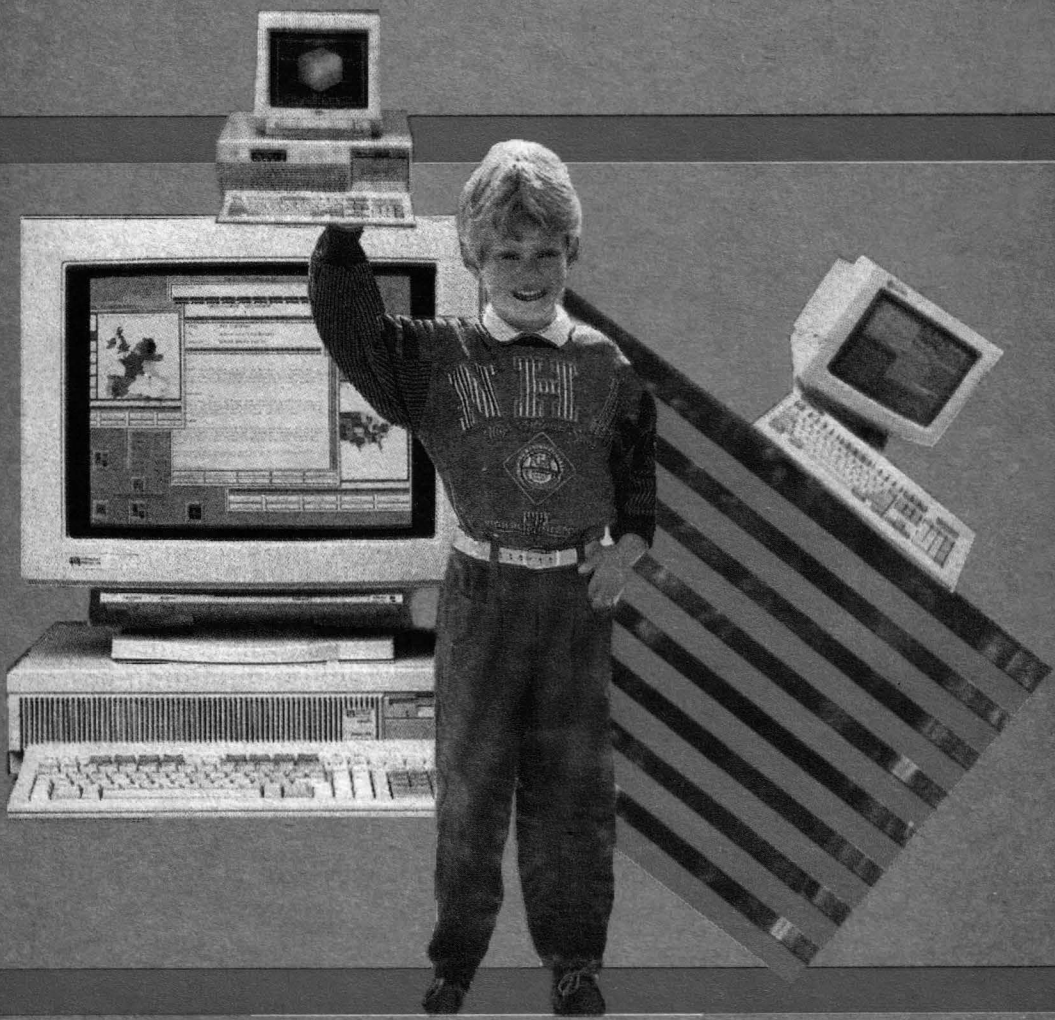
Biblioteca de Informatică

Ion DIAMANDI

CALCULATORUL, coleg de bancă



Editura Agni

Ion Diamandi

Calculatorul, coleg de bancă

Editura Agni
Bucureşti, 1993

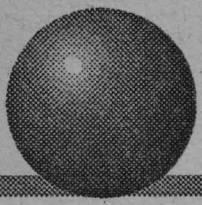
Tehnoredactarea pe calculator: Ioana Varga, Florin Ilia
Coperta: Iolanda Malamen

C.P. 30-107, Bucureşti

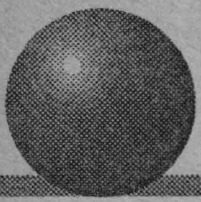
ISBN 973-95626-3-9

Ion Diamandi

Calculatorul, coleg de bancă



Cuprins



Cuvînt înainte

Introducerea recentă a informaticii în programa şcolară pentru gimnaziu şi liceu face necesară studierea unor probleme precum şi elaborarea unor metodologii de utilizare a tehnicilor informatice în orele de şcoală. În acest sens sînt chemaţi să-şi aducă contribuţia atît profesori cît şi informaticieni care au experienţă în domeniul utilizării calculatoarelor de către tînăra generaţie. Nu mai puţin importante sînt şi aspectele psihosociale legate de aceste abordări.

Este cunoscut faptul că în învăţămîntul din ţara noastră predomină aspectul informativ în defavoarea celui formativ. Introducerea informaticii ca materie şcolară şi, mai mult, introducerea tehnologiilor informatice în mediul şcolar trebuie să aibă ca efect, în primul rînd, tocmai aspectul formativ şi ar fi o greşeală ca acest lucru să fie considerat numai sub aspectul asimilării de către elev a unui plus de informaţii (termeni noi, instrucţiuni, comenzi). Lucrarea de faţă îşi propune să reprezinte un ghid care să ofere cîteva idei profesorilor din învăţămîntul preuniversitar în vederea organizării unor ore de clasă avînd şi calculatorul ca mijloc didactic. Scopul este deprinderea copiilor cu o gîndire ordonată şi logică sau, cu alte cuvinte, formarea unei gîndiri algoritmice. Bineînţeles că, în aceeaşi măsură, lucrarea poate fi utilă elevilor din cursul gimnazial şi liceal.

Pentru cel care doreşte să folosească lucrarea (pentru a învăţa utilizarea calculatorului, sau pentru a învăţa la matematică cu ajutorul calculatorului, sau în vederea organizării unor ore de şcoală cu calculatorul), se presupune cunoaşterea prealabilă a cîtorva elemente privind utilizarea calculatoarelor şi, anume, modul de operare (de introducere a instrucţiunilor şi comenzilor) precum şi cîteva dintre cele mai cunoscute instrucţiuni şi comenzi ale limbajului BASIC. În acest scop nu se recomandă introducerea clasică utilizată în informatică prin care se descrie forma unei instrucţiuni şi apoi se dau exemple de folosire, fiind preferabil să se pornească de la rezolvarea unor probleme concrete de şcoală corespunzătoare anului de studiu respectiv. Este de asemenea indicat un ritm lent de introducere a termenilor şi instrucţiunilor noi. Deşi accentul se va pune pe

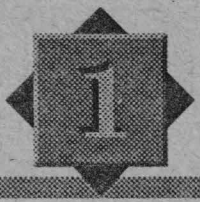
conceperea programelor (aspectul formativ), aceasta nu va reprezenta un scop în sine, ci, un mijloc de rezolvare a problemelor curente de şcoală. De aceea s-a ales ca temă principală rezolvarea unor probleme de matematică, dar lucrarea se doreşte a constitui un stimulent pentru profesori, în ideea de a descoperi, ei înşişi, şi de a integra în orele de clasă noi probleme (şi nu numai de matematică) a căror rezolvare se pretează a se realiza cu ajutorul calculatorului. De fapt astfel de rezolvări de probleme (fiind strîns legate de programa şcolară şi nesolicitînd cunoştinţe suplimentare de matematică faţă de cele prevăzute acolo), se pot integra şi în orele de la alte discipline şcolare.

Modul de tratare a problemelor este următorul: se prezintă problema eventual cu încercarea de rezolvare pe cale clasică şi explicarea necesităţii lucrului cu calculatorul. Necesitatea utilizării calculatorului în rezolvarea problemei poate proveni şi din posibilitatea astfel creată de explorare şi experimentare pe care o oferă acest nou instrument. Apoi se prezintă metoda (algoritmul) de rezolvare a problemei. S-a considerat că pentru acest tip de probleme nu este necesară şi descrierea algoritmului prin tehnici ca scheme logice sau diagrame de structură, fiind suficientă înţelegerea paşilor algoritmului. În final se expune un exemplu de program scris în limbajul BASIC. Astfel sînt abordate probleme cunoscute din programa de matematică pentru gimnaziu şi liceu: multipli numerelor,divizibilitatea numerelor, cel mai mare divizor comun şi cel mai mic multiplu comun, numere prime, descompunerea în factori primi, şirul lui Fibonacci, funcţia de gradul 1, grafice de funcţii, limite de şiruri etc. Deseori sînt propuse spre rezolvare şi anumite probleme distractive sau probleme pentru realizarea unor jocuri.

Orele de informatică se vor desfăşura sub forma activităţilor practice cu calculatorul, prin problemele propuse urmărindu-se nu numai dezvoltarea unei gîndiri logice şi convergente (algoritmice) ci şi formarea unor deprinderi de explorare şi experimentare cu calculatorul. În unele exemple am căutat şi dezvoltarea unor aptitudini pentru luarea unor decizii în condiţii restrictive de timp (gîndire intuitivă), foarte necesare în rezolvarea ulterioară a unor probleme din viaţa de zi cu zi.

Paralel cu activităţile propuse am introdus şi cîteva elemente fundamentale privind bazele proiectării şi funcţionării calculatoarelor. Informatica a ajuns la un stadiu de maturitate, care impune ca unele din rezultatele sale să intre în cultura generală a elevilor.

Programele exemplificate sînt funcţionale pentru cele mai răspîndite calculatoare din ţara noastră şi, implicit, din sistemul de învăţămînt preuniversitar şi anume, calculatoarele compatibile Sinclair Spectrum: HC, CIP, JET, COBRA, TIM. S-a căutat de asemenea ca programele să fie realizate într-un BASIC standard pentru a putea fi lesne transpuse şi pe alte calculatoare. Majoritatea programelor sînt funţionale şi pe calculatoare de tip PC în versiunea GWBASIC. Cele care utilizează tehnici specifice versiunii SINCLAIR BASIC (în special cele care utilizează elemente grafice şi şiruri de caractere), fiind, evident, nefuncţionale pe calculatoare PC, sînt prezentate în două variante (A şi B), pentru SINCLAIR SPECTRUM şi, respectiv, GWBASIC. Pentru a veni în sprijinul celor care vor să se iniţieze în utilizarea calculatoarelor, în final, se prezintă un index al cuvintelor cheie folosite în lucrare.



Logica maşinilor inteligente

Pentru a obţine rezultatul unor calcule, de exemplu 3+4, ştim că nu este suficient să cerem calculatorului 3+4 şi nici 3+4=. Este necesar să comandăm calculatorului să ne afişeze rezultatul operaţiei respective, iar acest lucru se va realiza cu comanda PRINT (AFIŞEAZĂ). Pentru tipul de calculator avut în vedere PRINT se va obţine prin acţionarea tastei P. Deci PRINT 3+4 şi apoi tasta **CR** (**ENTER**). Vom obţine afişat pe ecran rezultatul 7. În mod similar pentru calculul oricărei expresii se va folosi aceeaşi comandă. Încercaţi de exemplu:

```
PRINT 3.47 * (0.34 - 41.83/(SQR 70 -3)*2)
```

De asemenea pentru a obţine afişarea unui text pe ecran se va folosi PRINT iar textul va fi încadrat între ghilimele.

```
PRINT "Am trecut in clasa a 8-a"
PRINT " 2 + 3 = 4 "
```

Dar ce se va întîmpla dacă vom introduce comanda PRINT 2 = 2 ?

Evident, nu i-am cerut calculatorului nici să ne efectueze un calcul şi să ne afişeze rezulatul şi nici să ne afişeze un text ca 2=2, deoarece nu l-am încadrat între ghilimele. Sigur, aţi observat însă că la introducerea comenzii PRINT 2 = 2 pe ecran s-a afişat rezultatul 1, în cazul în care am lucrat cu HC şi -1 în cazul PC-ului. Acelaşi rezultat îl vom obţine şi la comanda:

```
PRINT 2 + 4 = 6
```

În schimb pentru PRINT 1 = 2 vom obţine rezultatul 0, la fel ca şi pentru PRINT 2 - 1 = 6. Acum lucrurile încep să se clarifice.

2 = 2 sau 2 + 4 = 6 reprezintă *propoziţii adevărate*, iar acestea au pentru calculatorul HC valoarea 1 (Adevărat = 1, respectiv -1 pentru PC) în timp

ce 1 = 2 sau 2 - 1 = 6 reprezintă propoziţii false care au pentru calculator valoarea 0 (Fals = 0).

Dar ce rezultat vom obţine cu `PRINT 1 = 1 = 1`*?*

Observăm că obţinem valoarea 1, ceea ce ne aşteptam, deoarece am scris numai lucruri adevărate. În schimb cu `PRINT 4 = 4 = 2 + 2` obţinem valoarea 0.

Pentru a înţelege rezultatul obţinut trebuie să ştim cum *evaluează* calculatorul ce urmează după comanda `PRINT`.

4 = 4 este o poziţie adevărată pe care calculatorul o evaluează cu 1, apoi evaluează expresia 2 + 2 la 4 şi, în sfîrşit, evaluează propoziţia 1 = 4 care, fiind falsă, va afişa rezultatul final 0.

De notat că mai întîi se evaluează expresiile aritmetice şi apoi cele logice.

De exemplu pentru `PRINT 3+2 < 3*2-4`, evaluează mai întîi 3+2 cu 5 apoi 3*2-4 cu 2 iar apoi comparîndu-l pe 5 cu 2 rezultă o propoziţie falsă, afişînd deci 0.

Puteţi explica de ce cu comanda `PRINT 1 = 1 = 3` *obţinem rezultatul 0, iar cu* `PRINT 1 = 1 > 3 < 2` *obţinem valoarea 1?*

Cu comanda `PRINT 1 = 1 = 3` obţinem rezultatul 0 deoarece calculatorul evaluează 1 = 1 cu 1 fiind o propoziţie adevărată, iar 1 obţinut nefiind egal cu 3 (propoziţie falsă) obţinem rezultatul final 0.

Cu comanda `PRINT 1 = 1 > 3 < 2` obţinem rezultatul 1 deoarece:

1 = 1 propoziţie adevărată —> valoarea 1
1 > 3 propoziţie falsă —> valoarea 0
0 < 2 propoziţie adevărată —> valoarea 1

În expresiile şi propoziţiile care vor fi evaluate după comanda `PRINT` se pot introduce nu numai numere, semne pentru operaţiile aritmetice, funcţii (de exemplu `SQR` = radical, `SIN` = sinus, etc.) şi paranteze (rotunde), dar şi semnele <, >, =<, >= precum şi operatorii logici `OR` (SAU) şi `AND` (ŞI).

Să vedem cum putem verifica cu ajutorul calculatorului operaţiile cu SAU care sînt date în următoarea tabelă de adevăr (Adevărat = A, Fals = F, SAU = ∨):

a) A ∨ A = A	sau	1 OR 1 = 1
b) A ∨ F = A	sau	1 OR 0 = 1
c) F ∨ A = F	sau	0 OR 1 = 1
d) F ∨ F = F	sau	0 OR 0 = 0

	Verificare cu calculatorul	Rezultat
a)	`PRINT 1 = 1 OR 2 = 1*2`	1
b)	`PRINT 1 = 1 OR 1 = 2`	1
c)	`PRINT 1 = 2 OR 1 = 1`	1
d)	`PRINT 1 = 2 OR 2 = 3`	0

În aceste exerciţii de logică se poate folosi şi operatorul logic `NOT` care are funcţia de a *nega* o propoziţie. De exemplu, `PRINT NOT (1 > 2)` va da rezultatul 1 deoarece 1 > 2 este o propoziţie falsă (rezultat 0), iar negarea ei va fi o propoziţie adevărată (rezultat 1).

Putem verifica faptul că printr-o negare `AND` se transformă în `OR` iar `OR` în `AND`. Astfel cu `PRINT NOT (1=2 AND 1=1)` obţinem ca rezultat 1 deoarece propoziţia din paranteză este falsă, iar negată va fi adevărată; forma va fi echivalentă cu `PRINT NOT 1=2 OR NOT 1=1` care dă ca rezultat final tot 1.

În mod similar verificării operaţiilor cu SAU se pot verifica şi operaţiile cu ŞI (`AND` = ∧):

e) A ∧ A = A	sau	1 AND 1 = 1
f) A ∧ F = F	sau	1 AND 0 = 0
g) F ∧ A = F	sau	0 AND 1 = 0
h) F ∧ F = F	sau	0 AND 0 = 0

Verificare cu calculatorul	Rezultat
e) PRINT 1 = 1 AND 2 = 2	1
f) PRINT 1 = 1 AND 1 = 2	0
g) PRINT 1 = 2 AND 1 = 1	0
h) PRINT 1 = 2 AND 2 = 3	0

Putem realiza şi propoziţii în care să combinăm cei doi operatori OR şi AND.

De exemplu putem verifica următoarea relaţie care demonstrează distributivitatea lui AND faţă de OR :

$$(P1 \vee P2) \wedge P3 = (P1 \wedge P3) \vee (P2 \wedge P3)$$

De exemplu să considerăm că P1 este adevărată iar P2 şi P3 sînt false. În acest caz (P1 ∨ P2) va fi adevărată iar (P1 ∨ P2) ∧ P3 va fi falsă. În consecinţă rezultatul va fi o propoziţie falsă (0).

Verificare cu calculatorul:

```
PRINT (1 = 1 OR 1 = 2) AND 1 = 3
```

Obţinem într-adevăr rezultatul 0, la fel ca şi în cazul:

```
PRINT (1=1 AND 1=3) OR (1=2 AND 1=3)
```

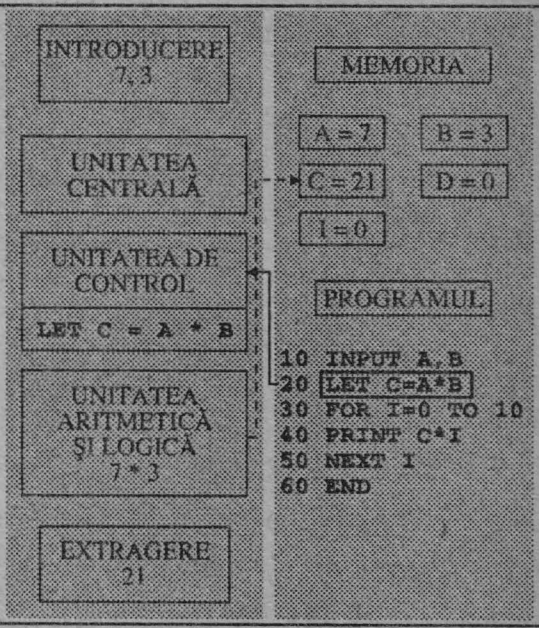
2 Calculatorul de sticlă

Ştiind *cum funcţionează un calculator* ne va fi mult mai uşor să realizăm programe şi, de asemenea, să le îndreptăm (depanăm) astfel încît să obţinem rezultate corecte.

Pentru a înţelege cum funcţionează un calculator ne-ar fi foarte util dacă am putea, oricînd am dori, să îl "secţionăm" sau, cu alte cuvinte, dacă am avea la dispoziţie un "*calculator de sticlă*" şi să vedem ce se petrece în interiorul său. Astfel am vedea, pe de o parte, părţile componente ale calculatorului (unitatea centrală, unitatea aritmetică logică, unitatea de introducere a datelor, unitatea de extragere a rezultatelor, etc.), iar, pe de altă parte, memoria calculatorului care conţine atît programul care se execută cît şi locaţiile (celulele) de memorie care conţin valorile variabilelor (parametrilor) din program.

fig.1

Calculatorul de sticlă



În figura 1 se distinge ce am "vedea" în "calculatorul de sticlă" sau în cazul "secţionării" unui calculator într-un anumit moment al funcţionării programului care se află în memorie.

O instrucţiune care se execută va fi preluată de unitatea centrală (şi anume de unitatea de control) care o va decodifica şi o va transmite la o altă unitate în funcţie de rezultatul decodificării. Astfel dacă este vorba de o instrucţiune de introducere de date, atunci va fi implicată unitatea de introducere.

Prin unitate de introducere înţelegem de cele mai multe ori tastatura, dar poate fi şi unitatea de discuri (sau casetofonul).

Astfel considerînd că în memorie s-a introdus un program, după ce s-a lansat comanda de execuţie a programului, RUN, calculatorul va lua prima linie de program (10 INPUT A,B) iar instrucţiunea respectivă o va trimite unităţii de control. Aceasta o decodifică şi, observînd că este vorba de o instrucţiune de introducere, (alte instrucţiuni de acest fel sînt READ, LOAD, etc.) va determina aşteptarea tastării de către utilizator a unor valori numerice. Să presupunem că s-au introdus valorile 7 şi 3. În acest caz aceste valori se vor depune în locaţiile de memorie A şi respectiv B după care controlul va fi trecut la următoarea linie de program (20 LET C=A*B). Acest moment este oglindit chiar în figură. După ce instrucţiunea este decodificată, unitatea aritmetică şi logică (**UAL**) va evalua expresia 3×7. Valoarea rezultată va fi depusă în locaţia de memorie **C** după care controlul va trece la următoarea linie de program. Dacă aceasta ar fi linia 25 PRINT C, fiind implicată o instrucţiune de extragere de rezultate, aceasta va avea efect asupra televizorului sau monitorului calculatorului afişîndu-se pe ecran numărul 21.

Instrucţiunile care activează unitatea aritmetică şi logică sînt acelea care conţin *expresii care necesită a fi evaluate* (calculate) (LET, PRINT, GO TO, IF etc) sau care necesită comparaţii aritmetice sau logice (IF), în timp ce, instrucţiunile care acţionează asupra unităţilor de extragere sînt: PRINT, PRINT AT, PLOT, DRAW, CIRCLE, LPRINT, LIST, LLIST, SAVE etc. De asemenea instrucţiunile care au efect asupra memoriei calculatorului (de fapt asupra locaţiilor de memorie) sînt aşa numitele instrucţiuni de atribuire, şi anume, LET, READ-DATA, INPUT, FOR-NEXT, executarea lor avînd ca urmare depunerea unor valori în locaţii de memorie. Observăm că multe din instrucţiuni activează mai multe unităţi. De

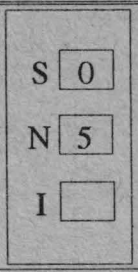
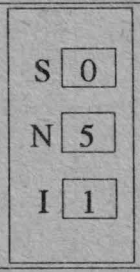
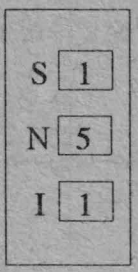
exemplu LET activează unitatea aritmetică logică pentru a calcula o expresie iar apoi depune într-o locaţie de memorie valoarea rezultată.

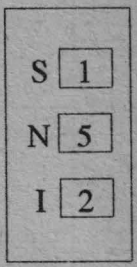
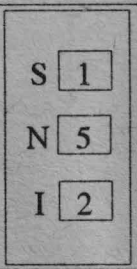
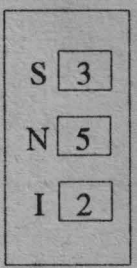
Din păcate calculatoarele cu care lucrăm nu sînt "calculatoare de sticlă" iar dacă secţionăm calculatorul nu vom vedea decît circuite electronice. Astfel încît nu ne rămîne decît să ne imaginăm ce se întîmplă într-un calculator atunci cînd se execută un program. Să încercăm acest lucru cu un program care realizează adunarea a **N** numere naturale:

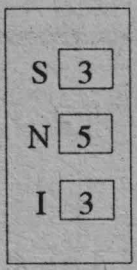
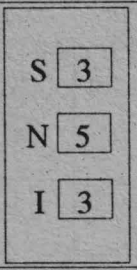
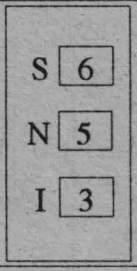
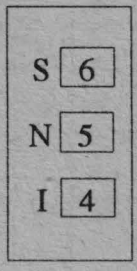
```
10 INPUT N
20 LET S = 0
30 FOR I = 1 TO N
40 LET S = S + I
50 NEXT I
60 PRINT S
```

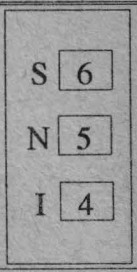
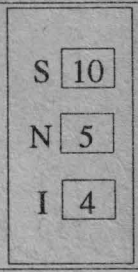
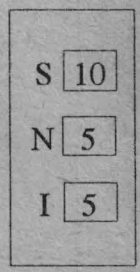
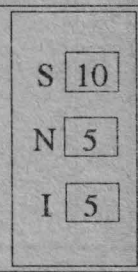
Paşii de program cu instrucţiunile care se execută, precum şi acţiunile corespunzătoare efectuate de calculator sînt oglindite în tabelul 1.

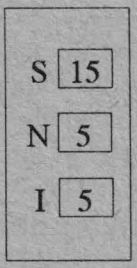
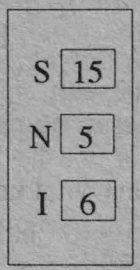
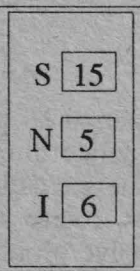
Momentul (paşi de program)	Instrucţiunea care se execută	Ce face calculatorul	Cum arată memoria după acţiunea calculatorului
1.	`10 INPUT N`	Calculatorul aşteaptă introducerea unui număr de la tastatură. Să presupunem că utilizatorul a introdus valoarea 5 şi apoi a acţionat **ENTER**. Valoarea 5 va fi depusă în locaţia de memorie cu numele N.	S N 5 I

2.	`20 LET S=0`	Calculatorul utilizează o locaţie de memorie numită S şi pune în ea valoarea 0.	
3.	`30 FOR I=1 TO N`	Calculatorul utilizează o locaţie de memorie numită I şi pune în ea valoarea 1.	
4.	`40 LET S=S+I`	Calculatorul trebuie să evalueze expresia din partea dreaptă a semnului egal, adică S + I. Pentru aceasta citeşte valorile 0 şi 1 din celulele de memorie S, şi respectiv I, adunarea lor realizîndu-se în unitatea aritmetică logică (UAL). Valoarea obţinută (1) se depune în locaţia de memorie S.	

5.	`50 NEXT I`	Calculatorul adaugă valorii din locaţia I o unitate (I = I + 1), adică trece la următorul I. Calculatorul citeşte valoarea din locaţia de memorie I şi o compară cu valoarea lui N (I > N ?). Cum valoarea lui I este 2, deci mai mică decît 5 atunci calculatorul transferă controlul la linia numărul 30.	
6.	`30 FOR I=1 TO N`	Calculatorul reia ciclul.	
7.	`40 LET S = S+I`	Calculatorul evaluează expresia S + I. El citeşte din locaţia de memorie S valoarea 1 şi din locaţia de memorie I valoarea 2, evaluînd deci expresia S + I la valoarea 3. Această valoare este depusă în locaţia de memorie S.	

8.	`50 NEXT I`	Calculatorul adaugă valorii din locaţia I o unitate (I = I + 1). Compară apoi această valoare cu valoarea din locaţia N şi, fiind mai mică (3), transferă controlul la linia cu numărul 30.	
9.	`30 FOR I=1 TO N`	Calculatorul reia ciclul	
10.	`40 LET S=S+I`	Deoarece S = 3 şi I = 3 rezultă că S + I = 3 + 3 = 6. Calculatorul depune în locaţia S valoarea 6.	
11.	`50 NEXT I`	Calculatorul trece la următoarea valoare din locaţia I. Aceasta este 4 fiind deci, mai mică decît cea din locaţia N care este 5. Calculatorul transferă controlul la linia cu numărul 30.	

12.	`30 FOR I=1 TO N`	Calculatorul reia ciclul.	
13.	`40 LET S=S+I`	Deoarece S = 6 şi I = 4 rezultă că S + I = 6 + 4 = 10. Calculatorul depune în locaţia S valoarea 10.	
14.	`50 NEXT I`	Calculatorul adaugă valorii din locaţia I o unitate (I = I +1), apoi compară valoarea din locaţia I cu valoarea din locaţia N. Valoarea din locaţia I nefiind mai mare decît cea din N (ele sînt egale), transferă controlul la linia cu numărul 30.	
15.	`30 FOR I=1 TO N`	Calculatorul reia ciclul.	

16.	`40 LET S=S+I`	Calculatorul evaluează expresia S + I. El citeşte din locaţia S valoarea 10 şi din locaţia de memorie I valoarea 5 evaluînd expresia S + I la valoarea 15. Această valoare este depusă în locaţia de memorie S.	
17.	`50 NEXT I`	Calculatorul adaugă valorii din locaţia I o unitate. Calculatorul compară valoarea din locaţia I cu valoarea din locaţia N şi aceasta fiind mai mare (I > N) nu mai transferă controlul la linia cu numărul 30, ci la linia următoare, care este linia cu numărul 60.	
18.	`60 PRINT S`	Calculatorul citeşte valoarea din locaţia de memorie S şi afişează această valoare. Pe ecran va apare valoarea 15.	

19.	—	Deoarece după linia 60 nu mai este nici o linie de program, calculatorul afişează un mesaj de terminare a programului: 0 OK 60: 1 ceea ce înseamnă că programul s-a terminat în linia 60 şi a fost executat fără nici o eroare.	S 15 N 5 I 6

tabel 1

Detalierea execuţiei programului pentru adunarea a N numere naturale

După cum se poate observa pentru executarea acestui program au fost necesari nu mai puţin de 19 paşi de program, rezultatul fiind cel corect (1 + 2 + 3 + 4 + 5 = 15). Iar pentru fiecare pas de program calculatorul a avut de executat una sau mai multe operaţii. De exemplu, pentru pasul 7, adică pentru executarea liniei `40 LET S = S + I` calculatorul a executat 4 operaţii:

1. a citit valoarea din locaţia S (care este 1)
2. a citit valoarea din locaţia I (care este 2)
3. a evaluat expresia S + I (1 + 2 = 3)
4. a depus valoarea obţinută (3) în locaţia S

Poate vi se pare monotonă şi plictisitoare funcţionarea în acest mod a calculatorului. Tocmai în aceasta constă şi tăria lui, şi anume, în faptul că poate executa foarte repede şi corect multe operaţii asemănătoare şi, de fapt, foarte simple. În speţă calculatorul, cu toate că în final poate să rezolve probleme grele şi să execute lucruri foarte interesante, nu "ştie" să facă decît cele 4 operaţii aritmetice.

Este bine ca pentru orice program (problemă) pe care îl dăm calculatorului să îl execute să ne imaginăm (în modul în care am făcut-o pentru programul care calculează suma primelor N numere naturale) felul în care va

funcţiona calculatorul şi care va fi evoluţia parametrilor programului (celulelor de memorie).

De obicei această *simulare* a funcţionării programului se face cu puţine valori. Dacă rezultatul va fi cel aşteptat atunci va fi de presupus că şi cu valori mai multe (reale) rezultatele vor fi corecte sau, cu alte cuvinte, programul "merge".

Dacă totuşi în cazul executării programului rezultatul nu este cel aşteptat atunci se poate proceda la *depanarea* programului prin întreruperea funcţionării acestuia în diferite momente şi prin "fotografierea" conţinutului locaţiilor de memorie (variabilelor) pentru a se vedea dacă evoluţia acestora este corectă. Întreruperea funcţionării programului se poate face prin inserarea unor instrucţiuni STOP după calculul în cadrul programului a valorilor parametrilor sau, pur şi simplu, prin întreruperea programului în mod comandă (prin BREAK sau STOP). Pentru "fotografierea" conţinutului variabilelor se pot insera după instrucţiunile STOP instrucţiuni de afişare (PRINT) a valorilor parametrilor care ne interesează. În cazul întreruperii programului cu BREAK sau STOP se vor da în mod direct comenzi de afişare (PRINT) a valorilor parametrilor care ne interesează.

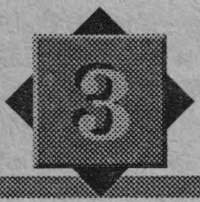
De exemplu, să presupunem că întrerupem programul nostru cu BREAK şi apoi citim valorile parametrilor cu:

```
PRINT N —> obţinem valoarea 5 (corect)
PRINT I —> (să presupunem că se afişează valoarea 3)
PRINT S —> obţinem valoarea 6 (corect)
```

Concluzionăm că programul "merge" deoarece pentru primele 3 numere naturale (I=3) suma este 6.

În ambele cazuri de întrerupere, după afişarea valorilor, execuţia programului se poate continua cu comanda CONT (CONTINUE), atîta timp cît valorile parametrilor nu au fost modificate. Vizualizînd astfel evoluţia parametrilor ne vom putea da seama în ce loc programul "nu merge bine", adică, de fapt, în ce loc valorile parametrilor nu mai corespund cu valorile aşteptate de noi.

Pentru unele calculatoare (nu şi pentru cele compatibile Sinclair Spectrum din păcate) există posibilitatea ca printr-o anumită comandă (opţiune) execuţia programului să fie însoţită de afişarea liniilor de program care se execută în acel moment. Această operaţie se numeşte **trasarea programului** iar comanda este TRACE (TRON pentru PC în cadrul sistemului GWBASIC). Se oferă o modalitate utilă de a se vizualiza în acelaşi timp atît rezultatele cît şi numerele de linie respective ale căror instrucţiuni conduc la aceste rezultate, fiind deci, *o unealtă de depanare a programelor.*



Rezolvarea problemelor cu calculatorul

Ne propunem să vă învăţăm cum să rezolvaţi probleme de matematică (dar nu numai) folosind ca instrument de lucru calculatorul. Bineînţeles că vă vom propune şi probleme de alt tip care se referă la alte domenii decît matematica, acestea incluzîndu-se în lucrare deoarece rezolvarea lor necesită de asemenea cunoştinţe şi metode matematice. Printre aceste probleme se pot enumera *problemele de grafică* şi cele de tip *jocuri logice*.

Desigur în abordarea problemelor propuse vom căuta iniţial o rezolvare uzuală (clasică) iar apoi vom încerca transpunerea acestei rezolvări în scopul tratării prin intermediul calculatorului. În situaţia în care nu găsim o rezolvare uzuală convenabilă se propune o nouă metodă mai apropiată de calculator. În ambele cazuri calculatorul este privit ca un instrument care ajută la rezolvarea problemei în special prin preluarea unor calcule repetate şi plictisitoare care ar fi mari consumatoare de timp pentru rezolvitor.

Deci, în primul rînd vom studia cum rezolvă calculatorul problemele. In acest scop va trebui să găsim o metodă pentru soluţionarea problemei, iar apoi să concepem un program cu care se va rezolva efectiv problema.

Metoda de a rezolva o problemă se numeşte **algoritm**.

Algoritmi

Algoritmul este reprezentat de o mulţime de reguli folosite într-o anumită succesiune, cu ajutorul cărora se poate obţine soluţia unei probleme prin executarea unui număr finit de operaţii. Aceasta este o definiţie şi nu este necesară memorarea ei. Mai importantă este înţelegerea ei, iar cum înţelegerea se realizează cel mai bine prin analogii, puteţi să vă imaginaţi că o reţetă dintr-o carte de bucate este, de fapt, un algoritm. De ce? Deoarece o reţetă reprezintă o mulţime de reguli care trebuie urmate, adică folosite într-o anumită succesiune pentru a obţine un anumit fel de mîncare sau o prăjitură, adică soluţia problemei.

Trebuie să remarcăm faptul că deseori obţinerea algoritmului (sau reţetei) nu este o treabă foarte uşoară. De exemplu pentru a se ajunge la reţetele unora dintre cele mai gustoase feluri de mîncare s-au făcut nenumărate încercări şi s-a folosit experienţa acumulată de gospodine şi bucătari timp de zeci (uneori chiar sute) de ani. Avînd însă reţeta (sau algoritmul) nu va fi foarte dificil de a obţine prăjitura dorită (sau *soluţia problemei*).

O metodă prin care se pot găsi algoritmi pentru rezolvarea problemelor este **metoda de abordare de sus în jos a problemelor**. Prin această metodă problema este descompusă în mai multe subprobleme şi, pe măsură ce aceste subprobleme vor fi mai mici, ele vor semăna din ce în ce mai mult cu paşii logici care pot fi reprezentaţi prin operaţii simple pe care le poate efectua calculatorul. Metoda este eficientă, dar, pentru rezolvarea problemelor care se descompun în foarte multe subprobleme, deseori este dificil de urmărit şirul logic al algoritmului. În scopul uşurării urmăririi şirului logic în aceste situaţii se utilizează scheme cum sînt diagramele de tip arbore. Prin *diagramele de tip arbore* se realizează o descriere grafică a subprogramelor şi paşilor logici de urmat pentru rezolvarea problemei.

Iată un exemplu de început de diagramă de tip arbore realizată în scopul rezolvării problemei de realizare a operaţiunilor logice cu două mulţimi A şi B (diferenţă, intersecţie şi reuniune) problemă care va fi reluată într-un capitol dedicat realizării acestui program (figura 2).

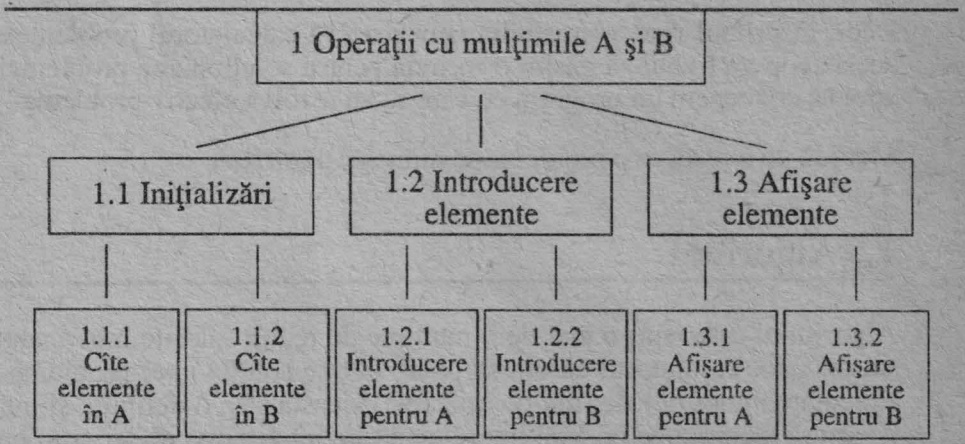


fig.2

Diagramă tip arbore pentru realizarea operaţiilor logice diferenţă, intersecţie şi reuniune a două mulţimi

Pentru exemplificarea metodei de abordare prin care problemele se pot descompune ("sparge") în subprobleme mai mici, vom evidenţia faptul că fiecare din subproblemele 1.2.1. şi 1.2.2. mai pot conţine cîte o sarcină 1.2.1.1. şi 1.2.2.1. de verificare a elementelor introduse pentru mulţimea A şi respectiv B.

După realizarea unei astfel de diagrame se caută să se obţină o descriere a algoritmului într-un limbaj succint astfel încît paşii de urmat să fie uşor de înţeles şi, în acelaşi timp, uşor de tradus în instrucţiuni pentru calculator, operaţie care se mai numeşte descrierea în **pseudocod** a algoritmului. Acest lucru va fi mult mai uşor de realizat din descrierea sarcinilor ce rezultă din diagrama de tip arbore. Bineînţeles, pentru a produce programul va trebui ca fiecare secţiune a pseudocodului să fie tradusă (translatată) în echivalentul său din limbaj BASIC. Iată cum arată descrierea în paşi logici a diagramei de structură prezentate:

Pentru a distinge mai bine problemele şi subproblemele de rezolvat acelaşi lucru îl vom scrie astfel (mai structurat):

- 1 Operaţii cu mulţimile A şi B
  - 1.1 Iniţializări.
    - 1.1.1 Cîte elemente în A.
    - 1.1.2 Cîte elemente în B.
  - 1.2 Introducere elemente.
    - 1.2.1 Introducere elemente pentru A.
      - 1.2.1.1 Verificare elemente introduse pentru A (să nu se repete).
    - 1.2.2 Introducere elemente pentru B.
      - 1.2.2.1 Verificare elemente introduse pentru B (să nu se repete)
  - 1.3 Afişare elemente.
    - 1.3.1 Afişare elemente din A.
    - 1.3.2 Afişare elemente din B.

Rezolvarea problemei

☞ *Găsirea metodei de a rezolva o problemă este, de obicei, cea mai dificilă parte a programării. De aceea, de la început este necesară organizarea şi planificarea activităţilor.*

Pentru a produce algoritmul de rezolvare a unei probleme vor trebui parcurse mai multe etape pe care le descriem în continuare, folosind pentru acomodare tot o descriere în paşi logici:

- 1 Privire generală asupra problemei
  - 1.1 Ce rezultate urmăreşti să obţii
  - 1.2 Ce date este necesar să introduci
  - 1.3 Ce operaţii vei efectua cu aceste date
- 2 Analiza problemei
  - 2.1 Analiza problemei pentru identificarea modului în care poate fi rezolvată cu calculatorul
  - 2.2 Identificarea formulelor, datelor şi relaţiilor pe care le vei folosi
  - 2.3 Identificarea tuturor datelor implicate
- 3 Proiectarea algoritmului de rezolvare a problemei
  - 3.1 Descompunerea problemei în subprobleme
  - 3.2 Utilizarea unei diagrame de structură (de exemplu de tip arbore) ca ajutor
  - 3.3 Identificarea tipului modulelor sau părţilor ca fiind de introducere, prelucrare sau extragere
  - 3.4 Utilizarea structurilor de control

Pentru evidenţierea celor mai importante structuri de control se pot utiliza **schemele logice**. Acestea reprezintă o metodă utilizată în realizarea de programe care constă din *blocuri logice* legate între ele. Fiecare *bloc* reprezentat prin diverse figuri geometrice, cu o anumită semnificaţie, descrie ce realizează programul în acel punct, *schema logică* descriind astfel fluxul controlului în algoritm şi deci în program. Schema logică evi-

denţiază deci, cele mai importante *structuri de control* şi uşurează astfel codificarea algoritmului în instrucţiuni BASIC, constituind, de asemenea, şi o parte importantă a documentaţiei programului.

Să exemplificăm o schemă logică care ar corespunde sarcinii 1.2.1.1. din pseudocod, şi anume, verificarea elementelor introduse pentru mulţimea A (să nu se repete elementele). Algoritmul de rezolvare a acestei sarcini este următorul: fiecare element din mulţimea A introdus, **a(i)** se compară cu toate elementele din stînga sa, adică, cu **a(1), a(2) .... a(i-1)**. Dacă se găseşte că este egal cu unul din elementele din stînga lui atunci se afişează mesajul " Valoare eronată. Introduceţi altă valoare", iar fluxul programului va merge în punctul în care se realizează introducerea valorii **a(i)**, punct care aparţine sarcinii 1.2.1. Dacă elementul **a(i)** nu este egal cu nici unul din stînga sa atunci valoarea introdusă este corectă şi se trece la introducerea valorii următoare, **a(i+1)**. Schema logică arată astfel (figura 3):

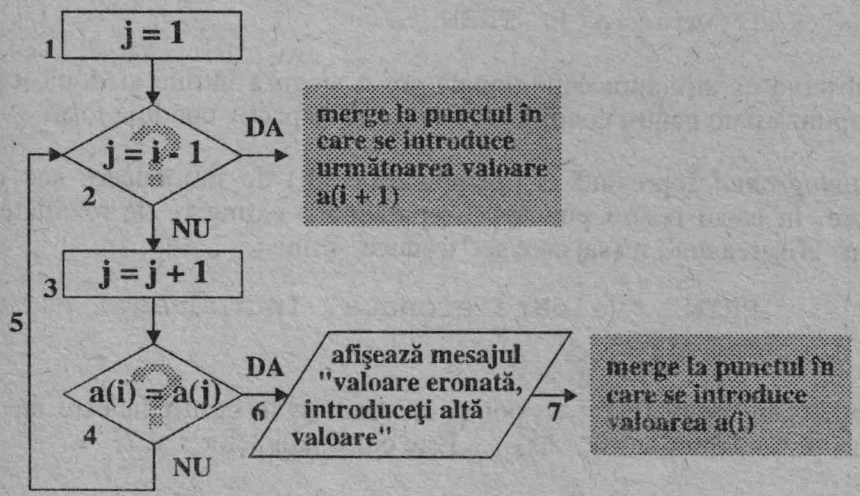


fig.3

Schema logică pentru verificarea elementelor introduse şi care aparţin unei mulţimi date

Figurile geometrice din desen se mai numesc **blocuri.**

Dreptunghiurile reprezintă *structuri de prelucrare*. În cazul nostru dreptunghiurile 1 şi 3 reprezintă *operaţii de atribuire*.

Acum putem să le "traducem" lesne în instrucţiuni BASIC. Astfel pentru punctul 1 se poate introduce o linie de program

```
LET j=1
```

iar pentru 3

```
LET j=j+1
```

Romburile reprezintă *structuri condiţionale*. Astfel punctul 2 se poate "traduce" în BASIC prin:

```
IF j=i-1 THEN GO TO ...
```

iar punctul 4:

```
IF a(i)=a(j) THEN ...
```

Se observă că structura condiţională are o singură intrare şi două ieşiri corespunzătoare pentru condiţie *adevărată* şi respectiv condiţie *falsă*.

Paralelogramul reprezintă o operaţie (operaţii) de introducere sau extragere. În cazul nostru punctul 6 reprezintă o extragere de rezultate şi anume afişarea unui mesaj care se "traduce" prin:

```
PRINT "Valoare eronata. Introduceti alta
valoare"
```

Săgeţile care arată fluxul se pot şi ele traduce în cazuri în care au loc salturi prin instrucţiuni `GO TO` .... Este cazul punctelor 5 şi 7.

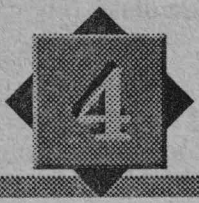
Deci următoarele etape pentru rezolvarea problemei vor fi:

- 4. Realizarea programului de rezolvare a problemei (codificarea în limbaj BASIC sau alt limbaj de programare)
- 5. Testarea (verificarea) programului
  - 5.1. Verificarea sintactică (cu îndepărtarea erorilor dacă acestea există)

⇨ 5.2. Verificarea logică cu mai multe seturi de date (cu îndepărtarea erorilor dacă acestea există)

⇨ 6. Obţinerea rezultatelor.

☞ *Lucrarea de faţă propune spre rezolvare numeroase probleme de matematică. În scopul clarificării modului de prezentare a rezolvării se pune mai mult accent pe analiza problemei, descrierea succintă (asemănătoare cu pseudocodul) a ideilor pentru rezolvarea ei şi prezentarea programului. Din acest punct de vedere rezolvarea oferită în lucrare trebuie privită ca un îndrumar în vederea rezolvării problemei, recomandîndu-se ca atunci cînd se lucrează practic să nu se sară paşii logici descrişi în acest capitol. Bineînţeles că soluţiile (programele) pe care le obţineţi pot diferi de cele oferite în lucrare. Important este ca ele să fie corecte, adică prin intermediul lor să se obţină rezultatele corecte.*



Probleme cu numere

Multiplii numerelor

Să se realizeze un program cu care să se determine multiplii unui număr.

Pentru calculul multiplilor unui număr se pot utiliza două metode:
a) calculul multiplilor prin adunare
b) prin înmulţire.

Pentru metoda a) se memorează valoarea numărului ai cărui multipli se doreşte să fie calculaţi într-o variabilă **M** (`LET M = A`). Valoarea **M** va reprezenta, de fapt, valoarea primului multiplu. Următorii multipli se determină prin acelaşi procedeu, şi anume se adună la numărul care reprezintă multiplul actual o valoare (pas) care reprezintă chiar valoarea numărului iniţial căruia dorim să-i calculăm multiplii. Programul P1 este:


P1.

```
5  REM **  program de calcul al   **
6  REM ** multiplilor unui numar **
7  REM **       prin adunare      **
10 INPUT A
20 LET M = A
30 PRINT M
40 LET M = M + A
50 GOTO 30
```

Astfel linia 40 este cea în care se calculează multiplii şi care evidenţiază metoda folosită şi anume aceea a adunării. Să presupunem că introducem la executarea programului valoarea 7. Se vor afişa la nesfîrşit multiplii

numărului introdus, adică 7, 14, 21, 28, 35, 42,... . Pentru oprirea programului se va acţiona

BREAK (**CS + SPACE**) pentru HC
CTRL + PAUSE pentru PC.

Pentru metoda b) multiplii se vor determina prin înmulţirea numărului iniţial cu 1, 2, 3 şi aşa mai departe. De exemplu, pentru determinarea multiplilor lui 5, aceştia se obţin prin 5×1=5, 5×2=10, 5×3=15 şi aşa mai departe. În acest scop va trebui să introducem o variabilă contor (C) care să ia valorile 1, 2, 3, ... şi cu care să înmulţim numărul iniţial pentru calculul fiecărui multiplu. Programul P2 este:

P2.

```
5  REM **  program de calcul al   **
6  REM ** multiplilor unui numar **
7  REM **      prin inmultire     **
10 INPUT A
20 LET C = 1
30 LET M = A * C
40 PRINT M
50 LET C = C + 1
60 GOTO 30
```

Linia 30 este cea în care se calculează multiplii şi care evidenţiază metoda folosită şi anume cea a înmulţirii. La fel ca în cazul programului P1, introducînd valoarea 7 se vor afişa multiplii lui unul sub altul: 7, 14, 21, 28, 35, 42,... .

Să se modifice programele de calcul al multiplilor unui număr astfel încît fiecare multiplu să fie afişat împreună cu numărul de ordine respectiv (al cîtelea multiplu).

Pentru cazul b) trebuie doar modificată linia 40 astfel:

```
40 PRINT C , M
```

Pentru cazul a) în vederea afişării numărului de ordine trebuie introdusă o variabilă contor I după modelul folosit în cazul b).

Se adaugă deci liniile:

```
25 LET I = 1
45 LET I = I + 1
```

linia 30 se modifică astfel:

```
30 PRINT I , M
```

La execuţia ambelor programe se vor afişa atît multiplii cît şi numărul care reprezintă ordinea lor. De exemplu, dacă se introduce valoarea 7 se va obţine:

```
1    7
2    14
3    21
4    28
⋮    ⋮
```

Dacă numărul de multipli pe care dorim să-i determinăm este cunoscut atunci programul se va putea realiza folosind un ciclu FOR-NEXT (programul P3):

P3.

```
5  REM **  program de calcul al  **
6  REM ** multiplilor unui numar **
7  REM **     cu ciclu FOR-NEXT   **
10 INPUT A
15 REM se introduce numarul de
16 REM multipli de calculat
20 INPUT N
30 FOR I = 1 TO N
40 LET M = A * I
50 PRINT I, M
60 NEXT I
```

În acest caz se va afişa un anumit număr de multipli ai unui număr. De exemplu, dacă se introduc valorile 17 şi 4 se vor afişa primii 4 multipli ai numărului 17.

Cel mai mic multiplu comun

Cum se poate determina cu ajutorul programului cel mai mic multiplu comun (CMMMC) a două numere?

Va trebui mai intîi să introducem cele două numere (**A** şi **B**) şi apoi să generăm multiplii acestor două numere. Generarea multiplilor o putem face înmulţind fiecare din numerele **A** şi **B** cu 1, 2, 3 şi aşa mai departe folosind metoda de calcul a multiplilor prin înmulţire (cazul b). Apoi vom afişa pe o coloană numărul de ordine (pentru a şti al cîtelea multiplu este), iar pe următoarele două coloane multiplii respectivi (**M1** şi **M2**). Programul P4 va arăta astfel:

P4.

```
5  REM ** program de calcul al CMMMC **
6  REM **          al doua numere      **
10 INPUT A
20 INPUT B
30 LET I = 1
40 LET M1 = A * I
50 LET M2 = B * I
60 PRINT I; " "; M1; " "; M2
70 LET I = I + 1
80 GOTO 40
```

Se vor genera multipli pînă cînd vom opri programul. De exemplu, introducînd numerele 3 şi 4 (pentru A şi respectiv B) vom putea citi din cele afişate pe ecran că al 7-lea multiplu al lui 3 este 21 iar al lui 4 este 28. Cînd s-a umplut un ecran întreg cu multipli şi nu mai avem nevoie de generarea altor multipli vom opri programul acţionînd

tasta **N** sau **BREAK** (**CS + SPACE**) pentru HC
CTRL + PAUSE pentru PC.

Cum vom identifica cel mai mic multiplu comun al celor două numere?

Simplu! Ne uităm pe coloana a 2-a şi a 3-a, iar primul număr pe care îl vom identifica ca fiind în ambele coloane va reprezenta CMMMC, deoa-

rece el este multiplu atît pentru primul număr (A), deoarece este în a 2-a coloană, cît şi pentru al doilea număr (B), deoarece este în a 3-a coloană şi, în plus, el este primul dintre multipli (deci cel mai mic) care este acelaşi pentru ambele numere. De exemplu, pentru A=3 şi B=4, vom observa că numărul 12 este afişat pe coloana a 2-a (al 4-lea multiplu al lui 3) şi, acelaşi număr este afişat şi pe coloana a 3-a (al 3-lea multiplu al lui 4). Deci 12 reprezintă CMMMC pentru 3 şi 4.

Ca alt exemplu, introducînd numerele 12 şi 42 vom observa numărul 84 afişat şi pe coloana a 2-a (al 7-lea) şi pe coloana a 3-a (al 2-lea). El reprezintă CMMMC pentru 12 şi 42.

Dacă dorim să determinăm CMMMC pentru mai multe numere putem să afişăm multipli pe mai multe coloane şi, în mod similar, primul număr care va fi afişat în toate coloanele (cu excepţia primei care nu exprimă multipli) va reprezenta CMMMC al numerelor respective.

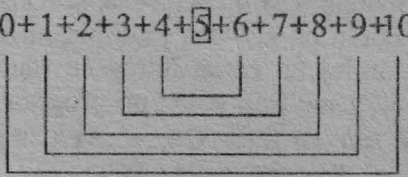
Sume şi produse de numere

Care este suma primelor 10 numere naturale ? Dar a primelor 100 de numere ?

Desigur, pentru suma primelor 10 numere naturale putem calcula 1 + 2 + 3 + ... şi aşa mai departe pînă la 10. Mai ingenios însă putem calcula observînd că se pot asocia 0 cu 10, 1 cu 9, 2 cu 8, 3 cu 7 şi 4 cu 6 toate aceste sume fiind 10 (vezi figura 4):

fig.4

Reprezentarea grafică a metodei de adunare a primelor 10 numere naturale



Avem 5 sume de 10 iar 5 din mijloc rămîne liber, deci: $5 \times 10 + 5 = 55$.

Puteţi da un răspuns aproximativ, dar rapid (fără să faceţi efectiv calculul) la întrebarea care este suma primelor 100 de numere?

Putem folosi aceeaşi regulă, adică să adunăm pe 0 cu 100, pe 1 cu 99 ... şi vom avea 50 de sume (de la 0 la 49) de cîte 100 iar 50 va ramîne singur, deci: $100 \times 50 + 50 = 5050$.

Răspunsul vostru la întrebarea precedentă a fost cît de cît apropiat de cel real? *Prin compararea rezultatului sumei primelor 10 numere cu cel obţinut pentru 100 de numere, puteţi spune cît va fi rezultatul sumei primelor 1000 de numere?*

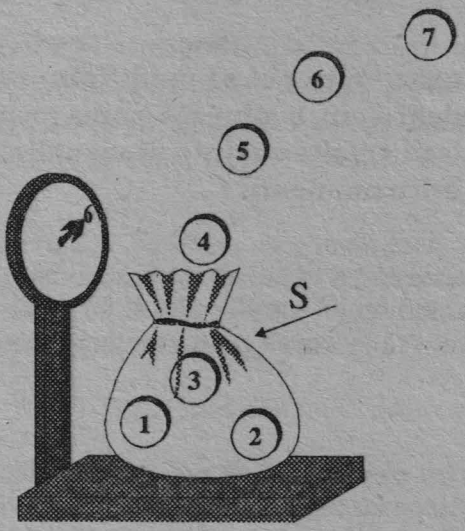
Desigur, aceasta va fi 500 500, iar al sumei primelor 10 000 va fi 50 005 000.

Cum se poate face un program astfel încît să realizăm calculul unor astfel de sume cu calculatorul?

Pentru aceasta trebuie să imaginăm un algoritm, adică să-l învăţăm pe calculator o regulă pentru a calcula astfel de sume.

fig.5

Adunarea numerelor naturale



Vom proceda în felul următor: luăm un sac (sau, dacă vreţi, o celulă de memorie) pe care îl botezăm S (îi punem o etichetă S). La început el este gol (deci `LET S = 0`) şi începem să punem în el cîte un număr (`FOR I = 1 TO N`). De fiecare dată suma va fi egală cu suma existentă pînă în

acel moment în sac la care se mai adaugă numărul respectiv (deci LET S = S + I). După ce am pus toate numerele ne uităm să vedem ce sumă avem în sac (PRINT S) şi astfel obţinem rezultatul dorit. Programul P5 va fi:


P5.

```
5 REM ** program de calcul al sumei **
6 REM ** primelor N numere naturale **
10 INPUT N
20 LET S = 0
30 FOR I = 1 TO N
40 LET S = S + I
50 NEXT I
60 PRINT S
```

Într-adevăr, acum, avînd afişate toate numerele putem verifica foarte uşor rezultatele pentru suma primelor 1000 sau 10 000 sau 100 000 de numere şi să regăsim regula de formare a numărului care reprezintă suma.

Cum va trebui să modificăm programul astfel încît să vedem cum evoluează suma pentru fiecare număr adunat (sau cu alte cuvinte să vizualizăm suma care este în sac în orice moment)?

Pentru aceasta va trebui să intercalăm în programul P5 o linie de vizualizare a conţinutului sacului (PRINT) după fiecare introducere a unui număr în sac (să facem sacul transparent). Deci introducem linia

```
45 PRINT S
```

iar în acest caz linia 60 nu mai are rost, deoarece ultima sumă afişată va fi chiar suma care este în sac după ce au fost introduse toate numerele. Dacă dorim să ştim de fiecare dată atît suma cît şi numărul de numere care adunate dau suma respectivă va trebui să vizualizăm, în afară de sumă şi numărul respectiv, care este, de fapt, I. Deci linia va fi :

```
45 PRINT I , S
```

Verificaţi acum cu programul de calcul al primelor N numere naturale următoarea formulă:

$$S = \frac{N \cdot (N+1)}{2}$$

Vom observa că formula este valabilă pentru 1, pentru 2 şi pentru orice număr I. Pentru această verificare vom modifica în programul P5 linia 45 astfel:

```
45 PRINT I; " "; S; " "; I * (I + 1)/2
```

şi vom vedea că pe fiecare rînd, adică pentru orice I se vor afişa două numere egale.

Care este suma primelor N numere impare ? Cum trebuie modificat programul pentru calculul sumei primelor N numere naturale astfel încît să calculeze numai suma numerelor impare ? Există şi în acest caz o regulă?

Pentru a calcula suma numerelor impare nu avem altceva de făcut decît să modificăm linia 30 făcînd calculul sumei cu numere din 2 în 2 (`STEP 2`). De asemenea trebuie să avem în vedere că în intervalul [0, N] jumătate din numerele naturale (N/2) sînt impare, iar restul pare. Deci:

```
30 FOR I = 1 TO N STEP 2
```

Pentru primele 10 numere, deci, pentru primele 5 numere impare obţinem suma 25, pentru primele 100 numere obţinem suma celor impare 2 500, iar pentru 1 000 obţinem 250 000.

Evident, în acest caz regula este mai simplă şi anume, de adăugare a două zerouri după grupul de cifre 25 pentru fiecare ordin de mărime. Deci, pentru 10 000 suma va fi 25 000 000.

Care este suma primelor N numere pare ? Care este în acest caz regula?

Singura modificare faţă de programul precedent P5 este faptul că se va începe contorizarea de la 2 şi nu de la 1 ca în cazul numerelor impare. În acest caz linia 30 va arăta astfel:

```
30 FOR I = 2 TO N STEP 2
```

Pentru primele 10 numere, deci pentru primele 5 numere pare (fără să-l includem pe 0 deoarece am început cu 2) suma este 30, pentru 100 este 2 550, iar pentru 1 000 este 250 500. Pentru 10 000 suma va fi 2 500 500. Deci se adaugă cîte un 0 atît grupului de cifre condus de 25 cît şi celui condus de 5.

Cum se poate calcula media aritmetică a N numere?

Ştim că media aritmetică a N numere se calculează împărţind suma numerelor la N. Pentru aceasta trebuie să ştim de la început cîte numere avem (linia 10). Se observă că în acest caz numerele pe care le avem de adunat nu le mai cunoaştem dinainte ci trebuie să le introducem. Acest lucru îl vom realiza prin inserarea liniei 35 în programul P5:

```
35 INPUT A
```

De exemplu, pentru a calcula media trimestrială la şcoală vom introduce separat mediile pentru fiecare materie. Linia 40 se va modifica în mod corespunzător avînd acum de adăugat de fiecare dată la vechea sumă noul număr introdus. În final nu vom fi interesaţi de suma numerelor ci de media lor care se obţine prin împărţirea sumei la N. În consecinţă în linia 60 vom afişa rezultatul S/N. Programul P6 va fi :

P6.

```
5 REM ** program de calcul al **
6 REM ** mediei aritmetice a  **
7 REM **       N numere       **
```

```
10 INPUT N
20 LET S = 0
30 FOR I = 1 TO N
35 INPUT A
40 LET S = S + A
50 NEXT I
60 PRINT S/N
```

Care este suma pătratelor primelor N numere naturale? Puteţi estima cam cît este, de exemplu, suma pătratelor primelor 100 de numere ? Există vreo regulă de formare a numărului care reprezintă suma?

Ştim că pătratul unui număr reprezintă numărul obţinut prin înmulţirea numărului cu el însuşi, sau, cu alte cuvinte, aria unui pătrat a cărui latură este egală cu însuşi numărul.

Pentru a calcula suma pătratelor numerelor va trebui să modificăm linia 40 în programul P5 de calcul al sumei numerelor în felul următor:

```
40 LET S = S + I * I
```

Pentru suma pătratelor primelor 10 numere vom obţine 385, pentru primele 100 338 350, iar pentru primele 1 000 vom obţine 333 833 500. Vă lăsăm să formulaţi singuri regula de obţinere a numărului.

Se poate verifica şi următoarea formulă:

$$S = \frac{N \cdot (N+1) \cdot (2N+1)}{6}$$

În acest scop vom introduce din nou linia 45 în felul următor:

```
45 PRINT I;" ";S;" ";I*(I+1)*(2*I+1)/6
```

Vom observa că pe fiecare rînd afişat, adică pentru orice **I**, ultimele două numere sînt egale.

Care este suma cuburilor primelor N numere ? Există vreo regulă de formare a numărului care reprezintă această sumă?

Cubul unui număr este reprezentat de numărul înmulţit de 3 ori cu el însuşi sau, cu alte cuvinte, volumul unui cub a cărui muchie este egală chiar cu numărul. Bineînţeles, pentru a calcula suma numerelor, linia 40 va avea următoarea formă:

```
40 LET S = S + I * I * I
```

Pentru primele 10 numere suma va fi 3 025, pentru 100, 25 502 500 iar pentru 1 000, 250 500 250 000. Vă lăsăm să formulaţi singuri regula de obţinere a numărului sumă.

Care este suma inverselor primelor N numere naturale ? Puteţi spune aproximativ cam cît este suma inverselor primelor 100 de numere?

Suma inverselor primelor **N** numere naturale va fi:

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\ldots+\frac{1}{N}$$

Observăm că numitorul fracţiilor care se adună este din ce în ce mai mare, deci şi numărul mai mic. Pentru a realiza calculul vom modifica linia 40 a programului P5 de calcul al sumei primelor **N** numere astfel:

```
40 LET S = S + 1/I
```

Vom nota că suma inverselor primelor 2 numere este 1,5 , suma inverselor primelor 3 numere este 1,83333 iar suma inverselor primelor 10 numere este 2,9289683. De asemenea, vom constata că suma inverselor primelor 100 de numere trece doar cu puţin de 5, iar suma inverselor a 1000 de numere este ceva mai mare de 7.

Care este suma inverselor pătratelor numerelor ? Puteţi spune aproximativ cam cît este suma inverselor pătratelor primelor 1000 de numere?

Suma inverselor pătratelor numerelor va fi:

$$1+\frac{1}{2\cdot 2}+\frac{1}{3\cdot 3}+\ldots+\frac{1}{N\cdot N}$$

Deoarece numitorul creşte mai repede şi numerele adunate vor fi mult mai mici. Pentru program vom modifica linia 40 astfel:

```
40 LET S = S + 1 / (I*I)
```

Veţi constata cu mirare (poate) că suma inverselor pătratelor primelor 1000 de numere este de numai 1,6439346, iar pentru 10 000 de numere suma va diferi doar cu foarte puţin de precedenta (1,6448341).

Avînd la dispoziţie programele de calcul al sumelor de numere, completaţi următorul tabel:

N	Suma primelor N numere naturale	Suma primelor N numere pare	Suma primelor N numere impare	Suma pătratelor primelor N numere naturale	Suma cuburilor primelor N numere naturale	Suma inverselor primelor N numere naturale
10	55	110	121	385	3025	2,9289683
100	5050	10100	10201	338350	25502500	. 5,
1000						
10000						
⋮	$\frac{N\cdot(N+1)}{2}$			$\frac{N\cdot(N+1)\cdot(2N+1)}{6}$		

Încercaţi să regăsiţi anumite reguli deja ştiute şi să descoperiţi altele.

De exemplu: este normal ca suma numerelor pare să fie dublă faţă de suma numerelor naturale ?

Da, deoarece fiecare număr care intră în componenţa sumei numerelor pare este dublu faţă de numărul omolog din cadrul sumei numerelor naturale.

De asemenea se poate observa că fiecare din numerele care intră în componenţa sumei numerelor impare este cu o unitate mai mare decît numărul omolog din cadrul sumei numerelor pare.

Dacă şahul Persiei ar fi avut calculator...

Inventatorul jocului de şah a cerut şahului ca recompensă, să-i dea pentru prima căsuţă a tablei de joc o boabă de grîu, pentru a 2-a, 2 boabe şi aşa mai departe, la fiecare căsuţă să-i dubleze numărul de boabe (vezi figura 6). Şahului i s-a părut o recompensă modestă. Calculaţi cu ajutorul calculatorului cîte vagoane de grîu trebuia să-i dea şahul recompensă inventatorului ştiind că 1000 de boabe de grîu = 1 kg şi că 1 vagon = 10 t.

Va trebui să calculăm o sumă de forma:

$$1+2+2\times2+2\times2\times2+...$$

adică:

$$2^0+2^1+2^2+2^3+2^4+...+2^{63}$$ (vezi figura 6)

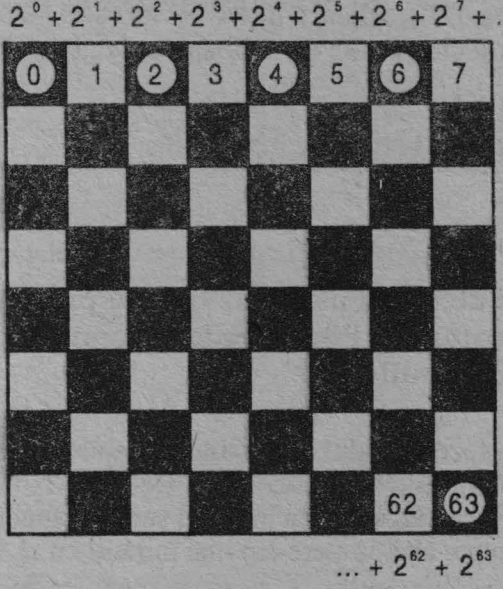
Ultimul termen al sumei este 2^{63} deoarece am început numărarea căsuţelor de la 0 (de la 0 la 63 sînt 64 de căsuţe).

Pentru calculul sumei boabelor de grîu va trebui să modificăm linia 40 din programul P5 de calcul al sumei primelor N numere astfel:

```
40 LET S = S + 2 ^ I
```

fig.6

Reprezentarea grafică a metodei de calcul a recompensei inventatorului jocului de şah



Nu uităm să modificăm şi linia 30 deoarece acum avem de repetat operaţia de adunare de un număr dat de ori şi anume de 64 de ori (de la 0 la 63), adică:

```
30 FOR I = 0 TO 63
```

Bineînţeles nu vom mai avea nevoie de linia 10 deoarece acum **N** este definit în linia 30 şi anume este egal cu 63. În final se va afişa numărul de vagoane. Acesta se va obţine prin împărţirea numărului de boabe rezultat la 1000, se rezultînd numărul de kilograme, apoi prin împărţirea acestui rezultat la 1000, obţinîndu-se astfel numărul de tone şi, în sfîrşit, prin împărţirea rezultatului obţinut la 10 se obţine numărul de vagoane. Programul P7 va arăta astfel:

P7.

```
5 REM **     program de calcul al     **
6 REM ** recompensei inventatorului **
7 REM **          jocului de sah      **
20 LET S = 0
30 FOR I = 0 TO 63
40 LET S = S + 2^I
50 NEXT I
60 PRINT S/1000/1000/10
```

Vom obţine ca rezultat 1.8446744 E+12 (E+12 înseamnă "înmulţit cu 10 la puterea 12"), adică 1 844 674 400 000 vagoane, adică un trilion, 844 miliarde şi cîteva sute de milioane de vagoane !

O problemă de intuiţie

Puteţi estima rapid cît este produsul primelor 10 numere naturale? Puteţi crede că numărul respectiv este mai mare de 3 milioane?

Pentru a face calculul produsului cu ajutorul calculatorului vom iniţializa o variabilă P (de la produs) cu 1. De ce cu 1 ? Deoarece dacă am iniţializa această variabilă cu 0 (ca în cazul sumei) atunci tot timpul produsul va fi 0 indiferent ce alte numere am mai înmulţi cu el. Programul P8 va fi:

P8.

```
5 REM ** program de calcul al   **
6 REM ** produsului primelor N **
7 REM **    numere naturale     **
10 INPUT N
20 LET P = 1
30 FOR I = 1 TO N
40 LET P = P * I
50 NEXT I
60 PRINT P
```

Pentru primele 10 numere vom obţine 3 628 800, iar pentru 100 de numere calculatorul ne va răspunde că este un număr prea mare pentru a-l putea calcula.

Care este produsul inverselor primelor N numere ?

În acest caz cu cît N va fi mai mare cu atît numărul cu care se va înmulţi va fi mai mic decît unitatea (deoarece numărul creşte) şi va avea ca efect

micşorarea numărului rezultat. Dar acesta nu va putea atinge practic niciodată 0. Linia 40 din programul P8 se va modifica astfel :

```
40 LET P = P / I
```

Pentru produsul inverselor primelor 10 numere obţinem 2.7557319E- 7 (E-7 înseamnă înmulţit cu 10 la puterea -7) adică obţinem de fapt numărul 0.0000002755.

Pentru produsul inverselor primelor 100 de numere obţinem 0 ! Rezultatul este incorect; calculatorul lucrează cu o anumită precizie şi în limita ei identifică numerele (în speţă, 0 cu $9 \cdot 10^{-9}$).

Să se realizeze un program cu care să se calculeze expresia:

$$1+\frac{1}{1\cdot 2}+\frac{1}{1\cdot 2\cdot 3}+\ldots+\frac{1}{1\cdot 2\cdot 3\cdot \ldots \cdot N}$$

Vom folosi atît metoda pentru calculul sumei (în primul rînd este vorba de o sumă de numere) cît şi metoda pentru calculul produsului a N numere. Este vorba, deci, de o *sumă de produse de numere*.

La fel cum pentru a construi o casă avem mai întîi nevoie de părţile ei componente (uşi, ferestre etc.), tot aşa pentru a calcula o sumă de produse vom calcula mai întîi produsele şi apoi, pe baza lor suma. Programul P9 este:

P9.

```
5 REM ** program de calcul al **
6 REM ** unei sume de inverse **
7 REM **       de produse      **
10 INPUT N
20 LET S = 0
30 LET P = 1
40 FOR I = 1 TO N
50 LET P = P * I
60 LET S = S + 1/P
70 NEXT I
80 PRINT S
```

Divizibilitatea numerelor

Modul de a testa cu calculatorul dacă un număr **A** este divizibil cu un alt număr **B** este următorul: dacă rezultatul împărţirii lui **A** la **B** este un număr întreg atunci **A** este divizibil cu **B**.

În limbajul calculatorului vom testa dacă **A/B** este egal cu `INT (A/B)`.

Într-adevăr 10 se divide cu 2, deoarece 10/2 este egal cu `INT (10/2)`, ambele expresii avînd valoarea 5.

Dar 9 nu se divide cu 2, deoarece:

$9/2 = 4{,}5$ iar `INT (9/2)` = `INT (4.5)` = 4 şi $4.5 \neq 4$

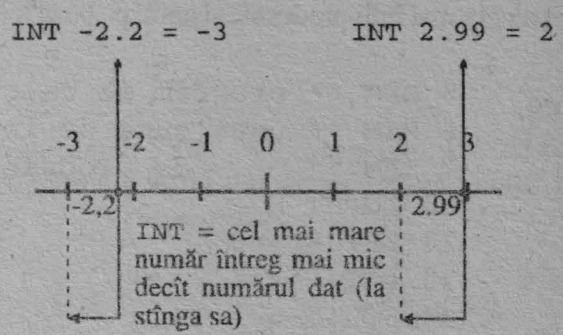
Se observă că pentru testarea divizibilităţii se foloseşte funcţia **INT** (ÎNTREG). Prin aplicarea ei asupra unei expresii se obţine valoarea întreagă a numărului rezultat în urma calculării expresiei.

O definiţie mai completă a funcţiei `INT` este însă: **cel mai mare număr întreg mai mic decît numărul dat**. Această definiţie se impune deoarece: `INT (2.99)` = 2 dar `INT (-2.2)` = -3.

Se observă, în acest ultim caz, că -3 este cel mai mare număr întreg mai mic decît numărul -2,1 (-2 ar fi fost un număr întreg mai mare decît -2,1 după cum se poate observa din figura 7).

fig.7

Reprezentare grafică pe axa numerelor pentru funcţia INT (ÎNTREG)



Să se facă un program prin care calculatorul să afişeze dacă un număr introdus este sau nu par.

Să presupunem că introducem numărul **N**, despre care vrem să ştim dacă este par sau nu (`10 INPUT N`). Apoi aplicăm modul de testare al divizibilităţii cu 2. Programul P10 va fi:

P10.
```
5 REM ** program de verificare  **
6 REM ** a paritatii unui numar **
10 INPUT N
20 IF N/2 = INT (N/2) THEN PRINT N;
   " PAR ": GOTO 10
30 PRINT N; " IMPAR"
40 GOTO 10
```

Introducînd valoarea 1244 vom obţine răspunsul PAR, iar pentru valoarea 1245 vom obţine IMPAR.

Pentru a opri programul vom introduce
STOP (**CS + A**) pentru HC
BREAK (**CTRL + C**) pentru PC.

Să se facă un program prin care să se afişeze dacă un număr introdus este divizibil cu 3.

Programul va fi asemănător cu P10. În linia 20, va trebui schimbat 2 cu 3 precum şi mesajele. În loc de "PAR" introducem "DIVIZIBIL CU 3", iar în linia 30 în loc de "IMPAR" punem "NU ESTE DIVIZIBIL CU 3".

Acum, introducînd valoarea 1244 vom obţine răspunsul "NU ESTE DIVIZIBIL CU 3", iar pentru valoarea 1245 vom obţine "DIVIZIBIL CU 3".

Numere prime

Care este cel mai mic număr prim mai mare ca 1000 ? Dar ca 10 000 ? Să se facă un program cu care să se stabilească dacă un număr întreg este prim sau nu.

Ştim că un număr este *prim* dacă nu are ca divizori decît pe 1 şi pe el însuşi. Deci, la început vom testa divizibilitatea numărului cu 2. Dacă numărul va fi divizibil cu 2 înseamnă că nu este prim şi ne oprim cu mesajul " Nu e prim". Dacă nu este divizibil cu 2 continuăm testarea. Vom testa dacă numărul este divizibil cu 3 şi, la fel, dacă este divizibil cu 3 ne oprim (nu e prim) şi, dacă nu, continuăm testarea.

Nu vom testa dacă numărul este divizibil cu 4 deoarece dacă nu a fost divizibil cu 2 înseamnă că nu va fi nici cu 4. Practic va trebui să testăm divizibilitatea cu numerele (în continuare) care sînt prime (cele din *Ciurul lui Eratostene*).

Este însă dificil să introducem în prealabil numere care sînt în *ciur* deoarece nu numai că ocupă o parte însemnată din memoria calculatorului dar nici nu ştim dinainte cîte numere prime anume să introducem în memorie, deoarece nu ştim cît de mare va fi numărul pe care dorim să-l testăm.

Vom proceda astfel: începînd cu 3 vom testa divizibilitatea numărului cu numere impare (deci din 2 în 2: `STEP 2`). În acest caz chiar dacă vom face şi unele operaţii inutile (de exemplu dacă am testat divizibilitatea cu 3 nu ar mai fi nevoie să o testăm şi cu 9) vom asigura o posibilitate pentru calculator de a repeta un anumit tip de operaţii.

Pînă unde vom merge cu testarea divizibilităţii ?

Evident dacă am ajuns pînă la jumătatea numărului şi nu am obţinut ca rezultat al testării divizibilităţii un rezultat întreg (care ar fi trebuit să fie 2), în continuare, nu mai are sens să continuăm testarea deoarece, următorul număr întreg mai mic decît 2 este 1 care nu ne interesează deoarece orice număr întreg este divizibil cu 1. Cu alte cuvinte nu ne interesează să testăm divizibilitatea numărului dat cu numere mai mari decît jumătatea lui, neputînd fi posibil, în acest caz, să se obţină un număr întreg. Deci vom merge cu testarea divizibilităţii pînă la jumătatea numărului.

Programul P11 este:

```
P11.  5 REM ** program pentru numar prim **
      10 INPUT N
      20 IF N/2 = INT (N/2) THEN PRINT N;
         " NU ESTE PRIM": GO TO 10
      30 FOR I = 3 TO N/2 STEP 2
      40 IF N/I = INT (N/I) THEN PRINT N;
         " NU ESTE PRIM": GO TO 10
      50 NEXT I
      60 PRINT N; " PRIM"
      70 GOTO 10
```

Dacă s-a parcurs tot ciclul dintre liniile 30 şi 50 şi numărul **N** nu a avut ca divizor nici unul din numere, înseamnă că el este prim.

Să aplicăm programul încercînd să răspundem la întrebarea *care este cel mai mic număr prim mai mare ca 1 000 ?*

Încercăm 1001: nu e prim, 1 003 nu e prim, 1 007 nu e prim, iar rezultatul corect este 1 009.

Într-un fel programul nu ne mulţumeşte pe deplin deoarece nu ne spune care a fost divizorul unui număr care nu este prim.

În linia 40 se testează divizibilitatea lui N. Dacă N/I este egal cu `INT (N/I)` înseamnă că numărul I este un divizor al lui N. Deci, pentru a afişa divizorul, linia 40 va deveni:

```
40 IF N/I = INT (N/I) THEN PRINT N;
   " NU ESTE PRIM. SE DIVIDE CU "; I : GO TO 10
```

Astfel vom afla că 1001 se divide cu 7; 1003 se divide cu 17 iar 1007 cu 19.

Care este cel mai mic număr prim mai mare ca 10 000?

Executînd programul observăm că:

✘ 10 001 nu este prim; se divide cu 73

✘ 10 003 nu este prim; se divide cu 7

✘ 10 007 este prim. El este numărul căutat.

Programul astfel realizat ne indică primul divizor pentru un număr dar, deseori, sîntem interesaţi să găsim toţi divizorii primi ai unui număr. Puteţi, deci, să realizaţi un program cu care să se afişeze divizorii primi ai unui număr natural N?

Vom verifica dacă fiecare din numerele 2, 3, 5, 7,... INT (N/2) reprezintă un divizor al lui N, iar dacă *da*, atunci se verifică dacă acest număr este prim sau nu, utilizîndu-se programul deja realizat. Dacă pentru acest număr se găseşte un divizor înseamnă că nu este prim şi se trece la testarea pentru următorul număr, iar dacă nu se găseşte nici un divizor înseamnă că este prim şi se afişează. Programul P12 este:

P12.
```
5 REM ** divizorii unui numar **
10 INPUT N
15 REM se cauta divizorii numarului N
20 IF N/2=INT (N/2) THEN PRINT 2; " ";
30 FOR I = 3 TO INT (N/2) STEP 2
40 IF N/I = INT (N/I) THEN GO TO 100
50 NEXT I
60 STOP
100 REM se verifica daca divizorul
101 REM gasit este un numar prim
110 FOR J = 3 TO INT (I/2) STEP 2
120 IF I/J = INT (I/J) THEN GO TO 50
130 NEXT J
140 PRINT I ; " ";
150 GO TO 50
```

Astfel aflăm că 1001 are ca divizori primi pe 7, 11 şi 13; 1003 are ca divizori primi pe 17 şi 59; 1007 pe 19 şi 53 , iar 10001 pe 73 şi 137.

Ciurul lui Eratostene

Ne punem problema să aflăm cîte numere prime sînt mai mici decît un anumit număr N.

În acest scop se foloseşte algoritmul pentru determinarea faptului dacă numărul este prim sau nu, iar pentru calcularea numărului de numere prime mai mici decît numărul dat se foloseşte algoritmul de calcul a unei sume, adăugîndu-se de fiecare dată o unitate la suma precedentă atunci cînd se identifică un număr prim (calculul reuşitelor).

Luăm în consideraţie faptul că 0 şi 1 nu sînt numere prime, primul număr prim fiind 2. Vom iniţializa în acest caz variabila **C** în care se păstrează numărul de numere prime găsit, cu 1 (avem iniţial un număr prim şi anume, 2) şi vom impune să se introducă un număr întreg mai mare sau egal cu 3. De asemenea vom testa toate numerele începînd cu 3 pînă la **N** (ne interesează toate numerele prime mai mici sau egale cu numărul **N**) luate din 2 în 2, deoarece numerele prime se găsesc printre cele impare. Programul P13 este:

P13.

```
5  REM ** cite numere prime **
6  REM ** pina la un numar N **
10 INPUT "Introduceti un numar intreg
   >= 3 "; N
20 LET C = 1
30 FOR I = 3 TO N STEP 2
40 FOR J = 3 TO INT(I / 2) STEP 2
50 IF I / J = INT(I / J) THEN GOTO 80
60 NEXT J
70 LET C = C + 1
80 NEXT I
90 PRINT C; " Numere prime mai mici
   sau egale cu "; N
```

Folosind acest program vom putea observa *cum variază numărul de numere prime atunci cînd N creşte* sau, cu alte cuvinte, *care este distribuţia*

numerelor prime. Astfel vom nota că sînt 4 numere prime mai mici ca 10 (deci 40% din numerele pînă la 10 sînt prime), 25 de numere prime mai mici ca 100 (deci 25% din numere pînă la 100), 168 de numere prime mai mici ca 1000 (17%) şi 1 229 numere prime mai mici ca 10 000 (12%). Observăm că procentul numerelor prime scade pe măsură ce **N** creşte sau, cu alte cuvinte, numerele prime sînt mai rare.

De asemenea putem să calculăm *cîte numere prime există într-un interval dat*. În acest scop vom introduce şi marginea din stînga a intervalului (presupunem că acesta este un număr par, deoarece, de obicei dorim să aflăm cîte numere prime sînt între 10 şi 20, între 100 şi 200 etc.):

```
6 INPUT S
```

Iniţializăm pe **C** cu 0, iar ciclul îl vom începe de la **S+1** (trebuie să începem cu un număr impar), linia 30 devenind:

```
30 FOR I = S TO N STEP 2
```

De asemenea linia 90 va deveni:

```
90 PRINT C; "Numere prime intre:"; S;
   " si "; N
```

Vom observa că sînt 4 numere prime în intervalul 10, 20; 21 numere prime în intervalul 100, 200 şi 135 numere prime în intervalul 1000, 2000.

Să se modifice programul realizat astfel încît să se afişeze şi numerele prime găsite sau, cu alte cuvinte, să se afişeze Ciurul lui Eratostene pînă la un anumit număr.

În acest scop în programul P13 iniţial, se vor insera liniile 15 pentru afişarea lui 2 (care este primul număr prim) şi 65 pentru afişarea numerelor prime **I** găsite :

```
15 PRINT 2; " ";
65 PRINT I; " ";
85 PRINT
```

Cel mai mare divizor comun

Să se facă un program cu care să se calculeze cel mai mare divizor comun (CMMDC) al două numere.

Va trebui mai întîi să introducem cele două numere (**A** şi **B**). Apoi ne vom gîndi la o metodă (algoritm) cu ajutorul căreia vom putea calcula CMMDC al celor două numere. Vom folosi, de exemplu, *algoritmul lui Euclid* cu care se poate calcula CMMDC a două numere întregi nenule (cu $A \leq B$) prin împărţiri repetate. Cum se procedează ? Se împarte **B** prin **A**. Dacă **B** se divide cu **A** (restul este 0) problema este simplă întrucît este evident că CMMDC este chiar **A**, întrucît divide şi pe **B** şi pe **A** şi nu poate exista alt număr mai mare decît **A** care să-l dividă pe **A**. Dacă la împărţirea lui **B** prin **A** restul nu este nul înseamnă că vom obţine un cît întreg, să îl notăm cu **C** ($C = B / A$) şi un rest **R**. Aplicînd proba împărţirii putem calcula şi valoarea lui **R**. Ştim că

$$B = A \times C + R, \text{ deci } R = B - A \times C.$$

Dar de această dată în formulă am ţinut cont de un cît întreg. Formula corectă este

$$R = B - A \times \text{INT } C.$$

Algoritmul lui Euclid ne indică apoi să înlocuim în împărţire pe **B** cu **A**, iar pe **A** cu **R** şi să continuăm procedeul pînă cînd obţinem un rest 0. În acest caz *CMMCD va fi ultimul rest nenul.*

Să exemplificăm funcţionarea algoritmului pentru numerele 72 şi 15. Am văzut că primul pas este împărţirea numărului mare (**B**) la numărul mic (**A**). În cazul nostru obţinem cîtul 4 şi restul 12. Mai departe, îl înlocuim pe **B** cu **A**, adică pe 72 cu 15 şi pe **A** cu restul **R** adică pe 15 cu 12. Vom repeta procedeul împărţind pe 15 la 12 şi obţinem cîtul 1 şi restul 3. Iarăşi înlocuim pe **B** cu **A**, adică pe 15 cu 12 şi pe **A** cu **R**, adică pe 12 cu 3, repetînd împărţirea. De data aceasta (12/3) obţinem cîtul 4 şi restul 0. Înseamnă că CMMDC este ultimul rest nenul, iar acesta a fost 3 pe care îl regăsim în valoarea actuală a lui **A**. Într-adevăr, putem verifica acest lucru

prin descompunerea numerelor iniţiale în factori primi şi aflarea CMMDC:

$$\begin{array}{r} 72 = 2^3 \times 3^2 \\ 15 = 3 \times 5 \\ \hline \text{CMMDC} = 3 \end{array}$$

Programul P14 va fi :

P14.
```
5  REM ** program de calcul al **
6  REM ** CMMDC al doua numere **
10 INPUT A
20 INPUT B
30 LET C = B/A
40 LET R = B - A * INT (C)
50 IF R = 0 THEN PRINT "CMMDC="; A : STOP
60 LET B = A
70 LET A = R
80 GOTO 30
```

Putem calcula cu ajutorul programului realizat şi cel mai mic multiplu comun (CMMMC) al numerelor?

Da, cunoscînd valorile iniţiale ale numerelor şi CMMDC, totdeauna *CMMMC va rezulta din împărţirea produsului numerelor iniţiale la CMMDC.* În acest scop va trebui să memorăm (salvăm) valorile iniţiale ale numerelor **A** şi **B** în alte două variabile, să zicem **P** şi **N**, şi, în final, să adăugăm în linia 50 o instrucţiune cu care se va calcula CMMMC. Deci adăugăm în programul P14:

```
25 LET P = B
27 LET N = A
50 IF R = 0 THEN PRINT "CMMDC="; A
51 PRINT "CMMMC="; P * N/A: STOP
```

Folosind tot o regulă (algoritm) de împărţire repetată să se arate cum se poate realiza descompunerea în factori primi a unui număr.

Ne propunem să găsim factorii primi ai unui număr întreg pozitiv. Ştim că orice număr întreg pozitiv **N** poate fi exprimat ca produs de numere prime ridicate la puteri naturale (ultimele le vom numi **indici**). Indicele unui număr prim reprezintă puterea la care trebuie ridicat numărul pentru descompunerea în factori primi ai lui **N**. De exemplu:

pentru N = 360

$360 = 2^3 \times 3^2 \times 5$ (indicii sînt 3, 2 şi 1).

Pentru a descompune în factori primi numărul **N** se va utiliza tot o metodă de împărţire repetată. Primul factor posibil este 2, deci putem scrie:

$$360 = 2 \times 180 = 2 \times 2 \times 2 \times 45 = 2^3 \times 45$$

Astfel, 2 este un factor al lui 360 şi a lui 180 şi al lui 90, dar nu este un factor al lui 45. Altfel spus împărţind pe 360 la 2 şi pe 180 la 2 şi pe 90 la 2 vom obţine restul nul, adică:

$$N - 2 * INT(N / 2) = 0$$

În schimb dacă N = 45 atunci $N - 2 * INT(N / 2) <> 0$

Vom încerca în continuare pe 3 :

$$360 = 2 \times 2 \times 2 \times 3 \times 15 = 2 \times 2 \times 2 \times 3 \times 3 \times 5 = 2^3 \times 3^2 \times 5$$

După cum se observă aplicarea metodei (algoritmului) se face astfel: se împarte succesiv **N** la divizorii **D** care sînt numere prime: (2, 3, 5, 7 etc). Dacă **D** divide pe **N** atunci se caută întregul **I** cel mai mare, astfel încît D^I să dividă pe **N**. In acest caz **N** fiind înlocuit în vederea împărţirii următoare prin N / D^I. Calculul se opreşte cînd produsul tuturor factorilor deja găsiţi este egal cu **N**.

Realizarea acestui program este ceva mai dificilă. Va trebui să memorăm mai multe valori ale variabilelor. De exemplu, va trebui să memorăm numărul

inițial **N**: `LET A = N`, deoarece în final avem nevoie de valoarea lui atît pentru afișare cît și pentru găsirea divizorilor care nu vor depăși ca valoare jumătate din numărul inițial. De asemenea va trebui să inițializăm cu 1 variabila **K** reprezentînd valoarea produsului tuturor divizorilor găsiți la puterile respective și cu 0 puterea curentă (`LET K = 1` și `LET E = 0`). Vom testa dacă 2 este divizor și apoi 3, 5, 7 etc pînă la jumătatea numărului (ca la testarea numărului prim). După alegerea fiecărui divizor potențial **D** vom apela o subrutină prin care cercetăm dacă **D** este divizor calculînd și puterea sa. Dacă numărul **A** se divide prin **D** - adică dacă A/D = INT(A/D) - atunci puterea lui **D** va crește cu o unitate (`LET E = E+1`), valoarea lui **K** se va multiplica cu **D**, iar numărul se va înlocui cu valoarea A/D pe care am notat-o cu **Q**, repetîndu-se secvența cu noul număr. Dacă **D** nu este divizor, atunci se afișează divizorul găsit la puterea respectivă. Folosim o subrutină pentru a nu repeta de două ori aceeași secvență de instrucțiuni, atît în cazul divizorului 2 cît și în cazul unui divizor impar.

Partea de program de la liniiile 90 la 180 realizează acest lucru și reprezintă practic un program de sine stătător (subprogram) în cadrul programului principal. De aceea, se numește **subrutină**. Ea se apelează cu instrucțiunea `GOSUB` (liniile 50 și 70), iar cînd ajunge la sfîrșit, instrucțiunea `RETURN` (linia 180) redă controlul programului la prima linie după instrucțiunea `GOSUB` care a apelat subrutina. Cînd K=N, adică atunci cînd produsul tuturor divizorilor găsiți la puterile respective este chiar **N**, programul se termină, deoarece nu mai putem găsi nici un alt factor prim. Programul P15 este:

P15.

```
5 REM ** program pentru descompunerea **
6 REM ** unui numar in factori primi  **
10 INPUT N
15 PRINT N; " = ";
20 LET A = N
30 LET K = 1
40 LET D = 2
50 GOSUB 90
60 FOR D = 3 TO INT (N/2) STEP 2
70 GOSUB 90
80 NEXT D
90 LET E = 0
100 LET Q = INT (A/D)
110 IF A/D <> Q THEN GOTO 160
```

```
120 LET E = E + 1
130 LET K = K * D
140 LET A = Q
150 GOTO 100
160 IF E <> 0 THEN PRINT D; "^"; E;"*";
170 IF K = N THEN STOP
180 RETURN
```

Radicali

Ştim că **pătratul** unui număr reprezintă numărul înmulţit cu el însuşi. De ce se numeşte pătrat al numărului? Deoarece reprezintă *aria unui pătrat a cărui latură este chiar numărul.* În mod similar **cubul** unui număr reprezintă numărul înmulţit cu el însuşi de 3 ori (la puterea a 3-a), deoarece reprezintă *volumul unui cub a cărei muchie este egală cu numărul.*

Se pune acum problema invers, adică, dîndu-se un număr, să se găsească altul care înmulţit cu el însuşi să ne conducă la primul număr (*operaţia radical*). Sigur că, avînd nişte numere pătrate, ne va fi uşor să identificăm numerele care, înmulţite cu ele însele, ne vor da ca rezultate pătratele respective. De exemplu, avînd 4, 9, 16, 25, 36 ... ştim că $2 \times 2 = 4$; $3 \times 3 = 9$; $4 \times 4 = 16$; $5 \times 5 = 25$ şi aşa mai departe, cu alte cuvinte, numerele corespunzătoare vor fi 2, 3, 4, 5, 6 ... Deci avînd numere pătrate este relativ simplu.

Dar cum putem găsi radicalul unui număr care nu este un pătrat perfect?

În acest caz operaţia este mai dificilă, deoarece rezultatul nemaifiind un număr natural, vom aproxima rezultatul prin numere rafinate (exprimate zecimal).

Putem însă să ne ajutăm de calculator.

Problema care se pune, deci, este să realizăm un program cu ajutorul căruia să găsim valoarea aproximativă a radicalului unui număr pozitiv.

Vom realiza un program cu ajutorul căruia vom putea calcula mai multe valori aproximative ale căror pătrate să fie din ce în ce mai apropiate de numărul dat. Metoda aceasta se numeşte **metoda de rezolvare prin aproximaţii succesive.**

În primul rînd vom introduce numărul din care dorim să extragem radicalul (`10 INPUT N`). Apoi vom introduce o încercare a noastră, adică numărul care credem noi că înmulţit cu el însuşi ne va da numărul iniţial **N** (`20 INPUT A`). Pentru a vedea cît de reuşită este încercarea noastră vom afişa valoarea încercării (**A**), rezultatul pe care îl obţinem cu încercarea respectivă şi numărul iniţial **N**. Desigur, cu cît diferenţa dintre ultimele două numere este mai mică, cu atît încercarea a fost mai bună, iar dacă cele două numere sînt egale atunci înseamnă că încercarea respectivă reprezintă exact radicalul numărului. Vom introduce diverse încercări (repetînd procedeul: `GO TO 20`) în funcţie de diferenţa observată între cele două numere.

Programul P16 este:

P16.

```
5 REM ** program de calcul al unui **
6 REM **  radical prin incercari   **
10 INPUT N
20 INPUT A
30 PRINT A; " "; A * A; " "; N
40 GOTO 20
```

Să aplicăm programul încercînd să găsim valoarea radicalului din numărul 10.

Ştim că $3 \times 3 = 9$ iar $4 \times 4 = 16$, deci numărul pe care îl căutăm va fi între 3 şi 4. Să încercăm 3,5. Vom obţine rezultatul 12,25 care este mai mare decît 10, deci, vom încerca o valoare mai mică: 3,4. Cu ea obţinem rezultatul 11,56 care este tot mai mare decît 10, deci, vom încerca o valoare mai mică: 3,3. Cu ea obţinem rezultatul 10, 89 iar cu următoarea (3,2) obţinem 10,24 care este tot mai mare ca 10. În continuare vom încerca o valoare mai mică (3,1) cu care obţinem rezultatul 9,61. Acesta este însă mai mic decît 10. Rezultă că valoarea pe care o căutăm este situată între 3,1 şi 3,2. Inseamnă că trebuie să căutăm o valoare cu 2 zecimale. Să încercăm valoarea 3,15. Obţinem rezultatul 9,9225 care este mai mic ca 10, deci, vom încerca o valoare mai mare. Cu 3,16 obţinem 9,9856 care este tot mai mică. Însă cu 3,17 obţinem 10,0489 care este mai mare. Deci, valoarea căutată este între 3,16 şi 3,17. Înseamnă că trebuie să căutăm o

valoare cu 3 cifre zecimale. Continuînd în acelaşi mod raţionamentul redăm rezultatele care se vor afişa pe ecran, mărind sau micşorînd valoarea aproximaţiei în funcţie de relaţia de inegalitate relativă la 10 (pătratul este mai mic sau mai mare ca 10):

3,165	10,017225	10
3,164	10,010896	10
3,163	10,004569	10
3,162	9,998244	10

Deci valoarea căutată este între 3,162 şi 3,163. Încercăm valoarea 3,1625;

3,1625	10,001406	10
3,1624	10,000774	10
3,1623	10,000141	10
3,1622	9,9995088	10

deci valoarea căutată este între 3,1622 şi 3,1623. Să încercăm valoarea 3,16225:

3,16225	9,9998251	10
3,16226	9,9998883	10
3,16227	9,99999515	10
3,16228	10,000015	10

Deci valoarea căutată este între 3,16227 şi 3,16228. Să încercăm valoarea 3,162275:

3,162275	9,9999832	10
3,162276	9,9999895	10
3,162277	9,9999958	10
3,162278	10,000002	10

Valoarea căutată este între 3,162277 şi 3,162278. Să încercăm valoarea 3,1622775.

3,1622775	9,999999	10
3,1622776	9,9999996	10
3,1622777	10	10

Deci valoarea căutată este chiar 3,1622777. Aceeaşi valoare o putem obţine (mult mai simplu) dacă am fi utilizat *funcţia pentru radical* (SQR):

```
PRINT SQR 10
```

se va afişa valoarea 3,1622777.

De fapt, chiar şi această valoare reprezintă o aproximaţie, calculatorul neputînd calcula o aproximaţie mai fină.

Rapid şi precis - cîteva probleme pentru dumneavoastră

Să se găsească perechile de numere a căror sumă este 1000, primul să fie divizibil cu 17 iar al doilea cu 19.

Pentru rezolvarea acestei probleme putem profita de viteza calculatorului şi astfel să-l punem să testeze toate perechile de numere a căror sumă este 1000. Prima pereche va fi 1 şi 999, a doua 2 şi 998 şi aşa mai departe. Astfel cu un contor I care ia valori de la 1 la 999, primul număr din pereche va fi chiar I, iar al doilea 1000-I. Programul este P17:

P17.
```
5 REM ** program pentru rezolvarea **
6 REM **    problemei cu numere     **
10 FOR I = 1 TO 999
20 IF I/17=INT (I/17) AND
   (1000-I)/19=INT ((1000-I)/19)
   THEN PRINT I,1000-I
30 NEXT I
```

Perechile de numere care întrunesc condiţiile şi care vor fi, deci, afişate sînt : 221 şi 779, 544 şi 456, 867 şi 133.

Să se găsească perechile de numere a căror sumă este 1000, primul număr să fie divizibil cu 17 sau cu 13 iar al doilea cu 19 şi cu 7.

Folosind acelaşi raţionament programul va fi asemănător, linia 20 a programului 17 modificîndu-se conform noilor condiţii:

```
20 IF (I/17=INT (I/17) OR I/13=INT (I/13))
   AND ((1000-I)/19=INT ((1000-I)/19)
   AND (1000-I)/7=INT ((1000-I)/7))
   THEN PRINT I , 1000-I
```

Să se găsească numărul $\overline{ABC}$ pentru care $A^2 + B^2 + C^2 = A + B + C$

Cifra **A**, fiind cea mai semnificativă, poate lua valori de la 1 la 9, în timp ce, cifrele **B** şi **C** pot lua valori de la 0 la 9. Vom lua pe rînd toate valorile posibile ale cifrelor **A**, **B** şi **C** şi vom testa care combinaţie îndeplineşte condiţia pusă. Programul P18 este:

P18.

```
5 REM **  program pentru gasirea **
6 REM ** valorilor care satisfac **
7 REM **       o identitate       **
10 FOR A = 1 TO 9
20 FOR B = 0 TO 9
30 FOR C = 0 TO 9
40 IF A*A + B*B + C*C = A + B + C
   THEN PRINT A ; B ; C
50 NEXT C
60 NEXT B
70 NEXT A
```

Se obţin următoarele numere: 100, 101, 110 şi 111. Prima cifră reprezintă valoarea lui A, a doua valoarea lui B, iar a 3-a valoarea lui C.

Să se genereze toate numerele de 4 cifre de forma $\overline{3A2B}$ care se divid cu 9.

După aceeaşi metodă A şi B pot lua valori de la 0 la 9. Vom lua toate combinaţiile posibile iar calculatorul va testa pentru care se îndeplineşte condiţia divizibilităţii cu 9, şi anume, cînd suma cifrelor numărului (adică 3+A+2+B) va reprezenta un număr divizibil cu 9. Programul P19 este:

P19.
```
5 REM ** program pentru gasirea  **
6 REM ** valorilor care satisfac **
7 REM **      o conditie         **
10 FOR A = 0 TO 9
20 FOR B = 0 TO 9
30 IF (3+A+2+B)/9 = INT ((3+A+2+B)/9)
   THEN PRINT 3;A;2;B
40 NEXT B
50 NEXT A
```

Se obţin 11 soluţii, prima fiind 3024, iar ultima 3924.

Să se găsească perechile de cifre A şi B pentru care numărul $\overline{7AB3}$ să fie divizibil cu 7 şi cu 3.

De data aceasta pentru divizibilitatea cu 7 nu mai avem nici o regulă, de aceea, va trebui să testăm dacă numărul (care este egal cu $7 \times 10^3 + A \times 10^2 + B \times 10 + 3$) este divizibil cu 7 după regula cunoscută.

Programul P20 este:

P20.

```
5 REM ** program de gasire a unor **
6 REM **   valori care satisfac    **
7 REM **       o conditie          **
10 FOR A = 0 TO 9
20 FOR B = 0 TO 9
30 IF (7*1000+A*100+B*10+3)/7=
   INT((7*1000+A*100+B*10+3)/7) AND
   (7+A+B+3)/3=INT((7+A+B+3)/3)
   THEN PRINT A,B
40 NEXT B
50 NEXT A
```

Vom obţine ca rezultat perechile 2 şi 0, 4 şi 1, 6 şi 2, 8 şi 3. Deci numerele 7203, 7413 , 7623 şi 7833 sînt divizibile cu 7 şi cu 3.

O problemă privind grafica în mişcare

Să se facă un program cît mai scurt cu ajutorul căruia să se mişte pe ecran un punct în 4 direcţii (sus, jos, stînga, dreapta) şi apoi un caracter grafic lăsînd urme în mişcarea sa.

Să presupunem că vrem să mişcăm punctul sau caracterul grafic care la început se află în centrul ecranului cu următoarele taste: 6 sau Q pentru sus; 7 sau A pentru jos; 8 sau P pentru dreapta şi 5 sau O pentru stînga. Cel mai scurt program se va realiza nu prin mai multe linii IF ci prin calculul coordonatelor în funcţie de tastele acţionate folosind propoziţiile logice.

De exemplu, pentru calculul coordonatei pe orizontală:

```
LET X = X+(INKEY$ = "8")-(INKEY$ = "5")
```

Dacă se acţionează tasta 8 atunci `INKEY$="8"` va fi o propoziţie adevărată şi va lua valoarea 1, în timp ce `INKEY$="5"` este o propoziţie falsă luînd valoarea 0. În concluzie noua coordonată va fi egală cu vechea coordonată la care se va adăuga o unitate (deci punctul se deplasează spre

dreapta). Dacă în schimb se acţionează tasta 5 atunci `INKEY$="8"` va lua valoarea 0 iar `INKEY$="5"` valoarea 1 şi, deci, valoarea coordonatei pe orizontală va scădea cu o unitate, asigurîndu-se deplasarea spre stînga. Programul P21 va asigura mişcarea punctului:

- ✘ la dreapta atunci cînd se acţionează tasta 8 sau tasta P
- ✘ la stînga cînd se acţionează tasta 5 sau O
- ✘ în sus cînd se acţionează tasta 6 sau Q
- ✘ în jos cînd se acţionează tasta 7 sau tasta A

P21.A

```
5 REM ** program pentru miscarea **
6 REM **   unui punct pe ecran    **
7 REM **  pentru calculator HC    **
10 LET X = 255/2
20 LET Y = 175/2
30 PLOT X , Y
40 LET X = X + (INKEY$ = "8" OR
   INKEY$ = "p") - (INKEY$ = "5" OR
   INKEY$ = "o")
50 LET Y = Y + (INKEY$ = "6" OR
   INKEY$ = "q") - (INKEY$ = "7" OR
   INKEY$ = "a")
60 GOTO 30
```

Totuşi veţi observa că, după ce punctul a atins deja marginea din stînga a ecranului, atunci cînd veţi comanda mişcarea la stînga, punctul se va deplasa spre dreapta ! De asemenea, după ce punctul a atins deja o dată marginea de jos ecranului, atunci cînd veţi comanda mişcarea în jos, punctul se va deplasa în sus ! Puteţi explica această anomalie ?

Explicaţia este următoarea: cînd se aplică instrucţiunea de afişare a unui punct pe ecran `PLOT X,Y` , limbajul BASIC calculează automat valoarea absolută a valorilor coordonatelor X şi Y (practic face `PLOT ABS X, ABS Y`). În acest caz atunci cînd valoarea coordonatei pe orizontală (X), scade sub valoarea 0, atingînd valorile -1, -2, -3 ... punctul se va desena pe coordonatele pe orizontală cu valorile 1, 2 ,3 ... realizîndu-se practic mişcarea la dreapta în loc de stînga. Pentru mişcarea în jos şi în sus fenomenul este similar. Însă atunci cînd vom comanda mişcarea punctului la

dreapta sau în sus, atunci cînd coordonatele pe orizontală (X) şi, respectiv, pe verticală (Y), vor depăşi valorile marginii ecranului (255 şi, respectiv, 175), atunci, punctul va ieşi din ecran, iar programul se va întrerupe cu un mesaj eroare.

Desigur în acest caz, vor trebui adăugate şi liniile care să asigure faptul că punctul nu va ieşi din ecran. Deoarece calculatorul consideră parametrii după PLOT în valoare absolută, condiţia ieşirii din ecran se va pune numai pentru partea dreaptă şi de sus a ecranului (X > 255 şi Y > 175) în timp ce, după cum am văzut acţionînd în continuare după atingerea marginilor tastele pentru ieşirea prin partea stîngă şi respectiv jos a ecranului, punctul se va depărta de aceste margini deoarece noile coordonate, deşi sînt negative, ele sînt luate în considerare în valoare absolută, ceea ce face ca pînă în momentul în care sînt atinse valorile -255 şi respectiv -175 punctele să poată fi afişate pe ecran.

Programul similar pentru calculatoare PC este P21.B.

P21.B.

```
5 REM ** program pentru miscarea unui **
7 REM **          punct pe ecran        **
8 REM **      pentru calculator PC      **
9 CLS
10 SCREEN 1
15 LET X = 320/2
20 LET Y = 200/2
30 PRESET (X,Y),7
40 LET X = X + (INKEY$ = "8" OR
   INKEY$ = "p") - (INKEY$ = "5"OR
   INKEY$ = "o")
50 LET Y = Y + (INKEY$ = "6" OR
   INKEY$ = "q") - (INKEY$ = "7"OR
   INKEY$ = "a")
60 GOTO 30
```

Se observă că s-a ales rezoluţia medie în modul grafic (320 × 200 puncte). Originea este acum colţul din stînga sus al ecranului, iar punctul se desenează în culoarea albă (cod 7). Mişcarea punctului se va realiza invers ca la HC (stînga se transformă în dreapta, iar sus în jos) deoarece condiţia de adevărat are valoarea logică -1.

Se observă că la atingerea marginilor ecranului nu se semnalează o eroare, mişcarea continuîndu-se în afara marginilor.

Pentru mişcarea unui caracter grafic (să zicem cel obţinut cu 8 şi CS în modul grafic) programul va fi similar, obţinîndu-se programul P22.A. pentru HC şi P22.B pentru PC:

P22.A
```
5 REM ** program pentru miscarea **
6 REM **    caracter pe ecran     **
7 REM **   pentru calculator HC   **
10 LET X = 11
20 LET Y = 16
30 PRINT AT X , Y; "■"
40 LET X = X + (INKEY$ = "6" OR
   INKEY$ = "q") - (INKEY$ = "7" OR
   INKEY$ = "a")
50 LET Y = Y + (INKEY$ = "8" OR
   INKEY$ = "p") - (INKEY$ = "5" OR
   INKEY$ = "o")
60 GOTO 30
```

Aceeaşi remarcă făcută pentru programul scris pentru grafica de rezoluţie înaltă este valabilă şi pentru programul actual scris pentru grafica de rezoluţie scăzută. În acest caz X va reprezenta numărul liniei, iar Y numărul coloanei. Se observă că atunci cînd X creşte, practic, numărul liniei va creşte, iar caracterul grafic se va deplasa în jos.

Programul similar pentru calculatoare PC este P22.B

P22.B
```
5 REM ** program pentru miscarea **
6 REM ** unui caracter pe ecran  **
7 REM **   pentru calculator PC  **
8 CLS
9 SCREEN 0
10 LET X = 25/2
20 LET Y = 20
30 LOCATE X,Y: PRINT "◆"
```

```
40 LET X = X + (INKEY$ = "6" OR
   INKEY$ = "q") - (INKEY$ = "7" OR
   INKEY$ = "a")
50 LET Y = Y + (INKEY$ = "8" OR
   INKEY$ = "p") - (INKEY$ = "5" OR
   INKEY$ = "o")
60 GOTO 30
```

Se observă că s-a ales modul text la lăţimea de 40 de caractere (SCREEN 0). De asemenea s-a ales caracterul grafic **caro** obţinut cu tastele **CTRL** şi **D**. Ca şi în cazul precedent mişcarea se va realiza invers deoarece condiţia de adevărat se evaluează cu -1.

Ieşirea caracterului din cadrul ecranului are însă ca urmare întreruperea programului şi afişarea mesajului de eroare **"Ilegal function call in 30"**.

Se observă că în cadrul propoziţiilor logice care folosesc funcţia INKEY$ se pot folosi numai operatorii logici OR care asigură posibilitatea alegerii unei singure taste pentru realizarea unei deplasări.

Operatorul logic AND nu se poate folosi în conjuncţie cu funcţia INKEY$ deoarece aceasta nu funcţionează cînd se acţionează deodată mai multe taste. În acest scop la calculatoarele HC se poate folosi funcţia IN.

Aceste programe pot asigura şi o *animaţie* cu condiţia ca punctul (şi respectiv caracterul grafic) să fie şters de pe vechea poziţie înainte de punerea sa pe următoarea poziţie.

De asemenea în acest caz este necesar să se găsească şi o viteză adecvată prin inserarea între două afişări consecutive a unei mici pauze (PAUSE) sau a unei note muzicale de o anumită durată (de obicei mică) pentru a da senzaţia unei mişcări reale.

Să regăsim "proporţia de aur"

Ştim că secvenţa de numere 0, 1, 1, 2, 3, 5, 8, 13, ... se numeşte *şirul lui Fibonacci*. Fiecare *nou termen* (număr) se calculează prin suma ultimelor două numere 0+1=1; 1+1=2; 1+2=3; 2+3=5; 5+8=13 şi aşa mai departe.

Realizaţi un program cu ajutorul căruia să se calculeze primele N numere din şirul lui Fibonacci. Calculaţi şi rezultatul împărţirii unui număr din şir la următorul. Ce puteţi spune în legătură cu rezultatul obţinut?

Va trebui să indicăm pînă la ce termen din şir vom calcula iar apoi să rezervăm un spaţiu de memorie pentru memorarea acestor numere (`20 DIM A (N)`). Apoi vom introduce valorile primilor doi termeni din şir (0 şi 1) iar următorii se vor calcula într-un ciclu de la 3 la N (adică de la al 3-lea termen pînă la al N-lea) după formula cunoscută. După acest calcul vom afişa valoarea contorului I (pentru a şti al cîtelea termen din şir este) apoi termenul respectiv şi raţia care îi corespunde (cîtul dintre termenul respectiv şi cel curent). Programul P23 este:

P23.

```
5 REM ** Fibonacci 1 **
10 INPUT N
20 DIM A(N)
30 LET A(1) = 0
40 LET A(2) = 1
50 FOR I = 3 TO N
60 LET A(I) = A(I-1) + A(I-2)
70 PRINT I ; " "; A(I) ; " "; A(I-1)/A(I)
80 NEXT I
```

Din cele afişate pe ecran la rularea programului, vom observa, de exemplu, că al 21-lea termen din şirul lui Fibonacci este 6765 iar raţia corespunzătoare este 0,618034. În legătură cu *raţia* observăm că ia valori diferite în jurul lui 0,6 pentru primii termeni ai şirului, apoi valori foarte apropiate de 0,618034 pentru termenii situaţi după termenul 20, iar începînd cu termenul 23 se menţine *constantă la valoarea 0,61803399*. Această raţie se mai numeşte şi **raţia (sau punctul) de aur**, ea reprezentînd proporţia cea mai naturală (s-a constatat că în natură dimensiunile sînt deseori bazate pe proporţia de aur), dar şi cea mai uzitată de oameni în construcţii, în arhitectură, în artă etc. Prin analize s-a descoperit că în marile capodopere ale lumii (sculpturi, picturi, creaţii cinematografice) se respectă riguros proporţia de aur.

Cum s-a descoperit acest şir de numere şi respectiv proporţie?

Pentru a afla cîte perechi de iepuri de casă se nasc într-un an dintr-o singură pereche de iepuri, cineva a aşezat o pereche de iepuri într-un loc îngrădit, ştiind că după o lună o pereche de iepuri aduce pe lume altă pereche. Numai prima pereche va avea descendenţi şi luna următoare, astfel încît în luna a doua vor fi trei perechi, iar dintre acestea în luna următoare două vor avea descendenţi, astfel încît în luna a treia numărul de perechi de iepuri în această lună este de cinci. Dintre acestea în aceeaşi lună, vor avea urmaşi trei perechi, iar numărul de perechi în luna a patra va fi de opt. Începeţi să recunoaşteţi şirul de numere?

Problema a fost formulată prin anul 1202, iar *Leonardo Fibonacci* (matematician italian, 1170-1250) a găsit *legea numerică* prin care se exprimă o însuşire a materiei vii, şi anume sub forma şirului de numere întregi pe care îl cunoaşteţi şi care mai poartă denumirea de *legea creşterilor organice*, deoarece prin el se exprimă dezvoltarea materiei vii, realizată prin compuneri care se însumează succesiv.

Dintre exemple menţionăm: distanţele dintre nodurile de creştere ale unei tulpini; dezvoltarea cochiliilor melcilor sau scoicilor; alungirea oaselor şi a coarnelor animalelor etc.

Modificaţi programul de calcul al termenilor şi raţiei pentru şirul lui Fibonacci astfel încît să începeţi secvenţa cu alte două numere dar folosind aceeaşi regulă pentru obţinerea fiecărui termen nou. Se modifică raţia?

În acest scop se modifică liniile 30 şi 40 din programul P23. De exemplu:

```
30 LET A(1) = 5
40 LET A(2) = 7
```

Se observă că termenii şirului vor fi diferiţi dar raţia este aceeaşi. În acest caz raţia se stabilizează la valoarea cunoscută chiar mai repede (de la al 20-lea termen). Se poate încerca şi cazul în care al doilea termen al şirului este mai mic decît primul termen. De exemplu:

```
30 LET A(1) = 10
```

```
40 LET A(2) = 2
```

Si în acest caz se poate observa că raţia va fi aceeaşi, ea stabilindu-se de la al 23-lea termen al şirului ca în cazul iniţial.

Pentru calculul punctului de aur verificaţi următoarea formulă de calcul :

$$r = \frac{1+\sqrt{5}}{2}$$

Cu

```
PRINT (1 + SQR 5) / 2
```

obţinem rezultatul 1,618034 care rezultă din împărţirea a doi termeni consecutivi : A(I) / A(I-1).

Modificaţi programul pentru calculul termenilor şi raţiei din şirul lui Fibonacci astfel încît să puteţi calcula doar al N-lea termen din şir şi raţia corespunzătoare.

Se scoate linia 70 din programul P23 şi se pune după NEXT în următoarea formă:

```
90 PRINT N;" " ;A(N) ;" "; A(N-1)/A(N)
```

Această modificare devine utilă atunci cînd se solicită al N-lea termen din şirul lui Fibonacci şi raţia corespunzătoare.

Utilizînd o variabilă multidimensională - A(N) - pentru calculul termenilor din şirul lui Fibonacci programul "merge" bine, dar este mare consumator de memorie atunci cînd vrem să calculăm termeni ai şirului de rang mare (de exemplu pentru N > 1000). De la un anumit rang în sus, acest lucru nici nu mai este posibil, calculatorul neavînd suficientă memorie.

Realizaţi un program pentru şirul lui Fibonacci fără a utiliza o variabilă multidimensională.

Pornim de la două valori iniţiale X1 şi X2. În orice situaţie următorul termen din şir va fi X3=X1+X2. Apoi punem valoarea lui X2 în X1 şi valoarea lui X3 în X2 şi îl calculăm de fiecare dată după aceeaşi formulă pe X3. Repetăm operaţia de cîte ori dorim. Programul P24 va fi:

```
5 REM ** Fibonacci 2 **
10 LET X1 = 1
20 LET X2 = 2
30 PRINT X1 ; " "; X2 ; " "; X1/X2
40 LET X3 = X2 + X1
50 PRINT X3 ; " "; X2/X3
60 LET X1 = X2
70 LET X2 = X3
80 GOTO 40
```

Pentru o reprezentare mai plastică putem da următoarele denumiri variabilelor:

X1 = BUNICUL
X2 = TATĂL
X3 = FIUL

Se observă că de fiecare dată FIUL, adică următorul termen din şir (de fapt următorul membru al familiei) va rezulta din X2 şi X1 iar pentru următoarea generaţie locul BUNICULUI va fi luat de actualul TATĂ, iar locul TATĂLUI va fi luat de actualul FIU.

Operaţii cu mulţimi de numere

Să se facă un program cu care să se realizeze operaţiile de reuniune, intersecţie şi scădere a două mulţimi.

Stim că în urma operaţiei de *reuniune* a două mulţimi rezultă o mulţime care *conţine elementele din prima şi din a doua mulţime, comune şi necomune*, luate o singură dată. De asemenea prin intersecţie rezultă o mulţime care conţine *numai elementele comune*, iar prin *scădere* o mulţime care conţine *toate elementele din prima mulţime dar care nu sînt în a doua.*

Să notăm cele două mulţimi cu A şi B. Vom parcurge următorii paşi logici:

- 1. **Iniţializare.** Va trebui să ştim cîte elemente conţine prima şi a doua mulţime, valori pe care le vom ataşa variabilei N (pentru prima mulţime) şi respectiv M (pentru a doua mulţime). Vom dimensiona cîte un vector pentru elementele mulţimii A şi elementele mulţimii B.
- 2. **Introducerea valorilor** A(I) pentru mulţimea A cu un ciclu `FOR-NEXT` precum şi a valorilor B(I) pentru mulţimea B cu un alt ciclu `FOR-NEXT`
  - 2.1. Deoarece o mulţime nu poate conţine acelaşi element de mai multe ori, în ciclul de introducere (atît pentru mulţimea A cît şi pentru mulţimea B) vom începe imediat după introducerea valorii un alt ciclu (imbricat) în scopul testării repetării valorii.Dacă aceeaşi valoare a mai fost introdusă atunci introducerea este respinsă solicitîndu-se altă introducere. De remarcat că indicele ciclului interior va lua valori de la 1 pînă la I-1 deoarece valoarea recent introdusă A(I) se compară cu valorile introduse pînă la acel moment, adică cu valorile A(1), A(2), ... , A(I-1). Dacă valoarea A(I) este diferită de valoarea A(K) , atunci verificarea se continuă, iar dacă se găseşte o valoare egală se afişează

mesajul de eroare şi se repetă introducerea. La fel se procedează şi pentru ciclul de introducere a valorilor pentru mulţimea B.

- 3. **Afişarea valorilor** mulţimii A şi mulţimii B. Afişarea se va face pe un rînd (valorile în continuare cu un spaţiu între ele) iar, după terminarea afişării valorilor unei mulţimi, se va introduce o linie PRINT pentru trecerea la rîndul următor.

- 4. **Determinarea mulţimii diferenţă A-B**. În acest scop vom lua pe rînd toate cele N elemente A(I) ale mulţimii A şi le vom compara cu fiecare din elementele B(J) ale mulţimii B. Dacă, de exemplu, un element A(I) nu este egal cu nici unul din elementele B(J), adică dacă se parcurge tot ciclul J, atunci se afişează elementul A(I) în continuare, deoarece el aparţine mulţimii A şi nu aparţine mulţimii B. Dacă însă se găseşte că elementul A(I) este egal cu un element B(J), atunci el se dă la o parte (nu se afişează, neaparţinînd mulţimii A-B) şi se trece la verificarea apartenenţei la mulţimea A-B a următorului element din mulţimea A (NEXT I).

- 5. **Determinarea mulţimii reuniune A ∪ B.** În acest scop se vor afişa în primul rînd toate elementele care aparţin mulţimii A. Deci:
  - 5.1. Se afişează toate elementele mulţimii A;
  - 5.2. Se iau pe rînd elementele mulţimii B şi se compară cu fiecare element din mulţimea A. Dacă se găseşte o egalitate atunci nu se mai afişează şi acest element deoarece el a mai fost odată afişat aparţinînd lui A (un element nu trebuie să apară de mai multe ori în reuniune) şi se trece la analizarea următorului element din B (NEXT J). Dacă după compararea cu toate elementele din A nu se găseşte nici o egalitate atunci elementul din B se afişează în continuare, el făcînd parte din reuniune.

- 6. **Determinarea mulţimii intersecţie A ∩ B.** Se iau pe rînd toate elementele din A şi se compară cu fiecare element din B. Dacă un element se găseşte şi în A şi în B atunci el se afişează

deoarece face parte din intersecţie. Apoi se trece la compararea următorului element din A cu fiecare din cele aparţinînd lui B (NEXT I). Programul P25 este:

P25.

```
5 REM **   program pentru calculul    **
6 REM **  diferentei, reuniunii si    **
7 REM ** intersectiei a doua multimi **
10 REM initializare
20 INPUT N , M
30 DIM A(N): DIM B(M)
40 REM introducere valori
50 FOR I = 1 TO N
60 REM introducere valoare multime A
70 INPUT A(I)
80 REM verificare valoare introdusa
90 FOR K = 1 TO I-1
100 IF A(I)  A(K) THEN GOTO 130
110 PRINT "Introduceti alta valoare"
120 GOTO 70
130 NEXT K
140 NEXT I
150 FOR I = 1 TO M
160 REM introducere valoare multime B
170 INPUT B(I)
180 REM verificare valoare introdusa
190 FOR K = 1 TO I-1
200 IF B(I)  B(K) THEN GOTO 230
210 PRINT "Introduceti alta valoare"
220 GOTO 170
230 NEXT K
235 NEXT I
240 REM afisare valori
250 REM afisare valori multime A
260 PRINT "A = ( ";
270 FOR I = 1 TO N
280 PRINT A(I) ; " ";
```

```
290 NEXT I
300 PRINT " ) "
310 REM afisare valori multime B
320 PRINT "B = ( ";
330 FOR I = 1 TO M
340 PRINT B(I) ; " ";
350 NEXT I
360 PRINT " ) "
370 REM diferenta
380 PRINT " A - B = ( ";
390 REM determinare elemente multime
391 REM diferenta
400 FOR I = 1 TO N
410 FOR J = 1 TO M
420 IF A(I) = B(J) THEN GOTO 460
430 NEXT J
440 REM afisare element multime
441 REM diferenta
450 PRINT A(I) ; " ";
460 NEXT I
470 PRINT " ) "
480 REM reuniune
490 PRINT " A U B = ( ";
500 REM determinare elemente multime
501 REM reuniune
510 FOR I = 1 TO N
520 REM afisare element multime
521 REM reuniune (originar din A)
530 PRINT A(I) ; " ";
540 NEXT I
550 REM evitarea repetarii afisarii
551 REM unui element in reuniune
560 FOR J = 1 TO M
570 FOR I = 1 TO N
580 IF B(J) = A(I) THEN GOTO 620
590 NEXT I
```

```
600 REM afisare element multime
601 REM reuniune (originar din B)
610 PRINT B(J) ; " ";
620 NEXT J
630 PRINT " ) "
640 REM intersectie
650 PRINT " A X B = ( ";
660 REM determinare elemente multime
661 REM intersectie
670 FOR I = 1 TO N
680 FOR J = 1 TO M
690 REM afisare element multime
691 REM intersectie
700 IF A(I) = B(J) THEN PRINT A(I);" ";
    : GOTO 720
710 NEXT J
720 NEXT I
730 PRINT " ) "
```

Deseori este bine ca în cadrul programelor să se facă o verificare prealabilă a datelor introduse, iar în cazul în care o anumită dată nu corespunde, să se repete introducerea. Să presupunem, de exemplu, că A(I) sînt nişte date care trebuie introduse şi care reprezintă numere întregi şi pozitive mai mici ca 1000.

Cum trebuie continuat programul astfel încît să se realizeze o verificare a datelor introduse?

Se va adăuga o linie programului P25 care face această verificare. Dacă data nu corespunde, se va afişa un mesaj prin care se va cere repetarea introducerii:

```
75 IF A(I) <> INT (A(I)) OR A(I) < 0 OR
   A(I) > 1000 THEN PRINT "data eronata;
   repetati introducerea" : GO TO 70
```

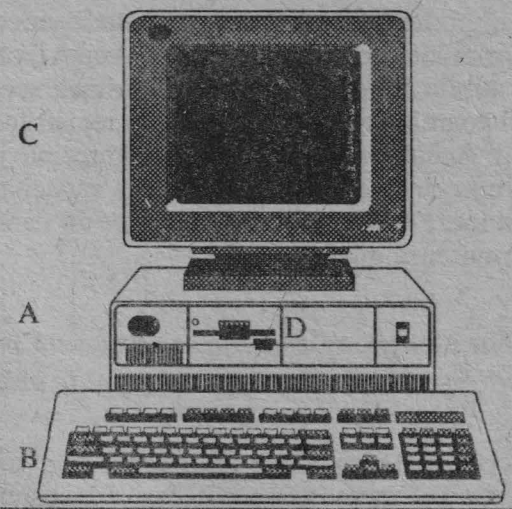
5 Din ce este format un calculator ?

Calculatoarele au pătruns în viaţa de toate zilele ca un instrument indispensabil pentru fiecare domeniu de activitate.

Cele de tip **PC** (Personal Computer) - figura 8 - sînt şi cele mai răspîndite şi mai utilizate dintre calculatoare datorită gradului lor de accesibilitate dar şi preţului relativ scăzut. Indiferent, însă, de tipul calculatorului, fie că este vorba de un sistem de dimensiuni mai mari (**mainframe**), un **minicalculator** sau unul de tip **PC**, un calculator este format din mai multe părţi conectate între ele şi care împreună alcătuiesc un tot, funcţionarea fiecărei părţi fiind integrată în funcţionarea întregului sistem.

fig.8

Calculatorul cu părţile lui componente



Unitatea centrală (vezi partea A din figura 8) este componenta principală (creierul calculatorului) care practic conduce şi controlează întregul proces din cadrul sistemului. În cadrul ei se găseşte **unitatea aritmetică şi logică** unde se desfăşoară toate operaţiile aritmetice şi logice. De obicei, din punct de vedere fizic, unitatea centrală este, de fapt, cutia (sau dulapul) pe care în mod "impropriu" o numim calculator. Pentru calculatoarele per-

sonale, întreaga unitate centrală se găseşte, de obicei, pe o singură placă ce conţine mai multe circuite integrate. Unitatea aritmetică şi logică, în acest caz, este unul din circuitele de pe placă şi este un *microprocesor*, o bucăţică de cristal de siliciu tratată special, încapsulată într-un înveliş de ceramică şi care poartă denumirea de *cip*. Microprocesorul asigură practic toate operaţiile în calculator, adică execută operaţii aritmetice şi logice, decodifică instrucţiuni, transmite altor circuite din sistem semnale de control etc. Pentru calculatoarele pe care le cunoaşteţi cel mai bine (HC, CIP, COBRA, TIM, JET) microprocesorul utilizat se numeşte **Z-80** şi conţine cîteva registre de memorie. Calculatoarele de tip PC sînt construite de obicei cu microprocesoare Intel 8088 sau 8086.

Memoria internă se găseşte tot în cutia amintită nefiind accesibilă vederii noastre. Este formată tot din circuite electronice - registre - (ceva mai mici decît microprocesorul) care au proprietatea de a memora informaţii. Practic ele memorează secvenţe de 0 şi 1 (se foloseşte, deci, **sistemul de numeraţie binar**) care reprezintă date sau programe. Astfel 0 şi 1 sînt cele mai mici unităţi de informaţie care circulă în calculator şi ele se mai numesc **biţi**.

Este important faptul că atît registrele microprocesorului cît şi cele ale memoriei interne pot memora o secvenţă fixă de biţi. Pentru calculatoarele compatibile Sinclair Spectrum mărimea secvenţei este de 8 biţi. Pentru calculatoarele de tip PC capacitatea regiştrilor este, de obicei, de 16 sau 32 de biţi. Această secvenţă de 8 biţi reprezintă un **octet** (sau **byte** în engleză) şi el reprezintă unitatea de măsură a capacităţii (mărimii) memoriei calculatoarelor. Deci, practic în calculator un caracter ca A sau 9, de exemplu, se memorează într-un octet.

☞ *Nu uitaţi: informaţiile se reprezintă prin biţi, dar se transmit şi se interpretează informaţiile conţinute în secvenţe de 8 biţi (octeţi).*

De aceea toate registrele unui calculator (indiferent dacă sînt din microprocesor sau memorie) au aceeaşi dimensiune (8 biţi la calculatoare Sinclair Spectrum) astfel încît ceea ce iese dintr-unul să poată intra în altul. Din acest motiv se mai spune că aceste calculatoare (Sinclair Spectrum) sînt calculatoare pe 8 biţi, spre deosebire de PC-urile construite cu microprocesoare INTEL (8088) ale căror registre, după cum am văzut, pot memora 2 sau 4 octeţi, fiind deci, calculatoare pe 16 sau 32 de biţi. Bineînţeles că dacă un calculator va avea registre de capacitate mai mare,

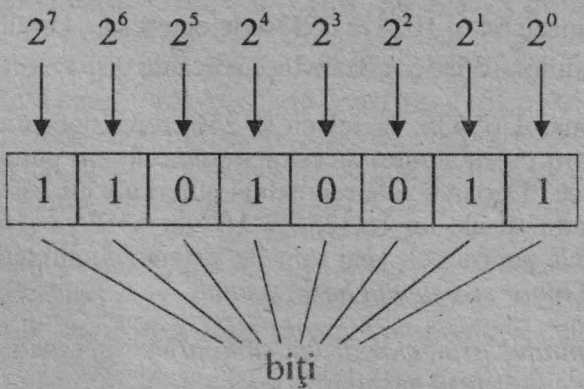
adică va putea memora şi transmite o cantitate mai mare de informaţie, va fi mai puternic.

Mărimea memoriei este vitală pentru viteza şi eficienţa cu care sistemul va opera. Tehnologia circuitelor imprimate pe siliciu (aşa numitele **cip**-uri) a evoluat foarte rapid ceea ce a permis o mare miniaturizare precum şi posibilitatea ca o capacitate de memorie mare să fie cuprinsă într-o unitate centrală de dimensiuni mici. Aşa că nu subevaluaţi micile calculatoare pe care le aveţi în faţă. Puterea lor (care este dată, în general, de doi factori importanţi, şi anume, *viteza de calcul* şi *capacitatea de memorie*) este similară cu cea a sistemelor de calcul care erau folosite de specialişti la nivelul anilor 1975.

Am văzut că pentru calculatoarele cu care lucrăm, în fiecare circuit de memorie "intră" un octet sau, cu alte cuvinte, 8 biţi. Fiecare din aceşti biţi va avea o valoare (pondere) în funcţie de poziţia pe care o ocupă în octet (vezi figura 9), întreaga înşiruire a celor 8 biţi formînd un număr (o valoare) pe care îl putem calcula. Primul bit (cel mai din stînga) este *cel mai semnificativ*, avînd ponderea 128 (2^7). Apoi ponderea biţilor scade pe măsură ce ne deplasăm spre dreapta (scăzînd puterile lui 2), ultimul bit (cel mai din dreapta) avînd valoarea 1 (2^0). Spunem că bitul cel mai din dreapta este *cel mai puţin semnificativ*. Valoarea pe care o reprezintă octetul o putem calcula în mod asemănător cu "citirea" unui număr (de fapt valoarea unui număr din sistemul zecimal, dar considerînd baza 2).

fig.9

Octetul



Octetul

De exemplu, înşiruirea de cifre 3524 din sistemul zecimal va reprezenta ca valoare 3 mii 5 sute 2 zeci şi 4, sau, de fapt, 3 mii + 5 sute + 2 zeci + 4, adică:

$$3 \times 10^3 + 5 \times 10^2 + 2 \times 10^1 + 4 \times 10^0$$

În mod similar, putem calcula valoarea octetului din figura 9, şi anume:

$$1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 =$$
$$128 + 64 + 16 + 16 + 2 + 1 = 211$$

Observăm că cel mai mic număr dintr-un octet se va forma atunci cînd toţi biţii vor fi "puşi pe 0", în acest caz numărul avînd valoarea 0 , iar, cel mai mare număr se va forma cînd toţi biţii vor fi "puşi pe 1", iar acest număr va fi 255.

Pentru calculatoarele pe care le cunoaşteţi, capacitatea lor de memorie este de 64 Ko (kilo octeţi). În sistemul zecimal, prefixul kilo indică 1000, dar în sistemul binar (care se utilizează, după cum am văzut pentru calculatoare), 1 kilo înseamnă 1024 deoarece trebuie să reprezinte o putere a lui 2 (şi anume 2 la puterea a 10-a).

Acum putem să ne reprezentăm mai clar ce înseamnă capacitatea de memorie internă a calculatorului ştiut: el poate la un moment dat să înmagazineze $64 \times 1024 = 65536$ de octeţi sau, cu alte cuvinte, peste jumătate de milion (65536×8) de biţi, adică de 0 şi 1.

Numărul 65536, la fel ca şi 256, reprezintă un număr foarte important pentru calculatoare. Acestor numere le-am putea spune **numere magice**. De ce? Deoarece ele reprezintă puteri ale lui 2 ($256 = 2^8$, iar $65536 = 2^{16}$) precum şi ale lui 16 ($256 = 16^2$ iar $65536 = 16^4$), iar 2 şi 16 reprezintă bazele pentru cele mai folosite sisteme de numeraţie la calculatoare: *sistemul binar* şi *sistemul hexazecimal.*

Avantajul principal al calculatoarelor personale este acela că memoria poate fi extinsă destul de uşor.

Multe dintre tipurile de calculatoare prezintă chiar în denumirea lor caracteristica referitoare la capacitatea de memorie. Astfel vechile sisteme de

calcul din ţara noastră se numeau FELIX C32, FELIX C256, FELIX C512, FELIX C1024. Numerele respective reprezentau de fapt, capacitatea de memorie în Kiloocteţi.

Observaţi că deşi aceste sisteme erau de dimensiune mare (dulapuri), ele prezentau o capacitate de memorie comparabilă cu a PC-urilor actuale.

Similar ZX SPECTRUM 128 înseamnă un calculator tip SINCLAIR SPECTRUM cu o capacitate de memorie internă extinsă de la 64 Ko la 128 Ko, iar COMMODORE 64 reprezintă un calculator COMMODORE cu o capacitate de memorie internă de 64 Ko.

Pentru a putea fi referiţi sau modificaţi, tuturor octeţilor memoriei interne le sînt asociate numere care reprezintă **adresele de memorie** ale octeţilor. La calculatoarele Sinclair Spectrum prima adresă de memorie este adresa 0, iar ultima este 65536, corespunzătoare capacităţii de memorie de 64 Ko.

Memoriile interne sînt de două feluri: unele care memorează programele (adică sînt deja "pline") care fac să "meargă" calculatorul şi altele care sînt "goale" aşteptînd ca noi să le "umplem" cu date sau programe. Primele memorii sînt de tip ROM (**R**ead **O**nly **M**emory) adică pot fi numai citite, noi putînd afla (citi) conţinutul lor dar nu şi scrie (adică memora) altceva în ele. Înscrierea conţinutului acestor memorii se face de fabricant, iar operaţia de înscriere cu programe (de fapt tot cu secvenţe de 0 şi 1) se mai numeşte **arderea memoriilor**.

În cazul calculatoarelor amintite, în memoriile ROM este înscris programul care face calculatorul să meargă (şi să înţeleagă) prin limbajul BASIC. De aceea aceste calculatoare se mai numesc **calculatoare BASIC** sau **cu BASIC-ul încorporat**, ele permiţîndu-ne, de la conectarea calculatorului la reţeaua electrică introducerea de programe BASIC. Memoria de tip ROM la calculatoarele Sinclair Spectrum se întinde de la adresa de memorie 0 pînă la adresa 16383, fiind vorba, deci, de 16 Ko de memorie.

Celelalte memorii interne sînt de tip RAM şi ele se mai numesc **memorii la dispoziţia utilizatorului**. Cînd conectăm calculatorul, aceste memorii sînt "goale", ele umplîndu-se pe măsură ce introducem programe şi date.

Putem iniţializa calculatorul acţionînd un buton de RESET, iar acest lucru realizează practic "golirea" memoriilor de tip RAM. Bineînţeles, memoriile de tip RAM reprezintă un volum de memorie mai mare decît cel reprezentat de memoriile tip ROM.

Să ne jucăm cu memoria

Există două instrucţiuni pereche în limbajul BASIC prin intermediul cărora se poate lucra cu memoria: PEEK şi POKE. Cu PEEK se poate citi ce este la o adresă de memorie, adică se poate vizualiza conţinutul respectiv. Din acest motiv instrucţiunea PEEK se utilizează în asociaţie cu instrucţiunea PRINT.

PRINT PEEK 0 va afişa pe ecran conţinutul primului octet al memoriei (bineînţeles un număr cuprins între 0 şi 255).

Cu POKE se poate modifica valoarea unui octet dintr-o adresă de memorie.

Să experimentăm ce am învăţat despre memoria ROM şi RAM prin intermediul intrucţiunilor PEEK şi POKE. .

Să vizualizăm conţinutul unei adrese de memorie (de exemplu adresa 1000) şi apoi să încercăm să modificăm această valoare:

```
PRINT PEEK 1000
```

Se va afişa numărul 9, ceea ce înseamnă că octetul de la adresa 1000 are valoarea 9. Să încercăm să modificăm această valoare de la 9 la 10:

```
POKE 1000 , 10
```

Să vizualizăm, din nou, conţinutul acestei adrese:

```
PRINT PEEK 1000
```

Observăm că a fost afişat tot numărul 9 deoarece adresa 1000 este o adresă de memorie ROM (aceste adrese sînt cele de la 0 la 16383) şi, deci, conţinutul său nu se poate modifica.

Să încercăm acum cu o adresă de memorie de tip RAM, deci, mai mare ca adresa 16384, să zicem 20000:

```
PRINT PEEK 20000
```

Obţinem valoarea 0. Adresa 20000 referă un octet din memoria de tip RAM ataşată ecranului grafic. Era normal să obţinem valoarea 0 deoarece

ecranul este gol. Să modificăm acestă valoare printr-o nouă valoare, şi anume, 170:

```
POKE 20000 , 170
```

Acum vizualizînd din nou valoarea conţinută în adresa 20000, obţinem, într-adevăr, noua valoare, şi anume, 170, deoarece 20000 este o adresă de memorie RAM.

Mai mult, se poate observa chiar pe ecran consecinţa modificării (practic se vizualizează octetul) şi anume o linie punctată, în partea de mijloc stînga a ecranului, corespunzătoare şirului de biţi 10101010 care reprezintă în sistemul binar numărul 170. Vizualizarea modificării s-a produs deoarece adresa de memorie 2000 este o adresă asociată ecranului grafic.

Uneori este necesar să memorăm o informaţie a cărei valoare reprezintă un număr superior valorii 255. In acest caz vom reprezenta această valoare pe 2 octeţi avînd, deci, nevoie de două adrese de memorie consecutive.

Cum se face citirea informaţiilor pentru valori mai mari ca 255? Să presupunem, de exemplu, că vrem să ştim care este adresa de început a unui program BASIC. Această adresă va fi un număr mai mare decît 16384, deoarece programul BASIC se situează (după cum am văzut) în partea de memorie RAM. Pentru memorarea numărului care reprezintă adresa sînt necesari 2 octeţi, şi anume, cei de la adresele 23635 şi 23636. Citirea informaţiei din aceşti 2 octeţi prin care vom afla adresa de început a programului BASIC se face astfel:

```
PRINT PEEK 23635 + 256 * PEEK 23636
```

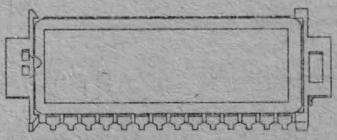
Observăm că valoarea octetului de la adresa mai mare se înmulţeşte cu numărul 256. Obţinem valoarea 23755 care reprezintă adresa de început a programului BASIC.

Dacă deschidem capacul "cutiei" calculatorului vom vedea placa pe care se situează circuitele: microprocesorul (care are dimensiuni ceva mai mari, de cîţiva centimetri) şi, ordonate sub formă de rînduri, mai multe circuite de memorie RAM de culoare închisă şi de dimensiuni mai mici precum şi 8 circuite de culoare deschisă (şi ceva mai mari) de memorie ROM.

După cum am văzut, de cîteva ori ne-am referit în descrierea părţilor componente ale calculatorului şi la programe. Într-adevăr, putem de la început să semnalăm faptul că un calculator este format practic din două

fig.10

Circuit ROM



părţi: o parte de echipamente (este o parte fizică, palpabilă, care include unitatea centrală, memoriile, etc) şi care se mai numeşte **hardware** (tocmai pentru că este o parte fizică, adică "tare") şi o parte de programe care se mai numeşte **software** (pentru că nu este o parte fizică, adică este "moale", nepalpabilă).

Programele înscrise de fabricant în memoriile ROM fac parte din software. În afara acestora tot din software mai fac parte şi programele realizate de noi pe măsură ce le introducem în memoria calculatorului.

Echipamente periferice

Pînă în acest moment am descris elementele (componentele) centrale din cadrul hardware-ului. Celelalte părţi componente ale calculatorului (adică tastatura, monitorul, casetofonul, unităţile de discuri, imprimanta etc) se mai numesc şi **unităţi periferice**, în totalitatea lor alcătuind periferia calculatorului.

Tastatura (vezi partea B din figura 8) este o componentă asemănătoare cu o maşină de scris. Prin intermediul ei instrucţiunile şi datele se introduc în calculator, de aceea se spune că tastatura este un **echipament (dispozitiv) de introducere (sau intrare)**.

Deşi la calculatoarele pe care le cunoaşteţi tastatura este pe "capacul" cutiei calculatorului ea face parte din *periferia calculatorului*. La multe calculatoare (de exemplu la cele de tip PC) ea se detaşează de unitatea centrală fiind legată de aceasta printr-un **cablu**.

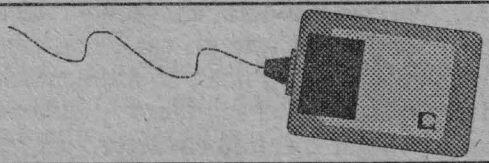
Tastatura este dotată cu taste care reprezintă litere, cifre, precum şi alte simboluri. La calculatoarele pe care le cunoaşteţi prin acţionarea unei taste se pot obţine şi cuvinte cheie care reprezintă **instrucţiuni** sau **comenzi**.

Alt dispozitiv de introducere poate fi cel de tip "**şoarece**" (figura 11), prin mişcarea sa şi prin acţionarea butoanelor putîndu-se introduce date sau comenzi. La calculatoarele de tip PC tastatura prezintă mai multe taste, iar

instrucţiunile (cuvintele cheie) se introduc prin tastarea lor caracter cu caracter.

fig.11

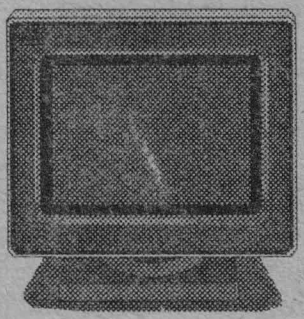
Mouse



Monitorul (sau televizorul) (vezi partea C din figura 8 şi figura 12) este utilizat pentru a afişa pe ecranul său diverse rezultate, date sau mesaje. De aceea se mai numeşte **dispozitiv de afişare**. Spunem că el este un **echipament periferic de extragere** (sau ieşire) deoarece cu ajutorul lui nu putem decît să citim rezultatele sau datele de pe ecran, adică să le extragem. Spre deosebire de televizoare, monitoarele (tuburile cu raze catodice CRT) sînt proiectate special pentru folosirea la calculator dispunînd de posibilităţi complexe de afişare (o rezoluţie mai mare).

fig.12

Monitor



Pe ecranul calculatorului se pot afişa caractere literale (litere, cifre, semne de punctuaţie, alte semne) şi caractere grafice (în modul G), iar în acest caz spunem că ecranul este în **mod text**. În afară de caractere se pot realiza şi desene (grafică), iar în acest caz ecranul va fi în **mod grafic**.

Pentru a afişa caractere sau grafică, toate aceste obiecte trebuie memorate. Deci ecranul calculatorului are un corespondent în memoria internă, acest corespondent numindu-se **memoria ecran**, adică memoria necesară reprezentării ecranului. Pentru a putea fi reprezentat în memorie ecranul este organizat într-un anumit mod în funcţie de obiectele afişate pe ecran (caractere sau grafică). Acest mod de organizare a memoriei (ecran) se mai numeşte **maparea memoriei**.

Să vedem cum se face "corespondenţa" dintre ecran şi memorie la calculatoarele HC în cazul în care se afişează caractere (modul text).

În acest caz ecranul este organizat pe linii şi pe coloane. Puteţi observa acest lucru şi singuri, introducînd, de exemplu, mai multe linii de caractere (text). Veţi nota în acest caz cum pe fiecare linie încap 32 de caractere (atenţie, şi spaţiul gol este un caracter), care se aliniază exact unele sub altele formînd linii. Veţi nota, de asemenea, că fiecare caracter, indiferent de mărimea lui, un punct sau litera A, de exemplu, ocupă acelaşi spaţiu pe ecran, spaţiu numit şi **celulă caracter**.

Astfel în *mod text* ecranul este organizat pe 32 de coloane (numerotate de la stînga la dreapta de la 0 la 31) şi pe 22 de linii (numerotate de sus în jos de la 0 la 21). În plus mai sînt două linii în partea de jos a cranului, pe care se afişează mesajele calculatorului. Concluzionăm că sînt 24 de lini şi 32 de coloane astfel încît ecranul poate fi acoperit de 24 × 32 = 768 caractere.

Ţinînd cont de faptul că fiecărei celulă caracter îi corespunde în memorie 8 octeţi, rezultă că întreaga memorie ecran are o mărime de 768 celule caracter × 8 octeţi = 6144 octeţi.

Pentru a fi afişate pe ecran caracterele au fost în prealabil "desenate" prin puncte, iar punctele s-au obţinut prin "punerea pe 1" a unor biţi din cadrul octeţilor care alcătuiesc caracterul. Să nu vă fie teamă! Caracterele au fost proiectate în prealabil de către constructor, întreg setul de caractere folosite de calculator (se regăsesc pe tastatură) fiind şi el memorat într-o anumită zonă a memoriei. Există însă posibiltatea ca în mod similar să vă construiţi propriile caractere (pentru ş, î, ţ, ă, de exemplu) şi să le obţineţi acţionînd tasta atribuită caracterului în mod grafic (G).

Să vedem cum se face "corespondenţa" dintre ecran şi memoria ecran în celălalt caz, adică atunci cînd vrem să afişăm desene (grafică). Evident va trebui să obţinem aceeaşi mărime pentru memoria ecran.

În acest caz desenul se compune din mai multe puncte, iar ecranul va avea 256 de puncte (se mai numesc **pixeli**) pe orizontală (de la 0 la 255) şi 176 puncte pe verticală (de la 0 la 175). Practic fiind necesar să se memoreze fiecare punct de pe ecran (pixel), iar fiecărui pixel corespunzîndu-i un bit numeric, vor fi necesari 256 × 176 = 45056 biţi, adică 46056 : 8 = 5632 octeţi. Am văzut însă că mai sînt două linii în partea de jos a ecranului necesare pentru a afişa mesaje, pe această porţiune de ecran nerealizîndu-se în mod normal grafică. Această zonă ridică cu 32 caractere × 2 linii × 8 octeţi = 512 octeţi mărimea memoriei, care devine astfel 5632 + 512 = 6144 octeţi, adică la fel ca în cazul precedent.

Se observă că în primul caz – modul text – caracterele fiind obiectele care se puteau pune pe ecran, acesta se acoperea cu numai 768 de caractere. De aceea acest tip de grafică se mai numeşte de **rezoluţie scăzută**.

În celălalt caz – modul grafic – punctele fiind obiectele care se puteau pune pe ecran, acesta se acoperea cu peste 45000 de puncte. De aceea acest tip de grafică se mai numeşte de **rezoluţie înaltă**.

Cum se fac în acest caz reprezentări grafice pe ecran? În mod similar, cînd unul din biţii corespunzători unui octet care reprezintă o adresă de memorie ecran este pus pe 1, atunci un punct se va desena pe ecran şi spunem că punctul (bitul) este *aprins*. Dacă bitul este pe 0, atunci nu apare nimic pe ecran, punctul (bitul) fiind *stins*.

Concluzionăm că cele 6144 de adrese de memorie, şi anume, cele cuprinse între 16384 şi 22528 reprezintă adrese de memorie ecran.

Cum se pot memora datele în afara memoriei interne a calculatorului ?

Unităţile de **memorie externă** înmagazinează datele şi instrucţiunile în afara unităţii centrale permiţînd ca acestea să fie utilizate repetat sau păstrate pentru cerinţe ulterioare.

Casetofonul este o unitate periferică cu ajutorul căreia se încarcă în memoria calculatorului programe (de exemplu, jocuri de pe casete) sau date. De asemenea tot cu el se pot "salva" adică înregistra pe casete, programe sau date pe care în prealabil le-am introdus în memoria internă a calculatorului şi pe care dorim să le folosim şi altă dată. Dacă nu le-am "salva" pe casetă, le-am pierde odată cu deconectarea calculatorului de la reţea. Spunem că unitatea casetofon este o **unitate de introducere/extragere**, deoarece cu ea putem, aşa cum am văzut, atît să introducem (să încărcăm) cît şi să extragem (adică să salvăm) programe şi date.

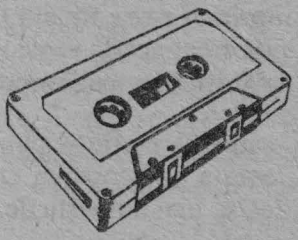
Salvarea (înregistrarea) programelor şi datelor se realizează pe **suporturi magnetice**. Acestea joacă rolul de memorie externă deoarece sînt în afara calculatorului. În timp ce memoria internă, fiind formată din circuite electronice (scumpe) este mai limitată, cea externă este mai ieftină şi poate înmagazina un volum mai mare de date şi programe.

Suportul magnetic pentru casetofon este obişnuita *casetă magnetică*.

Capacitatea de memorie externă se măsoară tot în octeţi (sau kilo octeţi). După cum ştiţi pe o casetă magnetică de o oră încap circa 10-12 jocuri de

fig.13

Casetă magnetică

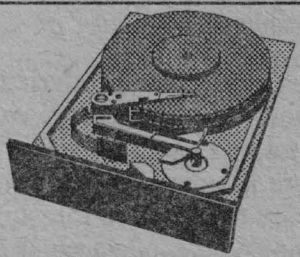


dimensiuni mari, adică din acelea care ocupă aproape întreaga memorie internă (de tip RAM) a calculatorului. Deci capacitatea de memorie a unei casete este de circa 10 ori mai mare decît capacitatea memoriei interne a calculatorului, adică, aproximativ 400-500 ko.

Cel mai frecvent utilizate unităţi de memorie externă pentru calculatoarele PC sînt **unităţile de discuri flexibile** (vezi partea D din figura 8 şi figura 14). Acestea sînt unităţi periferice tot de introducere/extragere cu ajutorul cărora, exact ca în cazul casetofonului se pot memora sau încărca în memoria internă programe sau date.

fig.14

Unitate de disc flexibil

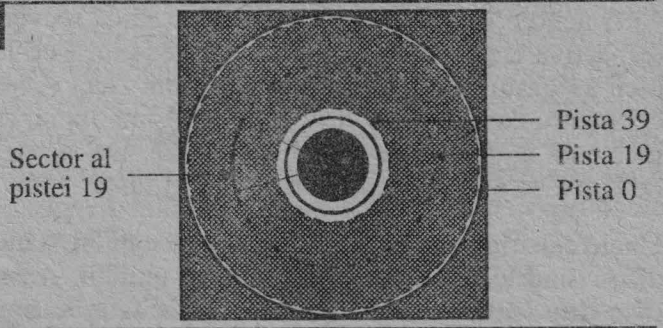


Suportul magnetic pentru unitatea de discuri este **discheta** sau **discul flexibil** (figura 15a, 15b) care este un disc magnetic care poate avea diverse dimensiuni (8 inch, 5,25 inch şi 3,5 inch). Capacitatea de memorie a unei dischete este în funcţie de tipul ei. Unele pot memora un volum de informaţii mai mic decît o casetă magnetică (256 Ko sau 360 Ko), altele pot memora un volum mai mare (720 Ko sau 1200 Ko). Acest volum variază în funcţie de mărimea dischetei şi densităţii înregistrărilor.

Unele discuri flexibile au o singură suprafaţă de înregistrare (faţă), altele prezentînd două suprafeţe de înregistrare fiind deci, dublă faţă. Pe suprafaţa de înregistrare se află dispuse cercuri concentrice care se înfăşoară dinspre centru spre margine. În figura 15a puteţi observa modul în care este organizată o dischetă pe piste şi pe sectoare, iar în figura 15b cum se

fig.15a

Organizarea unei dischete



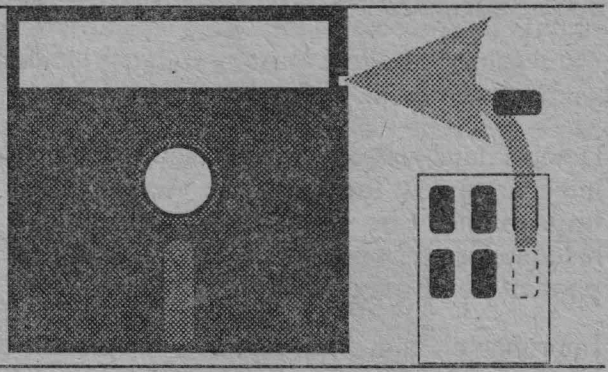
realizează protejarea unei dischete pentru ca informaţiile înregistrate (date sau programe) să nu fie şterse din greşeală.

În interiorul unităţii de discuri se găsesc montate mai multe capete de citire/scriere care înregistrează şi regăsesc (citesc) informaţii de pe **discul magnetic**.

Discurile magnetice ca şi banda (caseta) magnetică reprezintă, deci, suporturi fizice pe care se înregistrează informaţiile. Înregistrările sînt de tip magnetic.

fig.15b

Protejarea unei dischete la scriere



Dacă ar fi să alegem ca memorie externă între casetă sau dischetă, ce aţi prefera? Acasă probabil preferaţi casetofonul pickupului dar, desigur, discul magnetic este de preferat deoarece în cazul utilizării sale, *accesul la informaţie este direct*. Aşa cum atunci cînd dorim să ascultăm o anumită melodie de pe un disc muzical poziţionăm acul chiar la începutul său, tot la fel, în cazul unităţii de discuri de la calculator, atunci cînd dorim să încărcăm un anume program de pe disc şi dăm comanda respectivă, capul de citire al acelei unităţi de discuri se va poziţiona exact pe acel program. Spre deosebire, pentru caseta magnetică, dacă dorim un anumit program

va fi nevoie să *parcurgem secvenţial* (adică secvenţă cu secvenţă) banda respectivă (indiferent dacă derularea casetei este lentă sau rapidă) astfel încît să găsim începutul acelui program. Deci accesul pentru disc faţă de cel pentru casetă, fiind direct este mult mai rapid, iar durata operaţiilor de încărcare şi salvare este mult mai mică. De asemenea informaţia memorată pe disc magnetic este relativ în siguranţă.

Unele calculatoare de tip PC au încorporate în "cutia" calculatorului şi un disc cilindric fixat, deci, în interiorul unităţii. Acest disc se mai numeşte **disc dur (disc "hard")** şi are o capacitate de memorie foarte mare, de circa 10-100 ori mai mare decît capacitatea unei dischete. Aceste discuri au marele avantaj că pot reţine multe programe şi date chiar în interiorul calculatorului.

Unele calculatoare mai mari (sisteme sau minicalculatoare) mai folosesc ca unităţi de memorie externă şi **unităţile de benzi magnetice**. Acestea deşi sînt mai lente (accesul fiind tot secvenţial) au o capacitate de memorie foarte mare. Ele folosesc ca suport de memorie tot banda magnetică (figura 16).

fig.16

Bandă magnetică

Deseori după ce s-a lucrat cu calculatorul este necesar ca rezultatele să fie înregistrate pe o foaie de hîrtie. Acest lucru se realizează prin intermediul **imprimantei** care este un echipament periferic folosit pentru tipărirea rezultatelor prelucrate de calculator. Spunem că imprimanta este un **echipament de extragere**.

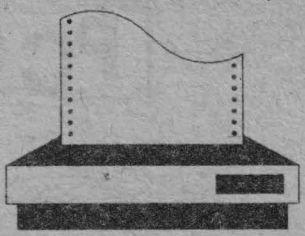
Imprimantele sînt foarte diferite între ele, deosebindu-se prin viteza de imprimare a caracterelor şi putînd fi lente sau rapide (acestea din urmă putînd imprima pînă la mai multe mii de caractere pe secundă). Viteza de imprimare este condiţionată de modul de imprimare folosit iar acesta poate fi lent, dacă se imprimă caracter cu caracter, sau mai rapid, dacă se tipăreşte o linie deodată. Unele imprimante permit tipărirea de la stînga la dreapta şi apoi de la dreapta la stînga, ceea ce măreşte foarte mult viteza de lucru prin eliminarea timpului pierdut la returul de car.

Foarte importantă este *metoda de imprimare*. Există imprimante care imprimă prin lovirea hîrtiei (*imprimare cu impact*) printr-un proces asemănă-

tor cu cel folosit la maşina de scris. Dintre imprimantele cu impact cele mai folosite sînt cele *matriciale*, la care capul de scriere este alcătuit din fire care reprezintă fiecare literă, cifră sau simbol (figura 17a). Un alt tip de imprimantă cu impact este cea cu "margaretă". Margareta reprezintă, de fapt, un cap de imprimare compus din rotiţe care au înglobate litere, cifre şi alte caractere. Roata de tipărire se învîrte pînă cînd caracterul dorit se găseşte într-o poziţie convenabilă pentru imprimare. Aceste imprimante sînt mai lente dar asigură o calitate deosebită a tipăririi.

fig.17a

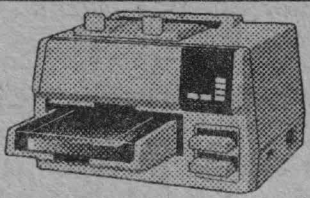
Imprimantă cu impact



Alte metode de imprimare sînt cele folosite la *imprimantele termice* care, după cum arată şi numele, generează caracterele pe hîrtie sensibilă la căldură şi imprimantele care imprimă pe baza unui *jet de cerneală*. Tehnologia imprimării evoluează foarte rapid, în prezent multe imprimante oferind facilităţi grafice ce pot genera ilustraţii, hărţi, planuri, etc. într-o diversitate de dimensiuni şi culori. Tehnologia cea mai cunoscută pentru astfel de imprimări este cea pe bază de *laser* (vezi figura 17b).

fig.17b

Imprimantă cu laser



Imprimanta cu laser îşi găseşte o întrebuinţare şi în realizarea cărţii de faţă. Punerea în pagină (tehnoredactarea) acesteia a fost făcută pe un calculator PC (AT 80386), apoi cartea a fost listată pe o imprimantă cu laser, iar după efectuarea corecturilor, rezultatul a fost fotografiat şi multiplicat de către tipografie.

Calculatoarele în competiţie

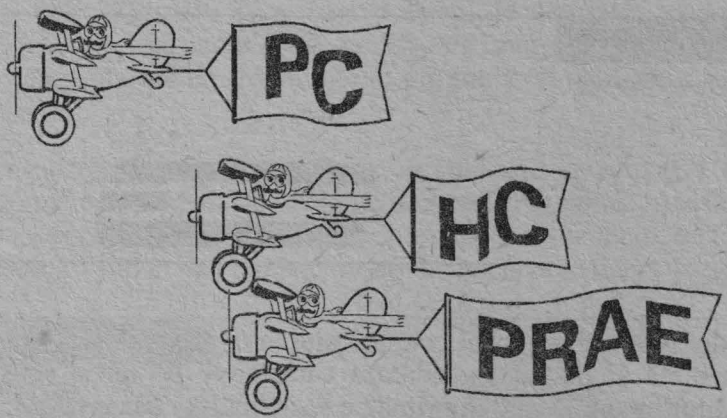


fig.18

Calculatoarele în competiţie

Deseori în utilizarea tehnicii de calcul se pot auzi expresii de genul: "*acest calculator este mai puternic*" sau "*calculatorul X este cel mai rapid*".

Deşi aceste expresii au intrat aproape în limbajul curent al utilizatorilor de calculatoare, cei mai mici dintre aceştia au numai o înţelegere intuitivă asupra acestor expresii. Să intrăm puţin mai în adîncul chestiunii şi să presupunem că avem în faţă cîteva calculatoare de tipuri diferite. Să luăm un calculator compatibil Sinclair Spectrum (HC, de exemplu) şi să tastăm:

```
PRINT (SIN 0.123456789)^2 + (COS 0.123456789)^2
```

Pentru PC nu uitaţi să puneţi argumentele pentru SIN şi COS între paranteze, regula generală la GWBASIC fiind să se utilizeze paranteze pentru argumentele funcţiilor.

Aşa cum ne aşteptam vom obţine valoarea 1 (deoarece $\sin^2 x + \cos^2 x = 1$). Să repetăm acum exemplificarea cu următoarea mică diferenţă (vom modifica argumentul la cosinus la a 9-a poziţie zecimală cu o unitate):

```
PRINT (SIN 0.123456789)^2 + (COS 0.123456788)^2
```

Vom obţine şi de data aceasta valoarea 1 ceea ce nu ne mai mulţumeşte pe deplin deoarece argumentul lui COS este puţin diferit de cel al lui SIN, iar calculatorul nu a fost aşa de puternic încît să sesizeze această mică diferenţă. În schimb cu un calculator PRAE (ei da, bătrînul PRAE care aşa cum îi arată numele a fost primul calculator personal realizat în ţară) vom obţine cu :

```
PRINT (SIN 0.123456789)^2 + (COS 0.123456789)^2
```

valoarea 1, iar cu:

```
PRINT (SIN 0.123456789)^2 + (COS 0.123456788)^2
```

valoarea 1.0000000002, de unde deducem că mica diferenţă a fost sesizată.

În cazul expus, prin "puterea" calculatorului am înţeles, de fapt, precizia sa de calcul, care este dată de numărul de cifre semnificative cu care sînt exprimate numerele. Într-adevăr, HC are o precizie de 9 cifre semnificative iar PRAE-ul de 11 cifre. Acest lucru se poate constata punînd calculatoarele să afişeze diverse numere. De exemplu, la HC cu

```
PRINT 0.123456789012345
```

se va afişa 0.12345679 în timp ce la PRAE se va afişa 0.12345678901.

Deci PRAE este mai puternic? Să nu ne grăbim cu afirmaţia deoarece putem avea surprize. Iată una dintre ele: să efectuăm cu PRAE un simplu calcul de ridicare la pătrat:

```
PRINT 10^2
```

şi spre stupoarea noastră vom obţine rezultatul 99.999999999 care este foarte apropiat de 100 dar nu identic.

Iată altă surpriză şi mai mare. Să luăm un calculator PC AT (care ştim că este mult mai puternic fiind din altă generaţie şi cu o capacitate de memorie mult mai mare) şi să repetăm exerciţiile făcute cu PRAE şi HC prin

intermediul lui GWBASIC încărcat de pe o dischetă. Vom observa că în primul caz cu sinusul şi cosinusul avînd acelaşi argument obţinem valoarea 0.9999999, iar în al doilea caz (cu argumentul lui cosinus modificat cu o unitate la a 9-a zecimală) acelaşi rezultat, ceea ce înseamnă că nici nu a calculat grozav şi nici nu a sesizat diferenţa. Iar acest calculator are o precizie la afişare de nu mai puţin de 16 cifre zecimale (se poate constata deoarece cu `PRINT 10.12345678901234567 89` se va afişa 10.123456789012345).

Puteţi explica de ce se obţin aceste rezultate atît de diferite şi, oarecum, contradictorii?

Viteza calculatoarelor

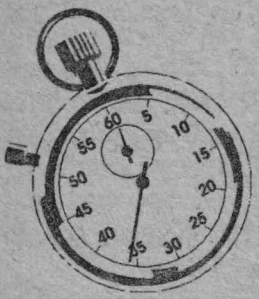
Cele două calculatoare (HC şi PRAE) sînt comparabile deoarece:

- sînt construite pe baza aceluiaşi microprocesor şi anume Z-80 ceea ce le conferă caracteristici asemănătoare (între care cea mai importantă este viteza de calcul)
- posedă aceeaşi capacitate de memorie (şi anume 64 Ko).

De fapt aceste două caracteristici stau la baza a ceea ce numim puterea unui calculator. În schimb calculatorul PC AT este construit pe baza unui microprocesor 80286 şi are o capacitate de memorie de 1Mo, deci are o putere mai mare.

Şi atunci cum se explică rezultatele obţinute?

Deoarece nu am luat în considerare un alt factor care determină puterea calculatoarelor, şi anume, faptul că nu am apelat direct la posibilităţile calculatorului ci printr-un intermediar (în cazul nostru interpretorul BASIC) care este proiectat diferit la diferite calculatoare. În funcţie de felul în care este construit acest interpretor numerele sînt reprezentate în simplă sau dublă precizie, memorîndu-se pe 4 octeţi (la PRAE) sau pe 5 octeţi (la HC) sau pe 8 (la PC în GW BASIC). În general proiectarea se face realizîndu-se un compromis între precizie şi viteză (care este în funcţie de modul în care se face analiza sintactică şi semantică – interpretarea), modul de alocare şi căutare a variabilelor etc.



Să punem acum *calculatoarele în competiţie din punctul de vedere al vitezei lor în BASIC*. Pentru aceasta vom folosi un program care identifică dacă 10 007 este sau nu un număr prim. (În realitate aşa cum am văzut este cel mai mic număr prim mai mare decît 10 000). Algoritmul folosit este cel clasic prin care se verifică dacă numărul este divizibil cu 2 şi apoi începînd cu 3, din 2 în 2, pînă la jumătatea numărului (vezi programul P11):

Ptest.

```
10 LET N = 10007
20 IF N/2 = INT (N/2) THEN PRINT N;
   " nu e prim": STOP
30 FOR I = 3 TO INT (N/2) STEP 2
40 IF N/I = INT (N/I) THEN PRINT N;
   " nu e prim": STOP
50 NEXT I
60 PRINT N ; " este prim"
70 STOP
```

Vom cronometra pentru fiecare calculator intervalul de timp dintre introducerea comenzii de executare a programului (RUN) şi afişarea rezultatului.

Şi iată rezultatele: PRAE dă răspunsul în 92 secunde, HC în 50 secunde, iar PCAT în 5 secunde.

Să calculăm aproximativ cam cîte operaţii (aritmetice şi logice) face calculatorul la executarea acestui program :

- ❑ linia 10: 1 operaţie (de atribuire)
- ❑ linia 20: 3 operaţii (una de împărţire, una de calcul pentru aflarea întregului şi una de comparare a celor două valori)
- ❑ linia 30: 2500 operaţii (5000:2 operaţii de atribuire)
- ❑ linia 40: 7500 operaţii (3 operaţii repetate de 2500 ori cît este ciclul)
- ❑ linia 50: 2500 operaţii (de comparare I cu N/2)
- ❑ linia 60: 1 operaţie (de afişare)

Total: 12505 operaţii

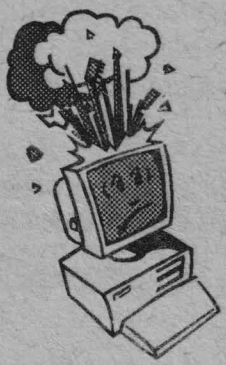
Astfel se poate spune că PRAE-ul are o viteză de 136 în timp ce HC de 250, iar PCAT de 2501 operaţii pe secundă.

Să optimizăm algoritmul modificînd în linia 30 `SQR (N)` în loc de N/2, adică să mergem cu testarea divizibilităţii numai pînă la $\sqrt{N}$. Calculatorul va avea de făcut mai puţini paşi şi, deci, mai puţine operaţii.

În acest caz rezultatele sînt: 3 sec. la PRAE, 2 sec. la HC şi sub 1 sec. la PC AT, ceea ce înseamnă că eficienţa a crescut de circa 25 de ori.

Ce n-a putut rezolva calculatorul

Să se verifice dacă numărul 100 895 598 169 este prim sau nu.



Sigur că, la prima vedere, ne putem gîndi că, avînd la dispoziţie programul optimizat pentru numere prime (mergînd cu testarea pînă la radical din număr), nu avem altceva de făcut decît să punem calculatorul la treabă şi să aşteptăm rezultatul. Dar, stupoare ! Atît calculatorul de tip HC cît şi cel de tip PC AT ne răspunde că 100 895 598 169 este divizibil cu 2 !

Numărul de testat depăşeşte limita de reprezentare pentru amîndouă calculatoarele.

Întrezăriţi voi o altă metodă de rezolvare? Dacă după mai mult timp nu v-a venit nici o idee, nu vă impacientaţi, vă vom da noi răspunsul.

Numărul 100 895 598 169 nu este prim; el are ca divizori pe 898 423 şi 112 303 (acest lucru se poate verifica şi cu calculatorul, dar nu chiar aşa de simplu, ci făcînd înmulţirea pe părţi). Problema trimisă într-o scrisoare lui Fermat de către baronul Messner a fost rezolvată de celebrul matematician de pe o zi pe alta (fără calculator !), acesta nelăsînd, din păcate, şi rezolvarea ei.

Calculatoarele sînt puse să rezolve şi astfel de probleme, munca de cercetare nemaputînd fi concepută fără ajutorul lor. Însă în astfel de cazuri se

folosesc calculatoare foarte puternice, de viteză şi capacitate de memorie foarte mari şi care, în plus, pot realiza şi calcule *în paralel*. De exemplu, recent, cu unul din calculatoarele de acest fel s-a găsit cel mai mare număr prim descoperit pînă în prezent (am văzut că, numerele prime sînt din ce în ce mai rare cînd sînt căutate printre numere foarte mari). Numărul despre care este vorba are nu mai puţin de 227 832 cifre!

Verificaţi-vă cunoştinţele despre calculatoare

Puteţi să vă verificaţi sau să vă îmbogăţiţi cunoştinţele generale despre calculatoare: la ce sînt şi cum sînt utilizate, cum sînt construite, istoricul lor. În acest scop veţi selecta una din opţiunile A (adevărat) sau F (fals) pentru fiecare din afirmaţiile din lista ce urmează verificînd, apoi, răspunsurile obţinute cu cele exacte, indicate în ANEXA A de la pagina 151.

1. Calculatoarele sînt inteligente. Ele pot răspunde practic oricărei întrebări. A F

2. Calculatoarele pot lucra cu cuvinte la fel ca şi cu numere. A F

3. Un calculator poate realiza mii de operaţii pe secundă. A F

4. Memoria calculatoarelor este foarte asemănătoare memoriei (creierului omenesc). A F

5. Benzile magnetice pot fi utilizate ca unităţi de memorie ale calculatorului. A F

6. Un program de calculator este un program TV despre calculatoare. A F

7. Un calculator poate avea o memorie suficient de mare pentru a memora o întreagă enciclopedie de informaţii. A F

8. Un calculator personal reprezintă un calculator aflat în proprietatea particulară a unei persoane. A F

9. Un calculator de buzunar este un calculator personal de dimensiuni foarte mici cu alimentarea prin baterii electrice şi care poate fi transportat eventual în buzunarul unei haine. A F

10. Atît un calculator personal cît şi unul de buzunar poate fi utilizat pentru rezolvarea unor probleme. A F

11. Atît un calculator personal cît şi unul de buzunar poate prelucra atît numere cît şi cuvinte. A F

12. Unul dintre primele dispozitive care ajutau la efectuarea calculelor matematice, abacul, a fost inventat în urmă cu aproape cinci mii de ani. A F

13. Primele calculatoare mecanice cu 4 funcţii au fost construite la începutul secolului XVII-lea. A F

14. Cartelele perforate au fost utilizate în dispozitivele create de Jacquard pentru automatizarea muncii la războaiele de ţesut la începutul secolului al XIX -lea. A F

15. Telefonul, automobilul, radioul şi televiziunea au fost inventate înaintea calculatorului electronic. A F

16. În Anglia, în timpul celui de-al II-lea Război Mondial, Allan Turing a construit un calculator electronic cu care s-au decodificat mesajele secrete transmise de naziști. A F

17. Primul calculator electronic universal este considerat a fi ENIAC, construit în Statele Unite ale Americii în anul 1945. A F

18. Primul calculator electronic universal conținea tranzistoare și avea memoria internă pe ferite. A F

19. Primul calculator electronic utiliza tuburile de vacuum. A F

20. În anul 1947 erau deja instalate în întreaga lume cîteva zeci de calculatoare electronice. A F

21. În anul 1947, IBM singurul producător de calculatoare electronice, a hotărît să nu mai investească în domeniul producției de calculatoare pe motiv că ele nu sînt cerute îndeajuns pe piață. A F

22. Un program pentru calculator reprezintă un set de instrucțiuni care arată ce pași trebuie parcurși pentru rezolvarea unei anumite probleme. A F

23. Sistemul de operare reprezintă un set de programe care are funcția de coordonare și control asupra calculatorului și care facilitează dialogul dintre utilizator și calculator. A F

24. Un program scurt poate fi utilizat uneori pentru a prelucra volume foarte mari de date. A F

25. Majoritatea roboţilor sînt cel puţin tot atît de inteligenţi ca şi o persoană medie. A F

26. Un elevator automat poate fi încadrat în categoria "maşinilor inteligente". A F

27. O formulă, de exemplu, $S = V \times T$ reprezintă un model matematic. A F

28. Regăsirea informaţiilor reprezintă utilizarea cea mai răspîndită a calculatoarelor. A F

29. Calculatoarele sînt utilizate inclusiv pentru realizarea de desene. A F

30. Dacă un program este destul de scurt (mai puţin de 20 de instrucţiuni) rularea (execuţia) sa va dura mai puţin de o secundă. A F

31. O persoană care scrie instrucţiuni pentru calculator se numeşte **operator de calculator**. A F

32. O persoană care instalează şi menţine un calculator se numeşte **inginer de sistem**. A F

33. Înainte de a folosi un calculator este absolut necesară cunoaşterea unor amănunte cum ar fi: cum este construit un calculator, care sînt părţile lui componente, pe ce tehnologie se bazează etc. A F

34. Un program de calculator se poate utiliza şi fără cunoştinţe privitoare la modul în care a fost realizat. A F

35. În urma executării următorului program, pe ecranul calculatorului se va afişa 3.6: A F

```
10 REM
20 PRINT "Rezultatul este: "
30 PRINT 10 - 2 * (5.3 - 2.1)
40 PRINT
45 CLS
50 PRINT 3.6
```

36. În urma executării următorului program, pe ecran se va afişa: A F

```
5
3 + 2 = 5
3 + 2 = 6
3 * 2 = 6
```

```
10 PRINT 3 + 2
20 PRINT "3 + 2 = 5"
30 PRINT "3 + 2 = 6"
40 PRINT 3 * 2 = 6
```

37. Ordinea corectă a liniilor următorului program este ACBDE. A F

```
A CLS
B LET A = B + 2
C INPUT B
D PRINT A , B
E STOP
```

38. Ordinea corectă a liniilor următorului program este ABCDEFG. A F

```
A CLS
B INPUT A
C LET B = A * B
D LET A = A + 2
E LET B = A + 1
F PRINT A , B
G STOP
```

39. După executarea următorului program care afişează suma primelor 10 numere naturale, introducînd comanda `PRINT I` se va afişa 1. A F

```
10 LET S = 0
20 FOR I = 1 TO 10
30 LET S = S + I
40 NEXT I
50 PRINT S
```

40. După executarea următorului program prin care se compară două numere A şi B, pe ecran se va afişa: A F

```
A este egal cu B
A nu este egal cu B
```

```
10 LET A = 3
20 LET B = 3
30 IF A = B THEN PRINT "A este egal cu B"
40 PRINT "A nu este egal cu B"
```

41. Instrucţiunile `INPUT`, `LET` şi `READ` permit citirea unor valori dintr-o listă. A F

42. După executarea următorului program prin care se compară două numere A şi B, pe ecran se va afişa: A F

```
A este egal cu B
A nu este egal cu B
```

```
10 LET A = 3
20 LET B = 3
30 IF A = B THEN PRINT "A este egal cu B":STOP
40 PRINT "A nu este egal cu B"
```

43. După executarea următorului program, pe ecran se va afişa: A F

```
A este mai mic ca B
```

```
10 LET A = 3
20 LET B = 3
30 IF A <> B THEN PRINT "A este mai mic ca B":
   PRINT "A este egal cu B":STOP
40 PRINT "A este mai mic sau egal cu B"
```

44. Instrucţiunile cuprinse în următoarea buclă FOR-NEXT se vor executa de 15 ori: A F

```
30 FOR I = 1 TO 30
40 PRINT I + 1
50 LET I = I + 1
60 NEXT I
```

45. Instrucţiunile din următoarea buclă se vor executa de un număr infinit de ori: A F

```
10 FOR X = 1 TO 10
20 LET X = X - 1
30 NEXT X
```

Mecanismul hazardului

Descori pentru simularea unor situaţii sau fenomene este necesară generarea unor numere întîmplătoare (aleatoare). Putem realiza acest lucru prin intermediul calculatorului folosind funcţia RND. Aceasta generează numere întîmplătoare, cuprinse între 0 şi 1. Este de notat că valoarea 0 poate fi luată nu însă şi valoarea 1.

Să se genereze cu ajutorul calculatorului numere aleatoare întregi şi pozitive mai mici decît un număr dat.

Desigur va trebui ca numărul să fie înmulţit cu RND iar la rezultat să se aplice funcţia întreg (INT). Programul P26 este:

P26.
```
5 REM ** program de generare numere **
6 REM **    aleatoare intregi si    **
7 REM **   pozitive mai mici ca N   **
10 INPUT N
20 PRINT INT (RND * N)
30 GOTO 20
```

De exemplu, dacă dorim să generăm numere aleatoare întregi mai mici decît 20, vom introduce numărul 20 iar apoi:

- dacă cumva RND ia valoarea 0, atunci se va afişa valoarea 0, care este un număr mulţumitor pentru ceea ce se cere
- dacă RND ia o valoare apropiată de 1 (să presupunem că este chiar valoarea 1, cu toate că ea nu poate fi atinsă), atunci se va afişa valoarea 20, care este şi ea mulţumitoare

✘ orice altă valoare pe care o ia RND va avea ca rezultat afişarea unui număr întreg cuprins între 1 şi 20

Am văzut totuşi că însăşi valoarea 20 nu o va lua niciodată, cu toate că acest rezultat ar fi şi el mulţumitor şi n-ar trebui să-l trecem cu vederea. În acest scop vom modifica linia 20 a programului P26 astfel:

```
20 PRINT INT (RND *( N+1))
```

Motivul este lesne de înţeles. De data aceasta nu se va mai atinge în exemplul nostru valoarea 21 care nu ne interesează, în schimb va putea fi atinsă valoarea 20 dar şi oricare altă valoare întreagă mai mică de 20, inclusiv 0 în cazul în care RND a luat valoarea 0.

Un joc cu numere

Să se realizeze un program cu care să se poată juca jocul "Ghiceşte numărul". Calculatorul memorează un număr cuprins între 0 şi 100 iar jucătorul trebuie să-l ghicească introducînd încercări (numere) la care calculatorul va răspunde "Este prea mare" sau " Este prea mic".

Desigur se va folosi funcţia RND pentru a se memora un număr aleator între 0 şi 100. Programul P27 va fi:

P27.
```
5 REM ** joc ghiceste numarul **
10 PRINT "GHICESTE NUMARUL"
20 LET N = INT (RND * 101)
30 INPUT X
40 IF X<N THEN PRINT "Prea mic":GOTO 30
50 IF X>N THEN PRINT "Prea mare":GOTO 30
60 PRINT "Ai ghicit !"
70 PRINT
80 GOTO 10
```

Să se modifice jocul "Ghiceşte numărul" astfel încît atunci cînd numărul este găsit să se indice numărul de încercări din care a fost găsit.

Pentru a ţine socoteala (contabilitatea) încercărilor, vom folosi o tehnică similară cu cea utilizată pentru realizarea sumelor cu calculatorul. Vom lua o variabilă (sac) S. La început în ea nu este nimic deci are valoarea 0:

```
25 LET S = 0
```

Apoi, de fiecare dată cînd se face o încercare vom pune în sac o bobiţă (adică adăugăm la vechea valoare încă o unitate):

```
35 LET S = S + 1
```

Vom modifica şi linia 60 astfel încît să se afişeze numărul de încercări:

```
60 PRINT "Ai ghicit din ";S ; "incercari"
```

Acestă tehnică este des folosită în special la jocuri pentru ţinerea evidenţei scorului, pentru ţinerea evidenţei încercărilor şi, de aceea, se mai numeşte tehnica *"numărării încercărilor"*.

Să-i ajutăm pe cei mici

Să se realizeze un program pentru antrenament la tabla înmulţirii.

Vom genera două numere aleatoare întregi cuprinse între 0 şi 10 şi vom întreba rezultatul înmulţirii lor. Dacă rezultatul este corect se vor genera alte 2 numere şi altă întrebare va apare, iar dacă este incorect se va repeta întrebarea şi se va introduce altă încercare. Programul P28 este:

P28.

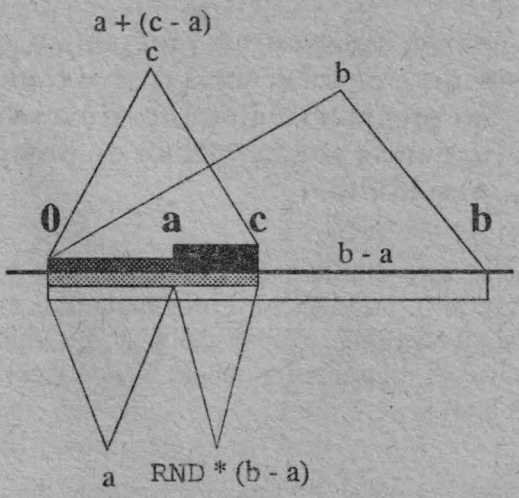
```
5 REM ** program pentru antrenament **
6 REM **      la tabla inmultirii     **
10 CLS: PRINT "Tabla Inmultirii"
20 LET A = INT (RND*11)
30 LET B = INT (RND*11)
40 PRINT "Cit fac "; A; " x "; B; " ? "
50 INPUT C
60 IF C=A*B THEN PRINT " Bine": GOTO 20
70 PRINT "Nu e bine": GOTO 40
```

Să se genereze cu ajutorul calculatorului numere aleatoare întregi cuprinse într-un interval dat.

Să presupunem că dorim să generăm numere aleatoare întregi în intervalul [a, b].

fig.19

Generarea de numere aleatoare întregi într-un interval dat



Desigur aceste margini vor trebui introduse. Problema este asemănătoare cu precedenta (de generare a unor numere aleatoare întregi şi pozitive mai mici decît un număr dat) considerînd faptul că orice număr C din intervalul [A, B] are în componenţa sa valoarea A la care se adaugă o anumită

parte (depinde de numărul aleator generat) din valoarea B-A (vezi figura 19). Programul P29 este:

P29.

```
5 REM ** program de generare numere **
6 REM ** aleatoare intregi cuprinse **
7 REM **    intr-un interval dat     **
10 INPUT A,B
20 PRINT A + INT (RND * ( B - A + 1))
30 GOTO 20
```

Cînd RND ia valoarea 0 atunci numărul generat va fi A, cînd RND ia o valoare apropiată de 1 atunci numărul generat va lua valoarea A + B - A = B, iar cînd RND ia orice altă valoare numărul generat va lua o valoare între A şi B.

Cîteva probleme cu numere şi grafică

Să se realizeze un program cu ajutorul căruia să se deseneze pe tot ecranul cît mai multe puncte. Să se realizeze un program cu ajutorul căruia să se deseneze cît mai multe puncte situate într-un dreptunghi în partea dreaptă sus a ecranului.

Problema este similară cu generarea unor numere aleatoare întregi cuprinse între 0 şi 255 (pentru orizontala punctului) şi între 0 şi 175 (pentru verticala punctului), pentru calculatoare compatibile Sinclair Spectrum, aceste valori menţinînd punctele în interiorul ecranului grafic. Programul P30.A va fi:

P30.A

```
5 REM ** program de desenare puncte **
6 REM **          pe ecran           **
7 REM **   pentru calculator HC      **
10 PLOT INT(RND * 256), INT(RND * 176)
20 GOTO 10
```

Pentru calculatoare PC programul similar este P30.B. În acest caz numerele aleatoare întregi vor fi cuprinse între 0 şi 320 (pentru orizontală) şi 0 şi 200 (pentru verticală):

P30.B

```
5 REM ** program de desenare puncte **
6 REM **          pe ecran           **
7 REM **    pentru calculator PC     **
8 CLS
9 SCREEN 1
10 PSET (INT(RND*320),INT(RND*200)),7
20 GOTO 10
```

In al doilea caz problema este similară tot cu generarea unor numere aleatoare cuprinse într-un interval dat. Pentru a se încadra într-un dreptunghi situat în partea dreaptă sus a ecranului să presupunem că orizontala punctelor va fi cuprinsă între limitele 170 şi 220, iar verticala între 110 şi 140. Programul P31 va fi:

P31.A

```
5 REM **     program de desenare     **
6 REM ** puncte intr-un dreptunghi **
7 REM **     pentru calculator HC    **
10 PLOT 170+INT(RND*51),110+INT(RND*31)
20 GOTO 10
```

Valorile 51 şi 31 s-au calculat astfel:

$$220 - 170 + 1 = 51$$

şi

$$140 - 110 + 1 = 31$$

Pentru calculatoare PC programul similar este P31.B. Dacă se păstrează aceleaşi valori pentru coordonate ca în cazul programului pentru HC, atunci se formează un dreptunghi în partea dreaptă jos a ecranului deoarece acum punctul de origine este în colţul din stînga sus.

P31.B

```
5 REM **     program de desenare     **
6 REM ** puncte intr-un dreptunghi **
7 REM **    pentru calculator PC     **
8 CLS
```

```
9 SCREEN 1
10 PSET(170+INT(RND*51),
   110+INT(RND*31)),7
20 GOTO 10
```

Să se facă un program care să afişeze în diverse locuri pe ecran litera a. Apariţia literei a va fi însoţită de o notă muzicală întîmplătoare.

Fiind vorba de afişarea unui caracter va trebui să generăm numere aleatoare întregi cuprinse între 0 şi 21 (pentru a indica linia pe care se va afişa caracterul) şi între 0 şi 31 pentru coloana pe care se va afişa caracterul. Nota muzicală poate avea aceeaşi durată, dar înălţimea va fi aleatoare. Programul P32.A pentru calculatoare HC este:

P32.A

```
5 REM ** program pentru afisarea **
6 REM **      unei litere         **
7 REM **   pentru calculator HC   **
10 PRINT AT INT(RND*22),INT(RND*32);"a"
20 BEEP .5, INT (RND * 70)
30 CLS
40 GOTO 10
```

Programul calculatoare PC programul similar este P32.B. S-a indicat modul text cu 40 de coloane (SCREEN 0) şi 25 de rînduri linii numerotate de la 1.

P32.B

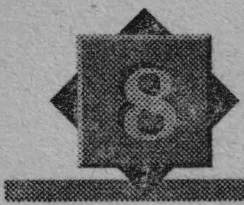
```
5 REM ** program pentru afisarea **
6 REM **       unei litere        **
7 REM **   pentru calculator PC   **
8 CLS
9 SCREEN 0
10 LOCATE 1+INT(RND*24), 1+INT(RND*39)
20 PRINT "a"
30 CLS
40 GOTO 10
```

Să se genereze numere aleatoare întregi de N cifre.

Problema este similară generării numerelor aleatoare întregi într-un interval, dacă se observă că cel mai mic număr de N cifre (corespunzător limitei stînga a intervalului) este $10^{(N-1)}$ iar cel mai mare număr de N cifre (corespunzător limitei dreapta a intervalului) este 10^N-1. Într-adevăr, pentru numere întregi de 3 cifre cel mai mic număr este 100 care este egal cu $10^{(3-1)}=10^2$, iar cel mai mare număr este 999 care este egal cu 10^3 - 1. Programul P33 va fi :

P33.

```
5 REM ** program de generare numere **
6 REM **    aleatoare intregi        **
7 REM **       de N cifre            **
10 INPUT N
20 PRINT 10^(N-1)+INT(RND*((10^N-1)-
   10^(N-1)+1))
30 GOTO 20
```



Probleme cu căutări şi inversiuni

Să se genereze un şir de N numere aleatoare întregi pozitive mai mici ca 100, apoi cel mai mare număr să-şi schimbe poziţia cu cel mai mic şi să se afişeze şirul.

După rezervarea spaţiului de memorie pentru N numere şi generarea numerelor va trebui să aflăm cel mai mare dintre numere (*maximul*) şi, respectiv, cel mai mic (*minimul*).

Pentru aflarea maximului procedăm astfel: considerăm că maximul este chiar primul număr (poziţia C a maximului fiind prima) şi îl comparăm succesiv cu fiecare din numerele din şir. La o comparare dacă numărul cu care comparăm maximul este mai mic atunci se trece la următoarea comparare (cu numărul următor), dar dacă la un moment dat numărul din şir este mai mare decît maximul, atunci îl luăm pe el ca fiind maximul. In acest mod după compararea maximului cu toate numerele din şir, în final, în variabila care memorează maximul se va afla chiar cel mai mare număr din şir.

Pentru aflarea minimului procedăm invers faţă de modul de aflare a maximului(operaţiile de aflare a minimului şi maximului se vor realiza în paralel): considerăm că cel mai mic număr este chiar primul (poziţia D fiind şi ea prima), apoi, în mod asemănător comparăm acest minim, succesiv, cu toate numerele din şir. Dacă găsim un număr mai mic decît minimul îl luăm pe el ca minim. În final în variabila minim vom regăsi cel mai mic număr din şir.

Avînd minimul şi maximul va trebui să le schimbăm între ele. În acest scop vom reţine în prealabil atît poziţia minimului cît şi cea a maximului. Inversarea celor două numere se va face prin **metoda celor trei pahare.** Conform acestei metode, pentru a schimba între ele conţinutul diferit a două pahare, avem nevoie de un al treilea pahar (gol) care va avea rolul de a primi (memora) conţinutul unuia dintre cele două pahare. Acesta golindu-se, va putea "primi" conţinutul celui de-al doilea pahar, care, la rîndul său golit va putea "primi" conţinutul păstrat doar temporar în cel de-al 3-lea pahar.

Programul P34 va fi:

P34.

```
5 REM **  program 1 cu cautari    **
6 REM ** si inversiuni de numere **
10 INPUT N
20 DIM A(N)
30 FOR I = 1 TO N
40 LET A(I) = INT (RND*100)
50 NEXT I
60 LET MAX = A(1) : LET C = 1
70 LET MIN = A(1) : LET D = 1
80 FOR I = 2 TO N
90 IF MAX < A(I) THEN LET MAX = A(I):
   LET C = I
100 IF MIN > A(I) THEN LET MIN = A(I):
   LET D = I
110 NEXT I
120 LET P = A(C)
130 LET A(C) = A(D)
140 LET A(D) = P
150 FOR I = 1 TO N
160 PRINT A(I); " ";
170 NEXT I
180 PRINT
```

Să se realizeze un program cu ajutorul căruia să se caute un element dintr-o mulţime (de exemplu, un număr dintr-o listă de numere) şi, dacă se găseşte, să se treacă la sfîrşitul listei în locul ultimului, toate celelalte elemente situate după el mutîndu-se înapoi cu o poziţie.

Va trebui să introducem cîte numere sînt în listă şi să dimensionăm un spaţiu de memorie pentru aceste numere. Apoi vom introduce numerele respective A(I).

În exemplul dat vom genera numere aleatoare întregi cuprinse, de exemplu, între 0 şi 100. Vom indica şi numărul pe care vom încerca să îl găsim introducînd, deci, un număr întreg X cuprins între 0 şi 100. Căutarea elementului se va realiza prin compararea numărului X cu fiecare număr A(I) din listă.

Acest tip de căutare se mai numeşte **lineară** şi reprezintă singura metodă de căutare a unui element dintr-o mulţime de elemente care s-a format aleator.

Dacă lista de elemente ar fi fost generată după o anumită regulă, de exemplu, dacă numerele ar fi fost puse în ordine crescătoare, atunci s-ar fi putut utiliza o metodă mai eficientă (de exemplu, cea prin care se încearcă elementul situat în mijlocul listei, metodă care se mai numeşte **binară**, deoarece se bazează pe împărţirea listei în două subliste).

Dacă prin parcurgerea întregii liste nu se găseşte nici un element identic cu cel introdus aceasta înseamnă că elementul căutat nu se găseşte în listă şi procesul se va repeta, adică, se va introduce de la tastatură alt număr pe care ne propunem să îl găsim. Dacă elementul este găsit atunci se memorează în variabila **C** poziţia **I** în care se găsea elementul şi se va începe procesul de mutare a elementelor care se găsesc după **X** (adică sînt în dreapta lui) cu o poziţie spre stînga. Practic se va trece conţinutul unei variabile (locaţie de memorie) aflate în poziţia **I+1** în variabila (locaţia) din poziţia **I**. Nu trebuie să ne temem de pierderea conţinutului variabilei din poziţia **I**, deoarece acesta este reprezentat de valoarea **A(I)**, pe care o cunoaştem, fiind egală cu valoarea **X**. Apoi prin transferul conţinutului a

cîte unei variabile din dreapta în variabila situată cu o poziţie spre stînga se eliberează locaţia pentru următorul transfer.

În final nu uităm să punem şi pe ultima poziţie **N**, valoarea elementului **X** şi apoi să afişăm lista astfel obţinută (vezi figura 20 în care se indică şi ordinea logică de executare a operaţiilor).

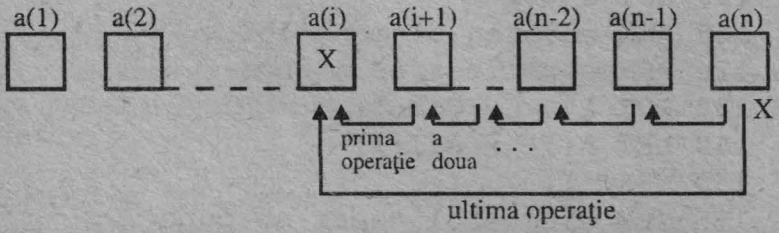


fig.20

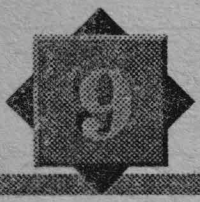
Reprezentarea grafică cu indicarea ordinii logice a operaţiilor pentru găsirea unui element dat într-o listă de elemente şi trecerea sa la sfîrşitul listei

Programul P35 este:

P35

```
5 REM **  program 2 cu cautari    **
6 REM ** si inversiuni de numere **
10 INPUT N
20 DIM A(N)
25 REM generare numere intregi si
26 REM pozitive mai mici
27 REM sau egale cu 100
30 FOR I = 1 TO N
40 LET A(I) = INT (RND * 101)
50 NEXT I
55 REM introducere element de cautat
60 INPUT X
65 REM cautare element
70 FOR I = 1 TO N
80 IF A(I) = X THEN GOTO 120
90 NEXT I
95 REM elementul cautat nu e in lista
```

```
100 PRINT "Elementul ";X; " nu se afla in
    lista. Introduceti pentru cautare
    alt element"
110 GOTO 60
115 REM elementul a fost gasit. Se
116 REM memoreaza pozitia lui in lista
120 LET C = I
125 REM mutarea elementelor cu o
126 REM pozitie spre stinga
130 FOR I = C TO N-1
140 LET A(I) = A(I+1)
150 NEXT I
155 REM elementul gasit se pune
156 REM pe ultima pozitie
160 LET A(N) = X
165 REM se afiseaza lista obtinuta
170 FOR I = 1 TO N
180 PRINT A(I) ; " ";
190 NEXT I
200 PRINT
```



Probleme cu cuvinte

Ca şi în cazul numerelor se pot rezolva cu ajutorul calculatorului şi probleme cu cuvinte (sau în termeni informatici "*şiruri de caractere*"), acestea putînd fi memorate (adică puse în locaţii de memorie) în mod asemănător.

Calculatorul va trebui prevenit de faptul că într-o locaţie de memorie urmează să se "pună" un cuvînt iar acest lucru se face prin numele atribuit locaţiei (variabilei) care trebuie să fie o literă urmată de semnul dolarului ($). Spunem că este o **variabilă alfanumerică** sau de **tip şir de caractere**. Deosebirea lucrului cu cuvinte faţă de cel cu numere constă în faptul că atunci cînd se rezervă un spaţiu de memorie pentru cuvinte va trebui indicat nu numai numărul maxim de cuvinte care se intenţionează a se memora ci şi mărimea cuvîntului maxim (din cîte caractere va fi format cel mai lung cuvînt).

Să presupunem că citim cu o instrucţiune `READ` mai multe nume de familie (şi anume 12) preluate dintr-o linie `DATA`, iar cel mai lung dintre nume presupunem (iniţial) că nu va depăşi 10 caractere (litere). Într-o astfel de linie după cuvîntul cheie `DATA` se trec datele (valori, şiruri de caractere) care urmează a fi citite, unele după altele separate prin virgulă.

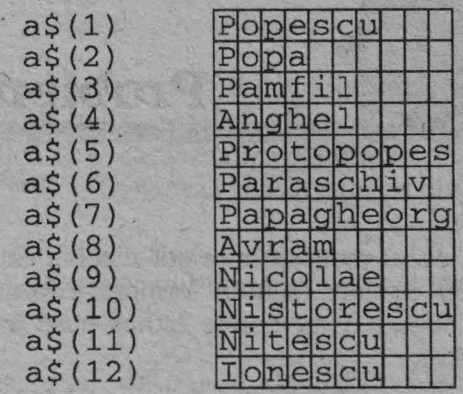
Printr-o instrucţiune `DIM A$ (12,10)` vom rezerva un spaţiu de memorie de 12 cuvinte (cîte au fost citite din linia `DATA`) formate din maxim 10 caractere.

În figura 21 se poate observa cum va arăta memoria în cazul în care linia `DATA` arată astfel:

```
DATA "Popescu", "Popa", "Pamfil", "Anghel",
"Protopopescu", "Paraschiv", "Papagheorghe",
"Avram", "Nicolae", "Nistorescu", "Nitescu",
"Ionescu"
```

fig.21

Reprezentarea unui spaţiu de memorie prin utilizarea unei variabile alfanumerice bidimensionale pentru memorarea unor nume



Se observă că toate cuvintele vor avea, de fapt, aceeaşi lungime pe care am declarat-o, şi anume 10 caractere:

- ✘ dacă şirul de caractere iniţial avea mai puţin de 10 caractere atunci locaţia de memorie se "umple" cu spaţii goale (blancuri) pînă la a 10-a poziţie. Este cazul numelor Popescu, Popa, Pamfil, Anghel, Paraschiv, Avram, Nicolae, Niţescu şi Ionescu
- ✘ dacă şirul de caractere iniţial avea 10 caractere atunci el intră normal în locaţia de memorie. Este cazul numelui Nistorescu
- ✘ dacă şirul de caractere iniţial avea mai mult de 10 caractere, atunci vor intra în locaţie numai primele 10 caractere, urmînd ca restul să nu mai fie memorate. Este cazul numelor Protopopescu şi Papagheorghe. Spunem că acest fel de memorare este de **tip procustian**.

În legenda "Patul lui Procust", acesta tăia, după dimensiunea patului său, din picioarele oamenilor mai înalţi care nu încăpeau în pat. În mod similar caracterele (literele) care nu mai au loc în locaţia de memorie dimensionată la o anumită lungime şi o depăşesc, nu se vor mai memora (vor fi "tăiate").

În ceea ce priveşte compararea cuvintelor în vederea punerii lor în ordine alfabetică, acest lucru se va realiza de către calculator în mod similar cu compararea numerelor şi punerea acestora în ordine crescătoare (sau des-

crescătoare) deoarece caracterele sînt memorate sub forma codului ASCII, care este un cod numeric.

Să se realizeze un program prin care să se citească mai multe nume de familie apoi să se introducă de la tastatură o literă mare. Să se afişeze toate numele care încep cu acea literă.

Explicaţiile pentru realizarea acestui program au fost deja date. Va trebui să rezervăm un spaţiu de memorie de N locaţii şi să imaginăm din cîte litere este format cel mai mare nume. Dacă un nume nu va încăpea în locaţie acest lucru nu ne deranjează prea mult deoarece ne interesează numai începutul cuvîntului nu şi-sfîrşitul său. Primul caracter din fiecare nume se va compara cu litera introdusă X$ şi dacă acestea sînt identice atunci se va afişa numele. Programul P36.A pentru calculatoarele Sinclair Spectrum este:

P36.A

```
5 REM **  program cu care se pot **
6 REM ** afisa numele care incep **
7 REM **   cu o anumita litera   **
8 REM **   pentru calculator HC  **
10 INPUT N
20 DIM A$(N,10)
30 FOR I = 1 TO N
40 READ A$(I)
50 PRINT A$(I)
60 NEXT I
70 PRINT
80 INPUT X$
90 FOR I = 1 TO N
100 IF A$(I,1) = X$ THEN PRINT A$(I)
110 NEXT I
120 PRINT
130 GOTO 80
```

```
140 DATA "Popescu", "Popa", "Pamfil",
    "Anghel", "Protopopescu",
    "Paraschiv", "Papagheorghe",
    "Avram", "Nicolae", "Nistorescu",
    "Nitescu", "Ionescu"
```

Dacă introducem litera P obţinem:

```
Popescu
Popa
Pamfil
Protopopescu
Paraschiv
Papagheorghe
```

iar dacă introducem litera A obţinem:

```
Anghel
Avram
```

Să se realizeze un program prin care să se introducă mai multe nume de familie apoi să se introducă de la tastatură o literă şi să se afişeze toate numele care se termină cu acea literă.

Pornind de la programul precedent pentru HC şi avînd dimensionat un spaţiu de memorie de N cuvinte a cîte 10 caractere practic am declarat fiecare cuvînt ca avînd o lungime de 10 caractere. În acest caz putem folosi funcţia LEN pentru a memora lungimea fiecărui cuvînt (şi astfel a identifica a cîta literă este ultima) deoarece toate cuvintele sînt de lungime 10. Nu putem face altceva decît să folosim un artificiu şi, anume, să cercetăm locul (poziţia) în cadrul cuvîntului unde apare primul spaţiu liber (blancul), iar ultimul caracter al cuvîntului va fi precedentul. Folosind această metodă va trebui să avem însă grijă ca să dăm o lungime în orice caz acoperitoare pentru orice nume (de exemplu 20) astfel încît să fim siguri că în cadrul numelor (la coadă) va apare cel puţin un spaţiu liber. Programul P37.A (numai pentru calculatoare HC) este:

P37.A

```
5 REM ** program pentru afisarea **
6 REM ** numelor care se termina **
7 REM **   cu o anumita litera   **
8 REM **   pentru calculator HC  **
10 INPUT N
20 DIM A$(N,20)
30 FOR I = 1 TO N
40 INPUT A$(I)
50 PRINT A$(I)
60 NEXT I
70 PRINT
80 INPUT X$
90 FOR I = 1 TO N
100 FOR J = 1 TO 20
110 IF A$(I,J) = " " THEN GOTO 130
120 NEXT J
130 IF A$(I,J-1) = X$ THEN PRINT A$(I)
140 NEXT I
150 PRINT
160 GOTO 80
```

Altă metodă este să folosim totuşi funcţia LEN pentru a identifica ultima literă a cuvîntului. Pentru aceasta se va introduce numele fără a se rezerva un spaţiu de memorie şi se va memora lungimea numelui. Apoi se va trece conţinutul variabilei care memorează numele **A$** în altă variabilă **B$** pentru care s-a declarat un spaţiu de memorie şi o lungime suficientă. Fiecare ultimă literă a variabilei **B$** adică **B$(I, L(I))** se va compara cu litera căutată. Programul P38 este:

P38.A

```
5 REM ** program pentru afisarea **
6 REM ** numelor care se termina **
7 REM **   cu o anumita litera   **
8 REM **  folosind functia LEN   **
9 REM **  pentru calculator HC   **
10 INPUT N
20 DIM B$(N,20) : DIM L(N)
30 FOR I = 1 TO N
```

```
40 INPUT A$
50 PRINT A$
60 LET L(I) = LEN A$
70 LET B$(I) = A$
80 NEXT I
90 PRINT
100 INPUT X$
110 FOR I = 1 TO N
120 IF B$(I,L(I)) = X$ THEN PRINT B$(I)
130 NEXT I
140 PRINT
150 GOTO 100
```

Programul similar pentru calculatoare PC este P38.B:

P38.B

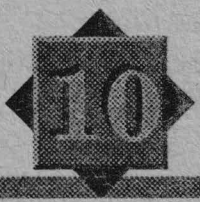
```
5 REM  ** program pentru afisarea **
6 REM  ** numelor care se termina **
8 REM  **   cu o anumita litera   **
9 REM  **  folosind functia LEN   **
10 REM **  pentru calculator PC   **
11 INPUT N
20 DIM B$(N) : DIM L(N)
30 FOR I = 1 TO N
40 INPUT A$
50 PRINT A$
60 LET L(I) = LEN (A$)
70 LET B$(I) = A$
80 NEXT I
90 PRINT
100 INPUT X$
110 FOR I = 1 TO N
120 IF MID$ (B$(I),L(I),1) = X$ THEN
    PRINT B$(I)
130 NEXT I
140 PRINT
150 GOTO 100
```

Să se realizeze un program cu care să se facă un catalog, adică numele să fie trecute în ordine alfabetică.

După introducerea celor **N** nume de maximum **M** caractere se procedează la fel ca la ordonarea numerelor, comparîndu-se numele între ele pe perechi. Indicele **I** arată a cîta trecere prin şirul de elemente se face, iar indicele **J** arată poziţia cuvîntului (al cîtelea nume este). Vor fi necesare maximum **N-1** treceri. Dacă numele dintr-o pereche nu sînt în ordine bună, adică primul este mai mare decît al doilea, atunci se schimbă ordinea între ele, aplicîndu-se metoda celor trei pahare la fel ca la numere. Programul P39.A este:

P39.A

```
5 REM ** program pentru afisare **
6 REM **  in ordine alfabetica  **
7 REM **  pentru calculator HC  **
10 INPUT N , M
20 DIM A$(N,M)
30 FOR I = 1 TO N
40 INPUT A$(I)
50 PRINT A$(I)
60 NEXT I
70 PRINT
80 FOR I = 1 TO N-1
90 FOR J = 1 TO N-I
100 IF A$(J) <= A$(J+1) THEN GOTO 140
110 LET B$ = A$(J)
120 LET A$(J) = A$(J+1)
130 LET A$(J+1) = B$
140 NEXT J
150 NEXT I
160 FOR I = 1 TO N
170 PRINT A$(I)
180 NEXT I
190 PRINT
```



Funcţii

7 probleme pentru funcţia de gradul I

Problema 1. **Să se deseneze prin puncte un segment de 256 pixeli (puncte) în partea de jos a ecranului grafic al calculatorului (este de fapt axa OX).**

Rezolvare: deoarece lungimea liniei este de 256 de pixeli, variabila de ciclare va avea limitele 0 şi 255. Se va reprezenta grafic funcţia

$$y = 0$$

Pentru trasarea punctelor se va modifica coordonata pe orizontală (X) care va creşte de la 0 la 255, în timp ce coordonata pe verticală (Y) va rămîne constantă (0). Pentru o înţelegere mai bună, să notăm şi în program variabila pe orizontală (abscisa) cu X, iar cea care reprezintă coordonatele pe verticală (ordonata) cu Y. Programul P40.A va fi:

P40.A

```
5 REM ** program de trasare a unui **
6 REM **    segment prin puncte    **
7 REM **  pentru calculatorul HC   **
10 FOR X = 0 TO 255
20 PLOT X , 0
30 NEXT X
```

Pentru calculatoare PC programul similar este P40.B în care se ţine seama de dimensiunile ecranului grafic:

P40.B

```
5 REM ** program de trasare a unui **
6 REM **    segment prin puncte    **
7 REM **  pentru calculatorul PC   **
```

```
8 SCREEN 1
9 CLS
10 FOR X = 0 TO 319
20 PSET (X , 199) , 7
30 NEXT X
```

Problema 2. **Să se deseneze prin puncte un segment de 100 pixeli la înălţimea de 80 de pixeli faţă de partea de jos a ecranului grafic.**

Rezolvare: variabila de ciclare va avea limitele 1 şi 100. Practic se va reprezenta dreapta care este graficul funcţiei

$$y = 80$$

Programul P41.A va fi:

P41.A

```
5 REM ** program de trasare a unui **
6 REM **    segment prin puncte    **
7 REM **   la o anumita inaltime   **
8 REM **   pentru calculatorul HC  **
10 FOR X = 1 TO 100
20 PLOT X , 80
30 NEXT X
```

Pentru calculatoare PC programul similar care trasează o dreaptă la înălţimea de 50 pixeli de partea de jos a ecranului este P41.B:

P41.B

```
5 REM ** program de trasare a unui **
6 REM **    segment prin puncte    **
7 REM **   la o anumita inaltime   **
8 REM **   pentru calculatorul PC  **
9 SCREEN 1
10 CLS
11 FOR X = 0 TO 319
20 PSET (X , 150) , 7
30 NEXT X
```

Problema 3. **Să se deseneze prin puncte o linie cu lungimea de 100 de pixeli pe verticală începînd de la înălţimea de 50 de pixeli faţă de partea de jos din stînga a ecranului.**

Rezolvare: variabila de ciclare va avea limitele 50 şi 150. Prima limită are valoarea 50 deoarece segmentul începe să se traseze de la înălţimea de 50 de pixeli. A doua limită se obţine prin adăugarea la valoarea iniţială (50) a mărimii segmentului (creşterea pe verticală) care este 100. Linia desenată se va situa pe dreapta care este graficul funcţiei

$$x = 50$$

Programul P42.A este :

P42.A
```
5 REM ** program de trasare a unui **
6 REM ** segment pe verticala prin **
7 REM **          puncte            **
8 REM **  pentru calculatorul HC    **
10 FOR Y = 50 TO 50+100
20 PLOT 50 , Y
30 NEXT Y
```

Pentru calculatoare PC programul similar este P42.B

P42.B
```
5 REM ** program de trasare a unui **
6 REM ** segment pe verticala prin **
7 REM **          puncte            **
8 REM **  pentru calculatorul PC    **
7 SCREEN 1
8 CLS
10 FOR Y=200-50 TO (200-50)-100 STEP -1
20 PSET (50 , Y) , 7
30 NEXT Y
```

Problema 4. Să se deseneze prin puncte cel mai mare dreptunghi care se poate desena pe ecranul grafic al calculatorului.

Rezolvare: prima latură a dreptunghiului o avem deja prin programul de la problema 1. Pentru dreapta următoare, abscisele punctelor vor fi aceleaşi (255), în timp ce ordonatele vor varia de la 0 la 175, adică atît cît permit mărimile coordonatelor ecranului pe înălţime, rezultînd o dreaptă cu lungimea de 176 de pixeli. Liniile în continuarea programului P42.A sînt :

```
40 FOR Y = 0 TO 175
50 PLOT 255 , Y
60 NEXT Y
```

Pentru următoarea dreaptă şi anume lungimea dreptughiului situată în partea superioară a ecranului, punctul de pornire este cel corespunzător coordonatelor **X = 255** şi **Y = 175**. Ordonata va rămîne constantă, iar cea pe orizontală va scădea de la 255 la 0 (deci `STEP -1`). Programul în continuare este:

```
70 FOR X = 255 TO 0 STEP -1
80 PLOT X , 175
90 NEXT X
```

În sfîrşit, pentru ultima latură, de data aceasta, abscisa va rămîne constantă (X = 0), în timp ce, ordonata va scădea de la 175 la 0. Vom adăuga programului P42.A liniile :

```
100 FOR Y = 175 TO 0 STEP -1
110 PLOT 0 , Y
120 NEXT Y
```

Pentru ca programul să meargă mai "frumos" se pot insera instrucţiuni `BEEP` după fiecare punct care se înscrie pe ecran, deci la numerele de linie 25, 55, 85 şi 115 :

```
BEEP 0.01,0
```

Se observă că durata sunetului este foarte mică, iar nota este **do**.

Problema 5. **Să se deseneze prin puncte cel mai mare pătrat care se poate înscrie pe ecranul grafic al calculatorului şi apoi să se traseze şi diagonalele sale.**

Rezolvare: deoarece laturile pătratului sînt egale, cel mai mare pătrat care se poate desena va avea latura egală cu 176 de pixeli. Programul P43.A este :

P43.A

```
5 REM ** program de desenare prin  **
6 REM **  puncte a celui mai mare  **
7 REM **   patrat care se poate    **
8 REM ** inscrie in ecranul grafic **
9 REM **   pentru calculatorul HC  **
10 FOR X = 0 TO 175
20 PLOT X , 0
30 NEXT X
40 FOR Y = 0 TO 175
50 PLOT 175 , Y
60 NEXT Y
70 FOR X = 175 TO 0 STEP -1
80 PLOT X , 175
90 NEXT X
100 FOR Y = 175 TO 0 STEP -1
110 PLOT 0 , Y
120 NEXT Y
```

Faţă de programul pentru dreptunghi se observă că diferă practic doar liniile de program 10 şi 70 care dau o altă dimensiune maximă abscisei (176 faţă de 256).

Programul similar pentru calculatoare PC este P43.B:

P43.B

```
5 REM ** program de desenare prin  **
6 REM **  puncte a celui mai mare  **
7 REM **   patrat care se poate    **
8 REM ** inscrie in ecranul grafic **
9 REM **   pentru calculatorul PC  **
10 SCREEN 1
```

```
11 CLS
15 FOR X = 0 TO 199
20 PSET (X , 199) , 7
30 NEXT X
40 FOR Y = 199 TO 0 STEP -1
50 PSET (199 , Y) , 7
60 NEXT Y
70 FOR X = 199 TO 0 STEP -1
80 PSET (X , 0) , 7
90 NEXT X
100 FOR Y = 0 TO 199
110 PSET (0 , Y) , 7
120 NEXT Y
```

Programul se poate scrie şi mai concis, desenîndu-se în acelaşi timp cîte 2 laturi şi folosind o singură variabilă pentru ciclare (programul P44.A şi respectiv P44.B pentru PC):

P44.A

```
5 REM ** program de desenare prin  **
6 REM **  puncte a celui mai mare  **
7 REM **   patrat care se poate    **
8 REM ** inscrie in ecranul grafic **
9 REM **   varianta imbunatatita   **
10 REM **  pentru calculatorul HC  **
15 FOR X = 0 TO 175
20 PLOT X , 0
30 PLOT 175 , X
40 NEXT X
50 FOR X = 175 TO 0 STEP -1
60 PLOT X , 175
70 PLOT 0 , X
80 NEXT X
```

P44.B

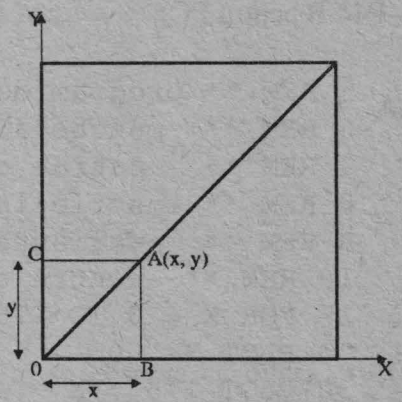
```
5 REM ** program de desenare prin  **
6 REM **  puncte a celui mai mare  **
7 REM **   patrat care se poate    **
8 REM ** inscrie in ecranul grafic **
```

```
9 REM **   varianta imbunatatita    **
10 REM **  pentru calculatorul PC   **
11 SCREEN 1
12 CLS
15 FOR X = 0 TO 199
20 PSET (X , 199) , 7
30 PSET (199 , X) , 7
40 NEXT X
50 FOR X = 199 TO 0 STEP - 1
60 PSET (X , 0) , 7
70 PSET (0 , X) , 7
80 NEXT X
```

Pentru prima diagonală se pune problema coordonatelor unui punct de pe diagonala unui pătrat (vezi figura 22).

fig.22

Coordonatele unui punct de pe diagonala pătratului



Să luăm la întîmplare un punct A de pe diagonala, de coordonate **X** şi **Y**. Ne interesează să exprimăm coordonata **Y** în funcţie de coordonata **X** sau, cu alte cuvinte, cît este ordonata cînd cunoaştem abscisa (abscisa o cunoaştem, deoarece este cea care va lua valori 0,1 ... 100, conform variabilei contor). Fiind vorba de un pătrat, *punctele de pe diagonală sînt egal depărtate de laturile pătratului (AB=AC)*. Dar AC = OB şi AB = OC, iar OB este abscisa, iar OC este ordonata. Deci ordonata (**Y**) va fi egală cu abscisa (**X**). Diagonala se va situa pe dreapta reprezentată de funcţia

$$y = x$$

În cazul nostru, pentru diagonală, coordonata pe orizontală creşte de la 0 la 175 (`FOR X = 0 TO 175`).

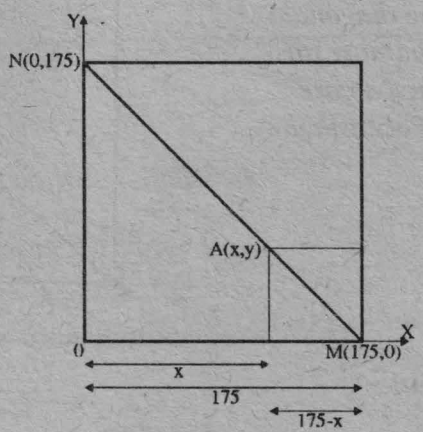
Programul P44.A se completează cu liniile:

```
90 FOR X = 0 TO 175
100 PLOT X , X
110 NEXT X
```

Acum să trasăm şi cealaltă diagonală care va porni, de exemplu, din colţul pătratului de coordonate (175,0) şi va ajunge în colţul de coordonate (0,175). *Pe ce dreaptă se va situa această diagonală?* (vezi figura 23).

fig.23

Reprezentarea grafică a diagonalei pătratului



O bună metodă pentru identificarea unei funcţii este să cercetăm extremităţile acesteia. Astfel, în punctul M : **x = 175** şi **y = 0**, iar în punctul N : **x = 0** şi **y = 175**. Ecuaţia care verifică aceste egalităţi este

$$y = 175 - x$$

deoarece pentru **y = 0** obţinem **x = 175**, iar pentru **x = 0** obţinem **y = 175**. În cazul nostru, abscisa va scădea de la 175 la 0. Programul P44 se completează cu:

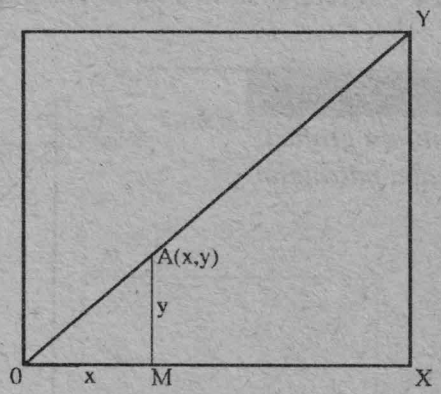
```
120 FOR X = 175 TO 0 STEP - 1
130 PLOT X , 175 - X
140 NEXT X
```

Problema 6. **Să se deseneze prin puncte diagonalele dreptunghiului realizat la problema 4.**

Rezolvare: problema care se pune, de fapt, este aceea de a se identifica funcţia a cărei reprezentare grafică este dreapta pe care se situează diagonala. Dacă luăm un punct A la întîmplare pe diagonală, de coordonate **x** şi **y**, va trebui să îl exprimăm pe **y** în funcţie de **x** (considerînd că vom cicla valoarea variabilei care exprimă abscisa) (vezi figura 24). Se observă că se formează un triunghi OAM *asemenea* cu triunghiul OYX (deoarece AM || YX).

fig.24

Coordonatele unui punct de pe diagonala dreptunghiului şi stabilirea funcţiei care reprezintă această diagonală



În acest caz:

$$\frac{x}{y}=\frac{OX}{OY} \text{ adică } \frac{x}{y}=\frac{255}{175}$$

iar din această relaţie îl putem scoate pe **y** în funcţie de **x**. Deci:

$$y=\frac{175}{255}\cdot x$$

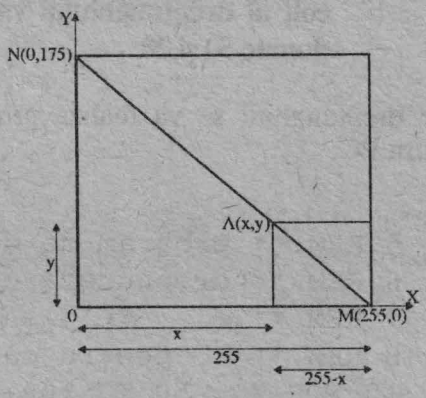
În consecinţă, *toate punctele de pe diagonală vor avea ordonatele egale cu 175/255 din valorile absciselor corespunzătoare.* Continuarea programului P42 de la problema 4 va fi:

```
130 FOR X = 0 TO 255
140 PLOT X , X * 175/255
150 NEXT X
```

Pentru cealaltă diagonală să cercetăm extremităţile ei (vezi figura 25):

fig.25

Stabilirea funcţiei care reprezintă diagonala opusă a dreptunghiului



În punctul M: **x = 255** şi **y = 0**;
În punctul N: **x = 0** şi **y = 175**

Funcţia de gradul I care verifică aceste egalităţi este:

$$y = 175 - \frac{175}{255} \cdot x$$

deoarece din figură (asemănarea triunghiurilor) rezultă că:

$$\frac{255 - x}{y} = \frac{255}{175}$$

de unde:

$$y = 175 - \frac{175}{255} \cdot x$$

În cazul nostru abscisa scade de la 255 la 0, iar programul se obţine completîndu-se P42.A cu liniile:

```
160 FOR X = 255 TO 0 STEP -1
170 PLOT X , 175 - X * 175/255
180 NEXT X
```

☺ ☺ ☺

Problema 7. **Să se deseneze prin puncte un dreptunghi ale cărei laturi sînt 100 şi 70 şi apoi să se traseze diagonalele sale. Un colţ al dreptunghiului va fi situat în punctul de coordonate 50 şi 20.**

Rezolvare: dreptunghiul se va realiza prin programul P45.A, respectiv P45.B pentru PC:

P45.A

```
5 REM ** program de desenare a unui **
6 REM ** dreptunghi cu diagonalele  **
7 REM **           sale             **
8 REM **    pentru calculator HC    **
10 FOR X = 50 TO 100+50
20 PLOT X , 20
30 NEXT X
40 FOR Y = 20 TO 20+70
50 PLOT 150 , Y
60 NEXT Y
70 FOR X = 150 TO 50 STEP -1
80 PLOT X , 90
90 NEXT X
100 FOR Y = 90 TO 20 STEP -1
110 PLOT 50 , Y
120 NEXT Y
```

P45.B

```
5 REM ** program de desenare a unui **
6 REM ** dreptunghi cu diagonalele  **
7 REM **           sale             **
8 REM **    pentru calculator PC    **
9 SCREEN 1
10 CLS
11 FOR X = 50 TO 100+50
20 PSET (X , 20) , 7
30 NEXT X
40 FOR Y = 20 TO 20+70
50 PSET (150 , Y) , 7
```

```
60 NEXT Y
70 FOR X = 150 TO 50 STEP -1
80 PSET (X , 90) , 7
90 NEXT X
100 FOR Y = 90 TO 20 STEP -1
110 PSET (50 , Y) , 7
120 NEXT Y
```

Pentru prima diagonală abscisa va creşte de la 50 la 150, iar ordonatele punctelor de pe ea vor fi proporţionale cu inversul raportului laturilor (70/100). De asemenea va fi necesar să se verifice condiţiile de la extremităţile diagonalei: **x = 50 ; y = 20** şi **x = 150 ; y = 90**.

În acest caz funcţia este :

$$y = 20 + (x - 50) \cdot \frac{70}{100}$$

deoarece din figura 26 rezultă că :

$$\frac{x - 50}{y - 20} = \frac{100}{70}$$

Se adaugă programului P45.A liniile :

```
130 FOR X = 50 TO 150
140 PLOT X , 20 + (X-50) * 70/100
150 NEXT X
```

Pentru porgramul P45.B la linia 140 va fi:

```
140 PSET (X, 20-(X-50)*70/100), 7
```

Pentru cealaltă diagonală, abscisa va scădea de la 150 la 50, iar pentru extremităţi se vor verifica condiţiile: **x = 150 ; y = 20** şi **x = 50 ; y = 90**.

Funcţia va fi :

$$x = 90 - (x - 50) \cdot \frac{70}{100}$$

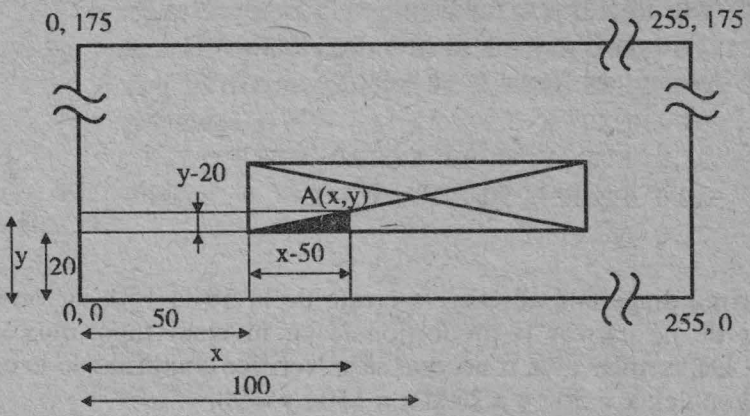


fig.26

Stabilirea funcţiei care reprezintă diagonala unui dreptunghi de coordonate şi poziţie date

Programul în continuare a lui P45.A este :

```
160 FOR X = 150 TO 50 STEP -1
170 PLOT X , 90 - (x-50) * 70/100
180 NEXT X
```

Pentru programul P45.B în linia 170 se va folosi `PSET` în loc de `PLOT`.

☺ ☺ ☺

Reprezentări grafice de funcţii

Deseori avem nevoie să reprezentăm grafic diverse funcţii. Cazul cel mai frecvent este cel în care avem mai multe puncte în sistemul de reprezentare prin coordonate, cunoscînd, deci, cîte două valori (corespunzătoare coordonatei pe orizontală şi, respectiv, coordonatei pe verticală) şi dorim să reprezentăm grafic aceste puncte pentru a putea observa tendinţa unui fenomen.

De exemplu, să presupunem că avem *10 puncte care reprezintă 10 înregistrări (informaţii) privind limita superioară a performanţelor calculatoarelor observate între anii 1971-1980.* Prin limita superioară a perfor-

manţelor înţelegem *viteza maximă atinsă de cele mai puternice calculatoare (supercalculatoarele) existente în anul respectiv, viteză măsurată în milioane de instrucţiuni pe secundă (MIPS).*

De exemplu în anul 1971 cel mai puternic calculator existent în lume era STARAN care prezenta o viteză de 70 MIPS. În anul 1972 a fost lansat calculatorul ILIAC IV care îmbunătăţea performanţele precedentului cu 5 MIPS. În anul 1973 nu a fost lansat nici un alt calculator care să prezinte performanţe superioare faţă de ILIAC IV. În anul 1976 firma CRAY a lansat supercalculatorul CRAY 1 care a fost primul calculator care a atins pragul celor 100 MIPS. În anul 1979 firma CDC a lansat calculatorul CYBER 203 care atingea performanţe de pînă la 400 MIPS iar, un an mai tîrziu, CYBER 205 a cărui viteză atingea 700 MIPS.

Performanţele descrise şi anii corespunzători sînt oglindite în tabelul de mai jos:

Anii (X)	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980
Viteza – MIPS (Y)	70	75	75	83	83	100	100	200	400	700

Dorim să reprezentăm grafic aceste puncte (pe orizontală vom avea anii, iar pe verticală, vitezele) astfel încît să stabilim ce formă (evoluţie) are curba care descrie creşterea vitezei (performanţei) calculatoarelor în timp.

Să se realizeze un program cu care să se poată reprezenta grafic punctele din tabelul reprezentînd creşterea performanţelor calculatoarelor şi care exprimă valorile Y în funcţie de X.

Vom avea nevoie să introducem numărul N de puncte (înregistrări) ale căror valori le cunoaştem şi să rezervăm un spaţiu de memorie suficient pentru abscise – `DIM X(N)` – şi ordonate – `DIM Y(N)`. Apoi va trebui să introducem valorile perechilor X,Y şi să le afişăm pe un rînd împreună cu numărul de ordine al înregistrării respective. Introducerea şi afişarea (care se face pe măsură ce introducem valorile coordonatelor în vederea verificării corectitudinii introducerii datelor) se va realiza prin intermediul

unui ciclu FOR-NEXT de N paşi. Pentru a putea reprezenta grafic punctele va trebui să realizăm o transformare a valorilor reale ale coordonatelor în valori proporţionale cu coordonatele ecranului (*factor de scară*). În acest scop va trebui să cunoaştem care sînt valorile minime şi maxime ale coordonatelor pe orizontală şi respectiv pe verticală (XMIN, XMAX, YMIN, YMAX). Acest lucru îl putem realiza printr-un ciclu de N paşi cercetînd valorile pentru fiecare punct. Regula (algoritmul) pentru găsirea maximului şi respectiv minimului o cunoaştem deja.

Ştiind acum valorile minimă şi maximă, putem să "punem" pe ecran toate punctele folosind o instrucţiune PLOT, valorile coordonatelor pe ecran fiind:

✘ coordonate pe orizontală:

HC	PC
$\frac{255}{XMAX - XMIN} \cdot (X(I) - XMIN)$	$\frac{319}{XMAX - XMIN} \cdot (X(I) - XMIN)$

✘ coordonate pe verticală:

HC	PC
$\frac{175}{YMAX - YMIN \cdot (Y(I) - YMIN)}$	$200 - \frac{199}{YMAX - YMIN \cdot (Y(I) - YMIN)}$

Programul P46.A este:

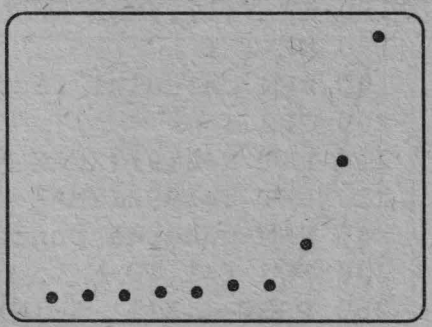
P46.A
```
5 REM **      program pentru      **
6 REM ** reprezentarea grafica    **
7 REM **       a unei functii     **
8 REM ** pentru calculatorul HC   **
10 INPUT N
20 DIM X(N): DIM Y(N)
25 REM introducere valori puncte
30 FOR I=1 TO N
40 INPUT "X=";X(I)
50 INPUT "Y=";Y(I)
60 PRINT I;" ";X(I);" ";Y(I)
70 NEXT I
```

```
95 REM calculul valorilor maxime si
96 REM minime pentru coordonate
100 LET XMIN=X(1)
110 LET XMAX=X(1)
120 LET YMIN=Y(1)
130 LET YMAX=Y(1)
140 FOR I=2 TO N
150 IF X(I)<XMIN THEN LET XMIN=X(I)
160 IF X(I)>XMAX THEN LET XMAX=X(I)
170 IF Y(I)<YMIN THEN LET YMIN=Y(I)
180 IF Y(I)>YMAX THEN LET YMAX=Y(I)
190 NEXT I
195 REM calculul factorului de scara
200 CLS
210 LET X=255/(XMAX-XMIN)
220 LET Y=175/(YMAX-YMIN)
245 REM punerea punctelor pe ecran
250 FOR I=1 TO N
260 PLOT (X(I)-XMIN)*X,(Y(I)-YMIN)*Y
270 NEXT I
```

Vom obţine rezultatul din figura 27 ceea ce ne sugerează o creştere exponenţială a performanţelor calculatoarelor în timp.

fig.27

Reprezentarea grafică prin puncte a creşterii performanţelor calculatoarelor electronice în timp



Pentru calculatoare PC programul similar este P46.B:

P46.B

```
5 REM **      program pentru      **
6 REM ** reprezentarea grafica    **
7 REM **      a unei functii      **
8 REM ** pentru calculatorul PC **
9 CLS
10 SCREEN 1 , 0
15 INPUT N
20  X(N):  Y(N)
25 REM introducere valori puncte
30 FOR I=1 TO N
40 INPUT "X=";X(I)
50 INPUT "Y=";Y(I)
60 PRINT I;" ";X(I);" ";Y(I)
70 NEXT I
95 REM calculul valorilor maxime si
96 REM minime pentru coordonate
100 LET XMIN=X(1)
110 LET XMAX=X(1)
120 LET YMIN=Y(1)
130 LET YMAX=Y(1)
140 FOR I=2 TO N
150 IF X(I)<XMIN THEN LET XMIN=X(I)
160 IF X(I)>XMAX THEN LET XMAX=X(I)
170 IF Y(I)<YMIN THEN LET YMIN=Y(I)
180 IF Y(I)>YMAX THEN LET YMAX=Y(I)
190 NEXT I
195 REM calculul factorului de scara
200 CLS
210 LET X=319/(XMAX-XMIN)
220 LET Y=199/(YMAX-YMIN)
245 REM punerea punctelor pe ecran
250 FOR I=1 TO N
260 PSET ((X(I)-XMIN)*X,
    199-(Y(I)-YMIN)*Y) , 7
270 NEXT I
```

Cum trebuie să modificăm programul astfel încît să trasăm şi liniile dintre puncte?

Desigur vom folosi o instrucţiune DRAW. Începînd cu al doilea punct vom trasa mai întîi dreapta care uneşte punctul precedent (X(I-1), Y(I-1)) cu cel actual (X(I), Y(I)), iar pentru aceasta avem nevoie să calculăm creşterile (sau scăderile) pe orizontală şi, respectiv, pe verticală. Acestea vor fi:

$$dX = X \cdot (X(I) - XMIN) - X \cdot (X(I-1) - XMIN) =$$
$$= X \cdot (X(I) - XMIN - X(I-1) + XMIN) =$$
$$= X \cdot (X(I) - X(I-1))$$

$$dY = Y \cdot (Y(I) - YMIN) - Y \cdot (Y(I-1) - YMIN) =$$
$$= Y \cdot (Y(I) - Y(I-1))$$

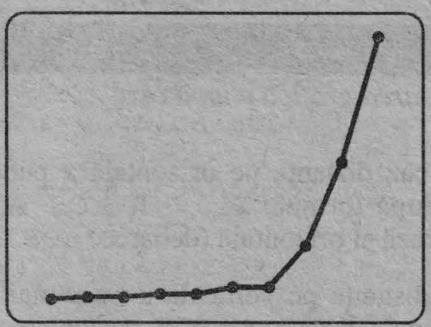
Vom insera în programul P46 următoarea linie în program:

```
255 IF I=2 THEN DRAW X*(X(I)-X(I-1)),
    Y*(Y(I)-Y(I-1))
```

În acest caz nu vom mai avea nevoie şi de linia 260 pe care putem să o ştergem. Vom obţine rezultatul din figura 28, care ne întăreşte convingerea unei creşteri exponenţiale a vitezei calculatoarelor.

fig.28

Reprezentarea grafică a creşterii performanţelor calculatoarelor electronice în timp



Să se traseze pe ecran un cerc prin puncte. De exemplu să se marcheze orele pe cadranul unui ceas.

Să presupunem că vrem să trasăm un punct P pe cercul de rază **R** şi, de asemenea, că centrul cercului este în origine (vezi figura 29).

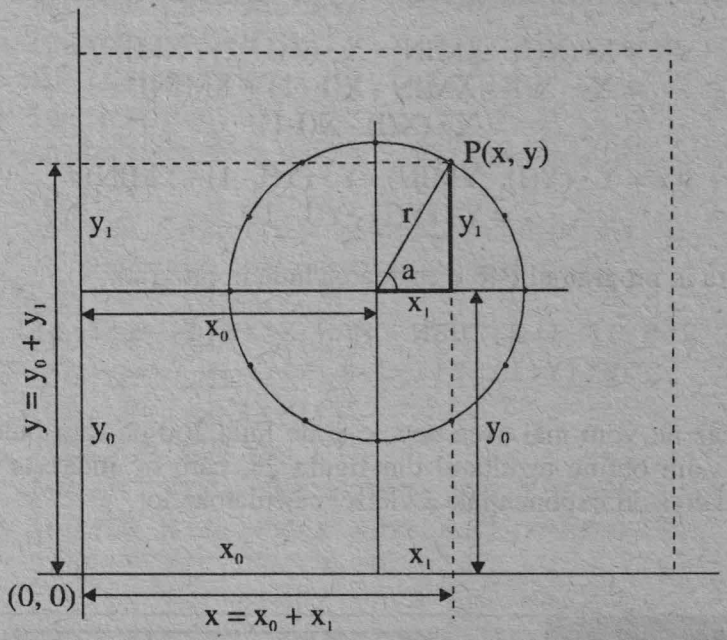


fig.29

Reprezentarea grafică a unui cerc

În acest caz distanţa pe orizontală a punctului P, şi anume X1, se calculează după formula `X1 = R COS A`, unde A este unghiul la centru făcut de rază şi orizontală (deoarece `COS A = X1/R`).

Similar, distanţa pe verticală a punctului P, şi anume Y1, se calculează după formula `Y1 = R SIN A`, deoarece `SIN A = Y1/R`. Astfel, în program, punctul P se va putea trasa prin:

```
PLOT R * COS A , R * SIN A
```

Dar în cazul real cercul nu este în origine. Originea se găseşte pentru ecranul grafic de înaltă rezoluţie în colţul din stînga jos al ecranului. Astfel, coordonata pe orizontală a punctului P, şi anume X, va fi X = X0 + X1, unde X0 este distanţa pe orizontală de la origine pînă la centrul cercului. Similar, coordonata pe verticală a punctului P va fi Y = Y0 + Y1. Acum vom putea trasa punctul P prin:

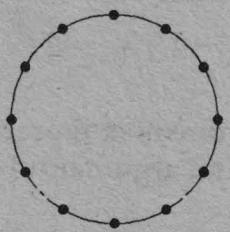
```
PLOT X0 + R * COS A , Y0 + R * SIN A
```

unde X0 şi Y0 reprezintă coordonatele pe orizontală şi pe verticală a centrului cercului.

Pentru a marca orele pe cadranul unui ceas va trebui să trasăm 12 puncte (vezi figura 30). Practic unghiul la centru va creşte de la 0 la 2π cu un pas de $\frac{2\pi}{12} = \frac{\pi}{6}$:

fig.30

Cadran de ceas



Programul P47 este:

P47.A

```
5 REM ** Program de trasare a unui **
6 REM **     cerc prin puncte       **
7 REM **   pentru calculator HC     **
10 INPUT "RAZA=";R
20 INPUT "COORDONATELE CENTRULUI
   CERCULUI ";X0 , Y0
30 FOR A=0 TO 2*PI STEP PI/6
40 PLOT X0 + R*COS A , Y0 + R*SIN A
50 NEXT A
```

Pentru calculatoare PC programul similar este P47.B:

P47.B

```
5 REM ** Program de trasare a unui **
6 REM **     cerc prin puncte       **
7 REM **   pentru calculator PC     **
8 CLS
9 SCREEN 1
10 LET PI=3.14
15 INPUT "RAZA=";R
20 INPUT "COORDONATELE CENTRULUI
   CERCULUI ";X0 , Y0
30 FOR A=0 TO 2*PI STEP PI/6
40 PSET (X0+R*COS(A), Y0+R*SIN(A)), 7
50 NEXT A
```

Reprezentarea grafică a unei funcţii de o variabilă

Ne propunem să reprezentăm grafic evoluţia unei funcţii atunci cînd argumentul său variază într-un interval.

În acest caz vom utiliza pentru reprezentare o funcţie **DEF FN** urmînd ca, pentru fiecare caz în parte, să introducem în program forma funcţiei respective. Va trebui să introducem apoi valorile domeniului de definiţie al funcţiei (XMIN şi XMAX), numărul de puncte N prin care se reprezintă funcţia şi să rezervăm un spaţiu de memorie Y(N) pentru toate valorile corespunzătoare ale funcţiei. Vom calcula şi valoarea pasului pe orizontală XP prin formula:

$$XP = \frac{XMAX - XMIN}{N - 1}$$

deoarece pentru N puncte se obţin N - 1 intervale.

Apoi vom calcula valorile corespunzătoare pentru funcţia Y mărind argumentul cu valoarea pasului. Vom calcula după metoda cunoscută valoarea minimă YMIN şi valoarea maximă YMAX a funcţiei Y, apoi printr-un

ciclu de N paşi vom trasa valorile funcţiei prin puncte, pentru valorile argumentului care cresc cu valoarea pasului.

Valorile funcţiei se vor calcula ţinînd cont de factorul de scară orizontal şi vertical.

Programul P48 pentru trasarea graficului funcţiei sinus va fi:

P48.A

```
5 REM **   Program de trasare a     **
6 REM ** graficului functiei SINUS **
7 REM **  pentru calculatorul HC    **
10 DEF FN Y(X)=SIN X
20 INPUT "domeniul de definitie ";
   XMIN, XMAX
30 INPUT " N = "; N
40  Y(N)
50 LET X = XMIN
60 LET XP = (XMAX - XMIN)/(N - 1)
70 FOR I = 1 TO N
80 LET Y(I) = FN Y(X)
90 LET X = X + XP
100 NEXT I
110 LET YMIN = Y(1)
120 LET YMAX = Y(1)
130 FOR I = 2 TO N
140 IF Y(I) < YMIN THEN LET YMIN = Y(I)
150 IF Y(I) > YMAX THEN LET YMAX = Y(I)
160 NEXT I
170 CLS
180 LET X = XMIN
190 FOR I = 1 TO N
200 PLOT (X-XMIN)*255/(XMAX-XMIN),
    (Y(I)-YMIN)*175/(YMAX-YMIN)
210 LET X = X + XP
220 NEXT I
```

Pentru calculatoare PC programul similar va fi P48.B

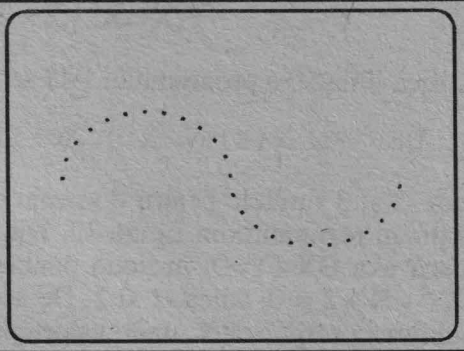
P48.B

```
5 REM **   Program de trasare a     **
6 REM ** graficului functiei SINUS **
7 REM **  pentru calculatorul PC    **
8 SCREEN 1 , 0
9 CLS
10 DEF FN Y(X)=SIN (X)
20 INPUT "domeniul de definitie ";
   XMIN, XMAX
30 INPUT " N = "; N
40  Y(N)
50 LET X = XMIN
60 LET XP = (XMAX - XMIN)/(N - 1)
70 FOR I = 1 TO N
80 LET Y(I) = FN Y(X)
90 LET X = X + XP
100 NEXT I
110 LET YMIN = Y(1)
120 LET YMAX = Y(1)
130 FOR I = 2 TO N
140 IF Y(I) < YMIN THEN LET YMIN = Y(I)
150 IF Y(I) > YMAX THEN LET YMAX = Y(I)
160 NEXT I
170 CLS
180 LET X = XMIN
190 FOR I = 1 TO N
200 PSET ((X-XMIN)*319/(XMAX-XMIN),
    200-(Y(I)-YMIN)*199/(YMAX-YMIN)),7
210 LET X = X + XP
220 NEXT I
```

Introducînd pentru domeniul de definiţie valorile 0 şi 6,28 (adică 2π), pentru 100 de puncte obţinem rezultatul din figura 31.

fig.31

Sinusoidă



Să se completeze programul astfel încît să se reprezinte şi axele de coordonate.

Vom calcula X0 şi Y0 astfel încît să "punem" punctele corespunzătoare dacă coordonatele lor "intră" în cele ale ecranului grafic. Vom folosi **OVER 1** pentru supratipărire, pentru HC. Adăugăm programului P48.A liniile:

```
230 LET X0=-XMIN*255/(XMAX-XMIN)
240 LET Y0=-YMIN*175/(YMAX-YMIN)
250 OVER 1
260 IF X0>=0 AND X0<=255 THEN
    PLOT X0,0 : DRAW 0,175
270 IF Y0>=0 AND Y0<=175 THEN
    PLOT 0,Y0 : DRAW 255,0
280 OVER 1
```

Pentru PC completarea similară este:

```
230 LET X0=-XMIN*320/(XMAX-XMIN)
240 LET Y0=-YMIN*199/(YMAX-YMIN)
260 IF X0>=0 AND X0<=319 THEN
    LINE (X0,0)-(X0,199)
270 IF Y0>=0 AND Y0<=199 THEN
    LINE (0,Y0)-(319,Y0)
```

Ca exemplu de aplicaţie să analizăm funcţia

$$Y(X)=X^2 - X - 2$$

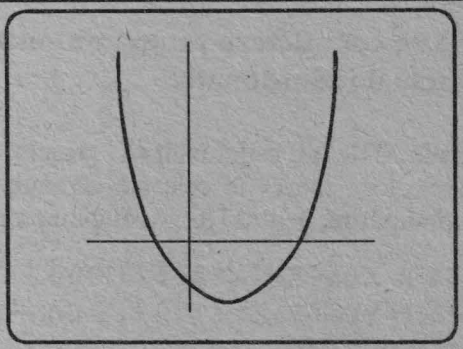
Vom modifica linia 10 a programului P48 astfel:

```
DEF FN Y(X) = X * X - X - 2
```

Introducem -2 şi 3 limitele pentru domeniu de definiţie iar pentru 100 de puncte obţinem rezultatul din figura 32. Din figură se observă cum curba intersectează axa OX (Y=0) în două puncte corespunzătoare rădăcinilor ecuaţiei $X^2 - X - 2 = 0$, adică -1 şi 2. De asemenea se observă că funcţia are un minim corespunzător unei valori pozitive mai mici decît 1 (şi anume valoarea 0,5) iar curba intersectează axa OY într-un punct corespunzător unui Y negativ (cînd X=0 şi Y=-2).

fig.32

Reprezentarea grafică a unei funcţii de gradul II date



Limite

Calculatorul poate fi un instrument foarte eficient în calculul şi studiul unor *limite de şiruri*. Afişarea rezultatelor expresiilor în cadrul unor repetiţii ne conduce de obicei la determinarea corectă a limitei.

Să se calculeze cu ajutorul calculatorului:

$$\lim_{n \to \infty} \frac{n}{n^2 + 1}$$

Vom studia limita acestui şir prin intermediul următorului program (P49):

P49.

```
10 FOR I=1 TO 100
20 PRINT I , I/(I*I+1)
30 NEXT I
```

Vom nota valorile obţinute:

0,5	pentru I = 1
0,1	pentru I = 10
0,04	pentru I = 2
0,0099	pentru I = 100

Dacă încă nu este evident că limita este 0 vom putea continua experimentarea cu valori mai mari ale contorului. De exemplu cu I să ia valori de la 1000 la 1100.

Obţinem 0,00099 pentru I = 1000 şi 0,00090 pentru I = 1100.

Să se calculeze cu ajutorul calculatorului:

$$\lim_{n \to \infty} \frac{3n+2}{2n+2}$$

Vom folosi programul P49 pentru diverse valori ale contorului şi cu linia 20 modificată astfel:

```
20 PRINT I , (3*I+2)/(2*I+1)
```

Vom obţine :

1,6666	pentru I = 1
1,52	pentru I = 10
1,512	pentru I = 20
1,5049505	pentru I = 50
1,5024876	pentru I = 100
1,5002499	pentru I = 1000

Şirul va converge către valoarea 1,5.

Să se studieze cu ajutorul calculatorului:

$$\lim_{n \to 0} (1+n)^{\frac{1}{n}}$$

Vom folosi programul P50:

P50.
```
10 FOR I=20 TO 1 STEP -1
20 PRINT I , (1 + I)^(1/I)
30 NEXT I
```

Vom obţine :

1,1644235	pentru I = 20
1,2709816	pentru I = 10
1,5874011	pentru I = 3
2	pentru I = 1

Deoarece valorile nu sînt încă concludente vom studia limita în continuare folosind un pas mai mic. Linia 10 a programului P50 va deveni:

```
10 FOR I = 1 TO 0.1 STEP - 0.1
```

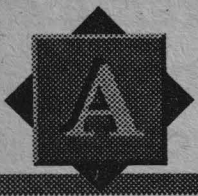
Vom obţine :

2,0404513	pentru I = 0,9
2,25	pentru I = 0,5
2,5	pentru I = 0,1

Restrîngînd din nou intervalul pentru contor, vom lua acum pasul 0.01 (`I = 0.100 TO 0.001 STEP -0.01`) şi obţinem rezultatele:

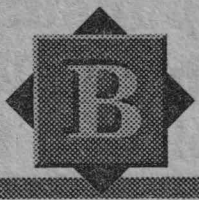
2,6052534	pentru I = 0,09
2,65	pentru I = 0,05
2,7048138	pentru I = 0,01

Devine evident că limita este **e**.



Răspunsuri la problemele din capitolul 6

1. F
2. A
3. A
4. F
5. A
6. F
7. A
8. F
9. F
10. A
11. F
12. A
13. A
14. A
15. A
16. A
17. A
18. F
19. A
20. F
21. A
22. A
23. A
24. A
25. F
26. A
27. A
28. F
29. A
30. F
31. F
32. A
33. F
34. A
35. A
36. F
37. A
38. F
39. A
40. A
41. A
42. F
43. F
44. A
45. A



Lista programelor prezentate în carte. Mic caiet de programe

» *Două programe pentru HC*

CREION MAGIC

Cu ajutorul tastelor 5, 6, 7 si 8 puteţi dirija o linie pe ecran şi să desenaţi figurile pe care le doriţi.

În timpul execuţiei, linia se alungeşte înspre ultima direcţie indicată pîna cînd veţi acţiona altă tastă.

```
 5 LET a$="": LET a=127: LET b=87
10 IF INKEY$"" THEN LET a$=INKEY$
20 LET b=b+(a$="7")-(a$="6")
30 LET a=a+(a$="8")-(a$="5")
40 IF a=256 OR a=0 THEN LET a=ABS (a-255)
50 IF b=176 OR b=0 THEN LET b=ABS (b-175)
60 PLOT a,b: GO TO 10
```

GRAFICĂ TRĂZNITĂ

Programul trasează linii de lungime întîmplătoare, de culori întîmplătoare, pornind din cele 4 colţuri de ecran.

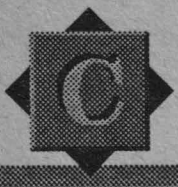
```
10 PLOT 0,0: DRAW INK RND*6;RND*255,RND*175
20 PLOT 255,175:DRAW INK RND*6;-RND*255,-RND*175
30 PLOT 255,0: DRAW INK RND*6;-RND*255,RND*175
40 PLOT 0,175: DRAW INK RND*6;RND*255,-RND*175
50 GO TO 10
```

```
10 REM  SIRENA DE AMBULANTA
20 FOR I=400 TO 800 STEP 4
30 SOUND I,0.3
40 NEXT I
50 FOR I=800 TO 400 STEP -4
60 SOUND I,0.3
70 NEXT I
80 GO TO 20
```

```
10 REM  SEMNAL
20 SOUND 5000 , 1
30 SOUND 10000 , 1
40 GO TO 20
```

```
10 REM  SIRENA POLITIE
20 SOUND 523 , 5
30 SOUND 783 , 5
40 GO TO 20
```

```
10 REM  LANSARE RACHETA
20 FOR I = 100 TO 3000 STEP 2
30 SOUND I , 100/I
40 NEXT I
```



Lista figurilor cuprinse în carte

Glosarul cuvintelor cheie

Prin **cuvinte cheie** se înţeleg cuvintele existente în memoria calculatorului şi care prezintă o anumită semnificaţie pentru acesta. Lucrarea de faţă face referire la cuvintele cheie ale limbajului **BASIC** utilizate pentru doua categorii de calculatoare: cele **compatibile ZX Spectrum** (HC, CIP, JET, COBRA, TIM, Sinclair) care utilizează versiunea Sinclair a limbajului **BASIC** şi cele **compatibile IBM PC** care utilizează versiunea **GW BASIC.**

Dacă pentru parcurgerea şi experimentarea lucrării în cazul calculatoarelor compatibile ZX Spectrum este necesară doar pornirea calculatorului, BASIC-ul fiind rezident în memoria acestor calculatoare, în cazul calculatoarelor compatibile IBM PC este necesară în prealabil încărcarea fişierului GW BASIC.

Atragem atenţia asupra modului diferit de introducere a cuvintelor cheie pentru cele două tipuri de calculatoare: *pentru cele compatibile ZX Spectrum orice cuvînt cheie se introduce prin acţionarea unei anumite taste sau a unei combinaţii de taste, iar pentru cele compatibile IBM PC prin tastarea cuvîntului sau combinaţiei de cuvinte, caracter cu caracter* (acesta este, de fapt, şi modul standard de lucru).

Orice cuvînt cheie poate face parte din următoarele categorii:

- **comandă** - cuvînt cheie al cărui introducere are ca urmare execuţia imediată de către calculator a unei acţiuni. Exemplu: `RUN`, `LIST`, `LOAD`, `SAVE`, etc.
- **instrucţiune** - cuvînt cheie care are ca urmare o acţiune executată de calculator şi care poate fi utilizat într-o linie de program. Acţiunea este îndeplinită numai atunci cînd programul va fi executat. Exemplu: `INPUT`, `PRINT`, etc.

Un cuvînt cheie care reprezintă o comandă va funcţiona ca o instrucţiune atunci cînd este folosit într-o linie de program.

De exemplu: `1000 LIST`, cuvîntul cheie `LIST` reprezintă o instrucţiune, acţiunea de listare a programului realizîndu-se cînd execuţia programului va ajunge la linia 1000.

Invers, o instrucţiune poate fi considerată comandă cînd va fi folosită ca atare. De exemplu, `PRINT 3+4`, cuvîntul cheie `PRINT` reprezintă o comandă, deoarece acţiunea de afişare a rezultatelor se realizează imediat după introducere.

În indexul cuvintelor cheie se va indica cu prioritate categoria cea mai reprezentativă în care se poate încadra cuvîntul cheie respectiv. De exemplu, `RUN` se va trece în categoria comandă/instrucţiune, deoarece, deşi se poate utiliza şi ca instrucţiune, se utilizează în primul rînd ca şi comandă. Invers, `INPUT` se va trece la categoria instrucţiune/comandă, deoarece, deşi se poate folosi şi în mod comandă, se utilizează în primul rînd în linii de program, deci ca instrucţiune.

- **funcţie** - cuvînt cheie a cărui folosire produce o valoare. Acest cuvînt se utilizează în combinaţie cu o comandă sau instrucţiune. Exemple: `INT`, `RND`, `ABS`, `SIN`, `SQR`. Exemple de utilizare: `PRINT INT 8.1`.

Atragem atenţia asupra unei deosebiri esenţiale în utilizarea cuvintelor cheie funcţie pentru cele două tipuri de versiuni BASIC aflate în discuţie: în cazul versiunii GW BASIC este obligatorie folosirea parantezelor pentru încadrarea argumentului unei funcţii, în timp ce, în cazul versiunii Sinclair BASIC folosirea parantezelor este opţională. Nefolosirea parantezelor în acest caz, deşi simplifică utilizarea, poate conduce la alte rezultate decît cele aşteptate.

De exemplu cu `PRINT INT 8.1+2.2` se va obţine rezultatul 10.2, deoarece se va evalua `INT 8.1` obţinîndu-se rezultatul parţial 8 iar acesta se va aduna cu 2.2 obţinîndu-se rezultatul final 10.2. Dacă se doreşte întregul din rezultatul lui 8.1+2.2 atunci se va introduce `PRINT INT (8.1+2.2)`.

- **operator logic** - cuvînt cheie care este utilizat într-o comandă sau instrucţiune şi care determină dacă o condiţie este adevărată sau falsă.
  Exemple de operatori logici sînt: `AND`, `OR`, `NOT`.

✗ **alte cuvinte cheie** - cuvinte cheie care sînt utilizate, în cadrul unor comenzi sau instrucţiuni şi care reprezintă particule de propoziţii care fac o legătură între cuvinte. Exemple: `THEN`, `AT`, `STEP` etc.

În scopul cunoaşterii şi folosirii cuvintelor cheie, adică a realizării de programe, vor trebui cunoscute :

- ❑ localizarea cuvintelor cheie pe tastatură (pe ce tastă se găseşte şi în ce mod se obţine), fapt important pentru calculatoarele compatibile Spectrum, pentru cele compatibile IBM PC introducerea cuvintelor cheie făcîndu-se, aşa cum am văzut, prin introducerea caracter cu caracter;
- ❑ ce fel de cuvînt cheie este (categoria);
- ❑ scopul utilizării cuvîntului cheie (în ce caz se foloseşte);
- ❑ formatul (sintaxa) cuvîntului cheie.

Deoarece în lucrare sînt oferite numeroase exemple de folosire a cuvintelor cheie în program, deseori fiind indicată şi modalitatea de obţinere s-a considerat util să se facă doar o trecere în revistă (memento) în ordine alfabetică a tuturor cuvintelor cheie folosite, indicîndu-se doar semnificaţia (din care rezultă şi scopul principal de folosire), categoria în care se încadrează cuvîntul, pentru ce tip de calculator se foloseşte (dacă nu se indică, se subînţelege că este utilizabil pentru amîndouă tipurile), precum şi paginile din lucrare în care puteţi regăsi cuvîntul cheie respectiv într-un anumit program. Indexul nu conţine toate cuvintele cheie din limbajul BASIC, ci doar majoritatea lor. Pentru restul de cuvinte, precum şi indicaţii referitoare la formatul complet (sintaxa) în care pot fi folosite, mai ales că acesta poate fi diferit pentru cele două tipuri de calculatoare avute în vedere (şi de asemenea poate conduce la rezultate neidentice), se pot consulta manuale de referinţă privitoare la limbajele Sinclair BASIC şi GW BASIC sau lucrări citate în bibliografie.

ABS. *Funcţie*. Produce valoarea absolută a unei valori numerice, adică valoarea fără semn pozitiv sau negativ.

AND. *Operator logic*. Leagă două sau mai multe condiţii în cadrul unei instrucţiuni. Dacă toate condiţiile legate prin AND sînt adevărate, atunci combinaţia va fi adevărată.

AT. *Particulă de legătură.* Se foloseşte în instrucţiunile PRINT, LPRINT şi INPUT pentru a localiza o afişare pe ecran sau o imprimare pe hîrtie. Se utilizează în Sinclair BASIC.

BEEP. *Instrucţiune/comandă.* Produce o notă muzicală de o anumită durată şi înălţime în Sinclair BASIC. În GW BASIC produce un sunet.

BIN. *Funcţie.* Transformă un număr binar într-unul zecimal. Se utilizează în Sinclair BASIC.

CLS. *Instrucţiune/comandă.* Şterge de pe ecran orice text sau grafică, lăsînd întregul ecran cu culoarea fondului.

CONT(INUE). *Comandă.* Reia execuţia unui program după o comandă BREAK. In Sinclair BASIC forma este CONTINUE.

COS. *Funcţie.* Introduce cosinusul unui unghi.

DATA. *Instrucţiune.* Specifică o listă de date. Se foloseşte în program în conjuncţie cu instrucţiunea READ.

DEF FN. *Funcţie.* Defineşte o funcţie.

DIM. *Instrucţiune.* Rezervă un spaţiu de memorie.

DRAW. *Instrucţiune/comandă.* Desenează pe ecran linii drepte sau curbe în Sinclair BASIC. Desenează un obiect definit printr-o secvenţă de subcomenzi grafice în GW BASIC.

FOR. *Instrucţiune/comandă.* Execută o serie de instrucţiuni de un număr specificat de ori (ciclu).

GO SUB. *Instrucţiune/comandă.* Realizează saltul la subrutina care începe la linia specificată.

GOTO. *Instrucţiune/comandă.* Execută salt necondiţionat la instrucţiunea de la linia specificată.

IF. *Instrucţiune/comandă.* Determină o execuţie condiţionată de valoarea logică a unei expresii.

INKEY$. *Funcţie.* Detectează ultimul caracter introdus de la tastatură. (citeşte tastatura).

INPUT. *Instrucţiune/comandă*. Se introduc datele de la tastatură în timpul execuţiei programului.

INT. *Funcţie*. Returnează cel mai mare întreg care este mai mic sau egal cu valoarea care reprezintă argumentul.

LEN. *Funcţie*. Returnează lungimea şirului de caractere specificat.

LET. *Instrucţiune/comandă*. Atribuie unei variabile valoarea expresiei specificate.

LINE. *Instrucţiune/comandă*. Desenează pe ecran o linie sau un dreptunghi. În GW BASIC. În Sinclair BASIC se utilizează împreună cu comanda SAVE pentru salvarea unui program cu lansare automată de la numărul de linii indicat.

LIST. *Comandă/instrucţiune*. Afişează lista instrucţiunilor programului din memoria calculatorului.

LOAD. *Comandă/instrucţiune*. Încarcă un program de pe un suport extern în memoria calculatorului.

LOCATE. *Instrucţiune/comandă*. Poziţionează cursorul în poziţia specificată. Se foloseşte în GW BASIC.

MID$. *Funcţie/instrucţiune*. Ca funcţie returnează un subşir al şirului indicat, iar ca instrucţiune, înlocuieşte o parte a unui şir cu un alt şir. Se foloseşte în GW BASIC.

NEW. *Comandă/instrucţiune*. Şterge din memorie programul curent şi toate variabilele.

OR. *Operator logic*. Leagă două sau mai multe condiţii în cadrul unei instrucţiuni. Dacă cel puţin o condiţie este adevărată, atunci combinaţia va fi adevărată.

OVER. *Instrucţiune/comandă*. Afişează un caracter pe deasupra altuia prin suprapunere.

PAUSE. *Instrucţiune/comandă*. Suspendă execuţia unui program pe o perioadă de timp. Se foloseşte în Sinclair BASIC.

PEEK. *Funcţie*. Returnează conţinutul octetului aflat la adresa de memorie specificată.

PI. *Funcţie*. Introduce valoarea lui PI (π = 3,141592). Se foloseşte în Sinclair BASIC.

PLOT. *Instrucţiune/comandă*. Desenează un punct într-o anumită poziţie pe ecran. Se foloseşte în Sinclair BASIC.

POKE. *Instrucţiune/comandă*. Introduce valoarea unui octet în locaţia de memorie specificată.

PRINT. *Instrucţiune/comandă*. Afişează pe ecran valoarea expresiilor indicate.

PSET. *Instrucţiune/comandă*. Desenează un punct pe ecran la coordonatele specificate. Se foloseşte în GW BASIC.

PRESET. *Instrucţiune/comandă*. Desenează/şterge un punct pe ecran la coordonatele specificate. Se foloseşte în GW BASIC.

READ. *Instrucţiune/comandă*. Citeşte valorile din instrucţiunea DATA şi le atribuie variabilelor din lista de variabile.

REM. *Instrucţiune*. Lansează comentarii într-un program.

RETURN. *Instrucţiune/comandă*. Termină execuţia unei subrutine şi revine în programul apelant.

RND. *Funcţie*. Generează un număr aleator mai mare sau egal cu 0 şi mai mic ca 1.

RUN. *Comandă/instrucţiune*. Începe execuţia unui program.

SAVE. *Comandă/instrucţiune*. Memorează (salvează) fişierele (programele sau datele) pe un suport extern.

SCREEN. *Instrucţiune/comandă*. Serveşte la declararea modului de utilizare a ecranului (text, grafică, culori). Se foloseşte în GW BASIC. Tot în GW BASIC se mai foloseşte la returnarea codului ASCII corespunzător caracterului din rîndul şi coloana specificate. Forma SCREEN$ se utilizează cu LOAD sau SAVE în Sinclair BASIC pentru a încărca/salva o imagine ecran.

SIN. *Funcţie*. Returnează sinusul unghiului specificat.

SQR. *Funcţie*. Returnează valoarea radicalului expresiei indicate.

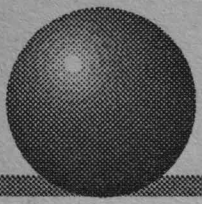
STEP. *Particulă*. Se utilizează în cadrul unei linii de program FOR pentru a evidenţia pasul cu care se modifică valoarea variabilei contor.

STOP. *Instrucţiune/comandă*. Termină execuţia unui program într-un anumit punct.

THEN. *Particulă*. Se utilizează într-o instrucţiune IF. Dacă este adevărată condiţia atunci instrucţiunea (instrucţiunile) care urmează după THEN se vor executa, iar dacă nu este adevărată se vor executa instrucţiunile începînd cu linia următoare de program.

TO. *Particulă*. Se utilizează într-o instrucţiune FOR pentru a determina valorile indicelui din ciclul FOR-NEXT.

TRON. *Comandă*. Determină listarea numerelor liniilor executate. Se foloseşte în GW BASIC.



Indexul cuvintelor cheie

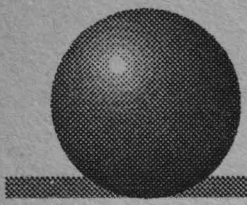
L

O

P

R

S



Bibliografie

Sinclair ZX Spectrum User's Guide, Dorling Kindersley Ltd. and Sinclair Research Ltd.

Informatica pentru elevi, Microinformatica S.R.L., Cluj, 1991

Calculatoare personale, de la hardware la software, Electronica NIS S.R.L., Cluj 1991

Gazeta de informatică nr. 1 - 3, Editura Petrion, Bucureşti, 1991; **nr. 4 - 10**, Editura Libris, Cluj, 1992

Culegere de probleme pentru concursuri, *Bălan D.M., Bălan G.*, Editura Licurici, Suceava, 1992

Limbaj BASIC, Editura Licurici, Suceava, 1992

Are You Computer Literate?, *Billings K., Moursound D.*, Dilithium Press, Portland, Oregon, 1979

Partenerul meu de joc, calculatorul, *Diamandi I.*, RECOOP, Bucureşti, 1989, 1990

Cum să realizăm jocuri pe calculator, *Diamandi I.*, Editura AGNI, Bucureşti, 1992

La conduite du ZX Spectrum, *Hartnell T., Jones D.*, Edition Eyrolles, 1993

L'enfant aux commandes de l'ordinateur, *Krieger D.*, Edition Eyrolles, 1984

ABC microcalculatoare, *Petrescu A. şi colectiv*, Editura Tehnică, Bucureşti, 1990

La conduite de l'IBM-PC, *Plonin A.*, Editions Eyrolles, 1984

Programarea structurată în BASIC, *Stan I., Şerban M., Sandor O., Singer H.*, Editura Libris, Cluj, 1992

Hello, BASIC, *State Lumniţa*, Editura AGNI, Bucureşti, 1992

Microsoft GW BASIC, *Vasiu L., Grama R.*, Editura Libris, Cluj, 1992

Programe de instruire – BASIC, *Vlad A. I.*, Editura Ştiinţifică şi Enciclopedică, Bucureşti, 1989

Să învăţăm să programăm în BASIC, *Zaharia C., Zaharia M.*, Editura Tehnică, Bucureşti, 1992

NUMELE _____

PRENUMELE _____

LOCALITATEA _____

STR _____

NR __ BL ____ ET ____ AP _____

JUDETUL _____ COD _____

Vă rog să-mi expediaţi prin colet poştal cu ramburs ___

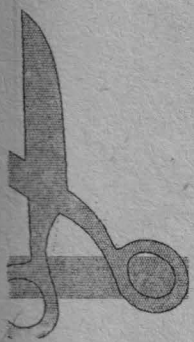
____ exemplare din cartea (cărţile) _____

_____

_____

_____

DATA _____ SEMNATURA _____



În curînd vor apărea:

◆

Mihaela Cârstea , Ion Diamandi
Un PC pentru fiecare

◆

Adrian Atanasiu
Cum se scrie un algoritm? simplu

◆

Marian Gheorghe
Cine eşti tu, BASIC ?

◆

Răzvan Andonie, Ilie Gârbacea
Algoritmi fundamentali în C++

Despre carte :

Cartea este un îndrumar pentru utilizarea instructivă a calculatoarelor de tip SPECTRUM (HC, JET, COBRA, CIP) atît la şcoală, cît şi acasă. Conţine rezolvarea a circa 50 de probleme de matematică, în limbajul BASIC (pentru calculatoarele de tip SPECTRUM) şi în versiunea GW-BASIC (pentru calculatoarele compatibile IBM-PC).

Instrument extrem de util pentru orele de informatică.

Despre autor :

Dl. **Ion Diamandi** este şeful compartimentului de Instruire Asistată de Calculator de la SOFTWARE-ITC-SA.

Cărţi publicate: ***Dialog cu viitorul*** (1988); ***Partenerul meu de joc, calculatorul*** (1988, 1989, 1990); ***LOGO - o nouă metodă de a învăţa cu ajutorul calculatorului*** (1991); ***Din spectacolul informaticii - calculatorul personal*** (1992); ***Cum să realizăm jocuri pe calculator*** (1992); ***40 jocuri logice în BASIC*** (1992).

ISBN 973-95626-3-9

Preţ 800 lei