şi locul unde ar trebui să fie `y=f(x)`.
120 Dacă `yr` începe pe ecran (`xr` începe sigur), atunci trasăm un punct al graficului.
130 Evident.

Variabile folosite:

`xl,xh` limitele intervalului pe Ox;
`yl,yh` limitele intervalului pe Oy;
`q$` expresia funcţiei în BASIC;
`rx` o constantă pentru trecerea din sistemul de coordonate cu limitele [`xl, xh`] în cel cu limitele [0, 255]
`ry` pentru trecerea [`yl, yh`] $\rightarrow$ [0, 175]
`ox` coordonata originii pe axă, în pixeli; poate fi în afara ecranului (de pildă, pentru trasare în intervalul [2, 3] pe Ox);
`oy` similar cu `ox`;
`xr` coordonata de pe ecran care îi corespunde lui `x` în intervalul dat;
`yr` coordonata `y` în sistemul ecranului.

2. Programul acesta este foarte spectaculos şi simplu de realizat. Pentru a mări literele, vom folosi următorul artificiu: le vom scrie invizibil într-un colţ al ecranului şi apoi vom explora acea zonă cu `POINT`. Iată programul şi nişte comentarii:

```
  1 REM    titluri tridimensionale
  2 LET lat = 6: LET ob = lat/1.6
  3 PRINT INK 7; AT 21,0; "TEXT."
  5 FOR i = 0 TO 7
 10   FOR j=0 TO INT (256/lat) - 2
 15      IF POINT(j,i)=0 THEN GOTO 99
 21      PLOT j*lat, i*lat + 75
 22      DRAW lat,0 : DRAW 0,lat : DRAW -lat,0 : DRAW 0,-lat
 25      DRAW ob,ob : DRAW lat,0 : DRAW -ob,-ob: DRAW ob,ob
 27      DRAW 0,lat : DRAW -ob,-ob: DRAW ob,ob
 31      DRAW -lat,0: DRAW -ob,-ob : DRAW ob,ob
 35      DRAW 0, -lat
 99   NEXT j
100 NEXT i
```

Linia	Comentariu
2	`lat` e lungimea laturii în pixeli; încercaţi şi alte valori (recomandăm 4); `ob` e lungimea celor două proiecţii ale unei muchii oblice
3	Scriem textul de mărit
5	Literele au câte 8 rânduri de pixeli
9	Calculăm câte cubuleţe încap: fiecare are latura `lat` şi avem 256 de puncte la dispoziţie. 2 se scade pentru perspectivă
15	Dacă punctul e stins, nu facem nimic
21	Colţul stânga-jos-faţă al cubului. Coordonatele (`i,j`) sunt înmulţite cu `lat` (mărim de `lat` ori); 75 centrează desenul (pentru `lat` peste 13, 75 e prea mare)
22	Faţa cubului
25	Faţa de jos
27	Faţa din dreapta
31	Spatele
35	Ce-a mai rămas
99,100	Totul pentru fiecare punct

Dacă daţi valori neîntregi pentru `lat`, din cauza erorilor de rotunjire cuburile n-o să mai fie perfecte, sau perfect suprapuse. Modificaţi acest program, pentru a elimina muchiile ascunse (o idee, care merge cam lent, este de a şterge locul pe care se desenează fiecare cub înainte de a-l trasa).

G.15 Capitolul 15

1. Se folosesc variabilele sistem VARS (23627) şi E_LINE (23641).

```
1 PRINT "adresa variabilelor :"; PEEK 23627 + 256*PEEK 23628
2 PRINT "ultimul lor octet :";  PEEK 23641 + 256* PEEK 23642
3 PRINT "lungimea :"; PEEK 23641-PEEK 23627 +
   256*(PEEK 23642-PEEK 23628) - 1
```

Pentru a lista numele tuturor variabilelor, programul este mult mai elaborat. Trebuie să ţinem cont că în timpul listării lor, spaţiul variabilelor îşi poate modifica forma (datorită creării de noi variabile sau modificării valorii lor). Trebuie cunoscut intim mecanismul atribuirilor. Anexa F descrie forma de memorare a variabilelor. Iată un model de program:

```
10 LET f=9000
15 LET x=23627 : GOSUB f : LET ad=x
17 LET x=23641 : GOSUB f : LET b=x-2
19 CLS: PRINT PAPER 5;"Variabile de la ";ad;" la ";b'
             "adr --- var --- tip --- lungime";PAPER 0; TAB 31;" "
20   LET a=INT ((PEEK ad)/32) : PRINT ad;"  "; CHR$(PEEK ad - 32*a + 96):
     PAUSE 0: POKE 23692, -1
21    IF a=3 THEN PRINT ,"num";TAB 27;5 : LET ad = ad + 6
22    IF a=4 THEN PRINT TAB 14;"mat num"; TAB 26;: LET x=ad+1: GOSUB f:
                  PRINT x: LET ad = ad + x + 3
23    IF a=7 THEN PRINT TAB 14;"var FOR";TAB 27;18 : LET ad=ad + 19
24    IF a=2 THEN PRINT ,"sir"; TAB 27;: LET x=ad+1: GOSUB f:
                  PRINT x: LET ad = ad + x + 3
25    IF a=6 THEN PRINT TAB 14;"mat sir"; TAB 26;: LET x=ad+1: GOSUB f:
                  PRINT x: LET ad = ad + x + 3
26    IF a<>5 THEN GOTO 30
27     LET a=0 : REM numaram literele din nume
```

```
 28     LET ad=ad+1 : PRINT CHR$(PEEK ad - 128*INT (PEEK ad/128));:
        LET a=a+1: IF INT (PEEK ad/128)=0 THEN GOTO 28
 29     PRINT ,"num"; TAB 27; a+5: LET ad=ad+6
 30 IF ad <= b THEN GOTO 20
 31 PRINT 'PAPER 1; TAB 31; TAB 31;" ": PAUSE 0 : STOP
9000 LET x=PEEK x + 256*PEEK (x+1) : RETURN
```

Subrutina de la 9000 citeşte conţinutul a două locaţii consecutive. `a` este tipul variabilei, iar `ad` adresa variabilei curente.

2. Iată pentru început soluţia, care este mult mai complicată decât ar fi lăsat enunţul să se înţeleagă:

```
   1 REM    contoare
   2 LET RAMTOP=65535 - 128 : CLEAR RAMTOP
   5 LET a=0 : LET b=0
  10 LET adr = PEEK 23606 + 256*PEEK 23607
  20 LET adr = adr + 8*CODE "0"
  30 FOR i=1 TO 8*10 : POKE RAMTOP+i, PEEK(adr+i-1) : NEXT i
  35 FOR i=1 TO 8 : POKE RAMTOP+80+i, PEEK (adr+i-1) : NEXT i
  40 PRINT AT 10,11;"COUNTER";TAB 12;"-----";TAB 12;"|000|"; TAB 12;"-----"
  50 LET adr = RAMTOP - 8*CODE "0" + 1
  60 POKE 23606, adr - 256*INT(adr/256) : POKE 23607, INT (adr/256)
  70   FOR i=0 TO 72: BEEP .001,5: POKE 23606,i: PRINT AT 12,15;0: NEXT i
  75   IF INKEY$="s" THEN GOTO 9999
  80   FOR i=1 TO 8 : BEEP .001,5 : POKE 23606,I :
            PRINT AT 12,14;A;9 : NEXT i : LET a=a+1
  90   IF a < 8 THEN GOTO 70
  94   FOR i=0 TO 72: BEEP .001,5: POKE 23606,i: PRINT AT 12,15;0: NEXT i
 100   FOR i=1 TO 8: BEEP .001,5 : POKE 23606,i:
            PRINT AT 12,13;B;99 : NEXT i : LET b=b+1 : LET a=0
 110   IF b < 9 THEN GOTO 70
 120 LET a=0 : LET b=0 : GOTO 70
9999 POKE 23606,0 : POKE 23607,60
```

Programul are 4 părţi principale:

(a) formarea unui nou set de caractere, care cuprinde doar cifrele (liniile 1-30) şi pregătirea ecranului (40);

(b) rotirea ultimei cifre a contorului (70);

(c) rotirea ultimelor două cifre ale contorului (80-90);

(d) rotirea tuturor celor trei cifre ale contorului (94-120).

Programul este o buclă infinită, care se părăseşte la apăsarea (prelungită) a unei taste.

Am ales în aşa fel adresa noului set, încât cel mai puţin semnificativ octet al adresei să fie în mod normal 0. Cifrei 0 îi corespunde adresa 65535 – 127. Am mutat acolo imaginile cifrelor 0-9 şi apoi din nou pe cea a cifrei 0!

Pentru a încetini mişcarea contorului am folosit un `BEEP` extrem de scurt (inauzibil): `BEEP .001,5`.

Variabilele `a` şi `b` reprezintă cifra din mijloc, respectiv cea din stânga a contorului.

Linia 70 face să defileze cifrele unităţilor între 0 şi 9.

Linia 80 roteşte cifra zecilor şi cea a unităţilor.

Linia 94 roteşte cifra unităţilor cât timp cifra zecilor este 9.

Linia 100 roteşte toate cele trei cifre simultan.

G.16 Capitolul 16

1. Tehnica folosită pentru a aduce imaginile caracterelor din ROM seamănă cu cea de la Exerciţiul 2, Capitolul 15. Litera â am pus-o pe tasta b (în mod G). Liniile 10-60 mută caracterele a, i, s, t, a în UDG-urile a, i, s, t, b. Liniile 70–100 construiesc semnele diacritice; linia 110 arată ce-a ieşit — introduceţi semnele în modul G al cursorului.

```
 10 FOR j=0 TO 4: READ a$
 15  LET adr = PEEK 23606 + 256*PEEK 23607 + 8*CODE a$
 20  FOR i=0 TO 7
 30    POKE USR a$ + 8*(j=4) + i, PEEK (adr+i)
 40  NEXT i
 50 NEXT j
 60 DATA "a","i","s","t","a"
 70 POKE USR "a", BIN 101000 : POKE USR "a"+1, BIN 10000
 80 POKE USR "i", BIN 10000 : POKE USR "i"+1, BIN 101000
 90 POKE USR "s"+7, BIN 110000 : POKE USR "t"+7, BIN 110000
100 POKE USR "b"+1, BIN 101000 : POKE USR "b", BIN 10000
110 PRINT "a i s t b"
```

2. Programul este conversaţional, în sensul că cere de la utilizator toate datele. A ieşit lunguieţ, dar foloseşte metode eficace şi are oarece aspect. Să vedem ce e nou:

   - se definesc două variabile care sunt adresele (etichetele) unor proceduri pentru citirea unui număr binar sau hexazecimal de la tastatură (`INBIN` şi `INHEX` respectiv);
   - interesant este modul de a citi o cifră cu `LET a = CODE INKEY$ - CODE "0"`; (`CODE "0" = 48`).

   Cele şase subrutine de calcul lucrează astfel:

   HEX→DEC de remarcat metoda extrem de rapidă a transformării, care foloseşte şase variabile: `A=10, B=11, ... F=15` pentru cifrele hex.

   DEC→HEX în linia 150 se forţează variabila sistem MODE la o valoare nonstandard, de unde rezultă un cursor de editare straniu (echivalent cu C).

   BIN→HEX numărul este dus întâi în baza 10, după care este transformat în baza 16, ca la DEC→HEX (puteam dealtfel sări la linia 180).

   HEX→BIN numărul este adus în baza 10 prin metoda HEX→DEC, apoi se sare la procedura DEC→BIN.

   BIN→DEC evident, foloseşte funcţia `BIN`.

   DEC→BIN transformarea este standard, însă la afişare din număr se reţin fie 8 fie 16 cifre.

   Subrutinele de citire a tastaturii:

   INHEX: din şirurile nule face 0, din şirurile cu mai mult de 4 caractere reţine numai ultimele 4, după care testează pentru fiecare caracter apartenenţa la intervalele 0-9, A-F.

   INBIN: ca la INHEX, păstrând ultimele 16 caractere ale numărului binar.

```
 1 BORDER 0: PAPER 0: INK 9: CLS
 2 LET INHEX=1000 : LET INBIN=2000
 5  PRINT AT 3,3;"ALEGETI CONVERSIA"
 8  RESTORE
10  FOR i=0 TO 6: READ a$: PRINT AT i+5,4;" ";i;". ";a$; TAB 24: NEXT i
20  DATA " STOP ", "HEX -> DEC","DEC -> HEX","BIN -> HEX","HEX -> BIN"
25  DATA "BIN -> DEC", "DEC -> BIN"
```

```
 30  LET a=CODE INKEY$-48 : IF a<0 OR a>6 THEN GOTO 30
 35  PRINT AT 3,3; TAB 31
 40  PRINT AT a+5, 4; OVER 1; PAPER 1; TAB 19; OVER 0; PAPER 3; INK 9;
           AT a+14,0; TAB 31;" "; PAPER 2; INK 4; a;". ";
 50  IF a=0 THEN STOP
 60  LET t=a : PAPER 8: GOSUB (1+a)*50 : PAPER 0: PRINT PAPER 1;
          AT t+14,0; OVER 1; TAB 31;" "
 70 GOTO 5
 99
100 GOSUB INHEX
105 LET R=0
107 LET A=10 : LET B=11 : LET C=12 : LET D=13 : LET E=14 : LET F=15
110 FOR I=1 TO LEN A$
120  LET R=R*16 + VAL A$(I)
130 NEXT I
140 PRINT A$; "="; R : RETURN
149
150 POKE 23617, 164: INPUT A
160 LET A=INT ABS A: IF A>65535 THEN GOTO 150
165 PRINT A; "=";
170 LET A$="0000"
180 FOR I=1 TO 4: LET R = A - 16*INT(A/16) : LET A = INT(A/16)
185  LET A$(5-I) = CHR$(48 + R + 7*(R>9)): NEXT I
190 PRINT A$ : RETURN
199
200 GOSUB INBIN
205 PRINT A$; "=";
210 LET A=VAL ("BIN "+A$)
220 FOR I=1 TO 4: LET R = A - 16*INT(A/16) : LET A = INT(A/16)
225  LET A$(5-I) = CHR$(48 + R + 7*(R>9)): NEXT I
230 PRINT A$ : RETURN
249
250 GOSUB INHEX
253 PRINT A$; "=";
255 LET A=10 : LET B=11 : LET C=12 : LET D=13 : LET E=14 : LET F=15
260 FOR I=1 TO LEN A$
265  LET R=R*16 + VAL A$(I)
270 NEXT I : LET A=R : GOTO 360
299
300 GOSUB INBIN
310 PRINT A$; "="; VAL ("BIN" + A$) : RETURN
349
350 POKE 23617, 164: INPUT A
355 LET A=INT ABS A: IF A>65535 THEN GOTO 150 :PRINT A; "=";
360 LET A$=""
370   LET R = A - 2*INT (A/3) : LET A = INT (A/2)
380   LET A$ = STR$ R + A$
390 IF A <> 0 THEN GOTO 370
400 LET A$="00000000"+A$
405 IF LEN A$ > 16 THEN LET A$ = A$(LEN A$-16 TO) :GOTO 420
410 LET A$ = A$(LEN $-8 TO )
420 PRINT A$ : RETURN
999
1000 REM  INPUT HEX NR
1010 POKE 23617, 164
1020 INPUT LINE A$ : IF A$="" THEN LET A$="0000"
```

```
1030 IF LEN A$>4 THEN LET A$=A$(LEN A$-4 TO )
1040 FOR I=1 TO LEN A$
1050   IF NOT (A$(I) >= "0" AND A$(I) <="9" )) AND
          NOT (A$(I) >= "A" AND A$(I) <= "F")) THEN GOTO INHEX
1060 NEXT I
1070 RETURN
1999
2000 REM  INPUT BIN NR
2005 POKE 23617, 164: INPUT LINE A$
2010 IF A$="" THEN LET A$="0000"
2020 IF LEN A$ > 16 THEN LET A$ = A$(LEN A$-16 TO )
2030 FOR I=1 TO LEN A$
2040  IF A$(I) <> "0" AND A$(I) <> "1" THEN GOTO INBIN
2050 NEXT I
2060 RETURN
```

3. Pentru a simplifica lucrurile, este recomandabilă alegerea celor 4 taste ca aparţinând la porturi diferite. Sugerăm următoarea combinaţie: sus q, jos a, stânga v, dreapta b. Pentru a simplifica şi mai mult lucrurile, vom accepta orice tastă citită de acelaşi port cu una din cele de mai sus. În rest, putem adapta programul de la Exerciţiul 4, Capitolul 12 schimbând doar liniile:

```
 1 LET N=IN 254 : REM VALOAREA PENTRU NICI O TASTA APASATA
50 LET x = x + (IN 32766<>n AND x<255) - (IN 65278<>n AND x>0)
55 LET y = y + (IN 64510<>n AND y<175) - (IN 65022<>n AND y>0)
```

   La momentul executării liniei 1 trebuie să nu fie nici o tastă apăsată (explicaţi de ce!).

4. Este adevărată! Dacă `0 <= x <= 31, 0 <= y <= 23`.
   Memoria atributelor începe la 22528 şi este organizată „firesc”. Pătrăţelul cu coordonatele `y, x` la `PRINT` are adresa atributelor `22528 + 32*y + x`, pentru că fiecare din rândurile `y` are 32 de caractere. Deci:
   `ATTR (y,x) = PEEK (22528 + 32*y + x)`

G.17 Capitolul 17

1. Iată soluţia:

```
  1 CLEAR 32767 : REM CLEAR sterge ecranul, deci trebuie dat inainte
  5 REM se deseneaza ...
  ...
100 REM salvam
110 FOR i=0 TO 6911 : POKE 32768+i, PEEK (16384+i) : NEXT i
120 SAVE "SCREEN$" CODE 32768, 6912
130 VERIFY "SCREEN$" CODE
```

2. Pentru că verificarea implică scrierea numelui fişierului respectiv pe ecran, acest nume trebuie să facă parte din chiar componenţa desenului de pe ecran! Dacă dorim să fie invizibil, îl putem scrie cu `PAPER`-ul şi `INK`-ul de aceeaşi valoare. Să nu uităm că, la scrierea titlului unui header, calculatorul scrie întâi un CR (`CHR$ 13`), deci numele va trebui să apară la început de rând, în nici un caz pe linia 0. Să zicem că în desen linia 6 este liberă. Atunci, înainte de a salva desenul, vom face:

```
100  PRINT AT 6,0;PAPER 8;"Bytes: nume"
```

Între literele numelui se inserează caractere de control pentru INK, de exact PAPER-ul respectiv. Apoi:

```
105 SAVE "nume" SCREEN$
110 PRINT AT 5,0; : PAPER 8 : VERIFY "nume" SCREEN$
```

Remarcaţi PRINT AT cu un rând mai sus.

O altă variantă ar fi să folosim faptul că un caracter scris de două ori cu OVER 1 lasă ecranul neschimbat. Atunci facem astfel:

```
 1 LET a$=CHR$ 22 + CHR$ 1 + CHR$ 0 + "Bytes:" : REM AT 1,0 Bytes:
 2 SAVE A$ SCREEN$
 3 OVER 1: FLASH 8: BRIGHT 8: PAPER 8: INK 8
 5 PRINT AT 0,0; : VERIFY a$ SCREEN$
```

La momentul întâlnirii header-ului, VERIFY va scrie în OVER 1, fără a perturba culorile textul de titlu pe linia 1 Bytes:, după care va scrie numele programului. Acesta începe cu caractere de control pentru poziţie, care se mută la începutul aceluiaşi rând şi scrie deasupra acelaşi text, lăsând ecranul în forma iniţială!

3. SCREEN$ este oarecum opusă lui PRINT AT.

G.18 Capitolul 18

1. Se desenează pe ecran (ştiţi cum) şi apoi se face COPY.
2. După imaginaţia fiecăruia.

Appendix H

Variabilele sistem la nivel de bit. Jonglerii cu variabile sistem

H.1 Variabilele sistemului BASIC

Notaţii:

- între paranteze rotunde:
  - numele unei variabile sistem semnifică valoarea ei; ex.: (REPPER) = PEEK 23652;
  - un număr reprezintă valoarea normală a variabilei; ex: (64) — variabila aceasta are de obicei valoarea 64;
- între semnele < şi >:
  adresa unei instrucţiuni sau proceduri din ROM care foloseşte această variabilă, în hexazecimal; ex: <1105> adresa 1105 în hexazecimal;
- punctul indică un octet al unei variabile mai lungi;
  ex.: KSTATE.0/4 = octeţii 0 şi 4 ai lui K_STATE;
  TVDATA.hi = primul octet al lui TVDATA; (hi – vine de la *HIgh byte*, adică MSB; lo este LSB.)
- bit x=y: ce semnificaţie are faptul că bitul x al variabilei are valoarea y;
- biţii care nu sunt trecuţi — autorul nu a putut identifica utilizarea lor (s-ar putea să aibă una!).

Descrierile sunt foarte succinte; ele sunt utile mai ales celui care explorează programele interpretorului BASIC din ROM.

Adresa Dec.	Hex.	Nume	Lg.	Descriere
23552	5C00	KSTATE	8	Folosite în citirea tastaturii: KSTATE.0/4 numărul tastei apăsate KSTATE.1/5 contor 5→0 pentru durată KSTATE.2/6 (REPPER) sau (REPDEL) KSTATE.3/7 codul caracterului
23560	5C08	LASK_K	1	Codul ultimei taste apăsate
23561	5C09	REPDEL	1	Durata necesară pentru a reciti o tasta apăsată mai mult timp (0.7 sec)
23562	5C0A	REPPER	1	Durata pentru a repeta o tastă apăsată continuu (0.1 sec)
23563	5C0B	DEFADD	2	Adresa parametrilor lui DEF FN
23565	5C0D	K_DATA	1	Al doilea parametru al caracterelor de control pentru culoare introduse de la tastatură <1105>
23566	5C0E	TVDATA	2	TVDATA.hi caracterul de control introdus TVDATA.lo primul parametru (Al doilea parametru în registrul A) <0A03>
23568	5C10	STRMS	38	Deplasamentul faţă de (CHANS) al adresei informaţiei de cale. Valorile iniţiale sunt: Calea Adresa Conţinutul Tip canal FD 5C10 0100 K FE 5C12 0600 S FF 5C14 0B00 R 00 5C16 0100 K 01 5C18 0100 K 02 5C1A 0600 S 03 5C1C 1000 P 04-0F 5C1E-5C34 0000 –
23606	5C36	CHARS	2	Adresa – 256 a imaginii setului de caractere (003B=15360). Adresa lui " "=15516.
23608	5C38	RASP	1	Lungimea bâzâitului. (64) <1167>
23609	5C39	PIP	1	Lungimea clic-ului unei taste. (00)
23610	5C3A	ERR_NR	1	Codul erorii – 1. (255)
23611	5C3B	FLAGS	1	bit 0=0: se tipăreşte un spaţiu înaintea unui cuvint cheie <187D> bit 1=0: nu se foloşeste imprimanta <1646> bit 2=0: se scrie în modul K de cursor <1937> bit 3=0: modul K al cursorului e selectat <1326> bit 5=0: nu s-a apăsat o nouă tastă <1303> bit 6=0: rezultatul este un şir bit 7=1: se execută o linie <12CF> =0: se verifică o linie (SINTAX FLAG)
23612	5C3C	TVFLAG	1	bit 0=1: se foloşeste canalul tastaturii <1634> bit 3=1: consideră că modul cursorului s-a schimbat <15D4> bit 4=1: listing automat <1835> bit 5=1: partea de jos a ecranului va fi ştearsă <1219>
23613	5C3D	ERR_SP	2	Adresa de pe stiva Z80 care va fi folosită ca punct de întoarcere în caz de eroare
23615	5C3F	LISTSP	2	Adresa de revenire dupa un listing automat <1795>

Adresa Dec.	Hex.	Nume	Lg.	Descriere
23617	5C41	MODE	1	Specifică forma cursorului: = 1 modul E > 1 modul G < 1 modul K/L/C (împreună cu FLAGS.bit2 şi cu FLAGS2.bit3)
23618	5C42	NEWPPC	2	Linia la care se sare (eticheta) <1E73>
23620	5C44	NSPPC	1	Nr. instrucţiunii la care se sare <1E73>
23621	5C45	PPC	2	Linia în curs de execuţie <1376>
23623	5C47	SUBPPC	1	Nr. instr. în curs de execuţie <1376>
23624	5C48	BORDCR	1	Atributele din partea de jos a ecranului (BORDER=PAPER de jos)
23625	5C49	E_PPC	2	Eticheta liniei cu cursorul ">"
23627	5C4B	VARS	2	Adresa variabilelor BASIC
23629	5C4D	DEST	2	Adresa variabilei in curs de atribuire (dacă aceasta există) <2AFF>
23631	5C4F	CHANS	2	Adresa datelor despre canale <1736>
23633	5C51	CURCHL	2	Adresa informaţiei curente utilizate pentru intrare/ieşire <1615>
23635	5C53	PROG	2	Adresa programului BASIC (23755)
23637	5C55	NXTLIN	2	Adresa următoarei linii din program
23639	5C57	DATADD	2	Adresa caracterului de după ultimul caracter citit din lista DATA (, : sau CR)
23641	5C59	E_LINE	2	Adresa liniei în curs de editare
23643	5C5B	K_CUR	2	Adresa cursorului de la editare (K, L, C, G, E)
23645	5C5D	CH_ADD	2	Adresa următorului caracter de interpretat <0020>
23647	5C5F	X_PTR	2	Adresa caracterului la care s-a obţinut o eroare Pointer la lista READ <1DEC> Pointer la linie/variabila la MERGE
23649	5C61	WORKSP	2	Adresa spaţiului de lucru
23651	5C63	STKBOT	2	Adresa vârfului stivei evaluatorului
23653	5C65	STKEND	2	Adresa sfârşitului stivei evaluatorului
23655	5C67	B_REG	1	Registrul B al evaluatorului
23656	5C68	MEM	2	Adresa memoriilor evaluatorului (23698)
23658	5C6A	FLAGS2	1	bit 0=1: a se şterge ecranul <12CF> bit 1=0: bufferul imprimantei e gol <0EE7> bit 2=0: caracterul nu e între ghilimele <1881> bit 3=1: modul C al cursorului <18F3> bit 4=0: nu se foloseşte canalul K <107F>
23659	5C6B	DF_SZ	1	Numărul de linii din partea de jos a ecranului (inclusiv una albă) (2)
23660	5C6C	S_TOP	2	Nr. liniei din partea de sus a ecranului la un listing automat
23662	5C6E	OLDPPC	2	Linia la care sare CONTINUE
23664	5C70	OSPPC	1	Nr. instrucţiunii la care sare CONTINUE
23665	5C71	FLGX	1	bit 0=1: se va şterge vechea valoare a acestei variabile şir <2B72> bit 1=1: o noua variabilă bit 5=1: mod INPUT <0FF3> =0: mod editare <1313> bit 7=1: se foloseste INPUT LINE <20DB>
23666	5C72	STRLEN	2	Lungimea şirului destinat atribuirii
23668	5C74	T_ADDR	2	Adresa următorului element în tabela de sintaxă T_DDR.lo indică tipul de header <1B55>

Adresa Dec.	Adresa Hex.	Nume	Lg.	Descriere
23670	5C76	SEED	2	Folosită în calculul RND
23672	5C78	FRAMES	3	Ceas de timp real în 1/50 sec. LSB primul.
23675	5C78	UDG	2	Adresa primului caracter definibil
23677	5C7D	COORDS	1	Coordonata x a ultimului punct
23678	5C7E		1	Coordonata y a ultimului punct
23679	5C7F	P_POSN	1	33–nr. coloanei la imprimantă
23680	5C80	PR_CC	2	LSB al adresei următorului caracter de trimis spre imprimantă (din buffer)
23681	5C81		1	Marcat nefolosit. În realitate, MSB al adresei următorului caracter de trimis spre imprimantă
23682	5C82	ECHO_E	1	33 – coloana
23683	5C83		1	24 – linia din partea de jos a ecranului (=ultimul caracter din INPUT buffer)
23684	5C84	DF_CC	2	Adresa în DF a următorului caracter de scris pe canalul S
23686	5C86	DF_CCL	2	Adresa în DF a următorului caracter de scris pe canalul K
23688	5C88	S_POSN	1	33 – coloana
23689	5C89		1	24 – linia poziţiei PRINT din partea de sus a ecranului
23690	5C8A	SPOSNL	1	33 – coloana
23691	5C8B		1	24 – linia poziţiei PRINT din partea de jos a ecranului
23692	5C8C	SCR_CT	1	Nr. de scroll de făcut înainte de a întreba scroll?
23693	5C8D	ATTR_P	1	Atributele permanente
23694	5C8E	MASK_P	1	Masca pentru atributele permanente (biţii 1 iau culoarea de pe ecran)
23695	5C8F	ATTR_T	1	Atributele temporare
23696	5C90	MASK_T	1	Masca pentru atribute temporare
23697	5C91	P_FLAG	1	bit 0=1: OVER 1 <22F0> bit 2=1: INVERSE 1 <22FD> bit 4=1: INK 9 <0BFA> bit 6=1: PAPER 9 <0BDB>
23698	5C92	MEMBOT	30	Zona de memorie a evaluatorului (6 zone a 5 octeţi)
23728	5CB0		2	nefolosit
23730	5CB2	RAMTOP	2	Adresa ultimului octet accesibil BASIC (65368)
23732	5CB4	P_RAMT	2	Adresa ultimului octet din RAM-ul fizic (65535)

H.2 Jonglerii cu variabile sistem

Să vedem câte ceva din ceea ce putem face dând valori cu POKE acestor variabile (în orice caz, nu un procedeu elegant de programare):

- Schimbând conţinutul lui REPDEL şi REPPER, se poate schimba viteza cu care se citesc tastele la INPUT şi la editare. Valoarea 0 înseamnă 256. Tastaţi POKE 23562,1 şi scrieţi apoi ceva!
- Variabila CHARS arată adresa unde se află memorată forma caracterelor tipăribile (cu codurile între 32 şi 127). Schimbând valoarea ei, o facem să puncteze o zonă oarecare din RAM, unde putem pune propriile noastre forme. Iată şi un program caraghios:

```
1 FOR i=0 TO 72
2    POKE 23606,i
3    PRINT AT 10,13;0 : BEEP .006, i/7
```

```
4 NEXT i
5 POKE 23606, 0
```

Se va întâmpla un lucru amuzant dacă opriţi programul cu BREAK. Ce?

- Modificând RASP se schimbă lungimea bâzâitului care apare la unele erori (de pildă la editarea unei linii mult prea lungi).
- PIP e lungimea clic-ului pe care îl produce o tastă apăsată. Normal e 0, însă, introducând alte valori, putem obţine sunete ce par mai înalte. De fapt este o iluzie, toate sunetele având aceeaşi înălţime; unele sunt prea scurte şi par mai joase. Tastaţi `POKE 23609,15`.
- Iată un divertisment cu ERR_NR:

```
1 FOR i=0 TO 28
2 POKE 23610,i
```

  Daţi `RUN` şi apoi mereu `NEXT i`. Veţi obţine şi nişte mesaje surprinzătoare. Valoarea ei normală (nu e eroare) este 255.
- ERR_SP indică o adresă (pe stivă) unde se găseşte adresa subrutinei care se foloseşte la apariţia unei erori. Modificarea ei folosind BASIC-ul este mai greu de făcut. Iată o metodă care permite ignorarea erorilor de către programul vostru (cu excepţia erorii C): `POKE 23613, PEEK 23730 - 5`. Erorile nu mai întrerup programul din execuţie, dar afectează variabila ERR_NR, ceea ce vă permite să le trataţi. „Protecţia” aceasta este desfăcută de folosirea lui `GOSUB`; folosirea ei în subrutine poate avea efecte catastrofale. Pentru a readuce sistemul la starea normală: `POKE 23613, PEEK 23730 - 3`.
- MODE specifică tipul cursorului (K, L sau C). `POKE 23617,1:INPUT a$` va trece cursorul în modul E pentru instrucţiunea `INPUT`. Puteţi obţine şi tipuri ciudate de cursor. Încercaţi 30 sau 164.
- NEWPPC şi NSPPC, dacă sunt schimbate în această ordine, forţează saltul la o anumită instrucţiune, dintr-o anumită linie.
- BORDCR (atributele din partea de jos a ecranului) poate da şi ea naştere la efecte interesante. Încercaţi `POKE 23624,200`. `POKE 23624,0` face 0 atributele rândurilor de jos, deci cursorul de la editare (K, L etc.) rămâne invizibil.
- E_PPC modificată poate muta cursorul de editare > la o anumită linie. De pildă la linia 10: `POKE 23625,10 : POKE 23626,0`
- PROG arată unde începe programul BASIC. O vom folosi pentru a forţa la 0 eticheta primei linii; nefiind o etichetă regulamentară, conferă acelei linii un statut privilegiat (de neşters). Folosind informaţiile din Anexa F cu privire la forma de memorare, vedem că putem schimba eticheta primei linii în 0 cu:

```
LET a=PEEK 23635+256*PEEK 23636 : POKE a,0 : POKE a+1,0
```

- DF_SZ reprezintă numărul de linii alocate părţii de jos a ecranului (canalului K). De obicei, acest număr este 2. Valori mai mici dau efecte secundare ciudate. Reducând această valoare la 0, tentativele de scriere pe canalul K vor distruge sistemul (inclusiv afişarea mesajelor de eroare, de pildă a celor cauzate de BREAK). În plus, putem folosi `PRINT` pe 23 de linii. Din păcate şi `CLS` blochează sistemul.

  Dacă punem DF_SZ la 1 şi apoi scriem ceva pe canalul K, întreg ecranul (inclusiv atributele) este umplut cu valoarea variabilei ATTR_P. Încercaţi:

```
5 FOR i=1 TO 8
6    POKE 23693, 2^i-1 : REM ATTR_P
7    POKE 23659,1 : REM DF_SZ
8    INPUT ""
9 NEXT i
```

- OLDPPC şi OSPPC pot face `CONTINUE` să sară într-un anumit loc.

```
 1 POKE 23662,50 : POKE 23663,0 : REM linia 50
 2 POKE 23664,2 : REM instructiunea a doua
 3 CONTINUE
 5 STOP
50 STOP : PRINT 0
```

  va tipări 0!

- SEED este iniţializată de `RANDOMIZE`. O putem folosi pentru a descompune rapid un număr în doi octeţi:

```
 5 RANDOMIZE nr
10 LET nrlo= PEEK 23670 : LET nrhi= PEEK 23671
```

- FRAMES este o variabilă care funcţionează ca un ceas. Valoarea celui mai puţin semnificativ octet este incrementată la fiecare 20ms. Al doilea octet este incrementat când LSB devine 0 ş.a.m.d. Incrementarea este oprită în timpul lucrului cu imprimanta, casetofonul sau al execuţării instrucţiunii `BEEP`. Timpul de când a fost resetat ultima oară calculatorul, în secunde, este:

```
(65536*PEEK 23674 + 256*PEEK 23675 + PEEK 23676)/50
```

  Pentru a potrivi ceasul, de pildă la ora 10, trebuie să fi trecut `10*60*60*50` de cinzecimi de secundă = 1800000. Descompunând acest număr avem 1800000 = 65536*27 + 256*119 + 65. „Reglarea" ceasului se face cu: `POKE 23674, 27: POKE 23673,119 : POKE 23672,64` Rescrieţi ceasul cu limbi folosind FRAMES.

- UDG este adresa formelor caracterelor grafice definibile. În mod normal, ea este imediat deasupra RAMTOP-ului. Ca să vă distraţi, puneţi-o peste E_LINE şi apoi tastaţi caractere grafice definibile. Explicaţi ce se întâmplă.
- COORDS pot fi folosite pentru a obţine trasări de segmente în coordonate absolute. Pentru a trasa până la punctul `x,y`:

```
DRAW x-PEEK 23677, y-PEEK 23678
```

- S_POSN o putem folosi ca să aflăm câte simboluri grafice are un şir: îl scriem pe ecran la 0, 0 cu ; după el şi apoi citim S_POSN ca să aflăm unde a rămas cursorul de tipărire.
- SCR_CT numără de câte ori poate fi „alunecat ecranul" înainte de a întreba `scroll?`. Variabila se decrementează la fiecare alunecare. Întrebarea survine la 0. `POKE 23692,255` ne scuteşte de `scroll?` o bună bucată de vreme. Încercaţi:

```
 1 POKE 23692,255
 5 FOR i=1 TO 10000
10   PRINT i
20 NEXT i
```

- Variabila 23681 este trecută în manuale ca nefolosită. În realitate, ea este MSB al adresei bufferului imprimantei. Pentru că SPECTRUM memora caracterele în buffer pe linii de pixeli (avea o imprimantă specială), putem obţine efecte ciudate, mutând bufferul peste memoria ecran şi trimiţând date spre imprimantă în absenţa acesteia (cu imprimantă conectată, acest program nu merge). Studiind cu atenţie memoria ecran veţi înţelege de ce următorul program face ceea ce face (rulaţi-l şi fără `NEXT` şi observaţi dispunerea rândurilor):

```
10 FOR i=64 TO 71: POKE 23681,i : LPRINT "Caractere mari": NEXT i
```

  Această linie contruieşte de 8 ori buffer-ul în memoria ecran şi îl umple cu imaginile caracterelor care apar pe feliuţe.

Iată câteva metode de a modifica însuşi programul BASIC, obţinând efecte ciudate. De pildă, putem insera în listing caractere de control pentru AT, TAB, backspace (CHR$ 8) etc.

```
    1 REM         Mesaj copyright protejat

LET a=PEEK 23635+ 256*PEEK 23636 : FOR a+5 TO a+12 : POKE a,8: NEXT i
```

transformă în CHR$ 8 spaţiile care le-am scris după REM.

Putem încerca şi alte combinaţii: inserând CHR$ 22 + CHR$ 30 + CHR$ 0 (AT 30,0), vom obţine la listarea acelei linii eroarea 5.

Index

[ ], 17
#, 138
$, 20
scroll?, 22

A

ABS, 44
ACS, 98
adevărat, 40
adresă, 114
aleator, 61
AND, 76
arccosinus, 98
arcsinus, 98
arctangentă, 98
argument, 15, 127
arhitectură, 112
asamblor, 112
ASCII, 88
ASN, 97
asociativitate, 28
ATN, 98
atribuire, 29
atribute, 109
ATTR, 108

B

Bach, 107
backslash, 169
bază de numeraţie, 152
BEEP, 107
BIN, 120
bistabil, 153
bit, 153
blanc, 16
boolean, 40
BORDER, 36
BREAK, 31
BRIGHT, 67
buclă, 53
buclă de întârziere, 80
buffer, 117
byte, 114

C

c.m.m.d.c, 46
c.m.m.m.c, 46
cât, 46
cale, 137
canale, 137
caracter, 16
caracter definibil, 121
caractere de control, 88
carriage return, 14
CHR$, 88
ciclu, 37, 53
cifre binare, 152
CIRCLE, 86
CLEAR, 39
CLOSE#, 138
CLS, 38
cod ASCII, 88
cod maşină, 112
CODE, 91
comandă, 15
comentariu, 43
compatibil, 137
compilare, 113
compilator, 113
concatenare, 29
constantă, 18
CONTINUE, 32
contor, 54, 163
contrast, 37
coordonate grafice, 84
coordonate relative, 85
COPY, 136
COS, 97
cosinus, 96
culori în mod grafic, 86
culori fundamentale, 151
cursor de editare, 63
cursorul de la tipărire, 34
cuvânt, 115
cuvânt cheie, 21
cuvinte cheie, 15, 16

D

DATA, 105
DEF FN, 127
DIM, 49
dolar, 20
DRAW, 84
driver, 136

E

e, 45
ecranul, 24
editare, 22
efecte laterale, 35
elastic, 174
ENTER, 14
eroare, 17
etichetă, 14
Euclid, 46
Euler, 45
EXP, 45
exponent, 72, 155

F

factorial, 129
fals, 40
fişier, 129
Fibonacci, 47
fill, 103
FLASH, 68
FN, 127
FOR - TO - STEP, 53
forma de prezentare, 21
formatul ştiinţific, 72
formatul exponenţial, 72
FP, 155
funţie exponenţială, 45
funcţia semn, 45
funcţie, 15

G

GOSUB, 94
GOTO, 35
grilă, 24

H

harta memoriei, 115
header, 131

I

IF - THEN, 40
imprimantă, 135
IN, 122
incrementare, 29
indice, 49
infixare, 74
INK, 37
INKEY$, 63
INPUT, 32
INT, 44
interactiv, 54
interpretare, 113
interpretor, 113
intrare-ieşire, 112
INVERSE, 66

K

kilooctet, 114

L

LEN, 80
LET, 28
lexicografic, 106
LIFO, 95
limbaj, 13
limbaj de asamblare, 112
limbaj maşină, 112
linia curentă, 22
linie, 13
LIST, 22
listare, 22
listing, 22
listing automat, 139
LLIST, 135
LN, 45
LOAD, 130
locaţie, 114
logaritm, 45
LPRINT, 135
LSB, 109, 154

M

mantisă, 155
matrice, 48
maximum, 51
memoria, 112, 114
memorie permanentă, 129
memorie video, 115
meniu, 165
MERGE, 131
mesaj de eroare, 17
metoda bulelor, 57
mişcare, 47
mici întregi, 155
microdrive, 140
microprocesor, 112
mnemonică, 112
modul, 44
modulo, 90

MSB, 154

N

Neper, 45
neprintabil, 88
NEW, 39
NEXT, 53
nivel înalt, 113
normalizare, 155
NOT, 75
număr prim, 47

O

octet, 114
OPEN#, 138
operaţii aritmetice, 28
operaţii logice, 40
OR, 76
ordine pentru şiruri, 106
originea grafică, 84
OUT, 123
OVER, 68
overflow, 156

P

PAPER, 36
parametru, 15
parametru opţional, 17
parametru simbolic, 127
parte întreagă, 44
PAUSE, 44
PEEK, 114
periferice, 112
PI, 93
pixel, 24
PLOT, 84
POINT, 87
pointer, 158
POKE, 114
port, 122
precedenţă, 28
precedenţa, 78
prefixare, 74
PRINT, 23
prioritate, 78
procedură, 94
procustean, 51
program, 13, 14
proiecţie, 85
punct, 16

R

radian, 85
radical, 45
RAM, 113
random access, 113
RANDOMIZE, 62
read only, 113
READ, 104
recursie, 175
recursiv, 96, 129
REM, 43
rest, 46
RESTORE, 105
RETURN, 94
rezoluţie, 24
RND, 61
ROM, 113
rotunjire, 44
RS232, 125
RUN, 27

S

sămânţă, 62
salt, 35
salvare, 129
sau exclusiv, 68, 81
SAVE, 129
SCREEN$, 126
seed, 62
semne speciale, 16
separatori, 23
setare, 115
SGN, 45
SIN, 97
sintaxă, 17
sinus, 96
sistem de operare, 137
slash, 28
sortare, 57
sortare prin selecţie, 58
SQR, 45
stiva, 95
stiva GOSUB, 94
STOP, 31
STR$, 121
subrutină, 94
sunet, 107
supraîncărcare, 29
sursă, 113

T

tabel de adevăr, 170
tablou, 48
TAN, 97
tangentă, 96
tip de date, 18
tipul şir de caractere, 18
tipul numeric, 18

TO, 69
transparenţă, 37
trigonometrie, 96

U

UC, 112
UDG, 121
underflow, 156
unitatea centrală, 112
USR, 118

V

VAL, 71
VAL$, 81
valoare absolută, 44
variabilă, 18, 19
variabile de sistem, 117
vector, 48
VERIFY, 132
virgulă flotantă, 155
von Neumann, 111

X

xor, 81

Z

Z80, 112