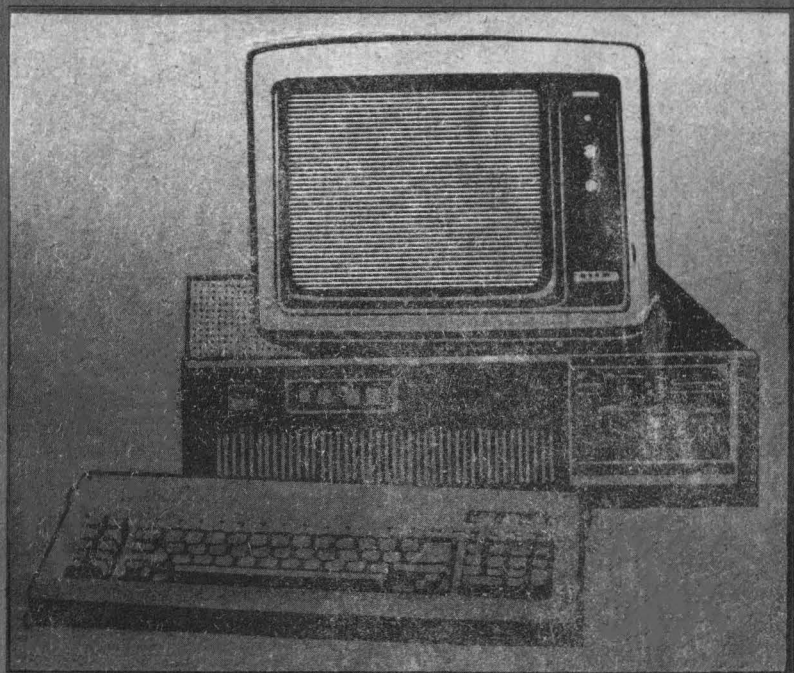


ION DIAMANDI

ION ODĂGESCU

DIN SPECTACOLUL INFORMATICII

calculatorul *personal*



DIN SPECTACOLUL INFORMATICII –
CALCULATORUL PERSONAL

ION ODĂGESCU

ION DIAMANDI

DIN SPECTACOLUL INFORMATICII—
CALCULATORUL PERSONAL



EDITURA MILITARĂ, BUCUREȘTI, 1991

Coperta : VICTOR ILIE

ISBN 973-32-0178-2

CUPRINS

Cap. 1. ACTUALITĂȚI ȘI PERSPECTIVE	7
1.1. Începuturi	7
1.2. Informatica personală	10
1.3. Supercalculatoarele	18
Cap. 2. ALGORITMI DE SORTARE, CĂUTARE ȘI MEMORARE	24
2.1. Algoritmi de sortare	25
2.2. Algoritmi de căutare	45
2.3. Memorarea masivelor	48
Cap. 3. PROGRAME BAZATE PE RECURENȚĂ ȘI PE ME- TODA CELOR MAI MICI PĂTRATE	53
3.1. Calcule recursive	53
3.2. Integralele $S_i(x)$ și $C_i(x)$	60
3.3. Integrarea unor funcții speciale	61
3.4. Calculul funcțiilor Bessel	63
3.5. Calculul funcției Laplace	65
3.6. Analize de regresie	69
Cap. 4. INSTRUIRE ASISTATĂ DE CALCULATOR	77
4.1. Tipuri de programe pentru instruire	78
4.2. Program pentru învățarea limbii engleze	81
Cap. 5. PROGRAME UTILITARE ȘI GRAFICĂ	84
5.1. Rutina de citire din memoria ROM pentru obținerea informațiilor referitoare la fișierele salvate	84
5.2. Rutină grafică pentru umplerea unor contururi	88
5.3. Reprezentarea grafică a funcțiilor	94

Cap. 6. PROGRAMARE ÎN STIL LOGO	100
6.1. Concepte generale	101
6.2. Variabile	115
6.3. Predicate	121
6.4. Evaluare condițională	124
6.5. Recursia	125
Cap. 7. PRODUSE PROGRAM SPECIFICE MICROCALCULATORARELOR	129
7.1. Gestiunea bazelor de date	131
7.2. Editare și prelucrare de tabele	136
7.3. Programe care realizează grafică de prezentare	141
7.4. Programe de editare și prelucrare texte	145
7.5. Poșta electronică	149
7.6. Sisteme automate pentru realizarea de publicații	150
Cap. 8. PROGRAME POSIBILE	152
8.1. Probleme ... matematice	152
8.2. Probleme ... cu operații numerice	162
Cap. 9. TENDINȚE	168
9.1. Mutații produse de IBM PS/2	168
9.2. Un contrast între strategii	176
9.3. Stații de lucru ingineresti	180
9.4. Securitatea informațiilor pe PC	186
ANEXĂ. MEMENTO BASIC-80	192
BIBLIOGRAFIE	204

ACTUALITĂȚI ȘI PERSPECTIVE

1.1. ÎNCEPUTURI

În multe țări tehnica de calcul a devenit în ultimii ani una dintre cele mai importante ramuri industriale, domeniu în care se înregistrează cel mai înalt ritm de înnoire a producției. Performanțele calculatoarelor și ale programelor aferente contribuie astăzi la modernizarea proceselor industriale, la ridicarea productivității muncii și a calității produselor, la dezvoltarea agriculturii, a transporturilor și telecomunicațiilor, la ocrotirea sănătății oamenilor și a mediului înconjurător, la instruirea tinerei generații.

Scopul producătorilor de sisteme de tehnică de calcul a fost permanent ca acestea să devină din ce în ce mai ușor de utilizat, mai fiabile, mai ușor de întreținut, să asigure siguranța datelor și un raport preț/performanță tot mai redus. În acest context apariția și dezvoltarea *calculatoarelor personale* a avut un rol hotărâtor dar și un impact uriaș asupra întregii societăți. De data aceasta, calculatorul nu mai reprezintă însă exclusiv o unealtă a specialistului în informatică, ci tot la fel de bine poate servi unui cercetător în orice domeniu, unui proiectant, unui lingvist, unui elev sau oricărei alte persoane. Ca să înțelegem evoluția fantastică pe care a avut-o domeniul tehnicii de calcul, în special prin intermediul calculatoarelor personale, să punctăm câteva aspecte relevante din scurta istorie a calculatoarelor și anume faptul că în anul 1947 existau în lume doar șase calculatoare instalate, IBM — singurul producător de calculatoare — hotărînd să nu mai investească în acest domeniu pe motiv că piața

prezintă un nivel prea scăzut; de menționat că pînă în anul 1967, la magazine și expoziții de calculatoare existau interdicții de intrare pentru tineri sub 18 ani. Astăzi, „parcul” de calculatoare personale instalate numără zeci de milioane de unități, apărînd reclame pentru ele chiar și în revistele de modă. Alt fapt semnificativ este acela că începînd cu 1984 — pentru prima oară în dezvoltarea electronicii și a calculatoarelor — ponderea cea mai mare pe piață este deținută de sisteme pe bază de microprocesoare, iar după puțin mai mult de un deceniu de la apariție în anul 1988, *sectorul calculatoarelor personale a devenit cel mai important segment pe piața electronicii în S.U.A.*

Fenomenul calculatoarelor personale este departe de a-și fi epuizat toate resursele. Creșterea rapidă a performanțelor calculatoarelor personale se va datora în continuare atît implementării rapide a tehnologiilor de vîrf în industria electronică, cît și folosirii unor soluții noi de arhitectură. Avîndu-se în vedere creșterea continuă a gradului de integrare a circuitelor electronice, abordarea noilor tipuri de circuite logice (Ga-As, bionica etc.), generalizarea treptată a unităților de memorie externă electronooptice pentru citire/scriere, cu capacități de zeci și sute de GB, se caută noi și noi soluții pentru realizarea calculatoarelor personale ale viitorului deceniu. Progresul tehnic și tehnologic extrem de rapid în domeniul microelectronicii favorizează elaborarea de noi generații de calculatoare personale de mare productivitate, răspîndirea lor pe scară foarte largă cu conectarea în rețele, apropierea lor de utilizatori și locuri de muncă (imagini grafice color, dialog prin voce, stații de lucru ingineresti orientate pe aplicații), dezvoltarea continuă de echipamente periferice și terminale aferente de mare performanță (discuri optice și rigide, imprimante cu laser, cu jet de cerneală, terminale grafice interactive de mare rezoluție) etc.

În evoluția calculatoarelor personale factorul determinant în succesul acestora a fost însă mai mult decît hardware-ul sau echipamentele periferice — software-ul.

În multe țări creșterea productivității și a calității produselor și serviciilor oferite de economiile naționale s-a realizat în mare măsură integrării software-ului.

Este de notat faptul că o bună parte din software-ul utilizat astăzi este demodat, mai ales în ceea ce privește capacitatea sa de a îmbunătăți productivitatea muncii.

Această învechire devine evidentă prin compararea cu noile produse software care tocmai își fac apariția, dar pînă la a căror generalizare se estimează că va trece un timp apreciabil. Producția de calculatoare personale, precum și puterea și capacitatea lor au crescut neîncetat. Astăzi, calculatoarele personale sînt la fel de puternice ca sistemele de calcul din decada trecută. Pentru a demonstra evoluția calculatoarelor, în general, și a celor personale, în particular, am selectat două calculatoare IBM din deceniile trecute (unul din seria 360 și altul din seria 370), ambele reprezentînd sisteme de calcul lider pe piața cal-

Tabelul 1.1.

Studiu comparativ

Anul	1965	1973	1981	1983	1984	1987
Model	360/ 30	370/138	PC	XT/370	PC/AT	PS2 Model 80
Preț* (K\$)	150	280	3	6	6	8
Performanță (MIPS)**	0,025	0,270+	0,300	0,500	1,2	3,5
Capacitate memorie internă (Ko)	10	500— 1 000	128— 640	640	256— 3 000	2 000— 18 000
Micropro- cesor/tehnolo- gie	—	—	8 088	8 088	80 286	80 386
Sistem de operare	DOS	DOS/ VSE VM/CMS	PC/DOS CP/M UNIX	VM/CMS PC/DOS PC/IX	XENIX PC/DOS 3,0 C-DOS	PC DOS 3.3 OS/2

* Configurație minimă tipică

** Milioane Instrucțiuni pe Secundă

culatoarelor la momentul respectiv. De asemenea, am luat în considerare patru calculatoare personale IBM din deceniul 1980, care au reprezentat standardul în acea perioadă. Tabelul 1.1 prezintă modelul, anul introducerii, capacitatea memoriei interne, performanța (viteza de prelucrare), tehnologia de realizare și prețul.

Se poate observa că pentru un anumit nivel de performanță prețul a scăzut considerabil.

Întrebarea cheie care se pune este la ce va fi folosită această putere de prelucrare. Răspunsul evident se referă la producerea unui software mai funcțional, mai productiv. Iar concluzia logică este că numai puterea hardware și vechiul software nu sînt suficiente, fiind imperios necesar un nou software pentru a utiliza efectiv și a beneficia de puterea hardului.

1.2. INFORMATICA PERSONALĂ

În 1973 la Orsay (Franța) se fondează societatea R2E care construiește primul microcalculator, Micral. Acesta este actul de naștere al microinformaticii; primul microcalculator a fost construit de francezi, dar „fenomenul micro” este de sorginte anglo-saxonă (IBM produce cîte un PC la fiecare 10 secunde, iar Apple — la fiecare 27 secunde). Specialiștii francezi nu au înțeles de la început fenomenul micro, nu au sesizat implicațiile economice ale acestuia. Microinformatica a reprezentat, printre altele, și transformarea calculatorului într-un instrument modern de lucru.

Astăzi, cei pasionați de informatică au devenit aproape la fel de numeroși ca și specialiștii. De aici derivă și aspectul de masă al microinformaticii.

Microcalculatoarele pot fi împărțite în prezent în mai multe grupe, în funcție de performanțe, caracteristici tehnice, utilizări, costuri etc. Astfel, se disting următoarele clase:

- a) calculatoare de buzunar (CBP) care se subdivid în:
 - a.1. calculatoarele de buzunar programabile în limbaje puțin evaluate, ce execută programe cu număr relativ mic

de instrucțiuni sau pași. Ele sînt larg răspîndite, fiind utilizate în special pentru calcule tehnico-științifice. Aceste calculatoare nu pot fi folosite pentru prelucrarea informației alfanumerice. Dintre produsele de acest tip amintim: CE109M, HP41, HP67, HP97, TI58, TI59 etc.;

a.2. calculatoarele de buzunar programabile într-un limbaj conversațional de nivel înalt, care permit introducerea datelor și instrucțiunilor de la tastatură alfanumerică miniaturizată. Memoria cu conținut permanent (PROM) stochează interpretorul pentru limbajul BASIC care dispune și de facilități de editare. În funcție de capacitatea memoriei RAM alocate (4-10 Ko), ele acceptă 1 000—65 000 linii de program BASIC. Dintre astfel de CBP-uri amintim: SHARP PC1 251, CASIO FX802 P, TANDY TRS80 PC2 etc.;

b) microcalculatoarele personale (individuale), care se subdivid în:

b.1. microcalculatoare portabile, ce dispun de un ecran de afișare matriceal, cu cristale lichide, iar tastatura este cea standard. Ele pot fi conectate la un televizor alb-negru sau color și la o miniimprimantă. Ca exemple se pot da: TI CC 40, CANON X 07, CASIO FP 200, SANYO TPC etc.;

b.2. microcalculatoare familiale, care posedă tastatură normală și folosesc, pentru afișare, un televizor alb-negru sau color, iar pentru stocarea externă a informației utilizează caseta magnetică. Memoria internă (RAM + ROM) este relativ mare, ceea ce permite lucrul cu programe mari. De aceea, aceste calculatoare au interpretare pentru BASIC, PASCAL, FORTH, MICROPROLOG etc. Ca domenii de aplicare, ele cuprind: învățămînt, proiectare, gestiune, jocuri, comenzi secvențiale etc.

Aceste microcalculatoare mai sînt cunoscute și sub numele de calculatoare personale sau individuale. Dintre tipurile utilizate pe scară mai largă amintim: aMIC, FELIX-Student, HC-85, PRAE, ZX81, SINCLAIR-SPECTRUM, ORIC1, DRAGON32, COMMODORE64 etc.;

c) microcalculatoare personal-profesionale (sînt cele mai performante micro-uri). Ele constau dintr-o tastatură, unitate centrală, monitor video, unități de discuri flexi-

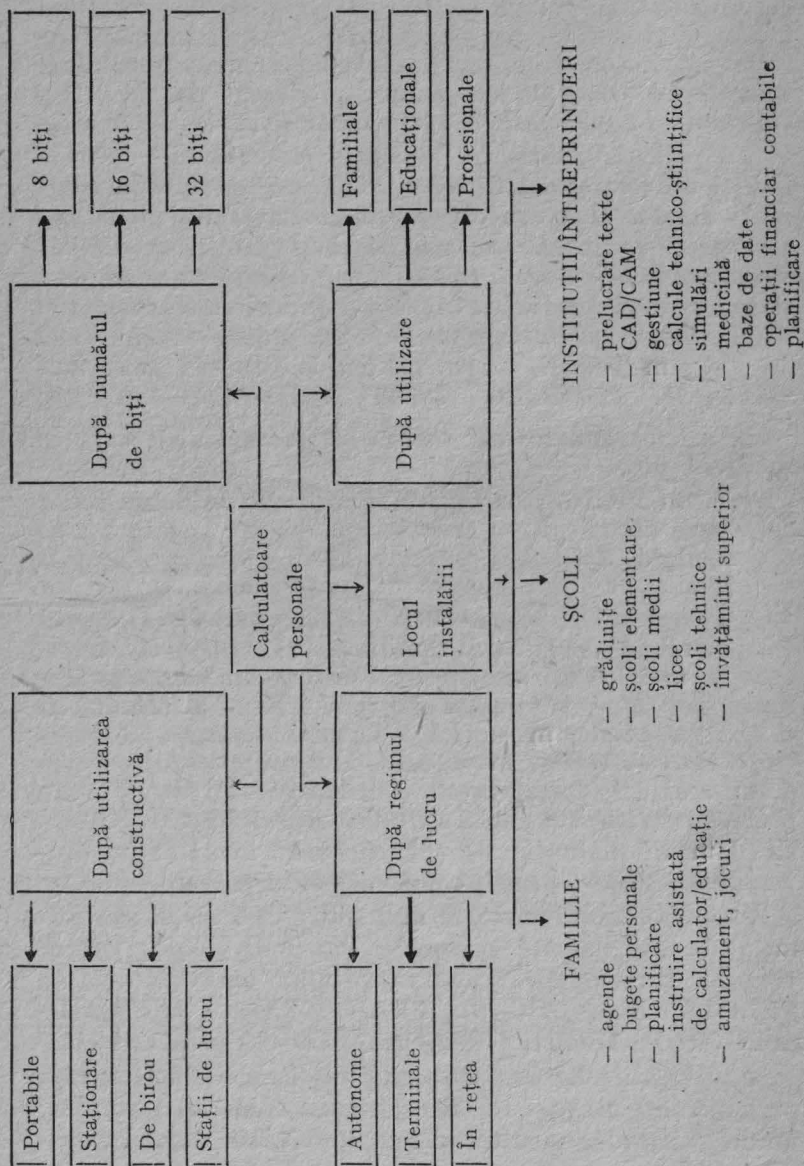


Fig. 1.1. Clasificarea și utilizarea calculatoarelor personale

bile, imprimantă și eventual alte echipamente periferice nestandard, fiind echipate cu microprocesoare orientate pe 8, 16 sau 32 biți. Limbajele evaluate BASIC, MODULA, FORTH, C, PROLOG, PASCAL etc. sînt implementate sub sistemele de operare CP/M, MS-DOS etc.

În funcție de capacitate, ele pot fi:

c.1. de capacitate medie, avînd următoarele caracteristici: memoria internă: 256-512 Ko; microprocesoare pe 16 biți (I 8 086, 8 088, Z 8 000, 68 000); unități de discuri flexibile, iar pentru versiunile extinse (XT) discuri în tehnologie Winchester de 10-80 Mo; sistemul de operare este MSDOS. Dintre sistemele din această categorie amintim: IBM PC lansat în 1981, avînd 256 Ko memorie internă și recunoscut ca standard internațional; IBM PC XT (1983, memorie extinsă; disc Winchester); Mac Intoch (Apple, S.U.A.) etc.;

c.2. de capacitate mare caracterizate prin: 512 Ko memorie internă; microprocesoare pe 16/32 biți (I 80 286, 80 386, 680 000 etc.); unități de discuri flexibile și discuri în tehnologie Winchester de 100-200 Mo; sistem de operare MS DOS sau OS/2. Ca reprezentanți ai clasei amintim: IBM PC AT (1984, IBM), IBM PS/2 (1987, IBM), MAC II (Apple, 1987);

c.3. stații de lucru — microcalculatoare evaluate pe 32 biți sau mini/microcalculatoare dedicate cu sistem de operare UNIX (exemple: IBM RT, SUN 3 000 etc.).

O clasificare a calculatoarelor personale, după mai multe caracteristici, poate fi observată în fig. 1.1. Dacă se notează: *a* — clasa calculatoarelor personale cu capacitatea redusă (*home computer*), *b* — clasa calculatoarelor de capacitate mică ce folosesc unități de discuri flexibile, *c* — clasa calculatoarelor profesionale, atunci principalele caracteristici ale produselor românești pot fi găsite în tabelul 1.2, iar ale celor mai răspîndite produse străine — în tabelul 1.3.

Domeniul calculatoarelor personale este dominat puternic în ultimii ani de cele profesionale. Piața calculatoarelor personale devine din ce în ce mai asemănătoare cu tradiționala piață de calculatoare. Atenția cu care sînt lansate produsele duce la apariția unor echipamente cu performanțe foarte spectaculoase, la prețuri scăzute. Copro-

Caracteristicile calculatoarelor personale românești

	aMIC	PRAE	HC85	TIM-S	CE 119	TPD Junior	CUB	FELIX PC
1	2	3	4	5	6	7	8	9
Memorie RAM	16-48 Ko	16-64 Ko	16-48 Ko	16-64 Ko	16-68 Ko	64 Ko	64 Ko	256-640 Ko
Compatibil	-	-	Spectrum	Spectrum	-	CP/M	CP/M	IBM PC
Tipul Microprocesor	a Z80	a Z80	a Z80	a Z80	a 8 080	b 8 080, Z80	b Z80	c 8 086 (+8 087)
Memorie externă	casetă	casetă	casetă	casetă	casetă 2 discuri flexibile 8 inch	1 disc, 8 inch sau 2 discuri 5 inch	2 unități de discuri flexibile	2 unități de discuri flexibile dublă față
Precizie BASIC	6 cifre	11 cifre	9 cifre	9 cifre	6 cifre	6-15 cifre	6-15 cifre	6-15 cifre
Limbaje	Basic	Basic	Basic, Forth, Pascal, C, Micro-Prolog	Basic, Forth, Pascal, C, Micro-Prolog	Basic, Forth, Editor	Basic, C, Forth, Pascal, Cobol, dBase	Basic, Pascal, C, Fortran, Cobol, Forth, dBase	Basic, C, Pascal, Cobol, Fortran, Lisp, dBase, Forth, Prolog

Tabelul 1.3.

Caracteristicile principalelor calculatoare personale produse în unele țări

Denumire	Firmă	Tip	Micro-procesor	Memorie internă (ko)	Memoria externă	Sistem de operare	Limbaje
ATARI 600 XL	ATARI (SUA)	a	6502	16—48	casetă magnetică	BASIC-ATARI	Basic, Pascal
IBM PC	IBM (SUA)	c	8088	256—640	discuri flexibile și disc fix	MS-DOS	Basic, C, Pascal, Fortran, Cobol, Lisp, Forth
Sinclair Spectrum	Sinclair (Anglia)	a	Z80A	16—48	casetă magnetică sau microdrive	BASIC-SINCLAIR	Basic, C, Pascal, Logo, MicroProlog Forth
Commodore 64	Commodore (SUA)	a/b	6510	64	casetă magnetică, disc flexibil	DOS (CP/M)	Basic, Pascal, Forth Logo
Apple II c	Apple (SUA)	b	6502	64—128	disc flexibil, casetă magnetică	DOS (CP/M)	Basic, Pascal, Forth Logo
HP 150	Hewlett Packard (SUA)	c	8088	256—640	discuri flexibile și disc fix	MS-DOS	Basic, C, Pascal, Fortran, Cobol, Lisp, Forth
MacIntosh	Apple (SUA)	b/e	68000	128 Ko-1 Mo	discuri flexibile și disc fix	Apple DOS	Basic, Pascal, C, Forth
IBM PC AT	IBM (SUA)	c	80 286	256 Ko-3 Mo	discuri flexibile (capacitate mare) și discuri fixe	MS-DOS	Basic, C, Pascal, Fortran, Cobol, Lisp, Forth

cesoarele, sistemele-expert, sinteza vocală, multiprogramarea au devenit tendințe obișnuite pentru calculatoarele personale.

Evoluția rapidă a microinformaticii conduce inevitabil spre realizarea de sisteme din ce în ce mai puternice dotate cu microprocesoare foarte rapide (viteze de 4—5 Mips) și facilități grafice remarcabile. Pe de altă parte, performanțele ultimelor modele de PC-uri sînt considerate apropiate de limita superioară permisă de structura componentelor și compatibilitatea programelor.

În aceste condiții, în aprilie '87 firma IBM lansa în fabricație o nouă familie de microcalculatoare, denumită PS/2 (*Personal System/2*), familie nouă din punct de vedere hardware sau/și software (tab. 1.4).

Familia de calculatoare PS/2 reprezintă fără îndoială un plus tehnologic, fapt ce determină mai multe firme de hard și de soft să anunțe planuri de susținere a noilor produse IBM, recunoscîndu-se astfel noul standard. De exemplu, toți realizatorii de produse program au anunțat că vor susține noul sistem de operare OS/2.

Firma Lotus a anunțat noi versiuni de 1-2-3 Symphony, Freelance Plus și Lotus Express pentru PS/2, utilizabile pe discuri de 3,5 inch. Ca și Lotus, Ashton-Tate va realiza noi versiuni pentru programe care vor rula cu dischete de 3,5 inch.

Dacă pentru producătorii de software și, respectiv, de calculatoare compatibile IBM lucrurile sînt clare, ei fiind imediat de acord cu noul standard, nu același lucru se poate spune despre utilizatori, care, acceptînd mai greu un nou standard după ce s-au obișnuit cu cel precedent, s-au împărțit deja în două tabere, unii susținînd cu frenezie noile produse, alții arătîndu-se mai reținuți.

În general, corporațiile (cei mai importanți utilizatori ai calculatoarelor PS/2) au apreciat lansarea noilor produse, care — fiind mai performante decît cele precedente — reprezintă incompatibilități și inconveniente. Se afirmă că existau motive tehnice solide pentru modificarea standardului. Alți utilizatori sînt însă mai reținuți, declarînd că vor achiziționa noile echipamente, dar că intenționează să testeze performanțele și compatibilitățile acestora.

Familia IBM PS/2

Model	Microprocesor	Memorie		Dischete		Discuri dure		Magistrale		Monitor grafic
		Standard	Maxim	Standard	Maxim	Standard	Maxim	Adrese	Date	
PS/2 30-002	8086/8MHz	640KB	2,6MB	2 x 720KB	—	—	—	20 biți	16 biți	MCGA*
PS/2 30-021	8086/8MHz	640KB	2,6MB	720KB	—	—	—	20 biți	16 biți	MCGA
PS/2 50-021	80286/10MHz	1MB	7MB	1,44MB	—	—	—	24 biți	16 biți	VGA
PS/2 60-041	80286/10MHz	1MB	15MB	1,44MB	2 x 1,44 MB	2 x 44 MB	2 x 44 MB	25 biți	16 biți	VGA
PS/2 60-071	80286/10MHz	1MB	15MB	1,44MB	2 x 1,44 MB	70MB	20 x 70 MB	24 biți	16 biți	VGA
PS/2 80-041	80386/16MHz	1MB	16MB	1,44MB	2 x 1,44 MB	44MB	2 x 44 MB	32 biți	32 biți	VGA
PS/2 80-071	80386/16MHz	1MB	16MB	1,44MB	2 x 1,44 MB	70MB	2 x 70 MB	32 biți	32 biți	VGA
PS/2 80-111	80386/20MHz	2MB	16MB	1,44MB	2 x 1,44 MB	115MB	2 x 115 MB	32 biți	32 biți	VGA
PC-XT/286	80286/6MHz	640KB	8,6MB	1,2MB	2 x 1,2 MB	20MB	—	24 biți	16 biți	—
PC-AT3	80286/8MHz	512KB	10,5MB	1,2MB	2 x 1,2 MB	30MB	2 x 30 MB	24 biți	16 biți	—

*) MCGA — Multi Color Graphics Array, VGA — Video Graphics Array

Există și sceptici, care invocă atît prețul exagerat al noilor calculatoare, cît și acele aspecte legate de apariția unei rețele nesigure formată din calculatoare de tip vechi IBM, de tip nou IBM și de calculatoare compatibile IBM, precum și de posibilitatea apariției unei avalanșe de noi tipuri de calculatoare din partea IBM și a producătorilor de calculatoare compatibile IBM.

O opinie justă pare a fi următoarea: noua tehnologie va mări desigur complexitatea muncii cu calculatoarele personale, astfel încît, cu toate că noile mașini vor deschide o nouă generație, tranziția nu va fi ușoară.

1.3. SUPERCALCULATOARELE

Sigur, cineva ar putea fi mirat de existența unui subcapitol pe tema supercalculatoarelor într-o lucrare despre calculatoare personale. Acest lucru este, însă, justificat din cel puțin două motive. Primul este legat de următorul fapt: dacă dezvoltarea calculatoarelor personale actuale este cauza principală a scăderii vânzărilor minicalcutoarelor, preconizîndu-se că acestea vor fi practic total înlocuite de calculatoarele personale care prezintă performanțe superioare sistemelor de calcul din anii 1960, atunci, cu siguranță, calculatoarele personale ale viitorului vor prezenta performanțe similare cu cele ale sistemelor și supercalculatoarelor de astăzi. Faptul este normal ținînd seama că viitoarele calculatoare personale vor „împrumuta” din ultimele cuceriri ale tehnologiei care se manifestă cel mai vizibil chiar la supercalculatoare. De fapt, dacă foarte multă lume cunoaște calculatoarele personale de mare succes (IBM-PC, Macintosh etc.) puțini cunosc faptul că există și *calculatoare personale pentru aplicații speciale*, dintre care unele, prin performanțele care le prezintă, sînt mai puternice decît sistemele de calcul (*main-frames*) uzuale. Aceste calculatoare folosesc circuite pe 32, 64 și 96 biți care lucrează la frecvențe de 100-200 MHz și memorii compacte, ajungînd la performanțe de 10-100 MIPS (!). În afară de circuitele obișnuite mai folosesc circuite pentru conversii digital-analogice și analogic-digitale care lucrează tot la frecvențe de 100-200 MHz.

Tehnologia de realizare a acestor calculatoare este cea folosită pentru Inițiativa de Calcul Strategic (0,25-0,5 μm), tehnologie ce va fi disponibilă pentru sectorul civil doar după anul 1990, în acest sector nelivrându-se, în prezent, tehnologii sub 1 μm . Nu rareori calculatoarele personale pentru aplicații speciale sînt realizate cu tehnologii specifice supercalculatoarelor cum ar fi, de exemplu, arseniura de galiu.

Al doilea motiv al includerii unui subcapitol de supercalculatoare este datorat modificării concepției de definire a supercalculatoarelor. Dacă de la apariția primului supercalculator (în urmă cu mai mult de 3 decenii) și pînă la jumătatea anilor '70 definiția supercalculatorului considera acest echipament drept cea mai puternică mașină existentă la un moment dat în cadrul unei game de modele sau al unei firme, în ultimii 15 ani optica s-a schimbat și definiția a trebuit să fie revizuită. La ora actuală, chiar dacă un calculator poate fi așezat fără dificultăți pe un birou obișnuit și este dedicat unui singur utilizator (calculator personal), el poate fi numit supercalculator dacă este proiectat cu o unitate de virgulă mobilă cu precizie extinsă și de foarte mare viteză pentru calcule științifice de mare performanță. Iar această denumire se aplică chiar dacă viteza sa este doar un procent din cea a unui sistem CRAY din ultimele modele.

În prezent, largă răspîndire a noilor generații de supercalculatoare ca mașini de clasa VIII a necesitat, pe lîngă reevaluarea definiției lor, și apariția unor noi tipuri de clasificări. Una dintre acestea se bazează pe însăși destinația calculatoarelor în virgulă mobilă de mare viteză și cuprinde mașini uniprosesor, mașini minimal paralele, avînd cîteva procesoare, și mașini cu un grad foarte ridicat de paralelism. Noile modele CRAY X-MP și Y-MP, precum și majoritatea supercalculatoarelor apărute în ultimii 5 ani cu arhitectura bazată pe cîteva procesoare și memoria partajabilă între acestea. În cea de-a treia categorie (a mașinilor cu grad ridicat de paralelism) intră noile echipamente IBM 3 090/VF, CRAY 2, 3 și viitorul 4, gama ETA-10, AHiant, Convex, Elxsi, Saxpy, Ametek, BB&N și Thinking Machines-1, ultimul avînd 65 536 de procesoare, la fel ca și Massively Parallel Processor utilizat la NASA.

Din punctul de vedere al tipurilor de supercalculatoare, piața pe plan mondial a acestor echipamente complexe oferea, în anul 1988, 65 de tipuri diferite provenind de la 28 de firme, în 1989 adăugându-se deja alte șase firme. Numărul pachetelor de aplicații variază, de obicei între 100 și 500, necesitând compilatoare specializate pentru calcule vectoriale sau în general paralele.

Domeniile de aplicații sînt din ce în ce mai numeroase, explicîndu-se astfel motivul pentru care la ora actuală cursa fabricării supercalculatoarelor a luat un avînt extraordinar. Domeniul prioritar rămîne cel militar, pentru prelucrarea digitală, în timp real, a semnalelor provenite de la sateliții militari de alarmă, sau pentru prelucrarea în timp real a imaginilor provenite de la sateliții de recunoaștere și fotografiere. Urmează apoi sectorul civil cu programe uriașe pentru modelarea și simularea fenomenelor legate de următoarele domenii: dinamica fluidelor și a gazelor, cercetările geologice și geofizice asupra resurselor terestre sau asupra sistemului planetar, construcția rezervoarelor și a clădirilor, prognoza vremii pe durată medie și lungă, calcule complexe din domeniul fizicii nucleare. Ca un exemplu concret, una din problemele acute ale fizicii nucleare este aceea a calculării masei protonului bazată pe formulele cele mai rapide și mai precise din cadrul cromodinamicii cuantice, teorie preponderentă ce urmărește descrierea structurii finale a materiei. Calculele care necesită circa 10^{17} operații în virgulă mobilă pe cuvinte de 256 biți și cu o precizie de numai 10% pentru problemă, necesitau în urmă cu trei ani rularea lor continuă timp de 15 ani pe cel mai rapid CRAY X/MP cu 4 procesoare și viteza de 800-1 000 MFlops, sau un CRAY-2.

Ca de obicei, firma IBM și-a propus să rezolve, mai rapid decît alți concurenți și această problemă, construind supercalculatorul GF/11-capabil să rezolve problema în numai patru luni, apoi cu o finanțare din partea DARPA a realizat o supermașină cu 576 de procesoare, numită RP3, și care tinde la viteza care să soluționeze problema în mai puțin de patru săptămîni. CRAY nu s-a lăsat mai prejos, anunțînd modelul CRAY 3 cu 16 procesoare și cu viteza comparabilă cu cea a lui IBM GF/11 și a lui ETA 10-GB.

Tipuri de supercalculatoare

Clasa	Tip	Firmă	Caracteristici – Conceptul de bază al arhitecturii	Performanțe medii	Anul apariti- iei
0	1	2	3	4	5
I	IBM 700	IBM	Memorie: 36Ko; lungime cuvinte: 36 biți	0,05 MIPS	1954
II	IBM 7000	IBM	Memorie: 32Ko; lungime cuvinte: 36 biți	0,25 MIPS	1959
III	CDC 6600	CDC	Memorie: 128Ko; lungime cuvint: 60 biți	1,25 MIPS	1965
IV	CDC 7600	CDC	Memorie: 512Ko; lungime cuvint: 60 biți	5 MIPS	1969
V	STAS 100	CDC	Procesor scalar și vectorial	80 MIPS	1971
	STARAN	Good year Aero space	Procesor aso- ciativ	70 MIPS	1971
	ASC	Texas Instru- ments	Procesor sca- lar multipipe- line		1972
	Iliac IV	Burro- ughs	Procesor para- lel și vectorial		1972
	BSC	Burro- ughs	Procesor para- lel și vectorial		1972
VI	CRAY 1	CRAY	Memorie: 1Mo; procesor vec- torial și scalar	100 MIPS (0,26 GFLOPS)	1976

Tabelul 1.5. (continuare)

0	1	2	3	4	5
	CYBER 203	CDC	Procesor vectorial și scalar	0,2—0,4 GFLOPS	1979
	CYBER 205	CDC	Procesor vectorial și scalar	0,8 GFLOPS	1980
VII	CRAY X-MP	CRAY	2 Procesoare tip conductă	0,4—0,6 GFLOPS	1981
	CRAY IS/100	CRAY	Memorie: 4Mo; procesor scalar și vectorial	600—1200 MIPS	1982
	CRAY X-MP/2	CRAY	2 Procesoare scalare și vectoriale	0,8 GFLOPS	1982
	HITACHI S-810	HITACHI	Memorie: 256 Mo; procesor scalar și vectorial	0,63 GFLOPS	1983
	FACOM M-380	FUJITSU	Procesor scalar și vectorial	0,26 GFLOPS	1983
	SX-1	NEC	Procesor scalar și vectorial	0,6 GFLOPS	1983
	CRAY X-MP/48	CRAY	Memorie: 4Mo; procesoare scalare și vectoriale	0,8 GFLOPS	1984
	SX-2	NEC	Multiprocesoare tip conductă	1,3 GFLOPS	1984
VIII	CRAY 2-2002	CRAY	2—4 procesoare scalare și vectoriale	1 GFLOPS	1985
	ETA 10	ETA SYSTEMS	Multiprocesor; 1—4 CPU	0,8—3,5 GFLOPS	1986

Tabelul 1.5. (continuare)

0	1	2	3	4	5
	RP3	IBM	512 procesoare, paralelism masiv	~ 100 GFLOPS	1987
	GF 11 (prototip)	IBM	paralelism masiv	11 GFLOPS	1988
	CRAY Y	CRAY	8 procesoare	1,4 GFLOPS	1988
	CRAY 3	CRAY	Multiprocesor; circuite utili- zând arseniură de galiu, ultra compact și răcit în întregime în- tr-o baie de fluorcarbon li- chid	10 GFLOPS	1989

Pentru amatorii de statistici prezentăm în tabelul 1.5 principalele tipuri de supercalculatoare, împreună cu câteva caracteristici de bază ale arhitecturii și performanțele pe care le pot realiza.

ALGORITMI DE SORTARE, CĂUTARE ȘI MEMORARE

Situațiile în care intervin probleme de căutare a unor valori sau de sortare a unor șiruri sînt foarte numeroase atît în activitățile economico-sociale (producție, organizare, servicii, administrație), cît și în activități școlare sau familiale.

În general, în lucrare vom folosi termenul de liste pentru a desemna un șir de numere.

Căutarea unor valori specificate într-o listă de numere (sau șiruri de caractere) poate fi efectuată mult mai eficient dacă numerele (sau șirurile de caractere) au fost în prealabil sortate într-o anumită ordine. De obicei se utilizează o ordine crescătoare pentru șiruri numerice și, respectiv, alfabetică pentru șiruri de caractere.

Pentru sortarea datelor au fost dezvoltate o serie întregă de modele (algoritmi). Există însă o mare diferență în ceea ce privește eficiența tehnicilor de sortare, aceasta depinzînd de tipul și volumul de date care vor fi sortate. De exemplu, un anumit algoritm poate conduce la rezultate bune în cazul unei liste de numere aleatoare, în schimb, folosirea sa poate fi neadecvată în cazul unei liste pentru care doar cîteva numere nu respectă o anumită ordine. Pentru liste formate din numere aleatoare algoritmul rapid de sortare (*Quick Sort*), precum și cel cunoscut sub numele de *Shell Sort* sînt mult mai eficienți decît algoritmul de sortare prin inversiuni. De obicei, alegerea celui mai eficient algoritm este o problemă care depinde foarte mult de experiență, în acest sens fiind necesară experimentarea mai multor tehnici pentru seturi de numere echivalente.

Vom prezenta în continuare cîțiva algoritmi de sortare începînd cu cei mai simpli, care, implicit, sînt și mai puțin eficienți.

2.1. ALGORITMI DE SORTARE

Sortare prin inversiuni (Bubble Sort). Principiul acestui algoritm constă în compararea numerelor din cadrul unei perechi de numere și în schimbarea poziției lor dacă sînt în ordine nedorită. Se formează deci perechi compuse din elementele 1 și 2, apoi 2 și 3, 3 și 4 pînă la $N-1$, N , iar la sfîrșitul acestei proceduri cel mai mare număr al listei va fi pe ultima poziție (a N -a). Procedura se va repeta și, la sfîrșit, următorul număr cel mai mare va ocupa ultima poziție din lista rămasă (poziția $N-1$). Astfel, procedura se va repeta pînă cînd ordinea va fi completă.

Următorul program (fig. 2.1) reprezintă algoritmul de sortare prin inversiuni și realizează ordonarea crescătoare a elementelor unei liste de numere.

Liniile 40-70 de program realizează introducerea elementelor listei, iar liniile 80-115 afișarea elementelor listei în ordinea în care acestea s-au introdus. Afișarea se va face pe un rînd (sau mai multe), între elemente intercalîndu-se cîte un spațiu. Linia 115 are rolul de a întrerupe scrierea în continuare după afișarea ultimului element al listei. Rutina de sortare se găsește între liniile 120 și 210.

```

10 REM ** PROGRAM ORDONARE **           120 REM MODUL ORDONARE
    ** PRIN INVERSIUNI **               130 FOR I=1 TO N-1
    *****                             140 FOR J=1 TO N-I
20 INPUT "numar de elemente ";         150 IF A(J)<=A(J+1) THEN GO TO
N                                         200
30 DIM A(N)                             160 LET C=A(J)
40 REM INTRODUCERE ELEMENTE           170 LET D=A(J+1)
50 FOR I=1 TO N                         180 LET A(J)=D
60 INPUT A(I)                           190 LET A(J+1)=C
70 NEXT I                                200 NEXT J
80 REM AFISARE LISTA NEORDONATA        210 NEXT I
A                                         220 REM AFISARE LISTA ORDONATA
90 FOR I=1 TO N                         230 FOR I=1 TO N
100 PRINT A(I); " ";                   240 PRINT A(I); " ";
110 NEXT I                               250 NEXT I
115 PRINT                                260 PRINT

```

Fig. 2.1. Sortare prin inversiuni

Indicele I semnifică numărul de treceri prin listă. Deoarece, după cum am mai arătat, după fiecare trecere, cel mai mare element din listă rămîne la coadă, înseamnă că este nevoie de cel mult $N-1$ treceri pentru ca lista să fie ordonată.

Indicele J semnifică locul (ordinea) unui element din listă, astfel încît perechile de numere care se vor compara vor fi $A(J)$ și $A(J + 1)$. Indicele va lua valori pînă la $(N - 1)$ deoarece, numai pînă în acest loc se vor compara perechile de numere, restul numerelor pînă la N fiind deja eliminate și puse la urma listei ca fiind mai mari.

Pentru a schimba ordinea valorilor în cadrul unei perechi de numere (în cazul în care ordinea nu este convenabilă), este necesară memorarea uneia din valori într-o variabilă auxiliară.

În exemplul de față s-a utilizat cîte o variabilă auxiliară C și respectiv D , pentru memorarea ambelor valori ale elementelor.

La sfîrșitul rutinei de sortare elementele sînt sortate, deci, pot fi listate (liniile 220-260).

Exemplu de utilizare:

Listă neordonată 129 267 56 41 69 43 99 90 4 8

Listă ordonată 4 8 41 43 36 69 90 99 129 267

Pentru a ilustra modul de funcționare a programului îl vom aplica doar pentru primele patru elemente ale listei inițiale luate ca exemplu, observînd cum se realizează ordonarea lor.

Sortarea a patru numere prin inversiuni: $A(1)$, $A(2)$, $A(3)$, $A(4)$

numere introduse: 129, 267, 56, 41

Proceduri:

(1) Trecere prin listă cu compararea succesivă a perechilor de numere.

De exemplu, $A(1)$ și $A(2)$, apoi $A(2)$ și $A(3)$. Dacă $A(1) > A(2)$, atunci se schimbă între ele, astfel încît $A(2)$ devine $A(1)$ și $A(1)$ devine $A(2)$. Dacă $A(1) = < A(2)$ atunci ele sînt lăsate în ordinea existentă.

Se observă că, în acest fel, cel mai mare număr din listă se va afla în final pe ultima poziție, adică a patra (267 va fi pe poziția $A(4)$).

(2) La prima trecere se vor face trei comparații și cel mai mare număr va trece pe poziția $A(4)$. La a doua trecere se vor face două comparații, iar cel mai mare număr (rămas) va trece în poziția $A(3)$. La a treia trecere se va face o singură comparație. Cel mai mare număr va fi $A(2)$. În acest moment nu mai sînt necesare alte treceri prin listă, cel mai mic număr rămînînd $A(1)$.

Sînt patru numere, deci, $N = 4$

Sînt necesare $(N-1)$ treceri, deci, $I = 1$ TO $(N-1) =$
 $= 1$ TO 3 treceri

Pentru fiecare pas sînt necesare $(N-I)$ comparații, deci,
 $J = 1$ TO $(N-1)$ comparații.

Iată și tabela de operații realizate în cursul sortării:

Tabela de operații

început	Trecerea 1 I=1			început trecere 2	Trecerea 2 I=2		început trecere 3	Trecerea 3 I=3	
	J=1	J=2	J=3		J=1	J=2		J=1	Sfîrșit
A(1)129	129			129	56		56	41	41
A(2)267	267	56		56	129	41	41	56	56
A(3)56		267	41	41		129	129		129
A(4)41			267	257			267		267

Sortare prin inversiuni cu controlul terminării. Acest algoritm prezintă o îmbunătățire față de precedentul, ceea ce îl face mai eficient (mai rapid).

În cazul algoritmului de sortare prin inversiuni poate apare situația în care, după mai puțin de $(N-1)$ treceri, ordonarea elementelor listei să fie deja realizată. În acest caz, algoritmul își continuă operațiile pînă la epuizarea

```

: 10 REM **PROGRAM ORDONARE **
      % INVERSIUNI CU
      *CONTROLUL TERMINARII
      *****
20 INPUT "Numar de Elemente ";
N
30 DIM A(N)
40 REM INTRODUCERE ELEMENTE
50 FOR I=1 TO N
60 INPUT A(I)
70 NEXT I
80 REM AFISARE LISTA NEORDONAT
.A
90 FOR I=1 TO N
100 PRINT A(I); " ";
110 NEXT I
115 PRINT
*120 REM MODUL ORDONARE
130 FOR I=1 TO N-1
135 LET S=0
140 FOR J=1 TO N-I
150 IF A(J)(<=A(J+1)) THEN GO TO
200
160 LET C=A(J)
170 LET D=A(J+1)
180 LET A(J)=D
190 LET A(J+1)=C
195 LET S=1
200 NEXT J
205 IF S=0 THEN GO TO 220
210 NEXT I
220 REM AFISARE LISTA ORDONATA
230 FOR I=1 TO N
240 PRINT A(I); " ";
250 NEXT I
260 PRINT

```

Fig. 2.2. Sortare prin inversiuni cu controlul terminării

întregului ciclu de treceri prin listă, deși nu se mai efectuează nici o operație de modificare a poziției elementelor listei, pierzându-se, astfel, un timp inutil. Pentru preîntîmpinarea unor astfel de situații, se utilizează o variabilă de control („flag”), S , care va indica dacă mai sînt necesare sau nu operații pentru ordonarea elementelor în liste. Dacă mai sînt operații de efectuat, algoritmul va continua normal, dacă nu, atunci se va trece direct la afișarea listei, întrucît aceasta este deja ordonată, fără a se mai face și alte treceri prin listă.

Programul din fig. 2.2 oglindește acest principiu.

Funcționarea este următoarea:

La începutul fiecărei treceri prin listă se inițializează variabila S cu 0 (linia 135). Dacă în cadrul trecerii prin listă, prin comparările dintre elementele listei, apare cel puțin un caz în care elementele unei perechi nu sînt în ordine, atunci fluxul programului va trece obligatoriu prin linia 195, în care variabilei S i se atribuie valoarea 1. După terminarea fiecărei treceri prin listă, algoritmul testează valoarea lui S (linia 205). Dacă $S = 1$ înseamnă că pozițiile elementelor listei au fost modificate și, în consecință, se va începe altă trecere prin listă (*NEXT I*); dacă $S = 0$ înseamnă că în cadrul listei nu mai sînt de efectuat modificări și, în consecință, lista se poate afișa, fiind ordonată.

Aplicînd algoritmul în noua sa formă pentru ordonarea listei date ca exemplu nu vom constata o îmbunătățire, în sensul scurtării timpului. Acest lucru este explicabil, deoarece dacă vom afișa valoarea indicelui I (care reprezintă numărul de treceri prin listă) înaintea începerii afișării listei ordonate, vom obține aceeași valoare atît în cazul inițial cît și în cel al controlului terminării. Adăugînd listei încă 10 elemente (de exemplu 1, 10, 8, 25, 100, 225, 400, 333, 100, 500), timpul de ordonare va scădea, în cazul aplicării algoritmului cu controlul terminării, de la 4 la 3 secunde (deci, scurtare cu 25%), efectuîndu-se doar 10 treceri prin listă în loc de 19.

Sortare alfabetică. Algoritmul de sortare prin inversiuni (sau oricare alt algoritm de sortare) se poate utiliza și în sortarea șirurilor de caractere, utilizîndu-se variabile tip șiruri de caractere. În cazul sortării alfanumerice trebuie indicată mărimea maximă a șirurilor de caractere comparate. Principiul după care se face ordonarea este același pentru litere, comparîndu-se, de fapt, codurile *ASCII* ale perechilor de caractere. Sortarea șirurilor de caractere, de exemplu, două cuvînte, se realizează prin compararea lor caracter cu caracter. Dacă primele două caractere sînt identice, atunci compararea se continuă asupra următoarelor două caractere și așa mai departe, pînă cînd este întîlnită o diferență. Dacă cele două șiruri nu au aceeași lungime, atunci cel scurt este inferior celui lung. Astfel se realizează o sortare a șirurilor de caractere în sens alfabetic. De exemplu, „*ABC*” este inferior lui „*ABD*”, deoarece codul *ASCII* al lui *C* este 67 care este mai mic decît 68 (codul *ASCII* al lui *D*). De asemenea, „*ABC*” este inferior lui „*ABCD*” fiind mai scurt. Bineînțeles „*ABCD*” va fi inferior lui „*CAB*”. După cum se poate observa, sortarea alfabetică poate fi de mare folos la realizarea multor lucrări practice: carte de telefon, agende personale, cataloage de școală, bibliografii etc. Programul respectiv este dat în fig. 2.3.

Pentru evitarea unor erori la rezultatele privind utilizarea acestui algoritm se impun cîteva observații:

— o literă mare nu are același cod *ASCII* cu litera mică corespunzătoare, făcîndu-se, deci, o deosebire între ele;

```

10 REM *SORTARE ALFANUMERICA*
*****
15 REM INTRODUCERE ELEMENTE
20 PRINT "INTRODUCETI NUMARUL
DE SIRURI DE CARACTERE (MAXIM 10
CARACTERE)"
25 INPUT N
30 DIM A$(N,10)
40 FOR I=1 TO N
50 INPUT A$(I)
60 NEXT I
70 REM AFISARE LISTA NEORDONAT
A
80 FOR I=1 TO N
90 PRINT A$(I); " ";
100 NEXT I

110 PRINT
115 REM MODUL SORTARE
120 FOR I=1 TO N-1
130 FOR J=1 TO N-I
140 IF A$(J+1)>A$(J) THEN GO TO 180
150 LET T=A$(J+1)
160 LET A$(J+1)=A$(J)
170 LET A$(J)=T
180 NEXT J
190 NEXT I
200 PRINT
210 REM AFISARE LISTA ORDONATA
220 FOR I=1 TO N
230 PRINT A$(I); " ";
240 NEXT I

```

Fig. 2.3. Programul de sortare alfanumerică

— spațiul liber (*SPACE*) conține, fiind considerat un caracter care are un anumit cod *ASCII*;

— utilizarea acestui algoritm pentru sortare numerică este posibilă, dar — în acest caz — trebuie ținut seama de faptul că este necesar ca numerele să aibă același număr de cifre.

Iată o utilizare incorectă în cazul sortării a 4 numere:

LISTA NEORDONATĂ

123 99 543 6

LISTA ORDONATĂ

123 543 6 99

Rezultatul este urmare a faptului că sînt comparate în primul rînd primele cifre ale numărului care nu au (în unele cazuri) aceeași pondere în cadrul numărului, unele reprezentînd unități, în timp ce altele — zeci sau sute. Utilizarea corectă este următoarea:

LISTA NEORDONATĂ

123 099 543 006

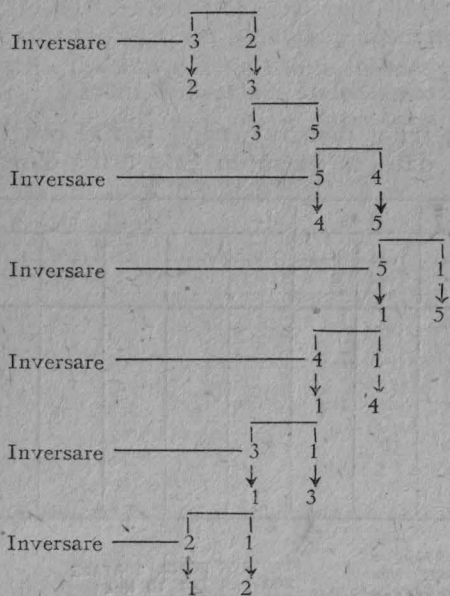
LISTA ORDONATĂ

006 099 123 543

Sortare prin inserare. Algoritmul de sortare prin inserare este mai eficient decît algoritmul de sortare prin inversiuni.

Să considerăm următoarea listă de numere: 3, 2, 5, 4, 1.

Principiul algoritmului de sortare prin inserare este următorul: se ia primul element din listă, apoi al doilea și se compară între ele, schimbându-se — dacă este necesar — ordinea. În continuare, al doilea element se compară cu al treilea, se schimbă ordinea, dacă este necesar, și în situația că se efectuează o schimbare de ordine atunci se compară din nou primul element cu al doilea și se schimbă ordinea dacă este necesar. Apoi, al treilea element este comparat cu al patrulea, și așa mai departe. Lista din exemplul dat se va sorta astfel:



Să considerăm lista $A(1), A(2), \dots, A(N)$. Pentru a insera elementul $A(I + 1)$ în poziția corectă se vor efectua următoarele operații:

Se face $T = A(I + 1)$, apoi

dacă $T > A(I)$ nu este necesară schimbarea între aceste elemente și, ca urmare, nu mai este necesară nici o altă comparație;

- dacă $T < A(I)$. atunci se schimbă ordinea între ele, $A(I+1) = A(I)$, și se continuă comparările;
- dacă $T \Rightarrow A(I-1)$ se face $A(I) = I$ și inserarea este terminată;
- dacă $T < A(I-1)$ se face $A(I) = A(I-1)$; și așa mai departe în josul listei.

Pentru realizarea modului de inserare se vor parcurge în program (fig. 2.4) următorii pași:

- 1) Let $J \Rightarrow I$ and $T = A(I+1)$
- 2) If $T \Rightarrow A(J)$ then let $A(J+1) = T$ and stop
- 3) Let $A(J+1) = A(J)$
- 4) Let $J = J - 1$
- 5) If $J < 1$ then let $A(J+1) = T$ and stop. If not, go to (2)
- 6) Repeat pentru fiecare valoare a lui I (de la 1 la $N-1$), unde N reprezintă numărul de elemente din listă.

Vizualizarea parametrilor pentru fiecare pas al programului, în cazul listei date ca exemplu este următoarea:

Început	I=1		I=2		I=3		I=4		
	J=1 T=3	J=2 T=5	J=3 T=4	J=2 T=4	J=4 T=1	J=3 T=1	J=2 T=1	J=1 T=1	J=0 T=1
A(1)=3	2								1
A(2)=2	3	3						2	
A(3)=5		5		4			3		
A(4)=4			5			4			
A(5)=1					5				

```

10 REM **ALGORITM DE SORTARE**
    * PRIN INSERARE *
    *****
15 REM INTRODUCERE ELEMENTE
20 INPUT "NR ELEMENTE ";N
30 DIM A(N)
40 FOR I=1 TO N
50 INPUT A(I)
60 NEXT I
70 REM AFISARE LISTA NEORDONAT

80 FOR I=1 TO N
90 PRINT A(I); " ";
100 NEXT I
110 PRINT

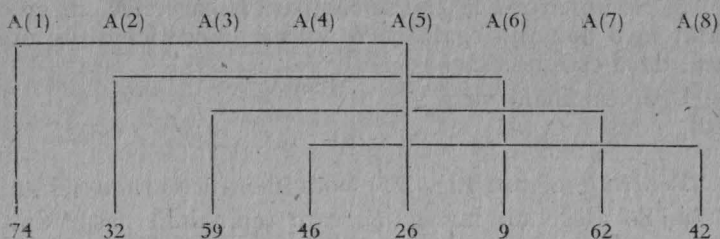
115 REM MODUL SORTARE
200 FOR I=1 TO N-1
210 LET J=I
220 LET T=A(I+1)
230 IF T>=A(J) THEN GO TO 270
240 LET A(J+1)=A(J)
250 LET J=J-1
260 IF J=1 THEN GO TO 230
270 LET A(J+1)=T
280 NEXT I
285 REM AFISARE LISTA ORDONATA
290 FOR I=1 TO N
300 PRINT A(I); " ";
310 NEXT I

```

Fig. 2.4. Sortare prin inserare

Sortare prin decojire (Shell Sort). Algoritmul de sortare prin decojire are la bază algoritmul de sortare prin inserare, însă sortarea prin inserare este precedată de un proces prin care elementele mai mici sînt mutate spre stînga, iar elementele mai mari spre dreapta, mai rapid. Pentru a evidenția principiul algoritmului să considerăm o listă de 8 elemente, ale căror valori sînt: 74, 32, 59, 46, 26, 9, 62, 42. Procesul de sortare va consta în următorii pași:

1) Se împarte 8 la 2, formîndu-se două liste distincte de cîte patru elemente. Se compară primul element din prima listă cu primul element din a doua listă, al doilea element din prima listă cu al doilea element din a doua listă și așa mai departe. Altfel spus, se compară fiecare element cu cel aflat peste patru poziții în lista inițială. Dacă în cadrul unei perechi formate numerele nu sînt în ordine convenabilă, atunci aceasta se schimbă:

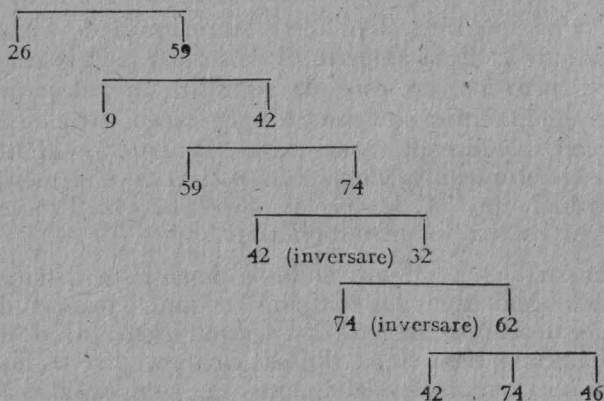


- Se compară: A(1) și A(5) → se inversează
 A(2) și A(6) → se inversează
 A(3) și A(7) → se lasă ordinea neschimbată
 A(4) și A(8) → se inversează

Noua listă va fi:

26 9 59 42 74 32 62 46

2) Se împarte 4 la 2 și se compară fiecare element cu cel aflat peste 2 poziții. Dacă cele două elemente comparate nu sînt în ordine convenabilă atunci se inversează:



Noua listă va fi:

26 9 59 32 62 42 74 46

3) Se împarte 2 la 2 și se compară fiecare element cu cel aflat la o distanță egală cu o poziție, modificându-se ordinea, dacă este necesar.

Ordinea finală va fi:

9 26 32 42 46 59 62 74

Pentru program (fig. 2.5) vom identifica următorii pași:

a) Se alege un întreg, S , care reprezintă pasul dintre elementele care sînt comparate. Alegerea se realizează,

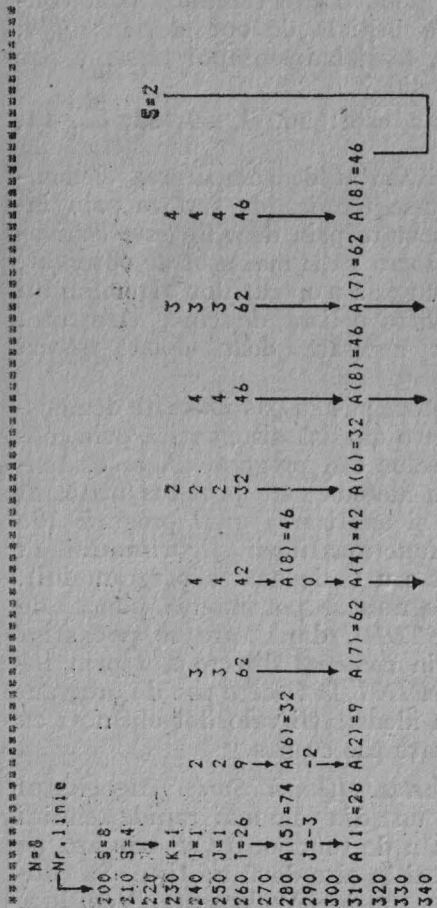
```

10 REM * ALGORITM DE SORTARE *
   * PRIN DECOJIRE *
   *****
15 REM INTRODUCERE ELEMENTE
20 INPUT "NUMAR ELEMENTE ";N
30 DIM A(N)
40 FOR I=1 TO N
50 INPUT A(I)
60 NEXT I
70 REM AFISARE LISTA NEORDONAT

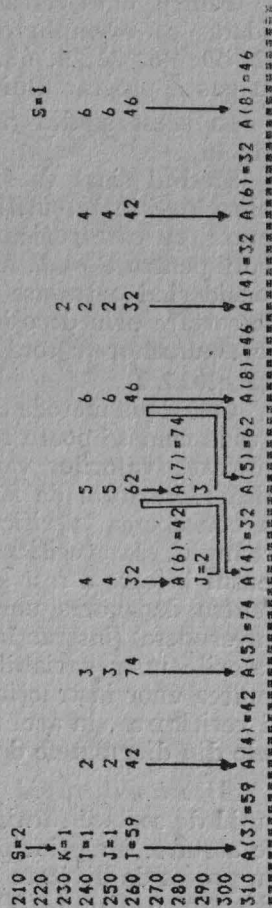
A
80 FOR I=1 TO N
90 PRINT A(I); " ";
100 NEXT I
110 PRINT
115 REM MODUL SORTARE
120 LET S=N
130 LET S=INT (S/2)
140 IF S<1 THEN GO TO 270
150 FOR K=1 TO S
160 FOR I=K TO N-S STEP K
170 LET J=I
180 LET T=A(I+S)
190 IF T>A(J) THEN GO TO 230
200 LET A(J+S)=A(J)
210 LET J=J-S
220 IF J=1 THEN GO TO 190
230 LET A(J+S)=T
240 NEXT I
250 NEXT K
260 GO TO 130
270 REM AFISARE LISTA ORDONATA
280 PRINT
290 FOR I=1 TO N
300 PRINT A(I); " ";
310 NEXT I

```

Fig. 2.5. Algoritmul Shell



 In acest stadiu (1), lista este: 26, 9, 59, 42, 74, 32, 62, 46.



Tabelul 2.1.
 Trasarea algoritmului
 Shell

de obicei, prin $INT(N/2)$, în care N reprezintă numărul de elemente ale listei;

b) Se sortează listele de articole de câte S elemente prin comparare și inversarea ordinii (dacă este necesar);

c) Dacă $S < 1$, *stop* (lista este ordonată);

d) Dacă $S \geq 1$, atunci se alege o nouă valoare pentru S (de obicei $INT(S/2)$) și se repetă pașii (b) și (d) de câte ori este necesar.

Pentru înțelegerea funcționării programului, vom considera ca exemplu o listă inițială de opt elemente (74, 32, 59, 46, 26, 9, 62, 42), realizându-se apoi trasarea pas cu pas a programului (tabelul 2.1).

În acest stadiu (2) lista este: 26, 9, 59, 32, 62, 42, 74, 46.

Stadiul final va fi reprezentat de compararea elementelor învecinate, utilizându-se tehnica de sortare prin inserare, cu care rutina de sortare prin decojire este echivalentă pentru $S = 1$. Acest lucru va fi mai ușor de observat, considerînd o trasare pas cu pas a operațiilor algoritmului de sortare prin decojire pentru o listă de cinci elemente, corespunzător cărora sînt necesare doar două treceri (tabelul 2.2).

Cele două metode de trasare pas cu pas ilustrate demonstrează cum se poate realiza o analiză sistematică prin modificarea valorilor variabilelor din program. Această tehnică reprezintă, de fapt, o metodă care poate fi utilizată la conceperea, verificarea și analizarea unui program (de exemplu, la verificarea funcționalității algoritmului în sensul în care a fost gîndit sau la depanarea programului). Pentru depanarea unui program se pot insera puncte de întrerupere (instrucțiuni *STOP*), după care se pot afișa valorile unor variabile prin comenzi directe sau prin inserarea unor instrucțiuni *PRINT* la fiecare pas de program și verificarea, în acel fel, a identității valorilor obținute cu cele din diagramele de trasare pas cu pas.

Algoritmul rapid de sortare (Quick Sort). Algoritmul rapid de sortare furnizează una din cele mai rapide tehnici de sortare, iar principiul său de funcționare se bazează pe un proces de divizare a listei în alte două subliste. Să considerăm o listă $A(N)$ care conține N elemente. Vom lua,

de exemplu, o listă de opt numere ($N = 8$). Principiul algoritmului rapid de sortare va fi descris de următorii pași:

1) Se inițializează doi pointeri, I și J , la cele două capete opuse ale listei. Primul din cele două elemente determinate de pointeri va fi denumit *număr de referință*. În exemplul prezentat, numărul de referință $A(I)$ este 63:

I							J
63	27	43	96	72	31	82	43

2) Se compară elementele corespunzătoare celor doi pointeri și se inversează ordinea dacă este necesar:

I							J
43	27	43	96	72	31	82	63

3) Se mută pointerul opus numărului de referință, un pas spre el:

	I						J
43	27	43	96	72	31	82	63

4) Se repetă pașii 2) și 3) pînă cînd $I = J$:

		I					J
43	27	43	96	72	31	82	63

			I				J
43	27	43	96	72	31	82	63

			I				J
43	27	43	63	72	31	82	96

			I			J	
43	27	43	63	72	31	82	96

			I		J		
43	27	43	63	72	31	82	96

			I		J		
43	27	43	31	72	63	82	96

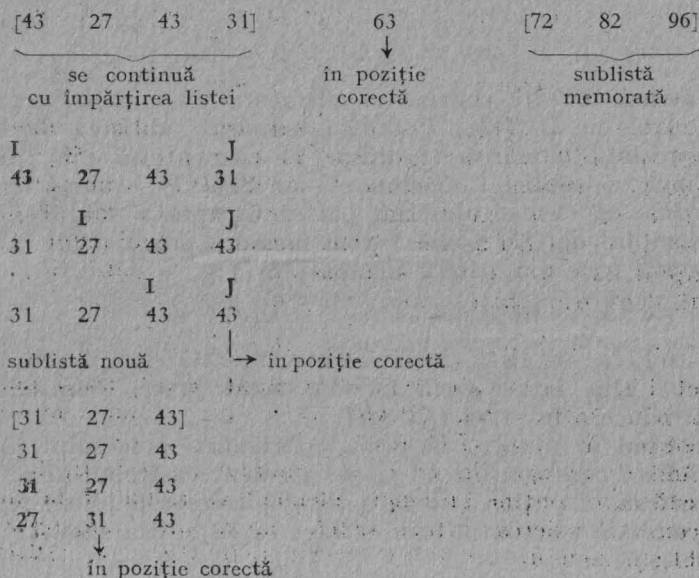
			I		J		
43	27	43	31	72	63	82	96

			I		J		
43	27	43	31	63	72	82	96

			I				
43	27	43	31	63	72	82	96

Atingerea acestui stadiu coincide, de fapt, cu împărțirea listei în două subliste. Numărul de referință este acum în poziția sa corectă din listă, iar sublistele sînt reprezentate de numerele din stînga și respectiv dreapta acestei poziții:

5) Una din subliste este trecută prin pașii (1) ÷ (4) iar cealaltă este memorată pentru o sortare viitoare:



6) Acest proces este repetat și, în fiecare caz, se memorează o sublistă. În final se sortează toate sublistele memorate anterior.

Pentru delimitarea elementelor din partea stîngă și dreapta a listei vom folosi două variabile, S și respectiv D , iar pentru pozițiile pointerului I și J se va întrebuița de asemenea un indicator, F , care poate lua două valori (1 și -1), specificîndu-se, în acest fel, care pointer determină numărul de referință. Astfel, $F = 1$ dacă numărul de referință este determinat de pointerul I și $F = -1$ dacă numărul de referință este determinat de pointerul J . Dacă

la sfârșitul pasului 4 numărul de referință este determinat de I (ca în exemplul dat), atunci lista a fost împărțită în:

sublistă	referință	sublistă
$(S, \dots, I-1)$	I	$(I+1, \dots, D)$

Lista din partea dreaptă este memorată prin crearea unei stive, folosindu-se o matrice $M(P,2)$ cu P inițializat la O . De fiecare dată când o sublistă este memorată, vom face:

$$P = P + 1, \quad M(P,1) = I + 1 \quad \text{și} \quad M(P,2) = D$$

Matricea M care conține N elemente este inițial dimensionată cu $M(N,2)$; P indică numărul sublistei, $M(P,1)$ reprezintă numărul de ordine al elementului din partea stângă a sublistei memorate, iar $M(P,2)$ — numărul de ordine al elementului din partea dreaptă a sublistei. În exemplul dat, la pasul 5 vom memora prima sublistă generată (cea din partea dreaptă) prin:

$$P = 1, \quad M(P,1) = 6 \quad \text{și} \quad M(P,2) = 8$$

Astfel, fiecare listă care trebuie memorată este plasată una după alta într-o zonă (stivă), acest proces numindu-se introducere în stivă (*PUSH*).

Când se ajunge, în urma continuării procesului, la o sublistă care conține un singur element va trebui să ne întoarcem, în vederea sortării listelor memorate. Vom regăsi o sublistă memorată prin extragerea listei din stivă (*POP*) prin:

$$S = M(P,1); \quad D = M(P,2) \quad \text{și} \quad P = P - 1.$$

Procesul va continua în acest mod pînă cînd toate listele sînt sortate (fig. 2.6). Liniile de program au semnificațiile următoare:

— în liniile 140 ÷ 180 se inițializează P , precum și valorile pentru S , D și pointerii I și J ;

— în linia 190 se stabilește indicatorul care indică poziția pointerului de referință ($F = -1$);

— în liniile 200 ÷ 240 se compară elementele $A(I)$ și $A(J)$, procedîndu-se o inversare a lor (dacă este cazul), iar apoi se reface valoarea indicatorului;

```

10 REM * ALGORITHM RAPID *
   * DE SORTARE *
   *****
15 REM INTRODUCERE ELEMENTE
20 INPUT "NUMAR ELEMENTE ";N
30 DIM A(N)
40 DIM M(N,2)
50 FOR I=1 TO N
60 INPUT A(I)
70 NEXT I
80 REM AFISARE LISTA NEORDONAT
A
90 FOR I=1 TO N
100 PRINT A(I); " ";
110 NEXT I
120 PRINT
130 REM MODUL SORTARE
140 LET P=0
150 LET S=1
160 LET D=N
170 LET I=S
180 LET J=D
190 LET F=-1
200 IF A(I)<=A(J) THEN GO TO 25
0
210 LET T=A(I)
220 LET A(I)=A(J)
230 LET A(J)=T
240 LET F=-F
250 IF F=1 THEN LET I=I+1
260 IF F=-1 THEN LET J=J-1
270 IF I<J THEN GO TO 200
280 IF I+1=D THEN GO TO 320
290 LET P=P+1
300 LET M(P,1)=I+1
310 LET M(P,2)=D
320 LET D=I-1
330 IF S<D THEN GO TO 170
340 IF P=0 THEN GO TO 400
350 LET S=M(P,1)
360 LET D=M(P,2)
370 LET P=P-1
380 GO TO 170
390 REM SFIRSIT SORTARE
400 REM AFISARE LISTA ORDONATA
410 PRINT
420 FOR I=1 TO N
430 PRINT A(I); " ";
440 NEXT I

```

Fig. 2.6. Algoritm Quick Sort

— în liniile 250 ÷ 270 se mută unul din pointerii I sau J (cel care trebuie mutat) în funcție de valoarea indicatorului;

— în linia 280 se verifică dacă pointerul I este la sfârșitul listei, iar apoi se trece la rutina de memorare a listei;

— în liniile 290 ÷ 300 se extrage (*PUSH*) lista din stivă;

— în liniile 320 ÷ 330 se verifică dacă sublista are mai mult de un element, controlul fiind condus în caz afirmativ la linia 170;

— linia 340 conduce controlul la rutina de afișare a listei ordonate dacă nici o sublistă nu este memorată;

— în liniile 350 ÷ 370 se introduce (*POP*) o sublistă în stivă;

— linia 380 conduce controlul la începutul rutinei de sortare pentru sublista introdusă în stivă.

Reamintim că unele extensii *BASIC* (de pildă, *BETA BASIC*) prezintă facilitatea de sortare a listelor de numere sau a celor de șiruri alfanumerice direct din cadrul limbajului *BASIC*. Pentru *BETA BASIC*, dacă a fost generată o listă $A(N)$, aceasta se va putea sorta foarte rapid cu

$SORT A(1 TO N)$ în ordine descrescătoare și cu $SORT INVERSE A(1 TO N)$ în ordine crescătoare. Cuvîntul cheie $SORT$ se va obține cu tasta M în modul $GRAPHICS$. Sortarea alfabetică pentru o listă de șiruri de caractere $A\$(N, M)$ se va obține cu $SORT A\$(1 TO N)$.

Sortare indexată. Înregistrările de fișiere pot fi constituite din mai multe cîmpuri. În acest caz, este necesară deseori sortarea articolelor după un anume cîmp (cheie) din cadrul articolelor. Acest tip de sortare se numește *sortare indexată*.

Să presupunem că avem o serie de înregistrări care formează un fișier pentru retribuțiile din cadrul unei întreprinderi și fiecare înregistrare conține următoarele opt cîmpuri: un număr de ordine, numele și prenumele, vechimea în muncă, funcția, categoria (gradația), secția, vîrsta, salariul. De exemplu:

10	POPESCU ION	11	ING	3	F12	36	3880
14	BADEA NICOLAE	17	EC	6	F6	44	4020
510	CRISTEA VASILE	15	TEHN	5	F1	35	3760

Ne putem propune să ordonăm aceste înregistrări în mod *alfabetic* după vechimea în muncă sau după vîrstă ori în funcție de salariu.

Vom prezenta o metodă de sortare cu care se poate realiza ordonarea înregistrărilor unui fișier după oricare cîmp al înregistrării. Deoarece pentru toate cîmpurile se utilizează variabile tip șiruri de caractere se va realiza o sortare alfabetică; din acest motiv este necesar ca înregistrările *numerice* (din același cîmp) să conțină același număr de caractere (cifre). Va trebui să înregistrăm 010, 014, 510 pentru numere de ordine și nu 10, 14, 510, utilizînd deci zerouri pentru menținerea aceleiași lungimi, fără modificarea valorilor.

Pentru sortarea înregistrărilor după un anumit cîmp se va proceda în felul următor:

1) Se utilizează un masiv $A\$(N, L, C)$ conținînd N înregistrări, fiecare de cîte L cîmpuri de maximum C caractere. De exemplu, se poate utiliza un masiv $A(10, 5, 20)$ pentru a reprezenta 10 înregistrări, fiecare cu cîte 5 cîmpuri, care pot conține fiecare pînă la 20 de caractere;

2) Se ia o decizie asupra cheii de sortare (cîmpul din cadrul înregistrării, după care se dorește să se realizeze sortarea, de exemplu al I -lea cîmp). În acest scop se inițializează un vector $K\$(N, C)$ și se face $K\$(R) = A\(R, I) pentru numărul de înregistrări ($FOR R = 1 TO N$), astfel încît lista $K\$(N, C)$ va conține cîmpurile pe care dorim să le aranjăm în ordine;

3) Se sortează cîmpul cheie în ordine crescătoare. În programul următor acest lucru se realizează în cadrul unei subrutine care începe la linia 900 și prin care se numără de cîte ori fiecare element din vectorul $K\$(N, C)$ este mai mare sau egal cu alte elemente (inclusiv elementul însuși). Dacă $K\$(N)$ este cîmpul cheie, atunci se face $X(P) = N$ și astfel se memorează, pentru fiecare element, rezultatul numărării (P) prin vectorul numeric $X(N)$.

La început se face $P = 1$ (fiecare cîmp este egal cu el însuși) și apoi se verifică condiția pentru alte cîmpuri aparținînd lui K \$(liniile 920-970), numărarea realizîndu-se prin adăugarea unei unități lui $P(P = P + 1)$, în cazul în care elementul este mai mare sau egal decît altul. Dacă elementele sînt egale, ordinea inițială în A \$ se păstrează (linia 960). De exemplu, se testează primul element al lui K \$ și se face $X(P) = 1$, se modifică valoarea lui P , se testează al doilea element al lui K \$ și se face $X(P) = 2$ și așa mai departe. Cu elementele 36, 44, 35, 22 care formează lista K \$, vectorul X va conține următoarele valori:

$$36 \quad X(3) = K\$(1)$$

$$44 \quad X(4) = K\$(2)$$

$$35 \quad X(2) = K\$(3)$$

$$22 \quad X(1) = K\$(4)$$

Prin afișarea lui K \$ ($X(1) TO X(4)$) se vor obține elementele sortate în ordine;

4) Masivul $A\$(X(N), L, C)$ va fi compus din înregistrările sortate în ordine după cîmpul ales și afișate utilizînd un ciclu $FOR-NEXT$ (liniile 270-320). Programul de realizare a acestei sortări este dat în fig. 2.7.

Introducînd în memorie datele exemplificate inițial pentru fișierul de retribuții (3 înregistrări a cîte 8 cîmpuri

```

10 REM ** SORTARE INDEXATA **
*****
20 REM INITIALIZARE DATE
30 INPUT "NUMAR MAXIM DE CA
RACTERE PE CIMP ";C
40 INPUT "NUMAR DE CIMPURI PE
INREGISTRARE ";L
50 INPUT "NUMAR DE INREGISTRAR
I ";N
60 DIM A$(N,L,C)
70 DIM K$(N,C)
80 DIM X(N)
100 REM INTRODUCERE DATE
110 FOR R=1 TO N
120 PRINT " INTRODUCETI ";L;" C
IMPURI PENTRU INREGISTRAREA ";
R
130 FOR I=1 TO L
140 INPUT A$(R,I)
150 NEXT I
160 NEXT R
170 PRINT
180 PRINT "AL CITELEA CIMP REPR
EZINTA CHEIA DE SORTARE?"
190 INPUT I
200 FOR R=1 TO N
210 LET K$(R)=A$(R,I)
220 N=K
230          900
240
250 PRINT "INREGISTRARILE SORTAT
E:"
260 PRINT
270 FOR R=1 TO N
280 FOR I=1 TO L
290 PRINT A$(X(R),I);" ";
300 NEXT I
310 PRINT
320 NEXT R
330 PRINT
340 PRINT "CONTINUARE?(D/N)"
350 INPUT Y$
360 IF Y$="D" THEN GO TO 180
370 STOP
380 REM SFIRSITUL PROGRAMULUI
890 REM SUBRUTINA DE SORTARE
900 FOR A=1 TO N
910 LET P=1
920 FOR B=1 TO N
930 IF K$(A)K$(B) THEN LET F=P
+1
940 IF K$(A)=K$(B) THEN GO TO 9
60
950 GO TO 970
960 IF A>B THEN LET P=P+1
970 NEXT B
980 LET X(P)=A
990 NEXT A
1000 RETURN
1010 REM SFIRSITUL SUBRUTINII

```

Fig. 2.7. Sortarea indexată

de maximum 15 caractere pentru un cimp), vom obține următoarele rezultate:

a) cheia = cimpul 2

014	BADEA NICOLAE	17	EC	6	F4	44	4020
511	CRISTEA VASILE	15	TEHN	5	F1	35	3760
010	POPESCU ION	11	ING	3	F12	36	3880

b) cheia = cimpul 3

010	POPESCU ION	11	ING	3	F12	36	3880
510	CRISTEA VASILE	15	TEHN	5	F1	35	3760
014	BADEA NICOLAE	17	EC	6	F4	44	4020

c) cheia = cimpul 4

510	CRISTEA VASILE	15	TEHN	5	F1	53	3760
010	POPESCU ION	11	ING	3	F12	36	3880
014	BADEA NICOLAE	17	EC	6	F4	44	4020

d) cheia = cimpul 5

Se obțin aceleași rezultate ca în cazul (c).

2.2. ALGORITMI DE CĂUTARE

Căutare liniară. Cea mai directă metodă de a căuta o anumită valoare într-o listă de numere nesortată este aceea prin care se examinează lista element cu element, de fiecare dată elementul fiind comparat cu valoarea căutată.

În programul din fig. 2.8 se creează o listă de numere aleatoare cuprinse între 100 și 199. Oricare ar fi numărul din listă generat, el nu va apărea decît o singură dată (liniile 45 ÷ 100). Elementele listei sînt afișate conform liniilor 110 ÷ 160, iar în liniile 200 ÷ 280 se realizează căutarea elementului dorit.

Dacă elementul dorit se găsește la începutul listei, el va fi depistat foarte rapid, dar pentru un element de la sfîrșitul listei găsirea se va realiza într-un timp mult mai lung. Pentru o listă care conține N elemente, numărul mediu de căutări va fi $N/2$.

Căutare binară. Această metodă, deși mult mai rapidă decît cea liniară, nu poate fi utilizată decît dacă în prealabil lista a fost sortată. În multe aplicații ne vom întîlni cu liste ordonate, iar în această situație căutarea binară va constitui metoda cea mai indicată.

În programul din fig. 2.9 tehnica de căutare binară se utilizează în liniile 500 ÷ 600. Principiul de bază constă

```
5 REM ** CAUTARE LINEARA **
*****
10 DIM A(100)
20 PRINT "INTRODUCETI NUMARUL
DE ELEMENTE DIN LISTA (<100)"
30 INPUT N
40 IF N>100 THEN GO TO 20
45 REM GENERARE ELEMENTE LISTA
46 REM ELEMENTELE DIN LISTA NU
SE REPETA
50 FOR I=1 TO N
60 LET A(I)=INT (100*RND)+100
70 FOR J=1 TO I-1
80 IF A(I)=A(J) THEN GO TO 60
90 NEXT J
100 NEXT I
110 REM AFISARE LISTA NEORDONA
TA
120 PRINT "LISTA NEORDONATA"
130 FOR I=1 TO N
140 PRINT A(I); " "
150 NEXT I
160 PRINT
200 REM RUTINA DE CAUTARE LINE
ARA
210 PRINT "INTROUCETI NUMARUL D
E CAUTAT (INTRE 100 SI 199)"
220 INPUT X
230 FOR I=1 TO N
240 IF X=A(I) THEN GO TO 280
250 NEXT I
260 PRINT "NUMARUL NU SE AFLA I
N LISTA DUPA ";N;" INCERCARI"
270 GO TO 300
280 PRINT "NUMARUL ";X;" GASIT
DUPA ";I;" INCERCARI"
300 REM END
```

Fig. 2.8. Căutare liniară

```

5 REM ** CAUTARE BINARA **
*****
10 DIM A(100)
20 PRINT "INTRODUCETI NUMARUL
DE ELEMENTE ALE LISTEI ((100))"
30 INPUT N
40 IF N>100 THEN GO TO 20
50 REM GENERARE ELEMENTE LIST
A
60 FOR I=1 TO N
70 LET A(I)=INT (100*RND)+100
80 FOR J=1 TO I-1
90 IF A(I)=A(J) THEN GO TO 70
100 NEXT J
110 NEXT I
120 REM AFISARE LISTA NEORDONAT
A
130 PRINT "LISTA NEORDONATA"
140 FOR I=1 TO N
150 PRINT A(I); " ";
160 NEXT I
170 PRINT
200 REM MODUL SORTARE PRIN INSE
RARE
210 FOR I=1 TO N-1
220 LET J=I
230 LET T=A(I+1)
240 IF T>A(J) THEN GO TO 280
250 LET A(J+1)=A(J)
260 LET J=J+1
270 IF J)=1 THEN GO TO 240
280 LET A(J+1)=T
290 NEXT I
300 REM AFISARE LISTA ORDONATA
310 PRINT "LISTA ORDONATA"
320 FOR I=1 TO N
330 PRINT A(I); " ";
340 NEXT I
350 PRINT
360 REM MODUL CAUTARE
370 PRINT "INTRODUCETI NUMARUL
CAUTAT (INTRE 100 SI 199)"
380 PRINT "LA SFIRSIT ISTATI 99
9"
390 INPUT X
400 IF X=999 THEN GO TO 700
410 PRINT
420 PRINT "CAUTAREA UNUI ELEMEN
T DIN LISTA DE ";N;" ELEMENTE"
500 REM CUTARE BINARA
510 LET L=1
520 LET H=N
530 LET C=0
540 LET M=INT ((H+L)/2)
550 LET C=C+1
560 IF X=A(M) THEN GO TO 630
570 IF L)=H THEN GO TO 650
580 IF X)>A(M) THEN GO TO 610
590 LET H=M-1
600 GO TO 540
610 LET L=M+1
620 GO TO 540
630 PRINT "ELEMENT GASIT ";X;"
DIN ";C;" " INCERCARI"
640 GO TO 370
650 PRINT "ELEMENT NEGASIT DUPA
";C;" INCERCARI"
660 GO TO 370
670 REM SFIRSIT CAUTARE
700 REM END

```

Fig. 2.9. Căutare binară

în compararea valorii căutate cu elementul aflat în mijlocul listei ordonate. Valoarea căutăată va fi fie mai mică (și în acest caz vom ști că ea se găsește în prima jumătate a listei), fie mai mare (și în acest caz se va cunoaște faptul că ea se găsește în cea de a doua jumătate a listei) decât elementul din mijloc. Iar dacă cumva nu este nici mai mică și nici mai mare, înseamnă că este egală, ceea ce semnifică că am dat peste valoarea căutăată. Procesul este repetat, în fiecare caz lista înjumătățindu-se. Să considerăm, de exemplu, căutarea valorii 30 în următoarea listă de 15 elemente:

1	2	4	6	8	10	12	14	16	18	20	24	28	30	36

Vom alege mai întâi 14 (elementul din mijlocul listei). Deoarece valoarea căutăată este mai mare înseamnă că ea

se găsește în partea dreaptă, iar lista în care vom căuta devine:

16	18	20	24	28	30	36
_____			_____			

Vom alege 24 (elementul din mijlocul listei). Deoarece valoarea căutată este mai mare, înseamnă că ea se găsește în partea dreaptă a listei, iar lista în care o vom căuta devine: 28 30 36. Vom selecta 30 (elementul din mijlocul listei) care este chiar valoarea căutată și acum găsită.

Astfel, am găsit un număr din trei căutări (încercări), comparativ cu 14, dacă am fi utilizat metoda de căutare liniară. În programul care realizează căutarea unei valori într-o listă de elemente vom observa următorii pași:

- 1) stabilirea (sau generarea) unei liste neordonate și afișarea ei (liniile 10 ÷ 170);
- 2) sortarea acestei liste și afișarea ei (liniile 200 ÷ 350);
- 3) căutarea binară cu afișare (liniile 500 ÷ 670).

Exemplu de utilizare:

INTRODUCEȚI NUMĂRUL DE ELEMENTE ALE LISTEI (< 100)
LISTA NEORDONATĂ

144	128	117	118	101	189	111	198
150	107	188	197	172	106	157	148
168	160	130	181	100	165	175	133
186	190	155	167	199	138	122	163
131	142	113	143	154	194	119	153

LISTA ORDONATĂ

100	101	106	107	111	113	117	118
119	122	128	130	131	133	138	142
143	144	148	150	153	154	155	157
160	163	165	167	168	172	175	181
186	188	189	190	194	197	198	199

INTRODUCEȚI NUMĂRUL CĂUTAT
(ÎNTRE 100 și 199)

LA SFÎRȘIT TASTAȚI 999

CAUTAREA UNUI ELEMENT DIN LISTA DE 40 DE ELEMENTE
NUMĂR GĂSIT 198
DIN 5 ÎNCERCĂRI

2.3. MEMORAREA MASIVELOR

Memorarea (salvarea) masivelor cu date numerice. Introducerea și memorarea datelor în masive se poate realiza prin intermediul instrucțiunilor *INPUT* în cadrul unor cicluri *FOR-NEXT*. În cazul în care se prelucrează un volum mare de date, se pot întrebuița instrucțiunile *READ-DATA*. Există însă posibilitatea de a genera liste de numere aleatoare prin intermediul funcției *RND*.

În programul din fig. 2.10 se creează un masiv $A(I)$ care se încarcă cu numere generate aleator. Programul se poate salva și, apoi, încărca din nou. În acest caz, dacă se va utiliza *RUN* în vederea lansării în execuție, variabilele (și deci masivul) se vor șterge. Din acest motiv, execuția se va realiza cu comanda *GO TO 10*.

Procedura generală este următoarea:

- 1) Realizarea și rularea unui program de generare a unei liste. Astfel se va crea $A(I)$;
- 2) Editarea liniilor și introducerea unor linii adiționale în funcție de necesități;
- 3) Salvarea programului final;
- 4) Încărcarea programului și execuția lui prin intermediul unei comenzi *GO TO*.

Pentru înlăturarea posibilității ca utilizatorul să folosească comanda *RUN* se poate folosi structura care lansează automat programul în execuție. Astfel, dacă într-un

```
10 DIM A(40)
20 FOR I=1 TO 40
30 LET A(I)=INT (100*RND)+100
40 NEXT I

10 REM "MASIV SALVAT"
20 REM PENTRU EXECUTIA PROGRAM
ULUI SE UTILIEAZA GOTO 10
30 PRINT "LISTA DE 40 DE NUMER
E ALEATOARE CUPRINSE INTRE 100 S
I 199"
40 FOR I=1 TO 40
50 PRINT A(I); " ";
60 NEXT I
70 PRINT
```

Fig. 2.10. Generare
numere aleatoare

```

*10 REM **RUTINA DE AUTO-RUN**
*****
20 REM AUTO-RUN VA PASTRA VAL
ORILE VARIABILELOR
30 DIM A(20)
40 FOR I=1 TO 20
50 INPUT A(I)
60 NEXT I
70 PRINT "PROGRAM CARE UTILIZE
AZA DATELE"
80 FOR I=20 TO 1 STEP -1
90 PRINT A(I)
100 NEXT I

110 REM ...ALTE LINII DE PROGRA
M
120 REM ...
8990 REM *SALVARE SI AUTO-RUN*
9000 CLS
9010 PRINT "INTRODUCETI NUME PRO
GRAM"
9020 INPUT A$
9030 PRINT "NUME PROGRAM:";A$.
9040 PRINT "NOTATI NUMELE PROGRA
MULUI"
9050 PAUSE 0
9060 SAVE A$ LINE 70

```

Fig. 2.11. Salvare și auto-run

program se introduce o linie de forma *9000 SAVE (program) LINE 200*; iar salvarea se realizează cu comanda *GO TO 9000*, atunci programul se va lansa automat în execuție după încărcare de la linia indicată (200).

Programul din fig. 2.11 realizează memorarea unui masiv prin procedura indicată, efectuându-se și o lansare automată în execuție.

Prin intermediul liniilor 40÷60 se creează un masiv de date. Aceste linii pot fi modificate, după ce datele au fost introduse, fără ca faptul respectiv să afecteze execuția programului sau, pot fi lăsate, în cazul în care se dorește să se introducă un nou set de date. Prin linia 9020 se solicită introducerea unui șir de caractere care va fi utilizat ca nume pentru program. Cu linia 9060 se realizează salvarea programului și a variabilelor sale. La încărcare, programul se va lansa automat în execuție de la linia 70, utilizând date care au fost deja introduse înainte de salvarea programului.

Memorarea masivelor cu șiruri de caractere: Programul din fig. 2.12 realizează un desen pe ecran (fig. 2.13) care poate fi salvat cu *SAVE "(nume)" SCREEN\$*. Pentru o încărcare ulterioară a imaginii-ecran se va folosi *LOAD "(nume)" SCREEN\$*.

Un ecran de caractere poate fi memorat și cu un masiv *A\$(704)*. În programul din fig. 2.14 se generează aleator 704 caractere din setul de caractere al calculatorului și apoi se plasează în fiecare celulă un caracter de pe cele 32 de coloane și 22 de linii ale ecranului. Bucla dublă din

```

10 FOR J=1 TO 40 STEP 4
20 FOR N=0 TO J*12 STEP 4
30 PLOT 125+J*2*SIN (N/(J*6)*PI
I),88+J*COS (N/(J*6)*PI)
40 NEXT N
50 NEXT J

```

Fig. 2.12. Program de desenare

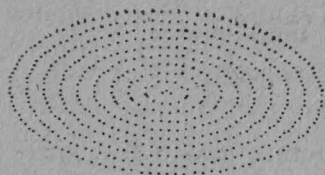


Fig. 2.13. Desenare prin program

```

5 DIM A$(704)
10 FOR L=0 TO 21
20 FOR C=0 TO 31
30 PRINT AT L,C:CHR$(32+RND*9
7)
40 NEXT C
50 NEXT L
60 FOR L=0 TO 21
70 FOR C=0 TO 31
80 LET A$(C+1+32*L)=SCREEN$(L
,C)
90 NEXT C
100 NEXT L

```

Fig. 2.14. Generare de caractere

liniile 60 ÷ 100 utilizează `SCREEN$(F, N)` pentru verificarea fiecărui caracter și plasarea sa în masivul `A$`.

După rularea programului, linia 5 se poate înlocui cu linia 5 `PRINT A$` și modifica sau șterge celelalte linii (10 ÷ 100). Acum există posibilitatea de a rula programul cu `GO TO 5`.

Deci, principiul rămâne variabil pentru orice masiv numeric sau tip șir de caractere. Odată rulat programul și datele inserate în masiv, ele rămân în siguranță putând fi accesate oricând, atât timp cât nu se utilizează `RUN`.

Memorarea datelor în șiruri de caractere. Șirurile de caractere pot fi utilizate pentru memorarea datelor, iar acestea pot fi folosite ulterior prin intermediul instrucțiunilor care mînuiesc șirurile de caractere. Datele pot fi, de asemenea, reasignate sau pot fi inserate noi valori. Există posibilitatea de a folosi și valori numerice, funcțiile `STR$` și `VAL` permițînd conversia valorilor numerice în șiruri de caractere și invers.

O metodă care permite cu succes eliberarea unor locații de memorie în cazul unor programe foarte mari (cu multe linii sau care folosesc multe date în instrucțiuni `READ-DATA`) ce nu încap în memorie se bazează pe faptul că se folosesc mai mulți octeți pentru memorarea valorilor în cazul variabilelor numerice, față de cazul celor de tip șir

de caractere. Practic, cu această metodă se vor economisi cîte trei octeți pentru fiecare dată (valoare) numerică. Aplicarea metodei constă în introducerea datelor numerice sub forma șirurilor de caractere, urmînd ca — la folosirea lor în program — să se utilizeze de fiecare dată conversia în valori numerice prin intermediul funcției *VAL*. În acest mod, se pot economisi în total cîteva sute de octeți în cazul utilizării unui număr de date de ordinul zecilor sau sutelor.

Calculul economiei de trei octeți pentru fiecare valoare numerică rezultă din următoarele considerente:

— fiecare valoare numerică este memorată în primul rînd ca un șir de caractere și, apoi, valoarea propriu-zisă, pe cinci octeți. Sînt necesari astfel șase octeți (un octet suplimentar este necesar ca delimitator) în afară de șirul de caractere;

— pentru memorarea fiecărui șir de caractere sînt necesari octeții pentru șirul propriu-zis (cîte un octet pentru fiecare caracter), precum și încă doi octeți pentru cele două caractere ghilimele. De asemenea, pentru utilizarea în program a valorii respective, folosirea cuvîntului cheie *VAL* (un octet) va ridica la trei octeți memorarea unui șir de caractere și folosirea sa ca valoare numerică, în afara octeților care reprezintă șirul propriu-zis. Astfel, economia finală va fi de trei octeți ($6 \div 3$) pentru fiecare valoare în parte.

În programul din fig. 2.15 datele (care reprezintă numele lunilor anului formate din primele trei litere) sînt memorate în variabila *A\$*. Ele pot fi accesate printr-un calcul simplu realizat în linia 70.

```

10 REM MEMORARE DATE TIP SIR
DE CARACTERE
20 LET A$="IANFEBMARAPRMAIIUNI
ULAUGSEPCTNOIDEC"
30 REM INTRODUCERE DATA
40 PRINT "INTRODUCETI LUNA (1
TO 12)"
50 INPUT LUNA
60 PRINT "LUNA ";LUNA;" ESTE "
;A$(LUNA*3-2 TO LUNA*3)

```

Fig. 2.15. Memora-
rea unui șir de ca-
ractere prescurtat

```

10 REM MEMORARE DATE TIP SIR
DE CARACTERE
20 LET A$=".IANUARIE.FEBRUARIE
.MARTIE.APRILIE.MAI.IUNIE.IULIE.
AUGUST.SEPTEMBRIE.OCTOMBRIE.NOIE
MBRIE.DECEMBRIE."
30 REM INTRODUCERE DATA
40 PRINT "INTRODUCETI LUNA (1-
12)"
70 INPUT LUNA
75 IF LUNA<1 OR LUNA>12 THEN G
O TO 30
80 LET P=0
90 LET A=1
100 IF A$(A)="." THEN LET P=P+1
110 IF P=LUNA+1 THEN GO TO 150
120 IF P=LUNA THEN PRINT A$(A+1
);
130 LET A=A+1
140 GO TO 100
150 PRINT "A FOST INTRODUSA"

```

Fig. 2.16. Memorarea unui șir de caractere extins

Următorul program (fig. 2.16) memorează în variabila A\$ numele întreg al lunilor, acesta fiind de lungime variabilă.

PROGRAME BAZATE PE RECURENȚĂ ȘI PE METODA CELOR MAI MICI PĂTRATE

3.1. CALCULE SUCCESIVE

Calculul factorialului. Realizarea unui program în limbaj BASIC pentru calculul factorialului, folosindu-se în acest scop mijloacele clasice de programare (iterative), nu reprezintă o problemă deosebită. Ca tehnică de programare, calculul factorialului se poate realiza elegant prin apelarea recursivă a unei subrutine, fapt evidențiat chiar de definiția factorialului, $n! = n \cdot (n - 1)!$.

Mecanismul intern al limbajului BASIC limitează însă drastic utilitatea apelurilor recursive, datorită faptului că toate variabilele în BASIC sînt globale.

Programul (fig. 3.1) și figura 3.2. asociată ilustrează o posibilitate de calcul al factorialului, folosind apelul recursiv în BASIC.

Se observă că programul conține dimensionarea stivei argumentelor n (linia 10) și a valorilor funcției $f(n)$ (linia 20), precum și un punct fix care nu face apel la recursie

```

2 REM *CALCULUL FACTORIALULUI
  FOLOSIND APELUL RECURSIV IN
  BASIC*
4 PRINT "Introduceți numarul"
5 INPUT n
10 DIM n(n+1)
20 DIM f(n+1)
30 LET fact=1
40 LET sp=n
50 LET n(sp)=n
60 GO SUB 100
70 PRINT "Rezultat=";f(sp+1)
80 STOP

100 PRINT "Apel cu argument :";
n: IF n=1 THEN LET f(sp+1)=1: RE
TURN
110 LET n(sp)=n: LET sp=sp-1: L
ET n=n-1: GO SUB 100
120 LET sp=sp+1: LET n=n(sp): L
ET f(sp+1)=n*f(sp)
130 PRINT "fact('";n;"')=";f(sp+
1)
140 RETURN

```

Fig. 3.1. Calculul factorialului

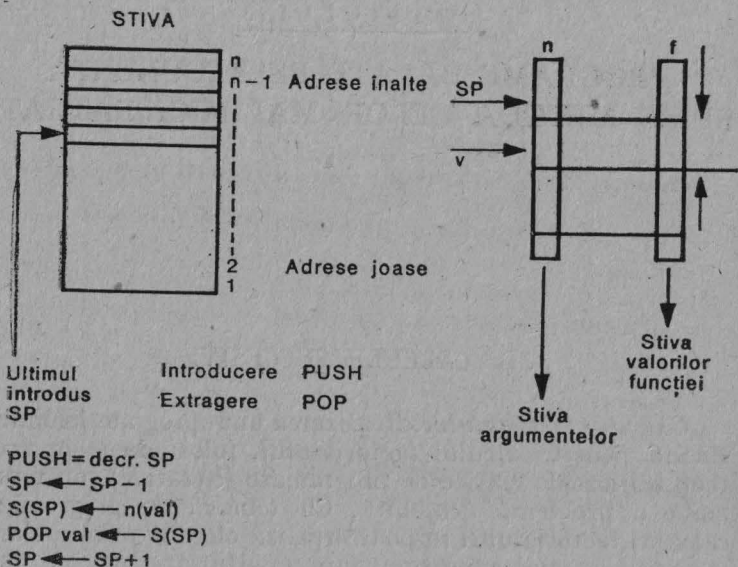


Fig. 3.2. Schema stivei pentru factorial

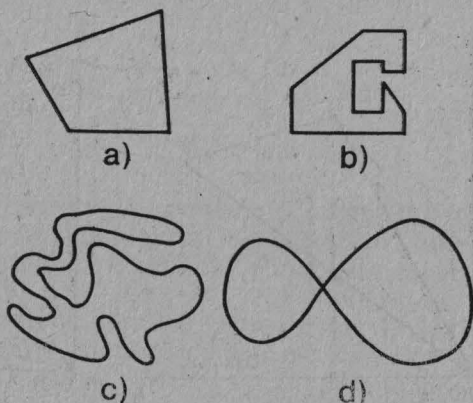
(linia 100). Dacă nu s-a ajuns la capăt cu calculul factorialului, mecanismul implică definiția recursivă prin apelarea funcției însăși (linia 110).

Calculul ariei unei suprafețe neregulate. Primul pas în determinarea ariei este reprezentarea obiectului pe o hartă fotografie, imagine proiectată etc. Utilizându-se apoi o tabletă grafică sau alt procedeu de transformare numerică, se va trasa perimetrul conturului suprafeței, generându-se un set de puncte ale căror coordonate carteziene se cunosc.

Calculatorul are rolul de a estima aria cuprinsă între punctele din eșantion. Precizia estimării va depinde de densitatea punctelor din eșantion. De asemenea, cu cât forma curbei (conturului) prezintă mai multe neregularități, cu atât sînt necesare mai multe puncte pentru o estimare bună.

Cu ajutorul algoritmului prezentat se vor calcula ariile suprafețelor delimitate de curbe închise simple, similare cu cele din fig. 3.3 a, b și c. Totuși algoritmul nu va putea

Fig. 3.3. Contururi
de delimitare a su-
prafetelor



să măsoare aria unei suprafețe delimitate de o curbă care se autointersectează ca în fig. 3.3 d. Aceste forme vor trebui „sparte” în curbe închise constituente, care vor fi analizate una câte una.

Să considerăm triunghiul OAB în spațiul cartezian (vezi fig. 3.4 a). Vom observa că:

$$\begin{aligned}
 \text{Aria } (OAB) &= \text{Aria } (OCB) + \text{Aria } (ABCD) - \text{Aria} \\
 &\quad (ODA) = \\
 &= x_2 y_2 / 2 = (x_1 - x_2)(y_1 + y_2) / 2 - x_1 y_1 / 2 = \\
 &= 1/2(x_2 y_2 + x_1 y_1 + x_1 y_2 - x_2 y_1 - x_2 y_2 - x_1 y_1) = \\
 &= 1/2(x_1 y_2 - x_2 y_1).
 \end{aligned}$$

Substituind valorile actuale în formulă vom observa că un triunghi parcurs într-o direcție va da un rezultat pozitiv, în timp ce parcurs în direcția opusă va da un rezultat negativ. Deci, pentru a obține cu siguranță un rezultat pozitiv, va fi necesar să luăm valoarea absolută a rezultatului.

Să considerăm suprafața mai complicată din fig. 3.4 b. Vom observa, de asemenea, că

$$\begin{aligned}
 \text{Aria } (ABCD) &= \text{Aria } (OAB) + \text{Aria } (OBC) + \\
 &\quad + \text{Aria } (OCD) - \text{Aria } (ODA).
 \end{aligned}$$

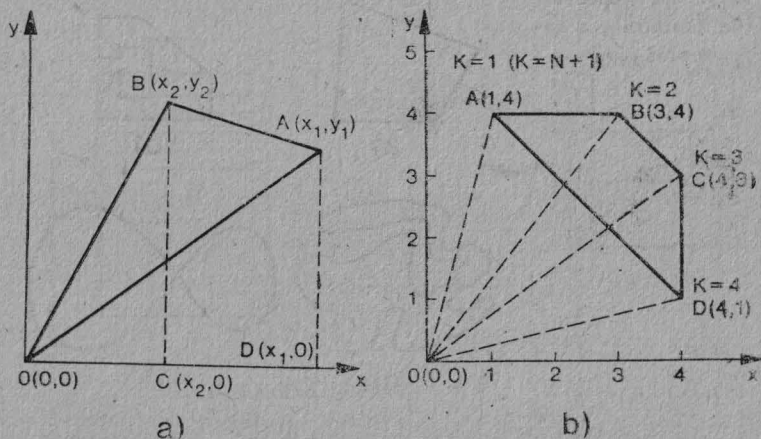


Fig. 3.4. Calculul ariei pe baza triunghiului

Putem aplica formula anterioară la fiecare din aceste triunghiuri, asigurându-ne permanent de același sens de parcurgere. În acest caz ariile componente vor fi toate ori pozitive, ori negative, permițându-ne astfel însumarea lor și aplicarea valorii absolute la rezultat.

Calculând aria lui $ABCD$ cu ajutorul valorilor fișate în fig. 3.4 *b*, respectiv punctele $A = (1, 4)$; $B = (3, 4)$; $C = (4, 3)$ și $D = (4, 1)$, vom obține: $\text{Aria } (ABCD) = = 1/2 \cdot | 1 \times 4 - 4 \times 3 + 3 \times 3 - 4 \times 4 + 4 \times 1 - 3 \times 4 + + 4 \times 4 - 1 \times 1 | = 0,5 \times | -8 | = 4$.

În vederea calculării ariei oricărei suprafețe (delimitate de o curbă închisă) se va generaliza formula anterioară. Dacă se iau coordonatele punctelor succesive apropiate, suprafața se va „sparge” într-o serie de triunghiuri apropiate, astfel încât chiar o suprafață mărginită de o linie curbă va putea fi aproximată cu un grad de acuratețe destul de ridicat.

Fie $\{x, y\}$ o secvență de n puncte, prin a căror unire se obține o curbă simplă închisă, C . Se definește un punct de pornire, (x_1, y_1) , care va fi același cu punctul final, (x_{n+1}, y_{n+1}) .

În acest caz,

$$\text{Aria } (C) = 1/2 \left| \sum_{k=1}^n (x_k y_{k+1} - x_{k+1} y_k) \right|.$$

Programul BASIC pentru calculatoare HC și TIM-S va fi dat în fig. 3.5.

Programul va citi o serie de coordonate x și y din linia DATA și va calcula aria mărginită de curba care unește aceste puncte. Se consideră că primul punct este conectat cu ultimul.

Linia 170 specifică numărul de puncte, iar linia 180 conține perechile x și y . În cazul de față s-a calculat aria unui dreptunghi, ale cărui vîrfuri sînt reprezentate de punctele de coordonate (1, 4); (3, 4); (3, 1) și (1, 1). Lățimea fiind 2 și lungimea 3, evident, aria va fi 6 (rezultat care se obține și prin rularea programului). Dacă coordonatele din linia 160 se vor înlocui cu cele din exemplul dat (fig. 3.4 b), adică (1, 4); (3, 4); (4, 3) și (4, 1), atunci se va obține exact aria calculată cu ajutorul formulei cunoscute (4).

Programul este utilizat pentru a calcula suprafețe delimitate de un eșantion de maxim 100 de puncte. Acesta poate fi mărit prin modificarea dimensiunii vectorilor din liniile 10 și 20.

În vederea folosirii cu un eșantion mare de puncte, programul se poate completa.

Calculul derivatei unei funcții. Pentru a evalua derivata unei funcții de o variabilă se poate aplica formula:

$$f'(x) = \frac{f(x+dx) - f(x)}{dx}, \text{ cu un } dx \text{ suficient de mic.}$$

```

10 REM *CALCULUL ARIEI UNEI SU
PRAFETE NEREGULATE DEFINITE PRIN
. PUNCTE DE COORDONATE DATE*
20 DIM X(100)
30 DIM Y(100)
40 READ N
50 FOR K=2 TO N+1
60 READ X(K),Y(K)
70 NEXT K
80 LET X(1)=X(N+1)
90 LET Y(1)=Y(N+1)

```

```

100 LET ARIA=0
110 FOR K=1 TO N
120 LET ARIA=ARIA+X(K)*Y(K+1)-X
(K+1)*Y(K)
130 NEXT K
140 LET ARIA=0.5*ABS ARIA
150 PRINT "ARIA CUPRINSA INTRE
PUNCTE ESTE ";ARIA
160 STOP
170 DATA 4
180 DATA 1,4,3,4,3,1,1,1

```

Fig. 3.5. Program de calcul al ariei

```

PRINT "Derivata unei functii"
PRINT : PRINT "Expresia functiei"
BEEP .2,20: INPUT "Tasteaza valoarea de variabila x":e$
PRINT PAPER 6: INK 9:AT 4.5
FN f(x)=VAL e$
LET eps=1e-6
PRINT : PRINT "valoarea lui f"
BEEP .2,20: INPUT "x ":xi
PRINT "Derivata este estimata la"
LET der2=0
FOR i=2 TO 20
150 LET der1=der2
155 LET dx=.5^i
160 LET der2=(FN f(x+dx)-FN f(x))/dx
165 LET prec=ABS (der1-der2)
170 IF prec<eps THEN GO TO 200
180 NEXT i
190:
200 LET derivata=2*der2-der1
210 PRINT : PRINT PAPER 6: INK 9:derivata
220:
230 BEEP .2,20: INPUT "Alta valoare a lui x (d/n)":xv
240 IF xv="d" THEN GO TO 100
250 IF xv("<n" THEN GO TO 230
260:
270 BEEP .2,13: BEEP .2,16
280 STOP

```

Fig. 3.6. Program de calcul al derivatei unei funcții

Luând $dx = 1/4$, se va obține o primă estimare a derivatei, care se va compara cu cea de a doua pentru un $dx = 1/8$. Se va proceda la fel pentru un $dx = 1/8$ și $1/16$ și așa mai departe, pînă cînd diferența dintre două estimări succesive ale derivatei va fi inferioară unei valori impuse. Această valoare este fixată, în cazul programului din fig. 3.6, la 10^{-6} . Numărul maxim de evaluări este 19, liniile $140 \div 180$ reprezentînd ciclul de estimări succesive ale derivatei. În linia 50 se realizează definiția funcției utilizator. Ca exemplu, se poate calcula derivata lui $\sin(x)$ pentru $x = 0$. Aceasta va fi estimată la 1,0 000 203 — deci, cu un grad de precizie rezonabil față de valoarea reală (1).

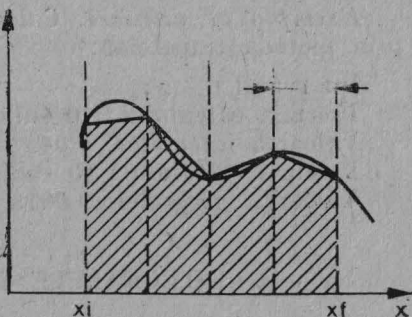
Calculul integralei unei funcții. Pentru o funcție a cărei expresie este cunoscută există, în general, două metode de calcul al unei integrale definite pe un interval: metoda trapezelor (pentru care funcția de integrat este asimilată cu o linie frîntă, fig. 3.7) și metoda Simpson (care înlocuiește segmentele acestei linii frînte prin arcuri de parabolă).

Formula de calcul este:

$$I = \left[f(x_i)/2 + \sum_{i=1}^{n-1} [f(x_i + i \cdot li)] + f(x_f)/2 \right] \cdot li,$$

unde n este numărul de subintervale (vezi fig. 3.8).

Fig. 3.7. Metoda trapezelor



```

5 DEF FN i(t)=SIN t
10 PRINT "Integrala unei funct
i i"
20 PRINT "a carei expresie est
e cunoscuta."
30 PRINT "Metoda trapezelor"
40 PRINT : PRINT "intervalele
E":
50 BEEP .2,20: INPUT "limita i
nferioara":xi: PRINT xi:" " :
60 BEEP .2,20: INPUT "limita s
uperioara":xf: PRINT xf:" ] "
70 PRINT "precizia integralei"
:
80 BEEP .2,20: INPUT "l : mica
. 1000 : mare " :ni
85 PRINT TAB 26-LEN STR$ ni:ni
: "sous-intervalles"
90 LET li=(xf-xi)/ni

100 LET integ=FN i(xi)/2
110 FOR i=1 TO ni-1
120 LET integ=integ+FN i(xi+i*1
i)
130 NEXT i
140 LET integ=integ+FN i(xf)/2
150 LET integ=integ*li
160:
170 PRINT : PRINT "valoarea int
egralei " :integ
180:
190 BEEP .2,20: INPUT "alta pre
cizie (o/n) " :rs
200 IF r$="o" THEN PRINT : GO T
O 70
210 IF r$="n" THEN GO TO 240
220 GO TO 190
230:
240 BEEP .2,13: BEEP .2,16
250 STOP

```

Fig. 3.8. Programul de calcul al integralei

În linia 5 se realizează definirea funcției de integrat, iar linia 90 mărimea unui subinterval. În linia 100 se definește primul termen al sumei (punctul x_i), iar în ciclul $110 \div 130$ se realizează cumulul pentru punctele intermediare. Pentru alte funcții se poate redefini linia 5. De exemplu, pentru funcția t^2 linia 5 va fi:

5 DEF FN $i(t) = t \times t.$

Valoarea este obținută prin exces (valoarea exactă este 9) și se ameliorează dacă numărul de subintervale crește.

Exemplu de utilizare. Calculul integralei unei funcții prin metoda trapezelor:

Intervalul 0,3

Precizia calculului: 10 subintervale

Valoarea integralei: 9.045

Precizia calculului: 50 subintervale

Valoarea integralei: 9.0018

3.2. INTEGRALELE SINUS ȘI COSINUS

Integrala sinus

$$Si(x) = SI(X) = \int_0^x \frac{\sin t}{t} dt$$

se calculează după relația

$$Si(x) = \sum_{n=0}^{\infty} [(-1)^n x^{2n+1} / (2n+1) (2n+1)!],$$

în care însumarea este oprită dacă termenul [] este, în modul, mai mic decât $\varepsilon = 10^{-9}$.

Programul din fig. 3.9, executat pentru $x = 0,1$ conduce la soluția $Si(0,1) = 0,099\ 944\ 461$, iar pentru $x = 10$ produce $Si(10) = 1,6583\ 476$.

```

5 REM * CALCULUL FUNCȚIEI *
  *           SI(X)           *
  * ***** *
10 INPUT "INTRODUCE X=";X
20 LET B=X: LET C=X
30 LET D=-(X*X)/2: LET I=0
40 LET I=I+1: LET E=(2*I+1)^2
50 LET B=((2*I-1)*D*B)/(I*E)
60 LET C=C+B
70 IF ABS(B)<1E-9 THEN GO TO
90
80 GO TO 40
90 PRINT "SI(N)=";C: GO TO 10
100 STOP

```

Fig. 3.9. Calculul funcției $SI(X)$

Fig. 3.10. Calculul funcției CI(X)

```

5 REM * CALCULUL FUNCȚIEI *
  * CI(X) *
  *****
10 INPUT "INTRODUCE X=";X
20 LET S=X*X: LET I=0: LET A=0
30 LET B=1: LET C=1
40 LET I=I+1: LET D=2*I
50 LET C=(D-1)*D*(-C): LET B=B
*S
60 LET E=B/(C*D): LET A=A+E
70 IF ABS (E)<1E-9 THEN GO TO
90
80 GO TO 40
90 PRINT "CI(X)=";A+0.57721566
49+LN (X): GO TO 10
100 STOP

```

Integrala cosinus

$$Ci(x) = CI(X) = \gamma + \ln x + \int_0^n \frac{\cos t - 1}{t} dt$$

este calculată cu formula:

$$Ci(x) = \gamma + \ln x + \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{2n \cdot (2n)!},$$

iar programul este dat în fig. 3.10. Dacă $x = 1$, atunci rezultatul execuției programului este $Ci(1) = 0,33\ 740\ 392$.

3.3. INTEGRAREA UNEI FUNCȚII SPECIALE

1. Integrala specială

$$E_n(x) = EN(X) = \int_1^{\infty} \frac{\exp(-xt)}{t^n} dt$$

se calculează cu ajutorul relației de recurență

$$E_{n+1}(x) = \frac{1}{n} [e^{-x} - xE_n(x)], \quad n = 1, 2, 3, \dots$$

```

5 REM * CALCULUL FUNCTIEI *
  *      EN(X)      *
  *****
10 INPUT "INTRODUCE H=";H
20 INPUT "INTRODUCE N=";N
30 INPUT "INTRODUCE X=";X
40 LET K=EXP (-X): LET E=K/X
50 IF N>0 THEN GO TO 70
60 PRINT "E0(X)=";E: GO TO 70
70 LET P=1: LET R=1: LET S=0:
LET I=0
80 LET I=I+1: LET P=(-X)*P: LE
T R=R*I
85 LET C=P/(I*R): LET S=S+C
90 IF ABS (C)>H THEN GO TO 80
100 LET E=-S-0.5772156647-LN (X
)
110 FOR I=1 TO N-1: LET Y=(K-E*
X)/I
120 LET E=Y: NEXT I
130 PRINT "EN(X)=";E: GO TO 30
140 STOP

```

Fig. 3.11. Integrala EN(X)

unde $E_0(x) = \exp(-x)/x$, $E_1(x) = -\gamma - \ln x - \sum_{i=1}^{\infty} (-1)^i x^i / (i \cdot i!)$, cu $\gamma = 0,5772156647$ numită constanta Euler.

Programul este dat în fig. 3.11, în care semnificația notațiilor este evidentă. Numărul H introdus reprezintă valoarea lui ε după care se controlează oprirea însumării. Astfel, dacă $|(-1)^i x^i / (i \cdot i!)| < \varepsilon$ se oprește însumarea.

Dacă, de exemplu, luăm $x = 0,5$, atunci $E_0(0,5) = 1,2130613$,

$E_1(0,5) = 0,5597736$, $E_2(0,5) = 0,32664386$, ...

$E_{10}(0,5) = 0,0634583$

2. Funcția

$$\alpha_n(x) = AN(x) = \int_1^{\infty} t^n e^{-xt} dt, \quad n = 0, 1, 2, \dots$$

folosește relația de recurență

$$\alpha_n(x) = [e^{-x} + n\alpha_{n-1}(x)]/x,$$

cu $\alpha_0(x) = e^{-x}/x$. Programul BASIC este dat în fig. 3.12, în care dacă dăm $x = 2$, $n = 6$ se obțin $\alpha_0(2) = 0,067667642$ și $\alpha_6(2) = 5,5994973$.

3. Funcția

$$\beta_n(x) = BN(X) = \int_{-1}^1 t^n e^{xt} dt, \quad n = 0, 1, 2, \dots$$


```

5 REM * CALCULUL FUNCTIEI *
      *      AN(X)      *
      *****
10 INPUT "INTRODUCE N=";N
20 INPUT "INTRODUCE X=";X
30 LET K=EXP (-X): LET A=K/X
40 IF N>0 THEN GO TO 50
45 PRINT "AN(X)=";A: GO TO 20
50 FOR I=1 TO N: LET A=(K+I*A)
/X
60 NEXT I: PRINT "AN(X)=";A
70 GO TO 20
80 STOP

```

Fig. 3.12. Integrala AN(X)

```

5 REM * CALCULUL FUNCTIEI *
      *      BN(X)      *
      *****
10 INPUT "INTRODUCE N=";N
20 INPUT "INTRODUCE X=";X
30 LET K=EXP (X): LET B=(K-1)
/X
40 IF N>0 THEN GO TO 50
45 PRINT "BN(X)=";B: GO TO 20
50 LET R=1: FOR I=1 TO N:
R=-R
60 LET B=(R*K-(1/K)+B)/X
70 NEXT I: PRINT "BN(X)="
80 GO TO 20
90 STOP

```

Fig. 3.13. Integrala BN(X)

se calculează după formula de recurență

$$\beta_n(x) = [(-1)^n e^x - e^{-x} + n\beta_{n-1}(x)]/x,$$

unde $\beta_0(x) = (e^x - e^{-x})/x$. Programul BASIC din fig. 3.13 va calcula pentru $n = 3$ și $x = 4$, valorile $\beta_0(4) = 13,644\ 959$, $\beta_1(4) = -10,242\ 877$, $\beta_3(4) = -7,2\ 614\ 763$.

3.4. CALCULUL FUNCȚIILOR BESSEL

O familie de funcții matematice de mare utilitate în inginerie o constituie funcțiile Bessel. Familia aceasta cuprinde: a) funcții de tipul unu, doi sau trei; b) funcții de ordin întreg, fracționar și neîntreg; c) funcții de tip regulat și modificat. Astfel de funcții sînt, de exemplu, funcțiile Kelvin, sau Riccati-Bessel. Funcțiile de tip unu

și ordin întreg sînt soluții ale ecuației diferențiale Bessel:

$$x^2 \frac{d^2 y}{dx^2} + x \frac{dy}{dx} + (x^2 - n^2) y = 0.$$

Această ecuație apare într-o mare varietate de probleme tehnice și științifice cum ar fi: ecuația coardei vibrante, ecuația transferului de căldură etc.

Pentru a fi programată și rezolvată funcția Bessel trebuie aranjată sub forma unei relații calculabile numeric. Una dintre aceste posibilități o constituie utilizarea dezvoltării în serie Taylor. Astfel, expresia Taylor pentru acest caz este:

$$J_n(x) = \left(\frac{x}{2}\right)^n \sum_{i=0}^{\infty} \left(\frac{-x^2}{4}\right)^i \left(\frac{1}{i!(n+i)!}\right).$$

Programul BASIC de rezolvare a problemei propuse este dat în fig. 3.14. În liniile 80÷110 se calculează primul termen al seriei, iar următorii și suma sînt determinați în instrucțiunile 120÷150. La începutul programului (liniile 20÷30) s-au introdus argumentul x_0 și ordinul n al funcției.

O altă modalitate de rezolvare o constituie găsire a unei relații de recurență. Aceasta este de forma:

$$J_{n+1}(x) = (2n/x) J_n(x) - J_{n-1}(x).$$

```

10 REM ** FUNCTIE BESSEL **
    * TIP 1 INTREG *
    ** RELATIE RECURENTA **
20 INPUT "ARGUMENT " : X0
30 INPUT "ORDINUL " : N
40 LET X=X0
50 IF ABS (X) < 1.E-10 THEN LET
X=1.E-10
60 LET Y=X
70 IF N>X THEN LET Y=N
80 LET N9=INT (Y+3*SQR (X)+9)
90 LET K3=0
100 LET K2=1.E-30
110 LET S=0
120 FOR I=N9 TO 0 STEP -1
130 LET K1=2*I/X*K2-K3
140 LET K3=K2
150 LET K2=K1
160 IF INT (I/2)=I/2 THEN LET S
=S+2*K3
170 IF I=N THEN LET J=K3
180 NEXT I
190 LET S=S-K3
200 LET J=J/S
210 PRINT "VALOARE CALCULATA=" :
J
220 STOP

```

Fig. 3.14. Funcție Bessel (formulă de recurență)

```

10 REM ** FUNCTIE BESSEL **
   *   TIP 1 INTREG   *
   ** SERIE TAYLOR  **
20 INPUT "ARGUMENT " :X0
30 INPUT "ORDINUL " :N
40 LET X=X0/2
50 LET X2=X*X
60 LET S=0
70 LET T=1
80 IF N=0 THEN GO TO 120
   90 FOR I=1 TO N
  100 LET T=X/I*T
  110 NEXT I
  120 FOR I=1 TO 999
  130 LET S=S+T
  140 LET T=-X2/I/(N+I)*T
  150 IF S(<)S+T THEN NEXT I
  160 PRINT "VALOARE FUNCTIE=";S
  170 STOP

```

Fig. 3.15. Funcției Bessel (serie Taylor)

Această relație este mai lentă decât dezvoltarea în serie Taylor pentru valori mici ale argumentului, dar este mai precisă pentru valori mari. În programul din fig. 3.15 valoarea argumentului x_0 și ordinul n sînt introduse prin liniile de program 20 ÷ 30. Bucla de recurență (liniile 120 ÷ 160) include normalizarea sumei la linia 150.

3.5. CALCULUL FUNCȚIEI LAPLACE

În teoria probabilităților funcției Laplace este o funcție de bază. Din cauză că aceasta nu poate fi descompusă în funcții elementare, ea a fost calculată și tabelată pentru diferite valori ale argumentului. Este irațional să introducem aceste valori în memoria calculatorului. De aceea este necesar să se calculeze funcția prin intermediul unui program (subprogram).

După cum se știe, expresia matematică a funcției Laplace este:

$$\Phi(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt,$$

unde x reprezintă valoarea curentă a variabilei aleatoare X , iar t este dat de expresia

$$t = \frac{x - m}{\sigma \sqrt{2}},$$

în care m este media variabilei X , iar σ — abaterea medie pătratică a lui X față de m .

Relația lui $\Phi(x)$ se evaluează prin diferite metode numerice.

Metoda dezvoltării în serie Taylor. Fiind dată o funcție $y = f(t)$, dezvoltarea sa în serie Taylor în jurul punctului de origine este

$$f(t) = f(0) + \frac{t}{1!} f'(0) + \frac{t^2}{2!} f''(0) + \dots + \\ + \frac{t^n}{n!} f^{(n)}(0) + \dots$$

Se consideră $f(t) = e^{-t^2}$; calculându-se derivatele acestei funcții în $t = 0$ și înlocuindu-le în dezvoltarea Taylor, se obține:

$$f(t) = 1 - \frac{t^2}{2!} \cdot 2 + \frac{t^4}{4!} \cdot 12 - \frac{t^6}{6!} \cdot 120 + \dots = \\ = 1 - \frac{t^2}{1} + \frac{t^4}{2} - \frac{t^6}{6} + \dots,$$

adică

$$\Phi(x) = \frac{2}{\sqrt{\pi}} \int_0^x \left(1 - \frac{t^2}{1} + \frac{t^4}{2} - \frac{t^6}{6} + \dots \right) dt = \\ = \frac{2}{\sqrt{\pi}} \left(\frac{x^1}{1 \cdot 0!} - \frac{x^3}{3 \cdot 1!} + \frac{x^5}{5 \cdot 2!} - \frac{x^7}{7 \cdot 3!} + \dots \right) = \\ = \frac{2}{\sqrt{\pi}} \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1) \cdot k!},$$

din care se observă că evaluarea lui $\Phi(x)$, pentru x dat, se reduce la calculul unei sume de termeni. Dacă se notează cu S valoarea sumei din paranteză și termenii cu U , atunci relația de recurență va fi:

$$S_{k+1} = S_k + U_{k+1}, \quad k = 0, 1, 2, \dots$$

unde

$$U_0 = x, \quad U_{k+1} = -U_k x^2 \cdot \frac{2k+1}{(2k+3)(k+1)},$$

$$k = 0, 1, 2, \dots; \quad S_0 = x.$$

```

10 REM ** FUNCTIA LAPLACE **      100 LET K=K+1
   *      SERIE TAYLOR      *      110 IF ABS (U))0.0001 THEN GO T
20 INPUT X                          O 80
30 IF X=0 THEN GO TO 140            120 LET S=S*2/SQR (3.1415)
40 IF X=3 THEN GO TO 160           130 GO TO 170
50 LET U=X                          140 LET S=0
60 LET S=X                          150 GO TO 170
70 LET K=0                          160 LET S=1
80 LET U=-U*X*X*(2*K+1)/(2*K+3     170 PRINT "VALOAREA FUNCTIEI=";
)/(K+1)                              S
90 LET S=S+U                        180 STOP

```

Fig. 3.16. Funcția Laplace (serie Taylor)

Consultându-se un tabel cu valorile funcției Laplace, se observă că $\Phi(0) = 0$ și $\Phi(x \geq 3) = 1$. Aceste două aspecte sînt cuprinse în programul BASIC din fig. 3.16. Programul se termină cînd ultimul termen, U_{k+1} , adunat la sumă este mai mic sau egal cu 0,0001. În acest caz, valoare S_{k+1} calculată se înmulțește cu $2/\sqrt{\pi}$ și se afișează rezultatul care este tocmai valoarea aproximativă a lui $\Phi(x)$ pentru x dat.

Metoda Simpson. Fie funcția $y = f(x)$ reprezentată în fig. 3.17, pentru care se calculează aria mărginită de curba $f(x)$, dreptele $x = a$, $x = b$, și axa Ox . În acest scop se va diviza intervalul $[a, b]$ în subintervale egale de lungime h . Ridicînd perpendiculare din punctele P_0, P_1, P_2, \dots se obțin, la intersecția cu $y = f(x)$, punctele M_0, M_1, M_2, \dots

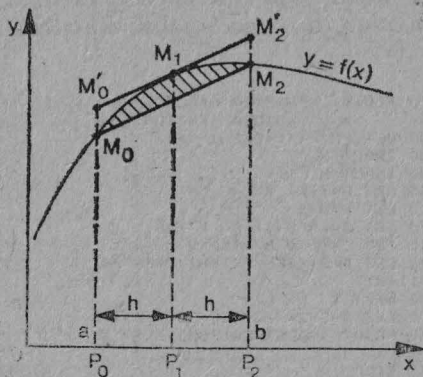


Fig. 3.17. Reprezentarea funcției

Din metoda lui Simpson se știe că aria se aproximează pentru n par și $[a, b] = [0, x]$ prin

$$f(x) dx = \frac{h}{3} (y_0 + 4y_1 + 2y_2 + 4y_3 + \dots + 2y_{n-1} + y_n),$$

unde $h = (b - a)/n = x/n$, $y_i = f(i \cdot h)$, $i = 0, 1, 2, \dots, n$.

Pentru funcția $f(t) = e^{-t^2}$ relația devine:

$$\Phi(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt = \frac{2}{\sqrt{\pi}} \cdot \frac{h}{3} \{f(0 \cdot h) + 4f(1 \cdot h) + 2f(2 \cdot h) + 4f(3 \cdot h) + 2f(4 \cdot h) + \dots + 2f[(n-2) \cdot h] + 4f[(n-1) \cdot h] + f(n \cdot h)\}.$$

Expresia din paranteză poate fi calculată recursiv cu relația:

$$S_{i+2} = S_{i+1} + 4f(i+h) + 2f[(i+1) \cdot h],$$

pentru $i = 1, 3, 5, \dots$ și $S_1 = f(0 \cdot h) - f(n \cdot h)$.

Programul BASIC corespunzător acestei metode este dat în fig. 3.18. Se observă că la începutul programului s-a definit funcția de calculat (linia 20), după care s-au introdus valorile X și N .

Față de metoda anterioară, metoda Simpson este mai puțin precisă, deoarece ciclul de calcul este dat numai de

```

10 REM ** FUNCTIA LAPLACE **
   *   *   *   *   *   *   *
   *   *   *   *   *   *   *
20 DEF FN F(T)=EXP (-T^2)
30 INPUT X
40 INPUT N
50 LET H=X/N
60 LET M=N-1
70 LET S=FN F(0)-FN F(N*H)
80 FOR I=1 TO M STEP 2
90 LET S=S+4*FN F(I*H)+2*FN F(
(I+1)*H)
100 NEXT I
110 LET S=S*H/3/SQR (3.1415)
120 PRINT "VALOAREA FUNCTIEI=";
S
130 STOP

```

Fig. 3.18. Funcția Laplace (metoda Simpson)

numărul de intervale h în care s-a împărțit $[0, x]$ și nu de un ε definit pentru termenii sumei. De aceea, pentru calcule mai precise se recomandă prima metodă.

3.6. ANALIZA DE REGRESIE

Regresia este o metodă de cercetare a unei relații pre-determinate, exprimând legătura ce există între o variabilă, y , numită variabilă *dependentă* (explicată, endogenă sau rezultativă) și una sau mai multe variabile, x_1, x_2, \dots , numite variabile *independente* (explicative, exogene, de influență). Fie y o variabilă dependentă de x_1, x_2, \dots , relația exactă sau *ecuația de regresie* a lui y în funcție de x_1, x_2, \dots , pusă sub forma $y = f(x_1, x_2, \dots)$, definește o *curbă* sau o *suprafață de regresie*; ea are menirea să permită, pentru valorile date x_1, x_2, \dots , calcul unei estimății a lui y .

Calculul elementar de aplicare a metodei regresiei îl constituie legătura dintre două variabile, $y = f(x)$.

Regresia liniară. Dependența liniară este un model determinist și nu reflectă exact legătura dintre y și x . Valorile observate (x_i și y_i) nu se găsesc exact pe dreapta

$$y(x) = b_0 + b_1x$$

și, de fapt, unei valori x_i îi pot corespunde mai multe valori y_i (fig. 3.19). Parametrii b_0 și b_1 se estimează prin *metoda celor mai mici pătrate* sau în *sensul celor mai mici pătrate*, adică în așa fel încât să se facă minimă suma pă-

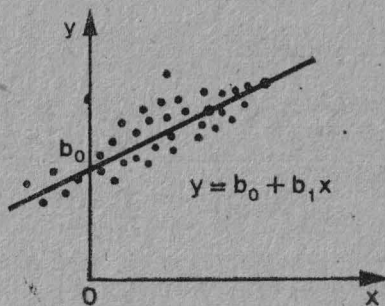


Fig. 3.19. Regresie liniară

tratelor abaterilor între punctele observate și punctele corespunzătoare ale dreptei. Dispunînd de o serie de valori observate (x_i, y_i) , suma pătratelor abaterilor de minimizat este:

$$S = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y_i - y(x_i))^2 = \sum_{i=1}^n (y_i - b_0 - b_1 x_i)^2$$

Valorile x_i și y_i fiind cunoscute, această sumă este în funcție numai de parametrii b_0 și b_1 . Anulînd derivatele parțiale în raport cu b_0 și b_1 ,

$$\frac{\partial S}{\partial b_0} = \frac{\partial S}{\partial b_1} = 0,$$

se obține sistemul de ecuații normale:

$$\begin{cases} \sum_{i=1}^n (y_i - b_0 - b_1 x_i) = 0 \\ \sum_{i=1}^n x_i (y_i - b_0 - b_1 x_i) = 0 \end{cases}$$

sau

$$b_0 n + b_1 \sum_{i=1}^n x_i = \sum_{i=1}^n y_i$$

$$b_0 \sum_{i=1}^n x_i + b_1 \sum_{i=1}^n x_i^2 = \sum_{i=1}^n x_i y_i,$$

care are soluțiile:

$$b_1 = \frac{\sum_{i=1}^n x_i \sum_{i=1}^n y_i - n \sum_{i=1}^n x_i y_i}{\left(\sum_{i=1}^n x_i\right)^2 - n \sum_{i=1}^n x_i^2},$$

$$b_0 = \frac{1}{n} \left(\sum_{i=1}^n y_i - b_1 \sum_{i=1}^n x_i \right).$$


```

3 PRINT "REGRESIE LINIARA"
10 INPUT "N=";N
20 LET A=0: LET B=0
30 LET C=0: LET D=0
40 PRINT "INTRODUCE X(I),Y(I)"
50 FOR I=1 TO N: PRINT "I=";I
60 INPUT "X(I)=";X: INPUT "Y(I)";Y
70 LET A=A+X: LET B=B+Y
80 LET C=C+X*X: LET D=D+Y*X
90 NEXT I
100 LET B1=(A*B-N*D)/(A*A-N*C)
110 LET B0=(B-B1*A)/N
120 PRINT "Y=";B0;"+";B1;"*";X
130 INPUT "INTRODUCE X=";X
140 PRINT "Y(X)=";B0+B1*X
150 GO TO 130
160 STOP

```

Fig. 3.20. Program de estimare liniară

Programul de estimare liniară este dat în fig. 3.20. Dacă de exemplu, $x_i = (2, 4, 6, 8, 10)$, $y_i = (5,5; 6,3; 7,2; 8,8; 8,6)$, atunci coeficienții b_0 și b_1 au valorile 4,75 și, respectiv, 0,395, iar dreapta de regresie este $y(x) = 4,75 + 0,395x$.

Regresie hiperbolică. Legătura dintre variabila dependentă y , și cea de influență, x , este de forma:

$$y(x) = b_0 + b_1/x.$$

Hiperbola, ca și parabola, este specifică exprimării dependențelor cu o alură ce tinde către un punct maxim (minim); dacă acest punct este depășit curbele sau capătă stabilitate sau descresc (cresc).

Procedînd în mod similar regresiei liniare, se obține sistemul de ecuații normale

$$\begin{cases} b_0 n + b_1 \sum_{i=1}^n (1/x_i) = \sum_{i=1}^n y_i; \\ b_0 \sum_{i=1}^n (1/x_i) + b_1 \sum_{i=1}^n (1/x_i^2) = \sum_{i=1}^n (y_i/x_i). \end{cases}$$

Soluția sistemului se obține ușor, iar programul de determinare a regresiei este dat în fig. 3.21. Pentru $n = 8$ și $x_i = (1, 2, 3, 4, 5, 6, 7, 8)$, $y_i = (12,2; 6,8; 5,2; 4,6; 3,9; 3,7; 3,5; 3,2)$ se obțin $b_0 = 1,9357619$; $b_1 = 10,160175$. Luînd $x = 2$, atunci $y = 7,0158495$.

Regresia cu funcția putere. Funcția putere (fig. 3.22) $y(x) = b_0 x^{b_1}$, implică un ritm de variație constant, de unde

```

5 PRINT "REGRESIE HIPERBOLICA
10 INPUT "N=";N
20 LET A=0: LET B=0
30 LET C=0: LET D=0
40 PRINT "INTRODUCE X(I),Y(I)"
50 FOR I=1 TO N: PRINT "I=";I
60 INPUT "X(I)=";X: INPUT "Y(I)
)=";Y
70 LET A=A+1/X: LET B=B+1/(X*X
80 LET C=C+Y: LET D=D+Y/X
90 NEXT I
100 LET E=N*B-A*A: LET F=C*B-D*N
A
110 LET K=N*D-A*C: LET F=F/E
120 LET K=K/E
130 PRINT "Y(X)=";F;"+";K;" /X"
140 INPUT "INTRODUCE X=";X
150 PRINT "Y(X)=";F+K/X
160 GO TO 140
170 STOP

```

Fig. 3.21. Program de estimare hiperbolică

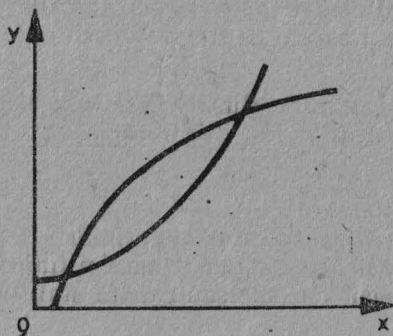


Fig. 3.22. Funcția putere

și valabilitatea ei pentru fenomene cu evoluție continuă, readmițînd — cel puțin în perioada de analiză și/sau pe-revizuire — nivel de saturație.

Construind în mod similar sistemul $\frac{\partial S}{\partial b_0} = \frac{\partial S}{\partial b_1} = 0$ și rezolvîndu-l, se obține soluția:

$$b_1 = \frac{\sum_{i=1}^n \ln x_i \sum_{i=1}^n \ln y_i - n \sum_{i=1}^n \ln x_i \ln y_i}{\left(\sum_{i=1}^n \ln x_i \right)^2 - n \sum_{i=1}^n (\ln x_i)^2},$$

$$b_0 = \exp \left[\frac{1}{n} \left(\sum_{i=1}^n \ln y_i - b_1 \sum_{i=1}^n \ln x_i \right) \right],$$

```

5 PRINT "REGRESIE CU FUNCTIE
PUTERE"
10 INPUT "N=";N
20 LET A=0: LET B=0
30 LET C=0: LET D=0
40 PRINT "INTRODUCE X(I),Y(I)"
50 FOR I=1 TO N: PRINT "I=";I
60 INPUT "X(I)=";IX: INPUT "Y(I)
)=";Y
70 LET A=A+LN (X): LET B=B+LN
(Y)
80 LET C=C+LN (X)*LN (Y)
90 LET D=D+X*LN (Y)
100 NEXT I
110 LET B1=(A*B-N*D)/(A*A-N*C)
120 LET B0=EXP ((B-B1*A)/N)
130 PRINT "Y(X)=";B0;"*";"EXP :
";B1;"*X)"
140 INPUT "INTRODUCE X=";X
150 PRINT "Y(X)=";B0+EXP (B1*X)
160 GO TO 140
170 STOP

```

Fig. 3.23. Program de estimare ca funcția putere

iar programul BASIC este dat în fig. 3.23. Execuția acestuia pentru $n = 6$, $x_i = (1, 2, 3, 4, 5, 6)$, $y_i = (3, 12, 27, 48, 75, 108)$ conduce la $b_0 = 3$, $b_1 = 2$, iar pentru $x = 2$ se obține $y = 12$.

Regresia exponențială. Una dintre formele posibile, (fig. 3.24), $y(x) = ab^x$, este potrivită când variabila de influență crește în progresie aritmetică, iar cea dependentă în progresie geometrică.

Sistemul de ecuații normale este:

$$n \lg a + \lg b \sum_{i=1}^n x_i = \sum_{i=1}^n \lg y_i,$$

$$\lg a \sum_{i=1}^n x_i + \lg b \sum_{i=1}^n x_i^2 = \sum_{i=1}^n (x_i \lg y_i)$$

iar programul este dat în fig. 3.25.

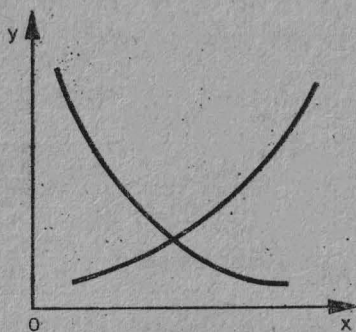


Fig. 3.24. Funcția exponențială $y = ab^x$

```

5 PRINT "REGRESIE
      EXPONENTIALA 1"
10 INPUT "N=";N
20 LET A=0: LET B=0
30 LET C=0: LET D=0
40 PRINT "INTRODUCE X(I),Y(I)"
50 FOR I=1 TO N: PRINT "I=";I
60 INPUT "X(I)=";X: INPUT "Y(I)
)=";Y
70 LET A=A+X: LET B=B+X*X
80 LET C=C+LN (Y)/LN (10)
90 LET D=D+X*LN (Y)/LN (10)
100 NEXT I
110 LET E=N*B-A*A
120 LET F=C*B-D*A
125 LET K=N*D-A*C
127 LET F=10^(F/E)
128 LET K=10^(K/E)
130 PRINT "Y(X)=";F;"*";K;"^";X
140 INPUT "INTRODUCE X=";X
150 PRINT "Y(X)=";F*K^X
160 GO TO 140
170 STOP

```

Fig. 3.25. Program pentru o estimare exponențială

Considerînd $n = 5$, $x_i = (1, 2, 3, 4, 5)$, $y_i = (6; 7; 8; 7; 10; 4; 12; 4)$, se obțin $a = 4,94 191$ și $y = 1,2 029 501$. Dacă $x = 2$, atunci $y = 7,1 514 993$.

Regresie exponențială. Forma funcțională

$$y(x) = b_0 \exp(b_1 x)$$

conduce la formulele

$$b_1 = \frac{\sum_{i=1}^n x_i \sum_{i=1}^n \ln y_i - n \sum_{i=1}^n x_i \ln y_i}{\left(\sum_{i=1}^n x_i\right)^2 - n \sum_{i=1}^n x_i^2}$$

$$b_0 = \exp \left[\frac{1}{n} \left(\sum_{i=1}^n \ln y_i - b_1 \sum_{i=1}^n x_i \right) \right]$$

și la programul din fig. 3.26.

Dacă $n = 9$, $x_i = (2, 3, 4, 5, 6, 7, 8, 9, 10)$, $y_i = (3,5; 5; 6; 2; 9; 13; 16; 23; 30; 40)$, atunci $b_0 = 1,9 394 813$, $b_1 = 0,30 528 331$. În aceste condiții dacă

```

5 PRINT "REGRESIE
      EXPONENTIALA 2"
10 INPUT "N=";N
20 LET A=0: LET B=0
30 LET C=0: LET D=0
40 PRINT "INTRODUCE X(I),Y(I)"
50 FOR I=1 TO N: PRINT "I=";I
60 INPUT "X(I)=";X: INPUT "Y(I)
)=";Y
70 LET A=A+X: LET B=B+LN (Y)
80 LET C=C+X*X
90 LET D=D+X*LN (Y)
100 NEXT I
110 LET B1=(A*B-N*D)/(A*A-N*C)
120 LET B0=EXP ((B-B1*A)/N)
130 PRINT "Y(X)=";B0;"*";X;"^";B1;"*";X
140 INPUT "INTRODUCE X=";X
150 PRINT "Y(X)=";B0*EXP (B1*X)
160 GO TO 140
170 STOP

```

Fig. 3.26. Program pentru regresie exponențială

în expresia $y = 1,9394813 \text{ EXP}(0,30528331x)$ se ia $x = 0$, rezultă $y = 2,9394813$.

Regresie parabolică. Legătura celor două variabile este

$$y(x) = b_0 + b_1x + b_2x^2.$$

Procedînd ca mai înainte, se obține sistemul de ecuații normale:

$$\begin{cases} b_0N + b_1 \sum_{i=1}^n x_i + b_2 \sum_{i=1}^n x_i^2 = \sum_{i=1}^n y_i; \\ b_0 \sum_{i=1}^n x_i + b_1 \sum_{i=1}^n x_i^2 + b_2 \sum_{i=1}^n x_i^3 = \sum_{i=1}^n x_i y_i; \\ b_0 \sum_{i=1}^n x_i^2 + b_1 \sum_{i=1}^n x_i^3 + b_2 \sum_{i=1}^n x_i^4 = \sum_{i=1}^n x_i^2 y_i. \end{cases}$$

Programul BASIC corespunzător este dat în fig. 3.27. Rulînd pentru $n = 7$, $x_i = (2; 4; 6; 8; 10; 12; 14)$, $y_i = (3,76; 4,44; 5,04; 5,56; 6; 6,36; 6,34)$, se obțin $B_0 = 2,8714287$, $B_1 = 0,45535711$; $B_2 = -0,014464284$. Dacă $x = 7$, atunci $y = 5,3501785$.

Regresia polinomială. Această legătură între variabilele x și y generalizează cazul precedent, adică

$$y(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n,$$

```

5 PRINT "REGRESIE PARABOLICA"
10 INPUT "INTRODUCERE N=":N
20 LET A=N
30 DATA 0.0.0.0.0.0.0
40 READ B,C,F,M,P,R,S
50 PRINT "INTRODUCE X(I).Y(I)"
60 FOR I=1 TO N: PRINT "I=":I
70 INPUT "X(I)=":X: INPUT "Y(I)
)=":Y
80 LET B=B+X: LET C=C+X*X: LET
F=F+X*X*X
90 LET M=M+X^4: LET P=P+Y: LET
R=R+X*Y
100 LET S=S+Y*X*X: NEXT I
110 LET D=B: LET E=C: LET K=C:
X
LET L=F
120 LET Q=D/A: LET E=E-Q*B: LET
F=F-Q*C
130 LET R=R-Q*P: LET Q=K/A: LET
L=L-Q*B
140 LET M=M-Q*M: LET S=S-Q*P: L
ET Q=L/E
150 LET B2=(S-R*Q)/(M-F*Q): LET
B1=(R-F*B2)/E
160 LET B0=(P-B*B1-C*B2)/A
170 PRINT "B0=":B0: PRINT "B1="
:B1: PRINT "B2=":B2
180 INPUT "INTRODUCE X=":X
190 PRINT "Y(X)=":B0+B1*X+B2*X*X
200 STOP

```

Fig. 3.27. Regresie parabolică

care conduce la sistemul:

$$\begin{cases} c_0 x_0 + c_1 a_1 + c_2 a_2 + \dots + c_m a_m = d_0, \\ c_1 a_0 + c_2 a_1 + c_3 a_2 + \dots + c_{m+1} a_m = d_1, \\ \dots \\ c_m a_0 + c_{m+1} a_1 + c_{m+2} a_2 + \dots + c_{2m} a_m = d_m \end{cases}$$

unde $c_j = \sum_{i=1}^m x_i^j$, $j = 0, 1, 2, \dots, 2m$, iar $d_k = \sum_{i=1}^n x_i^k y_i$,

$k = 0, 1, 2, \dots, m$.

O asemenea ajustare are ca rezultat o tendință neliniară.

Extrapolarea. Operația de extrapolare constă în stabilirea valorii unor termeni ai seriei cronologice de date care se situează în afara orizontului de analiză. Extrapolarea presupune: a) adoptarea unui model de evoluție, $y_t = f(t)$; b) introducerea în model a valorii convenționale a variabilei timp corespunzătoare momentului pentru care se efectuează extrapolarea. Extrapolarea devine posibilă după ce a fost determinată *tendința generală* a seriei de date cu unul din procedeele de ajustare anterioare; în acest caz, tendința este extrapolată pentru un orizont oarecare.

INSTRUIRE ASISTATĂ DE CALCULATOR

Instruirea asistată de calculator (CAI — Computer Assisted Instruction) este cel mai obișnuit termen folosit în descrierea utilizării calculatoarelor în scopuri educaționale. În multe țări utilizarea calculatoarelor și a tehnologiilor informatice a intrat în practica școlară, deschizând noi orizonturi procesului de instruire, iar acest fapt a fost în mod nemijlocit legat de apariția și dezvoltarea calculatoarelor personale. Să nu uităm că cei care au inventat calculatorul personal l-au dedicat în primul rînd scopurilor educative. În ultimul timp au fost efectuate numeroase studii comparative și evaluative referitoare la folosirea calculatoarelor personale în școală, iar în multe țări sînt în curs de desfășurare programe naționale de introducere a tehnologiilor informatice în procesul de învățămînt.

Iată cum se prezenta situația în domeniul CAI la mijlocul deceniului '80 în unele țări: ■

— în S.U.A. și Japonia, $\frac{2}{3}$ din școlile elementare și $\frac{1}{3}$ din cele secundare sînt posesoare de calculatoare personale pe care le folosesc nemijlocit în procesul de instruire;

— în școlile din S.U.A., Anglia, Japonia și Franța au fost inițiate proiecte naționale experimentale legate de instruirea asistată de calculator;

— în China este în curs de experimentare utilizarea calculatoarelor în ciclul primar și gimnazial;

— în Ungaria, Polonia și Bulgaria, învățămîntul beneficiază de mii de calculatoare, fiind inițiate de asemenea, experimente pe plan național etc.

4.1. TIPURI DE PROGRAME PENTRU INSTRUIRE

Există mai multe tipuri de instruire asistată de calculator care se referă în special la *tipurile de programe* pentru instruire: formarea aptitudinilor, preparatoare, simulatoare, rezolvarea de probleme etc. Trebuie menționat faptul că un program utilizat pentru instruire cu calculatorul poate fi realizat prin combinarea mai multor tipuri (tehnici) menționate. De exemplu, un program preparator, include în mod tipic exerciții pentru formarea aptitudinilor după ce o nouă noțiune a fost introdusă.

Includerea acestor programe în procesul de instruire implică, de fapt integrarea informaticii în procesul de învățămînt. Pe lângă această tendință, care este dominantă, mai există și alta legată de introducerea disciplinei informatice ca obiect de învățămînt, iar aceasta include atît noțiuni legate de calculatoare (*computer literacy*), cît și asimilarea unuia sau mai multor limbaje de programare (în funcție de vîrstă). De obicei se practică învățarea utilizării calculatoarelor prin intermediul limbajelor BASIC, LOGO, PASCAL.

Programele pentru formare de aptitudini (drill and practice exercises) permit celor care învață să vină în contact cu fapte, relații, probleme și vocabular pînă ce materialul a fost asimilat sau pînă ce o aptitudine a fost formată. Ele includ exerciții și practică.

Cele mai bune programe pentru formarea aptitudinilor folosesc formate interesante, care încurajează reușita elevului și stabilesc un stimulent pentru răspunsul pozitiv asociat.

Cu programele pentru formarea aptitudinilor poate fi atins orice subiect, cîteva domenii (materii) oferind totuși un mediu mai prielnic dezvoltării acestor programe. Astfel, acestea pot fi eficiente în aria dezvoltării vorbirii, calculatorul avînd posibilitatea de a realiza legătura între cuvînt și înțelesul său. De exemplu, calculatorul afișează o definiție, iar elevul tipărește cuvîntul corespunzător, procesul putînd fi repetat pînă cînd asocierile între cuvinte și definițiile lor sînt complete.

Alte domenii în care se recomandă folosirea programelor pentru formarea de aptitudini sînt: matematica, învățarea limbilor străine, expresii și construcții gramaticale, muzică (recunoașterea notelor, ritmurilor, termenilor muzicali) etc.

Este de remarcant faptul că în sistemul de instruire din S.U.A. programele pentru formarea de aptitudini ocupă 18% din timpul utilizat în instruirea asistată de calculator de către elevi.

Programe preparatoare sau tutoriale (tutorial) utilizează explicații scrise, descrieri, întrebări, probleme și ilustrații grafice pentru conceptele dezvoltate, în același mod în care un profesor „preparator” pregătește un elev. Deseori sînt utilizate preteste pentru a determina cel mai normal început de lecție sau cum se poate trece peste anumite lecții. După ce o noțiune (segment de lecție sau lecție) a fost prezentată este oferit, de obicei, și un exercițiu tip formare de aptitudini. În anumite cazuri, elevul are întregul control asupra programului.

În final un post test (pentru fiecare obiect sau grup de obiecte) va determina nivelul atins de către acel elev. Nota elevului poate fi afișată la sfîrșitul lecției, la fel ca și alte sugestii privitoare la studiu și/sau practică.

Autorul programului preparator trebuie să prevadă toate răspunsurile corecte și posibile și să permită (în unele cazuri) greșeli de ortografie ne semnificative (care pot fi puse pe seama tastării, de exemplu). Programul trebuie să răspundă inteligent la răspunsuri incorecte, să prevadă cele mai obișnuite răspunsuri incorecte și să ofere explicații special concepute în cazul unui răspuns incorect.

Există mai multe tipuri de programe preparatoare. Cele care progresează într-un mod liniar prezintă o serie de ecrane, fără a diferenția elevii. Răspunsurile incorecte pot servi la trecerea într-o secvență de recitare. Programele preparatoare branșate, pe de altă parte, nu cer tuturor utilizatorilor să urmeze același model, dar îi îndrumă spre anumite lecții sau părți de lecții după rezultatele pretestelor și posttestelor.

În dezvoltarea programelor preparatoare trebuie respectate mai multe principii, printre care :

— conceptele să fie dezvoltate într-o manieră secvențială, care să nu producă confuzii asupra celui care învață ;

— instruirea să fie completată cu grafică și sunete, în așa fel încît programul să capteze atenția și să mențină interesul elevului ;

— pretestele și posttestele să fie valide și să măsoare cu acuratețe progresele elevului.

Programele de simulare permit experimentarea unor situații, care ar fi dificil sau imposibil de realizat în clasă.

Simulatoarele asigură simularea unor situații, modelele, în care rezultatele finale să fie obținute din deciziile proprii ale utilizatorului. Ghidați după datele furnizate de simulator, elevii selectează anumite opțiuni sau aleg anumite situații, apoi obțin rezultatele deciziilor.

Simulările pot fi mai efective atunci cînd sînt utilizate pentru a ilustra idei și experimente explorate în prealabil prin alte mijloace — idei, texte, chestionare, discuții etc. Aceste programe pot fi de asemenea utilizate în procesul de instruire a elevilor în laboratoarele de fizică sau chimie, în simularea unor experiențe chimice sau fizice. Este totodată posibilă simularea experimentelor care sînt prea costisitoare, complicate sau mari consumatoare de timp. Simulările determină acumularea unei experiențe participative față de cea obținută prin simpla lecturare a experiențelor. Simulările pot fi, de asemenea utilizate la antrenament, în operarea diferitelor tipuri de echipamente. Prin aceasta elevii vor putea practica și învăța operarea echipamentelor respective fără riscuri pentru ei sau pentru echipament.

O utilizare în creștere o au *simulatoarele grafice* prin care se simulează grafic diverse procese și fenomene, păstrîndu-se totuși, interacțiunea dintre calculator și elev.

Programele de tipul „*rezolvitoare de probleme*” necesită o anumită strategie din partea elevului. Așa cum sugerează, denumirea, o problemă este prezentată, iar utilizatorul încearcă să o soluționeze. Elevul poate învăța din greșeli și cîștiga în dezvoltarea aptitudinilor pentru rezolvarea problemelor. În acest proces elevul învață să gîndească să

exploreze problemele și să nu dea răspunsuri pur și simplu. Unele programe de rezolvare de probleme pot fi descrise ca „jocuri logice educaționale“.

Programele pentru rezolvare de probleme promovează dezvoltarea modelelor de gândire sistematică, permițând în același timp elevilor lucrul practic într-un mod diferit față de programele tradiționale destinate formării de aptitudini și deprinderi.

4.2. PROGRAM PENTRU ÎNVĂȚAREA LIMBII ENGLEZE

Programul prezentat reprezintă un exemplu clasic de program pentru formare de aptitudini, și anume pentru învățarea limbii engleze de către copiii începători. Principiul funcționării se bazează pe memorare: pe ecran se afișează un timp scurt (în funcție de mărimea textului) câte o propoziție, apoi în mod aleator va dispărea câte un cuvânt pe care elevul trebuie să îl introducă prin tastare.

Mai multe principii de instruire asistată au fost respectate în proiectarea programului, printre care:

- interactivitatea;
- învățarea este neimpusă, desfășurându-se sub forma unui joc;
- ajutorul se acordă atunci când elevul îl cere, dar este posibilă și acordarea de calificative (prin punctaj și clasament);
- folosirea opțiunilor: niveluri diferite ale utilizatorilor (1 — începători, 2 — mediu, 3 — avansați), precum și listă de opțiuni pentru ajutor (1 — pentru indicarea următoarei litere, 2 — pentru indicarea cuvântului întreg, 3 — trecerea la altă propoziție, 4 — modificarea nivelului);
- posibilitatea modificării datelor de către profesor: propozițiile care se vor afișa pe ecran se găsesc între liniile 1 000 și 3 000.

Programul din fig. 4.1 este scris în BASIC și poate fi îmbunătățit substanțial (păstrându-se, bineînțeles, principiile menționate) prin intermediul unor subrutine în cod mașină (de exemplu, pentru efecte sonore, grafică etc.).

```

7 DIM c(10): DIM a$(10,13): F
OR i=1 TO 10: LET a$(i)="Spectru
m": NEXT i
10 BORDER 2: PAPER 5: INK 9: C
LS : LET l=0: LET scor=0: POKE 2
3609,40: LET b=0: LET a=1: LET c
u=1
20 PRINT "Program pentru invat
area limbii ENGLEZE"
'''TAB 6:"Nivel de cunostinte"
'''1.....incepatori. '''2...
.....nivel mediu. '''3.....
..avansati. ''' INK 2: PAPER 5:
"Aposati una din tastele (1 2 3)
"
40 LET a%=INKEY$: IF a$="" THE
N GO TO 40
45 BEEP .01,10: BEEP .02,20
50 IF a$="1" THEN LET lev=1: G
O TO 100
60 IF a$="2" THEN LET lev=2: G
O TO 100
70 IF a$="3" THEN LET lev=3: G
O TO 100
80 GO TO 40
100 IF lev<=3 AND lev>=1 THEN R
ESTORE lev*1000: GO TO 500
500 DIM c$(100,15): DIM n$(60):
DIM b(20): CLS : PRINT AT 8,5:"
Asteptati un moment !"
505 FOR a=0 TO 6: POKE USR "a"+
a,0: NEXT a: POKE USR "a"+7,126
510 REM
520 PRINT AT 17,1:"Scor":AT 17
,6:scor
525 PRINT AT 18,0:" [1]-pentru
urmatoarea litera [2]-pentru
tot cuvintul [3]-pentru
alta propozitie [4]-pentru
alt nivel."
530 PRINT AT 17,17:"Nivel":lev
535 LET b=0: LET a=1: LET cu=i
540 READ p$: IF p$="end" THEN G
O TO 9000
542 IF p$="lev 2" THEN LET lev=
2: PRINT #1:" Se trece la nivel
ul nr. 2
545 PRINT AT 17,17:"Aposati o tas
ta! " : PAUSE 0: PRINT #
1:"
" : CLS : GO TO 520
543 IF p$="lev 3" THEN LET lev=
3: PRINT #1:" Se trece la nivel
ul nr. 3
545 PRINT AT 17,17:"Aposati o tas
ta! " : PAUSE 0: PRINT #
1:"
" : CLS : GO TO 520

```

```

545 GO SUB 900: READ a: FOR h=1
TO a-1: READ b(h): NEXT h
550 PRINT AT 0,0:p$: FOR i=1 TO
LEN p$: LET b=b+1
560 LET a=p$(i): IF a$="" OR
a$="." OR a$="," OR a$="?" OR a$
="!" OR a$=":" OR a$=";" THEN LE
T c$(cu)=p$(a TO a+b-1): LET a=i
+1: LET cu=cu+1: LET b=0
570 LET a=p$(i): IF i<LEN p$
THEN IF (a$=" " OR a$="," OR a$=
"." OR a$="?" OR a$="!" OR a$=":
" OR a$=";" AND p$(i+1)=" " THE
N LET i=i+1: LET a=i+1: GO TO 57
0
580 NEXT i: LET cuv=cu-1: PRINT
AT 8,0:"
" : PAUSE 250*lev: FOR a
=1 TO cuv: PRINT AT 13,0:"
590 LET a=INT ((RND*cuv)+1): IF
a<=0 OR a>=cuv+1 THEN GO TO 590
593 LET s=c$(a): LET u=1
595 IF s$(u)<>" " AND s$(u)<>"
" AND s$(u)<>"." AND s$(u)<>"
" AND s$(u)<>"?" AND s$(u)<>"!" AN
D s$(u)<>"?" THEN LET u=u+1: GO
TO 595
596 LET s=s$( TO u-1)
600 LET c=a-1: FOR k=1 TO a-1:
LET v=c$(k): LET r=1
604 IF v$(r)<>" " AND v$(r)<>"
" AND v$(r)<>"." AND v$(r)<>"
" AND v$(r)<>"?" AND v$(r)<>"!" AN
D v$(r)<>"?" THEN LET c=c+1: LET
r=r+1: GO TO 604
609 NEXT k: LET c=c+1: LET j=1:
LET h=1
610 IF c+b(h)>32 THEN LET c=c-3
2+b(h): LET j=j+1: LET h=h+1: GO
TO 610
620 FOR x=1 TO LEN s$: PRINT AT
j-1,c+x-21"A": NEXT x: LET x=x
-1
630 PRINT AT 12,0:"Tastati cuvi
ntul care lipseste": LET z=0
640 LET a%=INKEY$: IF a$="" THE
N GO TO 640
645 BEEP .01,20: BEEP .007,40
650 LET b%=s$(z+1): GO TO 600
670 IF a%=b% THEN LET scor=scor
+5: LET z=z+1: PRINT AT j-1,c+z-
2:b%:AT 14,2+z:a%:AT 17,6:scor:
GO TO 680
675 LET scor=scor-1: PRINT AT 1
7,6:scor
680 IF z=x OR a$="3" THEN GO TO
700
690 GO TO 640
700 PAUSE 50: PRINT AT 12,0:"

```

Fig. 4.1. (Continuă la pag. 83)

```

": NEXT a: GO TO 530
800 IF a$="1" THEN LET z=z+1: L
ET scor=scor-2: PRINT AT j-1,c+z
-2:b$:AT 14.z+2:b$:AT 17.6:scor:
IF lev=1 THEN GO TO 670
810 IF a$="2" THEN FOR w=z TO x
-1: LET b$=s$(w+1): PRINT AT j-1
,c+w-1:b$:AT 14.3+w:b$: NEXT w:
LET z=x: LET scor=scor-15: PRINT
AT 17.6:scor: IF lev=1 THEN GO
TO 670
820 IF a$="3" THEN LET a=cuv: I
F lev=1 THEN GO TO 670
830 IF a$="4" THEN GO TO 9900
840 GO TO 670
900 FOR m=0 TO 10: PRINT AT m,0
:

```

```

": NEXT m: RETURN
1000 DATA "I read a book.",1."I
go to school.",1
1900 DATA "lev 2"
2000 DATA "Every Monday mother a
oes shopping.",2.7,"She f
ook a bus to get to the shops
quikly.",2,3
2900 DATA "lev 3"
3000 DATA "Yesterday was Sunday.
Dan and Doris were busy in th
e kitchen all morning.Doris was
the cook and Dan was her help.
",4,3,1,1
9000 CLS : PRINT AT 1,9:" Clasam
ent "
9010 FOR i=1 TO 10: IF scor=c(i
) THEN LET l=i: LET i=10
9012 NEXT i
9013 IF l=0 THEN GO TO 9040
9015 FOR i=9 TO 1 STEP -1. LET c

```

```

(i+1)=c(i): LET a$(i+1)=a$(i): N
EXT i
9020 LET c(1)=scor
9030 INPUT "Tastati numele dv.(m
ax 13 lit.)":a$(1)
9040 FOR i=1 TO 10
9050 PRINT AT 2*i+1,3:i:TAB 6:a$
(i):TAB 22:"-> ":c(i)
9060 NEXT i
9070 PRINT #0:"Apasati o tasta p
t. a reincepe.": PAUSE 0
9080 GO TO 10
9900 PRINT AT 18,0:" [7]-pentru
un nivel superior [8]-pentru
un nivel inferior [9]-pentru
sfirsit.

```

```

9910 LET a$=INKEY$: IF a$="" THE
N GO TO 9910
9920 IF a$="7" THEN LET lev=lev+
1: IF lev>=3 THEN LET lev=3: LET
a=cuv
9930 IF a$="8" THEN LET lev=lev-
1: IF lev<=1 THEN LET lev=1: LET
a=cuv
9940 IF a$="9" THEN GO TO 9000
9945 IF a$("<"7" AND a$("<"8" AND
a$("<"9" THEN GO TO 9910
9950 RESTORE lev*1000
9960 PRINT AT 12,0:"

```

llc

```

9970 GO TO 520
9998 DATA "end"
9999 CLEAR : SAVE "Lb.Engleza"
INE 0

```

Fig. 4.1. Program de învățare a limbii engleze

PROGRAME UTILITARE ȘI GRAFICĂ

**5.1. RUTINĂ DE CITIRE DIN MEMORIA ROM
PENTRU OBTINEREA INFORMAȚIILOR
REFERITOARE LA FIȘIERELE SALVATE**

Informațiile referitoare la un fișier salvat pe caseta magnetică (tipul fișierului, lungimea sa etc.) se regăsesc în partea de antet (*header*) care însoțește fișierul. Într-adevăr, fișierele sînt separate în două părți, fiecare avînd un format diferit. Prima parte sau antetul fișierului conține diverse informații relative la fișier. Ea ocupă 18 octeți organizați în felul următor:

— octetul 0 (zero) indică tipul fișierului și poate avea valorile: 0 — pentru program BASIC; 1 — fișier de date numerice; 2 — fișier de date alfanumerice; 3 — program în cod mașină (*bytes*); 4 — program în cod mașină creat prin Editor/Asamblor;

— octeții 1÷10 conțin numele fișierului;

— octeții 11÷12 indică lungimea programului în octeți, împreună cu variabilele (pentru tipul de fișier 0 — program BASIC) sau lungimea programului/fișierului în cod mașină (pentru tipul 1, 2, 3);

— octeții 13÷14 indică linia de autostartare (cînd tipul de fișier este 0) sau adresa de implantare a programului, dacă tipul fișierului este 3;

— octeții 15÷16 arată lungimea programului BASIC fără variabile (numai pentru tip fișier — 0).

Partea a doua a fișierului conține datele propriu-zise. Pentru un fișier care memorează un program în cod mașină această parte va conține toți octeții programului.

Interfața cu casetofonul. Bitul 3 al portului 254 este utilizat pentru scriere (OUT), iar bitul 6 — pentru citire (IN). Acești doi biți sînt înlocuiți de rutinele de scriere și citire casetă, care sînt situate în memoria ROM și permit scrierea sau citirea de pe casetă a unui număr de octeți. Numărul de octeți, precum și poziția sînt fixate de utilizator.

Rutina de scriere este situată la adresa 4C2H(1218). La început, IX va trebui să conțină adresa de început a zonei de transferat, iar DE, numărul de octeți care se transferă. Registrul A va avea valoarea 0 (dacă se dorește scrierea unui antet de fișier) sau FFH (dacă se dorește scrierea corpului unui fișier). Următoarea suită de instrucțiuni va permite scrierea uneia din părțile fișierului pe caseta magnetică (fig. 5.1).

Rutina de citire este situată la adresa 556 H (1366) și necesită aceleași valori de intrare. În plus, va fi necesar ca indicatorul Carry să fie poziționat pe 1. Următoarea suită de instrucțiuni va permite citirea de pe casetă a unei părți a fișierului (fig. 5.2). Suita de instrucțiuni

LD	IX	, adresa	: adresa de început a zonei
			: care conține octeții de transferat
LD	DE	, număr	: numărul de octeți de transferat
LD	A	, cod	: 00 sau FFH în funcție de partea de transferat
			: (antet sau fișier propriu-zis)
CALL	4C2H		: scriere

Fig. 5.1. Rutina de scriere

LD	IX	, adresa	: adresa zonei în care vor fi transferați
			: octeții citiți
LD	DE	, număr	: numărul de octeți de citit
LD	A	, cod	: 00 sau FFH în funcție de partea de citit
			: (antet sau fișier propriu-zis)
SCF			: '
CALL	556H		: citire

Fig. 5.2. Rutina de citire

27986 37	SCF	: set carry flag
27987 3E00	LD A , 00	: se incarca acumulatorul cu 0
27989 DB21606D	LD IX , 27000	: se incarca IX cu adresa zonei
		: in care vor fi transferati octetii
		: (28000)
27993 111100	LD DE , 00017	: se incarca DE cu 17 (numarul de
		: octeti ai antetului)
27996 CD5605	CALL 01366	: se apeleaza rutina de citire din ROM
27999 C9	RET	

Fig. 5.3. Citirea informațiilor din antet

(dezasamblate cu MONS) care permite citirea informațiilor din antet referitoare la fișier (fig. 5.3) conține: pe prima coloană — adresele instrucțiunilor în zecimal, în a doua coloană — codurile instrucțiunilor în hexazecimal, în a treia coloană — mnemonicele de instrucțiuni cu operanzi în zecimal, iar în a patra coloană — comentarii.

Următorul program (fig. 5.4) reprezintă o aplicație care va utiliza informațiile extrase din antetele fișierelor existente pe caseta magnetică prin intermediul rutinei de citire casetă prezentate anterior. Programul va crea un fișier de date pentru ținerea evidențelor referitoare la programele conținute într-o bibliotecă personală.

Rutina de citire a informațiilor din antet se va localiza la adresa 26 986 (vezi linia 1) și va fi apelată din linia 150.

Rutina de citire se introduce din BASIC prin intermediul unei linii DATA (linia 30), care conține codificarea instrucțiunilor în cod mașină și a operanzilor în zecimal. Se poate observa că $55 = 37 H$; $62 = 3E H$ etc. regăsindu-se astfel codurile instrucțiunilor în hexazecimal din programul dezasamblat (coloana a doua). Datele din linia DATA au fost introduse ca șiruri de caractere, urmărind a fi transformate, apoi, în valori numerice prin intermediul funcției VAL. În acest mod se realizează o economie de memorie.

În linia 50 se realizează o rezervare de memorie, începând de la adresa 27 000 (adică de unde se termină programul în cod mașină); de la această adresă se va memora fișierul de date cu informațiile conținute în antetele citite.


```

1 CLEAR 26985
2 LET c8=9: LET e=1: LET c=1:
LET g=300
3 POKE 27000,5
8 INPUT "Cod Caseta (2 cifre)
":y
10 BORDER 0: PAPER 0: INK 7: C
LS
20 DATA "55","62","0","221","3
3","120","105","17","0","20
5","86","5","201"
30 FOR a=26986 TO 26999: READ
a$: POKE a,VAL a$: NEXT a
40 DEF FN p(a)=PEEK a+256*PEEK
(a+1)
50 DEF FN a(p,c)=27000+17*(c-1
)+170*(p-1)
80 GO SUB 500
150 RANDOMIZE USR 26986: GO SUB
730: IF PEEK FN a(p,c)=5 THEN G
O TO 150
160 GO SUB 600
170 GO SUB 800
180 IF c<c8 THEN LET c=c+1: LET
et=130: GO TO 200
190 IF c=c8 THEN LET p=p+1: LET
c=1: LET et=80: GO TO 200
195 STOP
200 POKE FN a(p,c),5: POKE 2699
2,INT (FN a(p,c)/256): POKE 2699
1,FN a(p,c)-256*INT (FN a(p,c)/2
56)
210 GO TO et
300 LET c=c-1: IF c=0 THEN LET
p=p-1: LET c=10
310 IF p=0 THEN LET p=1: LET c=
1: GO SUB 550: GO TO 150
320 GO TO 170
500 REM Initializare tabel
520 CLS : PRINT PAPER 1;"DIR*Ca
seta ";y:" Adr.Lungime.Auto""
530 PLOT 0,165: DRAW INK 2;255.
0
540 PLOT 0,166: DRAW INK 2;255.
0
550 INPUT : PRINT #1: PAPER 1:
TAB 25;"PAG ";p:(" " AND p<cB)
560 RETURN
600 REM Decodificare HEDER
610 LET ad=FN a(p,c)
620 LET t=PEEK ad
630 DIM n$(10)
640 FOR i=1 TO 10: LET n$(i)=CH
R$ PEEK (ad+i): NEXT i
650 LET l=FN p(ad+11)
660 LET s=FN p(ad+13)
670 LET x=FN p(ad+15)
680 GO SUB 700+t
690 GO TO 710
700 PRINT "Prog. ";: RETURN
701 PRINT "Num.A ";: RETURN
702 PRINT "Chr.A ";: RETURN
703 PRINT "Bytes ";: RETURN
705 RETURN
710 PRINT PAPER 1;n$: PAPER 0:
":
720 IF t=3 THEN PRINT s:":":1::
GO TO 770
730 PRINT l:
740 IF t>0 THEN GO TO 770
750 PRINT "/"*:x:
760 IF s=0 AND s<10000 THEN PR
INT "/"*:s:
770 PRINT "": RETURN
780 BEEP .1,-16: BEEP .1,32: BE
EP .1,31: BEEP .1,48: RETURN
805 INPUT : PRINT #1: FLASH 1:
": FLASH 0: PAPER 1:TAB 25;"PA
GE ";p:(" " AND p<cB)
810 FOR u=1 TO 180
820 IF INKEY$="" THEN GO TO 100
0
830 LET a$=INKEY$: GO SUB 780
840 LET cc=c: LET pp=p
850 INPUT : PRINT #1: FLASH 1:
": FLASH 0: PAPER 1;"SusVirDo
s ReitoargereTiparire":TAB 25;"P
AG ";p:(" " AND p<cB)
860 IF INKEY$="" THEN GO TO 860
870 LET a$=INKEY$: GO SUB 780
880 IF a$="r" THEN LET p=pp: GO
SUB 500: FOR c=1 TO cc: GO SUB
600: NEXT c: LET c=cc: INPUT "Sa
lvare sau Noua caseta?":a$: IF a
$(1)="s" OR a$(1)="S" THEN GO TO
9999
881 IF a$(1)="n" OR a$(1)="N" T
HEN RUN
884 IF a$="t" THEN GO TO 2500
890 IF a$="s" THEN LET p=p-1: G
O TO 920
900 IF a$="v" THEN LET p=1: GO
TO 920
910 LET p=p+1
920 IF p<1 THEN LET p=1
930 IF p>pp THEN LET p=pp
940 GO SUB 500
950 FOR c=1 TO cB AND ((cB AND
p(pp)+(cc AND p=pp))
960 GO SUB 600
970 NEXT c: GO TO 850
1000 NEXT u
1010 GO SUB 550
1020 RETURN
2500 COPY : GO TO 850
8700 GO SUB 800
9000 BORDER 0: PAPER 0: INK 7: C
LS : LOAD ""CODE 27000,17*(c-1)+
170*(p-1)+1: GO SUB 800
9998 STOP
9999 LET y$="DIR*Cas "+STR$ y: S
AVE y$ LINE 9000: SAVE "director
"CODE 27000,17*(c-1)+170*(p-1)

```

Fig. 5.4. Crearea unui fişier de date

Programul prezentat are caracter utilitar, cu ajutorul său existînd posibilitatea de a ține evidența înregistrărilor pe casete magnetice. Memorarea informațiilor (evidențelor) se va face tot pe caseta magnetică.

Programul prezintă un tablou pe care vor fi afișate informațiile. După lansarea în execuție a programului, se pornește casetofonul pentru parcurgerea casetei înregistrate cu programe. Programul va afișa (eventual pe mai multe pagini — ecrane — care conțin cîte 10 titluri) denumirea fișierului, *tipul* (program BASIC — *Prag* —, program în cod mașină — *Bytes* —, fișier de date numerice — *Num A* — — sau fișier de date alfanumerice — *Chr A* —), *lungimea fișierului* și *linia de autostartare* (pentru programe BASIC) sau *adresa de implantare a programului* (pentru programe în cod mașină).

La terminarea parcurgerii unei casete sau cînd utilizatorul dorește se va acționa orice tastă (dar numai pînă la 4÷5 secunde de la afișarea informațiilor referitoare la fișierul citit) și se va intra în al doilea mod de lucru, ale cărui opțiuni sînt: *S* (*Sus*) pentru afișarea paginii inferioare, *V* (*Vîrf*) pentru afișarea primei pagini, *J* (*Jos*) pentru afișarea paginii superioare, *R* (*Reîntoarcere*) pentru intrare în primul mod de lucru, *S* (*Save*) pentru salvarea fișierului creat pe casetă, *I* (*Imprimare*) pentru imprimarea fișierului la imprimantă sau *N* (*New*) pentru o nouă casetă, caz în care fișierul se șterge, intrîndu-se din nou în primul mod de lucru. Se recomandă ca fișierul creat să se salveze la începutul casetei respective. La o ieșire accidentală programul se poate relansa cu comanda *RUN*.

5.2. RUTINĂ GRAFICĂ PENTRU UMLEREA UNOR CONTURURI

Prezentăm o rutină foarte performantă realizată în cod mașină și utilizabilă în programe pentru umplerea rapidă a unor contururi.

Principiul de realizare a rutinei. Se caută începutul liniei de pixeli în care se află punctul curent analizat. De aici (de la început) se umple linia pînă la găsirea unui

punct activat sau a sfârșitului (marginii) de ecran. Dacă se găsește o ramificație, aceasta este depusă în lista FIFO. Dacă în listă se găsește o ramificație, atunci se aplică umplerea de linie și se scoate din listă. Noua ramificație se depune la sfârșitul listei, iar citirea se face de la început.

În locul coordonatelor punctelor se utilizează adresa + + masca. În mască un bit setat va da un punct. Lista FIFO are 400 de locații (o locație = 3 octeți). WRITER indică următoarea locație liberă, iar READER, ramificația cea mai veche.

PUTTUB depune o ramificație, iar GETTUB extrage o ramificație. Umplerea unui rând se realizează cu subrutine PLINE. Prin intermediul subrutinelor HLDOWN și HLUP se solicită o adresă din memoria RAM pentru ecran și se furnizează adresa octetului care este „sub” și respectiv „deasupra” adresei.

Dacă lista FIFO este plină, PUTTUB returnează $Z = 1$.

Schema logică pentru rutina principală PAINT este dată în fig. 5.5 iar pentru subrutina PLINE, în fig. 5.6.

Performanțe. Rutina poate umple pînă la maximum 10 000 puncte (pixeli) pe secundă (de exemplu, un patru-later de 100×100 este umplut în circa o secundă).

Rutina de umplere în cod mașină pentru calculatoare HC (compatibile ZX Spectrum) este dată în fig. 5.7. Ea se apelează din BASIC de la adresa TEST, punctul de start fiind ultimul punct desenat cu PLOT sau PLOT INVERSE. Dacă se solicită umplerea, punctul de start este marcat cu PLOT INVERSE 1, iar dacă se solicită ștergere, cu PLOT. Următorul program BASIC (fig. 5.8) exemplifică utilizarea rutinei (care se lansează de la adresa 63 089), realizînd umplerea și apoi ștergerea unui cerc de rază 30. Utilizatorul va introduce coordonatele centrului cercului sau va modifica programul, astfel încît să se realizeze umplerea sau ștergerea pentru alte contururi.

În vederea funcționării pe alte calculatoare care au memorie ecran altfel organizată sînt necesare mai multe modificări. De exemplu, pentru calculatorul PRAE 48 KO (la care adresa de început a memoriei ecran este #E 000, iar cea de sfârșit #FFFF) sînt necesare unele modificări (vezi fig. 5.9).

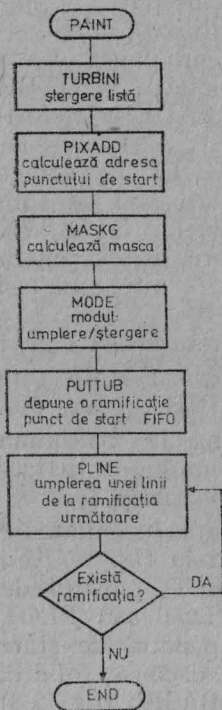


Fig. 5.5. Schema logică a rutinei PAINT

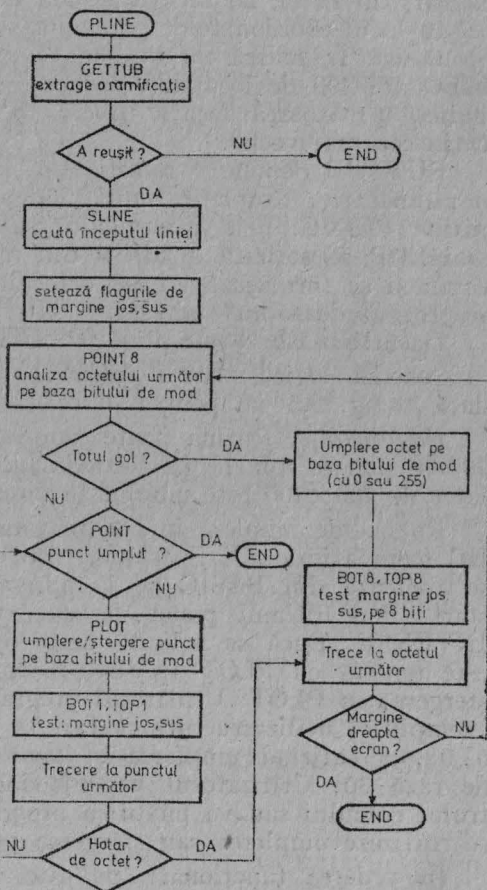


Fig. 5.6. Schema logică a subrutinei PLINE

10	ORG	#F000	
12			
14	PAINT:		128
16	PUSH	AF	130 PUTTUB:
18	PUSH	BC	132 EX DE.HL
20	PUSH	DE	134 PUSH DE
22	PUSH	HL	136 LD HL.(WRITER)
24	CALL	TUBINI	138 INC HL
26	CALL	PIXADD	140 LD (HL).E
28	CALL	MASKG	142 INC HL
30	CALL	MODE	144 LD (HL).D
32	CALL	PUTTUB	146 INC HL
34			148 LD (HL).B
36	PAINT1:		150 LD DE.FIFOE
38	CALL	PLINE	152 CALL CP16
40	JR	NZ.PAINT1	154 JR NZ.PUT1
42	POP	HL	156 LD HL.FIFO
44	POP	DE	
46	POP	BC	160 PUT1:
48	POP	AF	162 LD DE.(READER)
50	RET		164 CALL CP16
52			166 JR Z.PUT2
54	MODE:		168 LD (WRITER).HL
56	LD	C.O	170
58	CALL	POINT	172 PUT2:
60	RES	3.C	174 POP HL
62	RET	Z	176 RET
64	SET	3.C	178
66	RET		180 PLINE:
68			182 CALL GETTUB
70	TUBINI:		184 RET Z
72	LD	HL.FIFO	186 CALL SLINE
74	LD	(READER).HL	188 SET 0.C
76	LD	(WRITER).HL	190 SET 1.C
77	LD	DE.(WRITER)	192
78	RET		194 PLINE1:
80			196 LD A.(HL)
82	GETTUB:		198 CALL POINTB
84	LD	HL.(READER)	200 JR Z.PLINE2
86	LD	DE.(WRITER)	202
88	CALL	CP16	204 PLINE3:
90	RET	Z	206 CALL POINT
92	INC	HL	208 RET NZ
94	LD	E.(HL)	210 CALL PLOT
96	INC	HL	212 CALL BOT1
98	LD	D.(HL)	214 CALL TOP1
100	INC	HL	216 RRC B
102	LD	B.(HL)	218 JR NC.PLINE3
104	PUSH	DE	220 JR PLINE5
106	LD	DE.FIFOE	222
108	CALL	CP16	224 PLINE2:
110	JR	NZ.GET1	226 XOR A
112	LD	HL.FIFO	228 BIT 3.C
114			230 JR NZ.PLINE4
116	GET1:		232 CPL
118	LD	(READER).HL	234
120	POP	HL	236 PLINE4:
122	XOR	A	238 LD (HL).A
124	INC	A	240 CALL TOPB
126	RET		242 CALL BOTB

Fig. 57. (continua la pag. 92)

244.			372		
246	PLINES:		374	TOP12:	
248	INC	HL	376	POP	HL
250	LD	A.L	378	RET	
252	AND	#1F	380		
254	JR	NZ.PLIN1	382	TOPB:	
256	INC	A	384	CALL	POINT
258	RET		386	JR	Z.TOPB1
260			388	SET	1.C
262	POINT:		390	RET	
264	LD	A.(HL)	392		
266	AND	B	394	TOPB1:	
268	BIT	3.C	396	BIT	1.C
270	JR	NZ.P01	398	RET	Z
272	OR	A	400	CALL	PUTTUB
274	RET		402	RES	1.C
276			404	RET	
278	POI:		406		
280	XOR	B	408	HLDOWN:	
282	RET		410	INC	H
284			412	LD	A.H
286	PLOT:		414	AND	7
288	LD	A.(HL)	416	RET	NZ
290	OR	B	418	PUSH	DE
292	BIT	3.C	420	LD	DE.-#7E0
294	JR	Z.PL1	422	ADD	HL.DE
296	XOR	B	424	POP	DE
298			426	LD	A.H
300	PL1:		428	AND	7
302	LD	(HL).A	430	RET	Z
304	RET		432	LD	A.H
306			434	ADD	A.7
308	BOT1:		436	LD	H.A
310	PUSH	HL	438	RET	
312	CALL	HLDOWN	440		
314	LD	DE.SCREEN	442	HLVP:	
316	CALL	CP16	444	LD	A.H
318	JR	NC.BOT12	446	AND	7
320	CALL	BOTB	448	JR	Z.HLVP1
322			450	DEC	H
324	BOT12:		452	RET	
326	POP	HL	454		
328	RET		456	HLVP1:	
330			458	LD	A.L
332	BOTB:		460	AND	#E0
334	CALL	POINT	462	JR	Z.HLVP2
336	JR	Z.BOTB1	464	PUSH	DE
338	SET	O.C	466	LD	DE.#6E0
340	RET		468	ADD	HL.DE
342			470	POP	DE
344	BOTB1:		472	RET	
346	BIT	O.C	474		
348	RET	Z	476	HLVP2:	
350	CALL	PUTTUB	478	LD	A.L
352	RES	O.C	480	SUB	#20
354	RET		482	LD	L.A
356			484	DEC	H
358	TOP1:		486	RET	
360	PUSH	HL	488		
362	CALL	HLVP	490	SLINE:	
364	LD	DE.SCREEN	492	CALL	LEFTB
366	CALL	CP16	494	JR	O.SLINE1
368	JR	C.TOP12	496	RRC	B
370	CALL	TOPB	498	RET	
			500		

Fig. 5.7. (continuă la pag. 93)

502	SLINE1:		526	TOP8:	
504	RRC	B	628	PUSH	HL
506			630	CALL	HLVP
508	SLINE2:		632	LD	DE.SCREEN
510	LD	A.L	634	CALL	CP16
512	AND	#IF	636	JR	C.TOP8N
514	RET	Z	638	LD	A.(HL)
516	DEC	HL	640	OR	A
518	LD	A.(HL)	642	JR	Z.TOP81
520	CALL	POINT8	644	CPL	
522	JR	Z.SLINE2	646	JR	NZ.TOP32
524	RLC	B	648		
526	CALL	LEFT81	650	TOP81:	
528	RRC	B	652	CALL	TOP8
530	RET	NC	654	POP	HL
532	INC	HL	656	RET	
534	RET		658		
536			660	TOP82:	
538	LEFT8:		662	CALL	TOP8
540	RLC	B	664	RRC	B
542	RET	C	666	JP	NC.TOP82
544			668		
546	LEFT81:		670	TOP8N:	
548	CALL	POINT	672	POP	HL
550	JR	Z.LEFT8	674	RET	
552	OR	A	676		
554	RET		678	POINT8:	
556			680	BIT	J.C
558	CP16:		682	JR	Z.P081
560	LD	A.H :HL.DE	684	CPL	
562	CP	D	686		
564	RET	NZ	688	P081:	
566	LD	A.L	690	OR	A
568	CP	E	692	RET	
570	RET		694		
572			696	MASKG:	
574	B0TB:		698	LD	B.1
576	PUSH	HL	700	INC	A
578	CALL	HLDDWN	702		
580	LD	DE.SCREND	704	MG1:	
582	CALL	CP16	706	RRC	B
584	JR	NC.B0TB8N	708	DEC	A
586	LD	A.(HL)	710	JR	NZ.MG1
588	DR	A	712	RET	
590	JR	Z.B0TB1	714		
592	CPL		716		
594	JR	NZ.B0TB2	718	FIFO	EQU *
596			720	DEFS	400*3
598	B0TB1:		722	FIFOE	EQU *-1
600	CALL	B0TB	724		
602	POP	HL	726	READER	DEFS 2
604	RET		728	WRITER	DEFS 2
606			730		
608	B0TB2:		732	PIXADD	EQU #22AA :
610	CALL	B0TB		RUTINA	EPROM
612	RRC	B	734	SCREEN	EQU #4000
614	JR	NC.B0TB2	736	SCREND	EQU *-37FF
616			738		
618	B0TB8N:		740	TEST	LD BC.(#5C7D) :
620	POP	HL		ULTIMUL	PUNCT
622	RET		742	JP	PAINT
624			744	END	TEST
			746		

Fig. 5.7. Rutina de umplere pentru HC-85

```

9 INPUT I
10 CIRCLE I.I.30
20 PLOT INVERSE 1:I+1,I+1
30 RANDOMIZE USR 63089
32 PAUSE 0
40 PLOT I+1,I+1
50 RANDMIZE USR 63089
60 GO TO 9

```

Fig. 5.8. Utilizarea rutinei

1000			1038	RLA
1002	SCREEN:		1040	LD HL,SCREEN
1004	EQU	#E000	1042	ADD HL,BC
1006			1044	RET
1008	SCREND:		1046	
1010	EQU	#FFFF	1048	HLVP:
1012			1050	PUSH BC
1014	PIXADD:		1052	XOR A
1016	XOR	A	1054	LD BC,32
1018	SRA	B	1056	SBC HL,BC
1020	RR	C	1058	POP BC
1022	RRA		1060	RET
1024	SRA	B	1062	
1026	RR	C	1064	HLDOWN:
1028	RRA		1066	PUSH BC
1030	SRA	B	1068	LD BC,32
1032	RR	C	1070	ADD HL,BC
1034	RLA		1072	POP BC
1036	RLA		1074	RET

Fig. 5.9. Rutina pentru PRAE

5.3. REPREZENTAREA GRAFICĂ A FUNCȚIILOR

Definit la începutul anilor '70 în scopul înlesnirii învățării programării, limbajul PASCAL este, în prezent, unul dintre cele mai răspândite limbaje de programare în mediul liceal și, în special, universitar. El a fost adoptat însă și pentru scrierea aplicațiilor în majoritatea domeniilor în care se utilizează calculatorul, mai ales după aparițiile versiunilor TURBO. Limbajul PASCAL oferă avantaje atât în ceea ce privește instrucțiunile de control, care sînt chiar cele impuse de tehnica programării structurate (IF-THEN-ELSE, REPEATUNTIL, CASE, FOR), cît și a facilităților deosebit de puternice pentru repre-

zentarea datelor și a graficii. Evitarea sistematică a utilizării instrucțiunilor GO TO, precum și scrierea modularizată oferă limbajului o inteligibilitate și o claritate foarte bune.

Avantajul major față de alte limbaje disponibile este viteza mare la rulare (este compilator și nu interpretor, ca limbajul BASIC, inclusiv versiunea pentru calculatoare compatibile Sinclair Spectrum), datorită faptului că prin compilare se generează cod direct executabil. De aici rezultă și una din aplicațiile sale majore și anume realizarea de calcule matematice și statistice complexe. Alt domeniu important de aplicație al limbajului PASCAL îl constituie reprezentările grafice, oferindu-se facilități deosebite prin setul de primitive cunoscut sub numele de TURTLE GRAPHICS.

Programul prezentat în fig. 5.10 este realizat în versiunea HP4 T16 pentru calculatoare compatibile Sinclair Spectrum; prin intermediul său se pot reprezenta grafic funcții de două variabile. Funcțiile sînt definite pe intervalul $[-3, 3]$ și iau valori în mulțimea numerelor reale. Marginile intervalului se pot modifica cu ușurință. Definirea funcției se face natural prin $Z = F(X, Y)$.

Liniile de program 130 ÷ 830 conțin definirea funcțiilor grafice (de fapt, o simplă interfață PASCAL-BASIC), prin intermediul programului putîndu-se realiza puncte și drepte la fel ca în BASIC. Urmează definirea rutinei de hard-copy (pentru canalul binar al INTERFETEI 1 și imprimanta Robotron K 6 314). Programul începe efectiv în linia 1 360 cu definirea funcției de reprezentat $-F(X, Y)$. Funcțiile FA și FB realizează proiecția din sistemul $OXYZ$ (spațiul de definire) în planul OAB (spațiul de reprezentare).

Algoritmul de vizibilitate este de tipul „Z-buffer”. Reprezentarea corectă în acest caz este asigurată numai dacă trasarea se face într-o anumită ordine (se începe din punctul cel mai apropiat de observat și se scanează suprafața pentru toate punctele cu coordonata x și, respectiv, y egală cu cea a punctului curent). Testul de vizibilitate se efectuează în rutina PLOT 1, care lucrează direct în coordonate ecran. Procedura PUNE trasează o linie între punctele

```

27 PROGRAM GRAFIC;
30 CONST SI=0.5;
35 CO=0.866025;
40 VAR A,B:ARRAY[1..30,1..30] OF REAL;
45 C:ARRAY[0..255,1..2] OF INTEGER;
50 I,J,K,L,M,N,NN,KR:INTEGER;
55 X,Y,Z,AR,BR,CM,IN,AMAX,BMIN,BMAX,PASX,PASY,BS,AS:REAL;
60
65
70
75
80
85
90
95
100
105
110
115
120
125 ( ++++++
130
135 Extensia de grafica si
140 sunet pentru HP4T16
145 )
150 PROCEDURE PLOT(X,Y:INTEGER);
155
160 BEGIN
165 IN LINE(#D,#21,#3A,#5C,#DD,#46,#02,#DD,#4E,#04,#CD,#E5,#22)
170 END;
175
180 PROCEDURE DRAW(X,Y:INTEGER);
185
190 BEGIN
195 IN LINE(#D,#21,#3A,#5C,#DD,#56,#03,#DD,#5E,#05,#DD,
200 #46,#02,#DD,#4E,#04,
205 #21,#01,#FF,#AF,#8A,#55,#28,#03,#54,#90,#47,#AF,#8E,
210 #5D,#28,#03,#5C,#91,#4F,
215 #CD,#8A,#24)
220 END;
225
230 PROCEDURE SPOUT(C:CHAR);
235
240 BEGIN
245 IN LINE(#D,#21,#3A,#5C,#DD,#7E,#02,#D7);
250 END;
255
260 PROCEDURE BEEPER(A,B:INTEGER);
265
270 BEGIN
275 IN LINE(#DD,#6E,#02,#DD,#66,#03,#DD,#5E,#04,#DD,#56,
280 #05,#CD,#85,#03,#F3);
285 END;
290
295 PROCEDURE BEEP(FR:INTEGER;LE:REAL);
300
305 VAR I:INTEGER;
310 BEGIN
315 IF FR=0 THEN FOR I:=1 TO ENTIER(12000*LE) DO
320 ELSE BEEPER(ENTIER(FR*LE),ENTIER(437500/FR-30.125));
325 END;
330
335 PROCEDURE PAT(LIN,COL:INTEGER);
340
345 BEGIN
350 IN LINE(#D,#21,#3A,#5C,#DD,#7E,#04,#FE,#18,#30,#27,
355 #47,#DD,#7E,#02,#FE,#20,#30,
360 #1F,#4F,#78,#E6,#18,#F6,#40,#32,#85,#5C,#78,#E6,#07,
365 #0F,#0F,#0F,#81,
370 #32,#84,#5C,#3E,#21,#91,#32,#88,#5C,#3E,#18,#90,#32,
375 #89,#5C,#00)
380 END;
385
390 PROCEDURE ATRIB(CULOARE:INTEGER);
395
400 BEGIN
405 IN LINE(#D,#21,#3A,#5C,#DD,#7E,#02,#32,#8D,#5C,#32,
410 #8F,#5C)
415 END;
420
425 PROCEDURE CLS(CULOARE:INTEGER);
430
435 BEGIN
440 IN LINE(#D,#21,#3A,#5C,#AF,#32,#6B,#5C,#DD,#56,#02,
445 #21,#00,#58,#01,#00,#03,#72,
450 #0B,#23,#78,#81,#20,#F9);
455 ATRIB(CULOARE);PAGE;PAT(0,0)
460 END;
465
470 PROCEDURE BORDER(C:INTEGER);
475
480 BEGIN
485 IN LINE(#D,#21,#3A,#5C,#DD,#7E,#02,#E6,#07,#47,#07,
490 #07,#07,#78,#D3,#FE)
495 END;
500
505
510
515
520
525
530
535
540
545
550
555
560
565
570
575
580
585
590
595
600
605
610
615
620
625
630
635
640
645
650
655
660
665
670
675
680
685
690
695
700
705
710
715
720
725
730
735
740
745
750
755
760
765
770
775
780

```

Fig. 5. 10 (continuă în pag. 97)

```

C204 790      ATRIM
C204 800      CLS
C20A 810      BORDER
C20A 820
C20A 830      ++++++
C20A 840
C20A 850      FUNCTION POINT(X,Y:INTEGER):INTEGER;
C2CD 860      VAR I,J,K,L,ADR:INTEGER;
C2CD 870      BEGIN
C2E5 880          I:=Y DIV 64;
C2F9 890          J:=-Y-64+1;
C31C 900          K:=J DIV 8;
C330 910          L:=J-K*8;
C34E 920          ADR:=L*256+K*32+I*2048+16384+(X DIV 8);
C3A7 930          I:=X-(X DIV 8)*8;
C3CD 940          I:=7-I;
C3E5 950          J:=ORD(PEEK(ADR,CHAR));
C3F3 960          WHILE I>0 DO
C40E 970              BEGIN
C40E 980                  J:=J DIV 2;
C422 990                  I:=I-1;
C42F 1000             END;
C432 1010            K:=J MOD 2;
C445 1020            POINT:=K;
C451 1030            END;
C460 1040
C460 1050      PROCEDURE COPY;
C463 1060      VAR I,J,K,L:INTEGER;
C463 1070      BEGIN
C47B 1080
C47B 1090      ( PROGRAMARE IMPRIMANTA )
C47B 1100
C47B 1110      WRITE(CHR(16));
C482 1120      SPOUT(CHR(27)); SPOUT('A'); SPOUT(CHR(6));
C4AD 1130      SPOUT(CHR(27)); SPOUT('C'); SPOUT(CHR(96));
C4D8 1140      SPOUT(CHR(13)); SPOUT(CHR(10)); SPOUT(CHR(13));
C505 1145      SPOUT(CHR(10));
C514 1150
C514 1160      ( INTRARE MOD GRAFIC )
C514 1170
C514 1180      L:=0;
C51D 1190      REPEAT
C51D 1200          WRITE(' ');
C52E 1210      SPOUT(CHR(27)); SPOUT('.') SPOUT(CHR(5)); SPOUT(CHR(8));
C56B 1215      SPOUT(CHR(2));
C577 1220      FOR I:=0 TO 255 DO
C59A 1230          BEGIN
C59D 1240              K:=0;
C5A6 1250              FOR J:=0 TO 2 DO
C5C9 1260                  BEGIN
C5CC 1270                      K:=K*4+POINT(I,L+J)*3; END;
C610 1280                      SPOUT(CHR(K)); SPOUT(CHR(K));
C634 1290                  END;
C637 1300                  L:=L+3;
C646 1310          WRITELN(' '); SPOUT(CHR(13)); SPOUT(CHR(10));
C674 1320          UNTIL L>=192;
C68D 1330          SPOUT(CHR(13)); SPOUT(CHR(10)); SPOUT(CHR(27));
C6BA 1335          SPOUT('0'); WRITE(CHR(16));
C6CE 1340          END;
C6D9 1350
C6D9 1360      FUNCTION F(X,Y:REAL):REAL;
C6DC 1370      VAR R,F1:REAL;
C6DC 1380      BEGIN
C6F4 1390          R:=X*X+Y*Y;
C73F 1400          F1:=-COS(X)*COS(Y)*4;
C781 1410          END;
C790 1420
C790 1430
C790 1440      FUNCTION FA(X,Y,Z:REAL):REAL;
C793 1450      BEGIN
C7AB 1460          FA:=Y*0.95-SI*X;
C7EE 1470          END;
C7FB 1480
C7FB 1490
C7FB 1500      FUNCTION FB(X,Y,Z:REAL):REAL;
C7FB 1510      BEGIN
C813 1520          FB:=-Z-0.31*Y-CO*X;
C86B 1530          END;
C875 1540
C875 1550      PROCEDURE PLOT1(I,J:INTEGER);
C87B 1560      VAR N:INTEGER;
C87B 1570      BEGIN
C890 1580          IF (C[I,1]=0) AND (C[I,2]=0) THEN BEGIN C[I,1]:=J; C[I,2]:=1 END
C99D 1590          ELSE
C9A0 1600          IF (C[I,1]>J) AND (C[I,2]<J) THEN
CA36 1610          ELSE

```

Fig. 5. 10. (continuă în pag. 98)

```

CA39 1620 BEGIN
CA39 1630 PLOT(1,J);
CA50 1640 IF C[I,1]≠J THEN C[I,1]:=J ELSE C[I,2]:=J;
CE20 1650 END
CE20 1660 END;
CE20 1670
CE20 1680 PROCEDURE PUNE(I,J,K,L:INTEGER);
CE2E 1690 VAR X1,X2,Y1,Y2,I1,KK:INTEGER;
CE2E 1700 BEGIN
CE46 1710 X1:=ROUND((A[I,J]-AMIN)*AS+S);
CE88 1720 Y1:=ROUND((B[I,J]-BMIN)*BS+S);
CC30 1730 X2:=ROUND((A[K,L]-AMIN)*AS+S);
CCA5 1740 Y2:=ROUND((B[K,L]-BMIN)*BS+S);
CD1A 1750 N:=X2-X1;M:=Y2-Y1;
CD4A 1760 KK:=ABS(M);IF KK<ABS(N) THEN KK:=ABS(N);
CD7C 1770 KK:=KK*2;
CD89 1780 I1:=0;
CD92 1790 IF KK=0 THEN PLOT1(X1,Y1) ELSE BEGIN
CDB1 1800 REPEAT
CDB1 1810 PLOT1(X1+((N*I1) DIV KK),Y1+((M*I1) DIV KK));
CE1F 1820 I1:=I1+1;
CE2C 1830 UNTIL I1>=KK;
CE48 1840 PLOT1(X2,Y2);
CE5F 1850 END;
CE5F 1860 END;
CE6E 1870
CE6E 1880
CE6E 1890 ( MAIN )
CE6E 1900
CE6E 1910 BEGIN
CE77 1920 CLS(71);
CE80 1930 Z:=F(-3,-3);AMIN:=FA(-3,-3,Z);AMAX:=AMIN;
CE7F 1935 BMIN:=FB(-3,-3,Z);BMAX:=BMIN;
CF3E 1940 PASX:=6/29;PASZ:=6/29;
CF6E 1950 X:=-3;
CF83 1960 FOR I:=1 TO 30 DO
CF9D 1970 BEGIN
CFA0 1980 Y:=-3;
CFB5 1990 FOR J:=1 TO 30 DO
CFCF 2000 BEGIN
CFD2 2010 PAT(11,12);WRITE(900-30*(I-1)-J:4);
D00C 2020 Z:=F(X,Y);
D02E 2030 AR:=FA(X,Y,Z);
D059 2040 BR:=FB(X,Y,Z);
D084 2050 IF AR<AMIN THEN AMIN:=AR ELSE IF AR>AMAX THEN AMAX:=AR;
D0E9 2060 IF BR<BMIN THEN BMIN:=BR ELSE IF BR>BMAX THEN BMAX:=BR;
D14E 2070 A[I,J]:=AR;B[I,J]:=BR;
D1E0 2080 Y:=Y+PASZ;
D1FA 2090 END;
D1FD 2100 X:=X+PASX;
D217 2110 END;
D21A 2120
D21A 2130 BS:=165/(BMAX-BMIN);AS:=236/(AMAX-AMIN);
D26C 2140 FOR I:=0 TO 255 DO
D286 2150 FOR J:=1 TO 2 DO
D2A3 2160 C[I,J]:=0;
D2E6 2170
D2E6 2180 CLS(71);PAT(12,8);WRITE('CAROTIAJ ?(1,2,3)');REAPLN;
D31A 2190 KR:=3;IF NOT EOLN THEN READ(KR);
D32F 2200
D32F 2210 CLS(71);
D338 2220 FOR K:=30 DOWNT0 1 DO
D351 2230 BEGIN
D354 2240 FOR I:=1 TO K-1 DO
D373 2250 BEGIN
D376 2260 IF (KR=1)OR(KR=3)THEN PUNE(K,K-I+1,K,K-I);
D3C7 2270 IF (KR=2)OR(KR=3)THEN PUNE(K-I+1,K,K-I,K);
D418 2280 END;
D41C 2290 FOR I:=1 TO K-1 DO
D438 2300 BEGIN
D43E 2310 IF K>1 THEN
D451 2320 BEGIN
D451 2330 IF (KR=2)OR(KR=3)THEN PUNE(K,K-I,K-1,K-I);
D4A2 2340 IF (KR=1)OR(KR=3)THEN PUNE(K-I,K,K-1,K-1);
D4F3 2350 END;
D4F3 2360 END;
D4F7 2370 END;
D4FA 2380 READLN;COPY;
D500 2390 END;
End Address: D504
Run?C

```

Fig. 5.10. Program de grafică

de indici (i, j) și (k, l) . Funcția se calculează înainte pentru valorile din nodurile unei rețele plane cu pasul 0,1.

Programul principal asigură inițializările, calculul funcției în nodurile rețelei, scalarea automată și parcurgerea rețelei conform algoritmului „Z-buffer“.

În figura 5.11 se pot urmări câteva reprezentări grafice de funcții realizate cu programul prezentat.

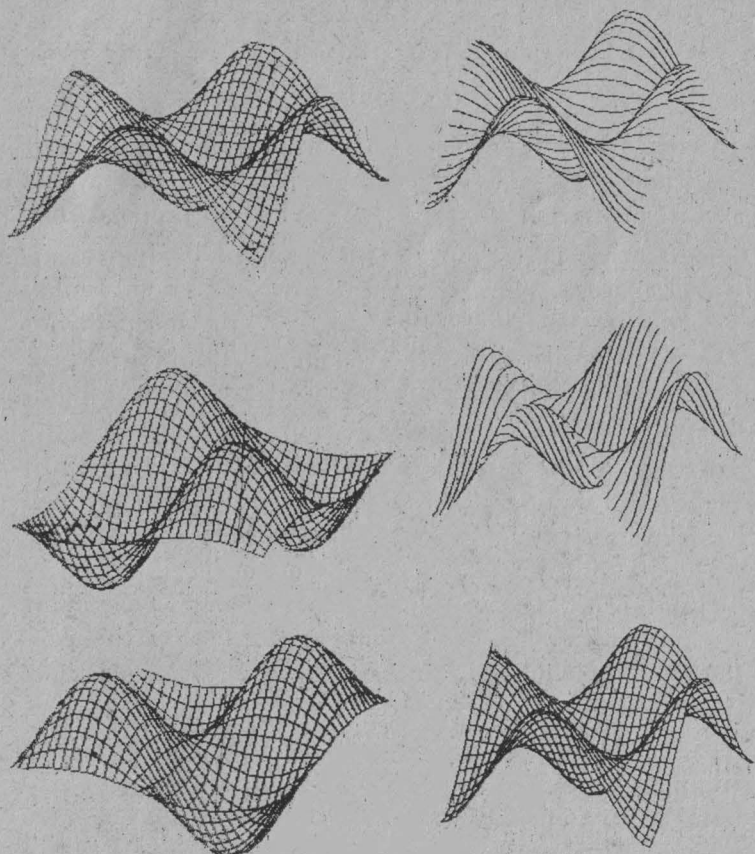


Fig. 5.11. Reprezentări grafice de funcții

PROGRAMARE ÎN STIL LOGO

În contrast cu primele limbaje de programare, al căror accent cade pe aritmetica computațională, *LOGO* a fost proiectat pentru prelucrarea cuvintelor și propozițiilor.

Programarea în *LOGO* poate fi înțeleasă la diferite niveluri. De aceea, *LOGO* reprezintă nu numai un limbaj de programare indicat pentru introducerea începătorilor (în special a copiilor mici), ci și o posibilitate de înțelegere mai profundă a informaticii pentru cei care au depășit un stadiu introductiv.

În acest capitol se urmăresc nu atât asimilarea și explicarea în stilul unui ghid de utilizare a unui număr cât mai mare de primitive *LOGO*, ci explorarea unor idei fundamentale ale programării calculatoarelor prin intermediul limbajului *LOGO*. Din acest motiv se subînțelege că sînt cunoscute cîteva aspecte elementare ca: încărcarea sistemului *LOGO*, modul de introducere (tastare) a comenzilor, utilizarea editorului *LOGO* etc.

Toate exemplele se vor referi la versiunea *LOGO* pentru calculatoarele, *HC*, *TIM-S*, *COBRA* etc., care este disponibilă prin caseta *RECOOP-ITCI* editată în 1989. Pentru lămuriri suplimentare atît cu privire la aspectele introductive, cît și a unor primitive *LOGO* se poate consulta manualul de referință care însoțește caseta menționată.

6.1. CONCEPTE GENERALE

Procedură. Convenim că într-un dialog vom sublinia răspunsurile date de LOGO pentru a le deosebi de introducerile unui utilizator. Presupunem, de asemenea că am introdus sistemul LOGO și, acesta a afișat promptul său specific (semnul de întrebare „?”) care arată că LOGO așteaptă instrucțiunile unui utilizator. Să introducem următoarea instrucțiune:

? PRINT 17

17

?

LOGO va răspunde la această instrucțiune afișând numărul 17 și, din nou, promptul său specific.

Cuvântul *PRINT* este numele unei proceduri. Aceasta este un mic segment dintr-un program și prin el se îndeplinește o anumită sarcină. Procedura numită *PRINT*, de exemplu, are drept sarcină să afișeze lucruri pe ecran.

Se cunoaște că în limbajele *BASIC* și *PASCAL* există mai multe tipuri de instrucțiuni, fiecare avînd o anumită sintaxă (o punctuație specială și un format). Spre deosebire de aceste limbaje, LOGO nu prezintă mai multe tipuri de instrucțiuni: în LOGO orice este realizat prin intermediul procedurilor. La începutul unei sesiuni de lucru LOGO, acesta „știe” circa 100-200 proceduri, numărul depinzînd de tipul calculatorului și de versiunea LOGO. Procedurile inițiale se mai numesc (proceduri) *primitive*. Programarea în LOGO se realizează îmbogățind repertoriul LOGO cu noi proceduri după nevoile proprii, iar acest lucru se va realiza prin asocierea mai multor proceduri existente.

Deși are o sarcină specifică, procedura *PRINT* nu va face totdeauna exact același lucru; prin intermediul ei utilizatorul poate afișa pe ecran orice dorește, nu numai numărul 17. Pentru a controla această flexibilitate, este nevoie de un mijloc care să indice procedurii exact lucrul pe care dorim să îl realizeze. Astfel, fiecare procedură conține o informație; poate fi un număr (ca în exemplu dat), dar există și alte feluri de informații care pot fi manipulate

de procedurile LOGO. Procedura numită *PRINT* necesită un *subiect* (*parametru de intrare*). Alte proceduri necesită mai mulți parametri de intrare, iar altele nici unul.

Exemple :

— *FORWARD 50* are ca urmare înaintarea „broaștei“ (o mică săgeată care în mișcarea ei lasă urme pe ecran; este dispozitivul cu care se realizează grafica în LOGO) cu 50 de pași, iar *RIGHT 90* are ca urmare rotirea „broaștei“ la dreapta cu un unghi de 90 de grade. Procedurile *FORWARD*, *RIGHT* ca și *PRINT* și multe altele necesită un singur parametru de intrare;

— *CLEAN* are ca efect ștergerea ecranului, iar *PENUP* — ridicarea creionului (de aici înainte „broasca“ se va deplasa fără a lăsa urme). Procedurile *CLEAN* și *PENUP* nu necesită nici un parametru de intrare;

— *SUM 32* are ca efect adunarea celor două numere, iar *PROD 32* înmulțirea lor. Procedurile *SUM* și *PROD* necesită câte doi parametri de intrare. Se observă că atât între proceduri și parametri, cât și între parametri se intercalează câte un spațiu care joacă rol de separator.

Într-o discuție obișnuită cuvinte ca *instrucțiune* și *procedură* au practic același înțeles: ele se referă la orice metodă, proces sau rețetă care realizează o anumită sarcină. Situația nu este aceeași când este vorba de programarea calculatoarelor. În LOGO, o *instrucțiune* reprezintă ceea ce se introduce pentru a se executa un ordin. (*PRINT 17* reprezintă un exemplu de instrucțiune). Unele instrucțiuni sînt mai complicate, fiind formate din mai multe părți. O instrucțiune trebuie să conțină informație suficientă pentru a specifica exact ceea ce este de dorit să execute LOGO. Pentru a face o analogie cu o instrucțiune (ordin) adresată unui operator, „Citește această pagină“, este o instrucțiune, în timp ce „Citește“ nu este, deoarece nu comunică ce se dorește a se citi.

O *procedură*, în schimb, reprezintă o tehnică pentru îndeplinirea unei sarcini oarecare (de exemplu, *PRINT* este numele unei proceduri.). Pentru a fi executată, procedura trebuie apelată de cineva. Se spune că se *invocă* o procedură. Procedurile sînt invocate prin instrucțiuni.

Comenzi și operații. În LOGO sînt două tipuri de proceduri: comenzi și operații. O *operație* este o procedură care calculează o valoare și o extrage (exemple de operații: *SUM*, *PROD* etc.).

O *comandă* este o procedură care nu extrage un rezultat, dar are un efect imediat (de pildă: mișcarea „broaștei“, un sunet, tipărirea pe ecran etc.); exemple de comenzi: *PRINT*, *SOUND* etc.

O *instrucțiune completă* constă în numele instrucțiunii urmat de atîtea expresii cîte sînt necesare pentru a extrage rezultatele.

O *expresie* este, de exemplu, *SUM 32* sau *17*. Se observă că operațiile sînt utilizate pentru a construi expresii.

Exemplu de instrucțiune completă: *PRINT SUM 3 2*.

Se pot pune deci pe aceeași linie mai multe instrucțiuni care formează instrucțiuni complete.

Cu succesiunea următoare de instrucțiuni se desenează pe ecran un pătrat cu latura de 50:

```
FORWARD 50 LEFT 90 FORWARD 50 LEFT 90 FORWARD 50  
LEFT 90 FORWARD 50 LEFT 90
```

Același lucru se poate obține mai ușor, folosind repetiția astfel: *REPEAT 4 [FORWARD 50 LEFT 90]*.

Modalități de adăugare a noi proceduri (cuvinte). Dacă ultima instrucțiune completă se va intercala între cuvintele *TO PATRAT* și *END*, atunci *PATRAT* va deveni numele unei noi proceduri; de fiecare dată cînd aceasta este invocată, va fi desenat un pătrat cu latura de 50:

```
TO PATRAT  
REPEAT 4 [FORWARD 50 LEFT 90]  
END
```

Cuvîntul *TO* este o prescurtare pentru „iată cum să ...“. Numele procedurii sugerează conținutul procedurii (o metaforă), iar *END* indică sfîrșitul procedurii. În mod similar, procedura pentru definirea unui triunghi cu latura 50 va fi:

```
TO TRIUNGI  
REPEAT 3 [FORWARD 50 LEFT 120]  
END
```

Se observă că pentru a desena orice figură geometrică închisă suma rotirilor „broaștei“ (este de fapt suma unghiurilor exterioare) trebuie să fie egală cu 360° (teorema rotirii complete a „broaștei“).

Desigur, noile proceduri create, avînd de acum același statut ca procedurile primitive, pot fi invocate în cadrul unor instrucțiuni complete și chiar în definirea unor noi proceduri. Iată un exemplu în care noile proceduri definite, *PATRAT* și *TRIUNGHI*, sînt invocate în cadrul unei instrucțiuni complete (vezi figura 6.1):

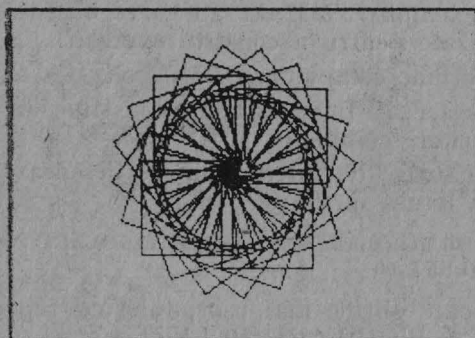


Fig. 6.1. Rezultatul unei instrucțiuni de tip *REPEAT...*

```
REPEAT 20 [FORWARD 2 PATRAT LEFT 18 TRIUNGHI]
```

Evaluare. Dacă instrucțiunile sînt constituite din nume de proceduri, iar procedurile invocate de instrucțiuni sînt compuse din mai multe instrucțiuni (așa cum am văzut), atunci cum se explică faptul că sistemul nu intră într-un cerc vicios, invocînd la nesfîrșit proceduri mai detaliate și, implicit, nerealizînd practic nimic. Pentru programarea în *LOGO* răspunsul este legat de faptul că, în final, atît instrucțiunile, cît și procedurile pe care le invocă trebuie să fie definite în termenii procedurilor primitive. Aceste proceduri nu sînt alcătuite din instrucțiuni *LOGO*. Sînt lucruri pe care calculatorul știe că trebuie să le realizeze în mod prioritar.

Să considerăm următoarea instrucțiune:

```
PRINT SUM 2 3
```

Dacă totul decurge conform celor stabilite, atunci *LOGO* nu va afișa cuvintele *SUM 2 3*, ci numărul 5. Subiectul (sau ce se introduce ori parametrul de intrare) pentru *PRINT* este expresia *SUM 2 3*. Însă limbajul *LOGO* evaluează parametrul de intrare înainte de a trece la procedura *PRINT* ceea ce înseamnă că *LOGO* invocă procedurile necesare (în acest caz, *SUM*) pentru a calcula valoarea expresiei (5). Astfel, rezultatul lui *SUM* devine parametrul de intrare pentru *PRINT*.

De remarcat că nu trebuie confundat rezultatul cu procesul de afișare. În *LOGO* cuvîntul „rezultat“ sau „output“ este un termen tehnic care se referă la o valoare care este calculată (furnizată) de o procedură și folosită de o alta, ce necesită un subiect (parametru de intrare). În exemplul dat, *SUM* furnizează numărul 5 pentru *PRINT*, dar *PRINT* nu furnizează nimic pentru altă procedură. Cînd *PRINT* afișează numărul 5, acest fapt reprezintă sfîrșitul fără a mai exista alte proceduri care așteaptă parametri de intrare.

Pentru a înțelege cum se realizează evaluarea în *LOGO* să imaginăm desfășurarea procesului în cazul următoarei instrucțiuni:

PRINT SUM 4 PRODUCT 10 2

Pentru evaluarea acestei instrucțiuni sistemul *LOGO* va parcurge următorii pași:

1. Primul lucru în această instrucțiune este numele procedurii *PRINT*. Sistemul *LOGO* știe că *PRINT* necesită un singur parametru de intrare, așa încît continuă să citească linia.

2. Următorul lucru găsit de sistemul *LOGO* este cuvîntul *SUM*. Acesta este de asemenea numele unei proceduri. Rezultatul furnizat de procedura *SUM* va reprezenta parametrul de intrare pentru procedura *PRINT*.

3. Limbajul *LOGO* știe că *SUM* necesită doi parametri de intrare; din această cauză *SUM* nu poate fi invocat pînă cînd *LOGO* nu găsește acești parametri pentru *SUM*.

4. Elementul următor în instrucțiune este numărul 4, acesta fiind deci primul parametru de intrare pentru

SUM. Parametrul respectiv necesită, de asemenea să fie evaluat. Din fericire, un număr se evaluează prin el însuși, astfel încît valoarea acestui parametru este 4.

5. Acum *LOGO* trebuie să găsească al doilea parametru de intrare pentru *SUM*. Următorul element în instrucțiune este cuvîntul *PRODUCT*, care reprezintă numele unei proceduri. *LOGO* trebuie să se orienteze spre această procedură, în scopul evaluării celui de al doilea parametru al lui *SUM*.

6. Limbajul *LOGO* știe că *PRODUCT* necesită doi parametri (deci, trebuie să-i caute). Între timp, *PRINT* și *SUM* sînt ambele în așteptare pentru evaluarea parametrilor lor (*PRINT* așteaptă pentru un singur parametru; *SUM*, pentru care s-a găsit un parametru, așteaptă pentru cel de-al doilea). Următorul element de pe linie este numărul 10. Acest număr se autoevaluează, primul parametru pentru *PRODUCT* fiind deci 10.

7. Sistemul *LOGO* are nevoie de încă un parametru de intrare pentru *PRODUCT*, așa încît el continuă să citească linia de instrucțiune. Următorul lucru pe care îl găsește este numărul 2. Acest număr se autoevaluează, astfel încît al doilea parametru pentru *PRODUCT* va fi valoarea 2.

8. Acum *LOGO* poate invoca procedura *PRODUCT* cu parametrii de intrare 10 și 2. Procedura *PRODUCT* va furniza rezultatul înmulțirii lui 10 cu 2, deci 20.

9. Acest rezultat (20) este o valoare care va reprezenta al doilea parametru pentru *SUM*. Acum *LOGO* poate invoca procedura *SUM*, cu parametrii 4 și 20. Procedura va furniza rezultatul 24.

10. Rezultatul procedurii *SUM* (24) reprezintă parametrul de intrare pentru *PRINT*, pe care *LOGO* îl poate acum invoca. Sistemul va afișa numărul 24.

Pentru a realiza o diagramă a acestei analize, fiecare procedură poate fi reprezentată ca un bloc cu una sau mai multe intrări (în partea superioară), în funcție de numărul de parametri de intrare ai procedurii, și o ieșire (în partea inferioară) dacă procedura furnizează sau nu un rezultat.

În fig. 6.2 poate fi urmărită reprezentarea grafică corespunzătoare procedurilor *PRINT* și *SUM*.

Reprezentările grafice ale procedurilor se pot conecta între ele, ieșirea unei proceduri devenind intrarea (para-

Fig. 6.2. Reprezentarea instrucțiunilor *PRINT* și *SUM*

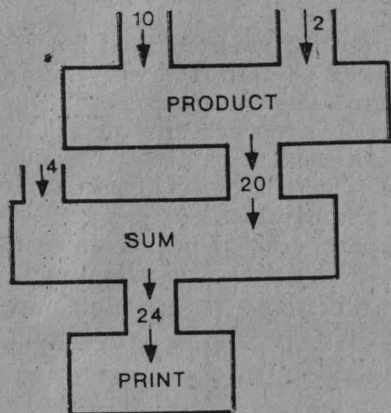
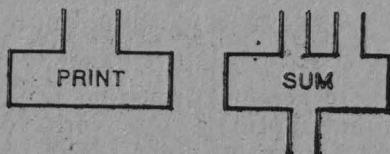


Fig. 6.3. Diagrama unei instrucțiuni complete

metrul de intrare) alteia. Astfel diagrama pentru instrucțiunea analizată va arăta ca în figura 6.3 (săgețile indică fluxul informațional în diagramă).

Trebuie remarcat faptul că atunci când se definește o nouă procedură liniile *LOGO* care se introduc nu se evaluează, fiind numai memorate ca parte a procedurii care se definește.

Mesaaje de eroare. După cum s-a subliniat, *LOGO* știe dinainte câți parametri de intrare sînt necesari pentru fiecare procedură. Ce se va întîmpla însă dacă unei proceduri îi va fi conferit un număr greșit de parametri? Încercați, de exemplu, *PRINT* și apoi *CR*. Se va afișa: „*PREA PUȚINE INTRĂRI ÎN PRINT*”.

Prin aceasta *LOGO* comunică simultan două lucruri: pe de o parte faptul că se întîmplă ceva cu o anumită procedură (prea puțini parametri, iar pe de altă parte, numele particular al procedurii care este în suferință. În exemplul dat este clar care procedură a fost implicată, fiind folosită doar una singură. Dar iată alt exemplu:

? PRINT REMAINDER PRODUCT 4 5
PREA PUȚINE INTRĂRI ÎN REMAINDER

În acest caz, mesajul *LOGO* este foarte folositor, subliniind faptul că este vorba de procedura *REMAINDER* care are nevoie de încă un subiect (și nu *PRODUCT* sau *PRINT*).

O greșeală frecventă pe care o fac programatorii începători este aceea de a ignora ceea ce transmite un anumit mesaj de eroare, de a nu încerca să-și imagineze problema reală care apare sau de a trece prea repede peste mesaj fără a desprinde din el foloasele potențiale. În *LOGO* mesajele de eroare sînt mai descriptive și mai la obiect, comparativ cu *BASIC* (de exemplu, în limbajul *BASIC* poate să apară deseori un mesaj de eroare care referă o anumită linie de program, iar eroarea să se găsească în altă linie).

Ce mesaj de eroare apare atunci cînd se introduc mai mulți parametri de intrare pentru o procedură? De exemplu,

? PRINT 2 3
2
NU AI SPUS CE SĂ FAC CU 3

Deci *LOGO* îndeplinește instrucțiunea *PRINT 2*, apoi găsește pe linie numărul 3, care este în plus.

Cuvinte și liste. Pînă acum exemplele date s-au referit la numere, aritmetică și grafică. Mai există prejudecata conform căreia calculatoarele pot realiza numai calcule dar, de fapt, mult mai interesantă este utilizarea calculatoarelor pentru a prelucra și alte informații în afară de numere.

Să presupunem că dorim ca *LOGO* să afișeze cuvîntul *SALUT*.

? PRINT SALUT
NU ȘTIU CUM SĂ SALUT

Deci *LOGO* interpretează *SALUT* ca fiind numele unei proceduri.

Mesajul de eroare apărut arată că nu există nici o procedură cu numele *SALUT* în repertoriu lui *LOGO*. Cînd limbajul evaluează instrucțiuni, el interpretează cuvintele

simplu ca nume de proceduri. Pentru ca un cuvînt să fie tratat ca *el însuși* trebuie să fie precedat de ghilimele ("):

```
? PRINT "SALUT
```

```
SALUT
```

Iată, de ce ghilimelele sînt utilizate în acest scop în LOGO: în informatică ghilimelele în fața unui cuvînt înseamnă prevenirea evaluării cuvîntului.

Pînă acum am văzut că numere se **evaluatează** prin ele însele, nefiind necesară inserarea ghilimelelor în fața lor. Acest fapt nu prezintă nici o contradicție, includerea ghilimelelor în fața numerelor neimplicînd nici o schimbare. De exemplu,

```
? PRINT SUM "2 "3
```

```
5
```

Se observă că, spre deosebire de alte limbaje de programare (*BASIC*, de exemplu), în LOGO ghilimelele se plasează doar în fața cuvîntului, fapt ce nu reprezintă o sintaxă arbitrară în cadrul sistemului studiat, ci reflectă faptul că un *cuvînt* LOGO nu este același lucru cu un *șir de caractere* folosit în alte limbaje. În LOGO mai multe cuvinte se pot combina, formînd astfel o *listă*. Cel mai simplu mod de a realiza o listă este de a închide cuvintele între paranteze pătrate, prin aceasta lista evaluîndu-se prin ea însăși:

```
? PRINT [CE MAI FACI?]
```

```
CE MAI FACI?
```

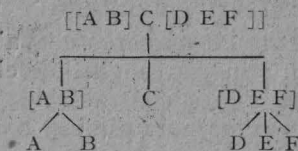
Parantezele drepte aplicate unei liste nu fac parte din listă, ele avînd drept scop atît delimitarea listei, cît și cotairea ei (evaluarea prin ea însăși).

Dacă parantezele drepte aplicate unei liste au același efect ca ghilimelele din fața unui cuvînt (adică evaluarea prin însăși lista sau însuși cuvîntul respectiv) atunci ce înseamnă evaluarea unei liste? Ei bine, fiecare linie de instrucțiuni care se introduce în LOGO este, de fapt, o listă care se **evaluatează** prin invocarea procedurilor care o compun.

În exemplul dat lista conține trei membri, fiecare din el fiind un cuvînt (primul este CE). Dar membrii unei liste pot fi nu numai cuvinte, ci și liste. Faptul că o listă poate

avea ca membru o altă listă conduce la o mare flexibilitate a modului de grupare a informațiilor, folosind ca tehnică listele.

O listă de tipul $[[A B] C [D E F]]$ conține trei membri, dintre care doi (primul și ultimul) sînt liste. O listă poate fi reprezentată și cu ajutorul unei diagrame de tip arbore, ca în exemplul următor:



În *LOGO* sînt două tipuri de informații care se pot prelucra — cuvintele și listele (numerele reprezintă, după cum am văzut, un caz special de cuvinte). Numele (denumirea) care se utilizează atît pentru cuvinte, cît și pentru liste este *obiect*.

O listă formată numai din cuvinte se mai numește și *frază* (*sentence*) sau *listă plată* (*flat*), diagrama de tip arbore în acest caz, prezentînd un singur nivel. Numele *frază* sugerează faptul că lista plată este utilizată deseori (deși nu totdeauna) pentru reprezentarea frazelor din limbajul curent. O frază în *LOGO* este un tip special de cuvînt.

Descrierea unei proceduri. În descrierea procedurilor o afirmație de tipul „SUM adună două numere”, deși nu este falsă, este total inadecvată. Iată în schimb un mod adecvat de descriere:

„SUM este o operație. Sînt necesari doi parametri de intrare (amîndoi trebuie să fie numere). SUM furnizează rezultatul adunării celor doi parametri”.

Iată și alt exemplu:

„PRINT este o comandă care necesită un parametru. Parametrul poate fi orice obiect. Efectul comenzii PRINT este afișarea pe ecran a obiectului parametru”.

Metoda folosită deci pentru descrierea unei proceduri trebuie să urmeze următorii pași:

1. Comandă sau operație?
2. Câți parametri de intrare?

3. Care este tipul obiectului pentru fiecare parametru ?

4. Dacă procedura este o operație care este rezultatul ei (ce furnizează)? Dacă procedura este o comandă ce efect va avea ?

Prelucrarea cuvintelor și listelor. LOGO conține o serie de operații care au drept scop extragerea obiectelor și dispunerea lor laolaltă. Cuvintele sînt formate din caractere (litere, cifre, semne de punctuație), dar un caracter nu reprezintă al treilea tip de obiect, el fiind de fapt un cuvînt format dintr-un singur caracter.

FIRST este o operație care necesită un parametru. (Parametrul poate fi orice obiect nenul). *FIRST* va furniza primul membru al parametrului, dacă acesta este o listă, sau primul caracter al parametrului, dacă acesta este un cuvînt. De exemplu,

```
? PRINT FIRST "SALUT
```

```
S
```

```
? PRINT FIRST [CE MAI FACI?]
```

```
CE
```

BUTFIRST este o operație care necesită un parametru. (Acesta poate fi orice obiect nenul). Rezultatul operației va fi o listă care conține toți membrii listei, în afară de primul (dacă parametrul a fost o listă), sau un cuvînt conținînd toate caracterele în afară de primul (dacă parametrul a fost un cuvînt).

```
? PRINT BUTFIRST "SALUT
```

```
ALUT
```

```
? PRINT BUTFIRST [CE MAI FACI?]
```

```
MAI FACI?
```

De remarcă faptul că *FIRST*-ul unei liste poate fi un cuvînt, dar *BUTFIRST*-ul oricărui obiect va fi totdeauna alt obiect de același tip. Atunci cînd operația *BUTFIRST* se aplică unui obiect care conține numai un singur membru, LOGO va afișa o linie fără nici un element.

```
? PRINT BUTFIRST "A
```

```
? PRINT BUTFIRST [SALUT]
```

```
?
```

În primul caz linia reprezintă un cuvânt „gol“, iar în al doilea caz o listă „goală“. Într-o instrucțiune se poate indica un cuvânt „gol“ utilizând ghilimelele cu un spațiu, iar o listă „goală“ prin două paranteze pătrate, care nu închid nimic între ele:

? PRINT " PRINT []

?

Similar cu *FIRST* și *BUTFIRST* există operațiile *LAST* (al cărui rezultat este reprezentat de ultimul membru al parametrului) și *BUTLAST* (al cărui rezultat este reprezentat de toți membrii parametrului, în afară de ultimul).

Cu ajutorul operațiilor *FIRST*, *LAST*, *BUTFIRST* și *BUTLAST* se extrag elemente din obiecte. Spre deosebire de ele, *SENTENCE* este o operație prin intermediul căreia se dispun la un loc mai multe obiecte. *SENTENCE* necesită doi parametri, care pot fi reprezentați de orice obiect. Rezultatul furnizat va fi întotdeauna o listă. În funcție de obiectele care formează cei doi parametri, precum și de tipul listei — plată sau nu — care reprezintă unul sau ambii parametri, rezultatul furnizat și obținut prin concatenarea obiectelor celor doi parametri va fi sau nu o listă plată.

Exemple:

? PRINT SENTENCE "A "MERGE

A MERGE

? PRINT SENTENCE [CE MAI] [FACI?]

CE MAI FACI?

? PRINT SENTENCE [CE MAI] "FACI?

CE MAI FACI?

? PRINT SENTENCE [] [CE MAI FACI?]

CE MAI FACI?

? PRINT SENTENCE [[LISTA 1A] [LISTA 1B]] [[LISTA 2A]
[LISTA 2B]]

[LISTA 1A] [LISTA 1B] [LISTA 2A] [LISTA 2B]

? PRINT SENTENCE [LISTA 1A] [[LISTA 2A] [LISTA 2B]]

LISTA 1A [LISTA 2A] [LISTA 2B]

Ultimul exemplu arată rezultatul folosirii operației *SENTENCE* în cazul în care primul parametru este reprezentat de o listă plată, iar al doilea — de una neplată.

WORD este o operație care necesită doi parametri. Este necesar ca amândoi parametri să fie cuvinte, între ele putînd fi și cuvîntul „gol”. Rezultatul furnizat de operația *WORD* va fi un cuvînt format prin concatenarea celor două cuvinte-parametru.

Example:

```
? PRINT WORD "MI "RE
```

MIRE

```
? PRINT WORD "MI [RE LA]
```

WORD NU MERGE CU [RE LA] CA INPUT

Operațiile descrise pot fi combinate în același mod în care s-au combinat *SUM* și *PRODUCT*. Să analizăm, de exemplu, următoarea linie *LOGO* care are ca rezultat afișarea cuvîntului *LOGO*:

```
? PRINT WORD WORD LAST "COCOSUL FIRST BUTFIRST  
"COMPUTER FIRST [GO TO ESTE O INSTRUCȚIUNE]
```

LOGO

Se ține seama de faptul că numerele sînt cuvinte, deci cu ele se pot combina operații aritmetice:

```
? PRINT WORD SUM 2 3 PRODUCT 2 3
```

56

```
? PRINT SUM WORD 2 3 PRODUCT 2 3
```

29

```
? PRINT SENTENCE SUM 2 3 WORD 2 3
```

5 23

COUNT este o operație care necesită un parametru. (Parametrul poate fi orice obiect). Rezultatul furnizat este un număr care indică lungimea parametrului. Dacă parametrul este un cuvînt, rezultatul va fi numărul de caractere ale cuvîntului. În situația că parametrul este o listă, rezultatul va fi numărul de membri ai listei.

```

? PRINT COUNT "SALUT
5
? PRINT COUNT [CE MAI FACI?]
3
? PRINT COUNT []
0

```

Aritmetica — formă specială de evaluare. Am văzut cum se evaluează o linie *LOGO*: se cercetează primul cuvânt și apoi al doilea, pînă se ajunge la sfîrșitul liniei, evaluarea avînd loc în continuare de la dreapta la stînga.

Deși acest proces este în general valabil, *LOGO* oferă și forme speciale de evaluare în scopul facilitării tasterii (introducerii) unor lucruri. Unul din aceste cazuri se referă la operații aritmetice infix (cu semne între numere). În loc de *PRINT SUM 2 3* se poate folosi *PRINT 2 + 3*. Cînd se utilizează operațiile infix se aplică următoarele reguli uzuale: înmulțirea (semnul „*“) și împărțirea (semnul „/“) se realizează înaintea adunării și scăderii dacă nu sînt folosite parantezele. Cu alte cuvinte, $2 + 3 \times 4$ înseamnă $2 + (3 \times 4)$, în timp ce $2 \times 3 + 4$ înseamnă $(2 \times 3) + 4$. De menționat că aceste probleme nu apar cînd sînt utilizate operațiile prefix. De exemplu, în fiecare din expresiile următoare se indică precis ordinea dorită a operațiilor:

```

SUM 2 PRODUCT 3 4
PRODUCT SUM 2 3 4
SUM PRODUCT 2 3 4
PRODUCT 2 SUM 3 4

```

A doua formă de evaluare se realizează în cazul anumitor operații ce necesită doi parametri, dar care pot utiliza și parametri suplimentari, folosind paranteze rotunde în jurul numelui procedurii și al parametrilor. De exemplu,

```

? PRINT SUM 2 3 4
5
NU AI SPUS CE SA FAC CU 4
? PRINT (SUM 2 3 4)
9

```

6.2. VARIABLE

S-a observat că noile proceduri definite (pînă acum) nu prezintă parametri și, în consecință, vor face absolut același lucru de fiecare dată cînd vor fi utilizate. Se va constata că în *LOGO* este posibilă depășirea acestei limitări.

Să definim o nouă procedură numită *SALUT* în felul următor:

```
? TO SALUT
> PRINT "SALUT
> PRINT [CE MAI FACI?]
> END
?
```

În timpul definirii unei noi comenzi, promptul *LOGO* își schimbă forma din semnul „?” în semnul „>”. Aceasta are drept scop atenționarea, atît asupra faptului că se definește corpul unei noi proceduri, cît și asupra faptului că, în acest context, liniile *LOGO* nu se evaluează în momentul introducerii, fiind doar memorate. Ele se vor evalua numai atunci cînd procedura va fi invocată.

Să încercăm să scriem acum o nouă procedură asemănătoare, dar care va avea ca parametru numele unei persoane și va funcționa astfel:

```
? SALUT1 "ALEX
SALUT, ALEX
CE MAI FACI?
? SALUT1 "MARIA
SALUT, MARIA
CE MAI FACI?
?
```

Procedura va fi asemănătoare cu procedura *SALUT*, dar rezultatul ei va depinde de parametrul dat:

```
? TO SALUT1 NUME
> PRINT SENTENCE "SALUT, THING1 NUME
> PRINT [CE MAI FACI?]
> END
?
```

Elementul de noutate în definierea procedurii *SALUT1* este, în primul rând, folosirea lui *NUME* după numele procedurii. Această adăugire comunică faptul că procedura *SALUT1* va necesita un parametru, iar numele acestui parametru va fi *NUME*. Parametrul respectiv poate fi imaginat ca un sertar sau o locație de memorie; când procedura *SALUT1* va fi invocată, un cuvânt (ca *ALEX* sau *MARIA*) va fi „pus“ în sertarul sau locația de memorie numită *NUME*.

Se observă că în fața numelui *NUME* este pus semnul „:“. Acest semn are o însemnătate specială pentru evaluatorul *LOGO*, și anume aceea de a face o distincție între un cuvânt, ca *SALUT1*, care este numele unei proceduri și cuvântul *NUME*, ce reprezintă numele unui parametru.

THING este o operație care necesită un parametru. Acesta trebuie să fie un cuvânt care să reprezinte numele unei locații de memorie (sertar). Rezultatul operației. *THING* va fi reprezentat de obiectul ce se găsește în locație. Termenul tehnic folosit pentru ceea ce am numit pînă acum sertar sau locație de memorie este *variabilă*. Orice *variabilă* poartă un nume și are un *lucru* sau o valoare atașată (corespunzător a ceea ce se găsește în interiorul locației). Atît numele, cît și lucrul fac parte din ceea ce este cunoscut sub denumirea generică de *variabilă*.

Cînd se introduce instrucțiunea:

SALUT1 "ALEX

LOGO va începe cu primul cuvînt de pe linie, *SALUT1*, pe care îl va interpreta ca numele unei proceduri. Sistemul *LOGO* va descoperi că procedurii *SALUT1* îi este necesar un parametru, astfel încît continuă cercetarea liniei, dînd peste un cuvînt cotat, „*ALEX*“. Deoarece, este cotat nu va necesita o evaluare, din care motiv cuvîntul *ALEX* va deveni parametrul de intrare pentru *SALUT1*. Acum *LOGO* este pregătit pentru a se putea invoca procedura *SALUT1*. Primul pas înainte de a evalua liniile de instrucțiuni în *SALUT1* este crearea unei variabile în care să fie pus parametrul. Această *variabilă* va avea numele *NUME*, iar ca lucru atașat, cuvîntul *ALEX*.

Examinarea valorii unei variabile este un fapt des întîlnit în procedurile *LOGO*; de aceea, există o abreviere

în acest scop. Astfel, în locul expresiei *THING* "*NUME*" se poate pune mai simplu: *NUME*, astfel încât vom putea înlocui linia din procedura *SALUT1* cu *PRINT SENTENCE* "*SALUT, :NUME*"

Se observă că semnul „:” reprezintă o abreviere a combinației *THING* și semnul “.”

Acum există posibilitatea de a scrie proceduri pentru realizarea, de exemplu, a figurilor geometrice cu latura variabilă.

Pentru pătrat:

```
TO PATRAT: LAT
REPEAT 4 [FORWARD: LAT LEFT 90]
END
```

Dacă se comandă *PATRAT 50*, se va desena un pătrat cu latura de 50, iar dacă se comandă *PATRAT 30*, se va desena un pătrat cu latura de 30.

O procedură poate fi definită cu mai multe variabile. Iată o procedură prin intermediul căreia se poate realiza o figură regulată cu latura variabilă și cu orice număr de laturi:

```
TO FIGURA: NR: LAT
REPEAT: NR [FORWARD: LAT LEFT 360/: NR]
END
```

Cu *FIGURA 4 50* se va desena un pătrat cu latura 50, iar cu *FIGURA 6 30* se va desena un hexagon cu latura 30.

Conversații cu calculatorul în LOGO. Pentru a realiza o conversație cu calculatorul trebuie definită o procedură interactivă, cu ajutorul căreia să se citească ceva introdus de la tastatură (acest deziderat se asigură prin intermediul operației *READLIST*). Operația menționată nu necesită nici un parametru și furnizează întotdeauna o listă care conține ceea ce s-a introdus de la tastatură (până la acționarea tastei *CR*). *READLIST* așteaptă introducerea liniei, apoi furnizează ceea ce s-a tastat.

Iată și o procedură cu care se poate realiza o conversație cu calculatorul:

```
TO CONVERSATIE
TEXTSCREEN PRINT [SALUT! CUM TE CHEAMA?]
PRINT SENTENCE [CE MAI FACI,] WORD FIRST READLIST?
PRINT FIRST LIST [ASTA E BINE READLIST]
END
```

Definirea de noi operații: Pînă acum procedurile definite erau comenzi, ele avînd un anumit efect (de exemplu, afișau ceva pe ecran), și nu furnizau un rezultat care să fie utilizat cu alte proceduri. Se pot, însă defini și operații noi. Iată un exemplu prin intermediul căruia se poate extrage al doilea membru al unei liste:

```
TO ALDOILEA: LUCRU
  OUPUT FIRST BUTFIRST: LUCRU
END
```

Procedura ALDOILEA se poate aplica astfel:

```
? PRINT ALDOILEA [CE MAI FACI?]
MAI
?
```

Se observă că procedura ALDOILEA folosește comanda *OUTPUT*.

Aceasta poate fi utilizată numai în corpul definirii unei proceduri și nu ca procedură de prim nivel. *OUTPUT* necesită un parametru care poate fi orice obiect. Efectul comenzii *OUTPUT* este acela de a face ca obiectul introdus ca parametru al ei să devină rezultat al procedurii care se definește.

Prezentăm în continuare cîteva reguli și tehnici de folosire a variabilelor în proceduri.

Dacă o procedură cu parametru invocă o altă procedură ca subprocedură, atunci este posibil ca ele să-și împartă variabilele (adică să le folosească atît una, cît și cealaltă). Este de asemenea posibil ca procedurile să conțină variabile separate.

Dacă o procedură face o referire la o variabilă care nu îi aparține, atunci *LOGO* caută o variabilă cu acel nume în superprocedura relativă la această procedură.

Să presupunem, de exemplu, că procedura *A* invocă procedura *B* și *B* invocă pe *C*. Să mai presupunem că o instrucțiune în procedura *C* referă o variabilă *V*. În primul rînd, *LOGO* încearcă să găsească o variabilă denumită *V* care aparține lui *C*. Dacă nu reușește (nu există nici o variabilă *V* în procedura *C*), atunci *LOGO* încearcă să găsească o variabilă *V* care aparține procedurii *C*. În final, dacă nici *C* și nici *B* nu conțin o variabilă numită *V*, atunci *LOGO* va căuta o astfel de variabilă în procedura *A*.

Variabilele care aparțin unei proceduri sînt *temporare*. Ele există numai atît timp cît procedura este activă. Dacă o procedură referă o variabilă cu același nume ca aceea care este referită de superprocedura sa, atunci variabila din superprocedură este temporar „ascunsă” în timpul rulării subprocedurii.

Modificarea valorii unei variabile. Pentru a schimba (modifica) lucrul atașat variabilei referite într-o procedură se poate folosi comanda *MAKE*. Aceasta necesită doi parametri: primul trebuie să fie un cuvînt care reprezintă numele variabilei, la fel ca parametrul pentru *THING*, iar al doilea poate fi orice obiect.

Comanda *MAKE* creează variabila cu numele respectiv (primul parametru) și depune în locația corespunzătoare variabilei lucrul care urmează după numele variabilei. Exemplu:

```
MAKE "A 50
```

Se observă faptul că *MAKE* este într-un fel similară cu instrucțiunea *BASIC LET*. Exemplul dat se poate „traduce” în *BASIC* prin *LET A = 50*. Deosebirea constă în faptul că, în *LOGO*, variabilei *A* i se poate atașa orice lucru (valoare, cuvînt, listă), în timp ce în *BASIC* trebuie explicitat dacă este vorba de o valoare numerică sau un șir de caractere.

De asemenea, este necesar să se observe faptul că, fiind vorba de „adresa de destinație” a unei valori, înaintea numelui variabilei se va pune semnul ghilimele și nu semnul „:”.

Conținutul variabilei se poate afișa. De exemplu,

```
? MAKE "A 20 PRINT: A PRINT: A+3
```

```
20
```

```
23
```

Variabile locale și variabile globale. Variabilele care apar ca parametri de intrare într-o procedură sînt create de procedură în momentul apelării și folosite în timpul execuției procedurii respective; la terminarea procedurii, aceste variabile sînt „distruse” (deci, conținutul lor nu se mai poate

utiliza). Din acest motiv, variabilele care sînt declarate ca parametri de intrare în proceduri se mai numesc *variabile locale*.

Deoarece variabile locale sînt create de fiecare procedură, se poate utiliza același nume de variabilă locală în diferite proceduri, fără ca prin aceasta să se creeze confuzii.

Variabila *A* definită anterior prin comanda *MAKE* poate fi însă utilizată de mai multe proceduri independente sau chiar de unele comenzi *LOGO* în afara procedurilor, fiind vorba deci de o *variabilă globală*. Variabilele globale pot fi create în proceduri sau în afara lor cu ajutorul instrucțiunilor *MAKE*. De remarcat că *MAKE* poate fi utilizată și pentru modificarea conținutului variabilelor (globale sau locale). Iată o procedură cu ajutorul căreia se poate număra pînă la un anumit număr indicat de utilizator cînd invocă procedura :

```
TO NUMARA: N
TEXTSCREEN MAKE "NUM 0
REPEAT :N [MAKE "NUM :NUM+1 PRINT :NUM]
END
```

Se remarcă faptul că variabila *N* este locală, fiind în lista de intrări a procedurii. În schimb, variabila *NUM* este o variabilă globală. Dacă introducem comanda *PRINT :NUM*, se va afișa ultima valoare care a fost numărată, dar dacă introducem comanda *PRINT :N*, se va afișa mesajul: "N nedefinit", deoarece după încheierea execuției procedurii variabila ei locală nu mai este utilizabilă.

Cu următoarele două proceduri se pot calcula suma și, respectiv, produsul primelor *N* numere naturale. Se observă că variabila pentru sumă (*S*) trebuie inițializată totdeauna cu 0, iar variabila pentru produs (*P*), cu 1. Se mai observă de asemenea că atît pentru calcularea sumei, cît și a produsului este necesară folosirea unei variabile auxiliare (*C*), care va îndeplini o funcție asemănătoare variabilei de tip contor (număr curent) dintr-un ciclu *FOR-NEXT* într-un program *BASIC* (*FOR C = 1 TO N*).

Iată procedurile :

```
TO SUMA :N
MAKE "S 0
MAKE "NC 0
REPEAT :N [MAKE "NC :NC+1 MAKE "S :S+:NC]
PRINT: S
END
TO PROD :N
MAKE "P 1
MAKE "NC 0
REPEAT :N [MAKE "NC :NC+1 MAKE "P :P* :NC]
PRINT: P
END
```

6.3. PREDICATE

În practică există o categorie specială de întrebări al căror răspuns este „da“ sau „nu“. Categoria corespunzătoare în *LOGO* este *predicatul*.

Un predicat este o operație care furnizează totdeauna ca rezultat cuvântul *TRUE* (adevărat) sau cuvântul *FALSE* (fals).

LISTP este un predicat care necesită un parametru de intrare, ce poate fi orice obiect. Rezultatul furnizat este *TRUE* dacă parametrul de intrare a fost o listă sau *FALSE* dacă parametrul de intrare a fost un cuvânt.

WORDP este un predicat care necesită un parametru de intrare. Acesta poate fi orice obiect. Rezultatul furnizat este *TRUE* dacă parametrul de intrare a fost un cuvânt sau *FALSE* dacă parametrul de intrare a fost o listă. Exemplu:

```
? PRINT WORDP "NUME
TRUE
```

EMPTYP este un predicat care necesită un parametru de intrare, ce poate fi orice obiect. Rezultatul furnizat este *TRUE* dacă parametrul de intrare este un cuvânt vid sau o listă vidă. Dacă parametrul de intrare este orice altceva, atunci rezultatul furnizat este *FALSE*.

Exemple :

```
? PRINT EMPTY []
```

```
TRUE
```

```
? PRINT EMPTY 0
```

```
FALSE
```

NUMBERP este un predicat care necesită un parametru de intrare, care poate fi orice obiect. Rezultatul furnizat este *TRUE* dacă parametrul de intrare este un număr și *FALSE* în caz contrar.

EQUALP este un predicat care necesită doi parametri de intrare. Rezultatul furnizat este *TRUE* dacă cei doi parametri de intrare sînt identici sau dacă amîndoi reprezintă numere egale. Completarea este necesară deoarece, de exemplu, 3 și 3.0 sînt numeric egale, deși nu reprezintă cuvinte egale. O listă nu va fi niciodată egală cu un cuvînt.

Exemple :

```
? PRINT EQUALP 3 3.0
```

```
TRUE
```

```
? PRINT EQUALP "NUME" [NUME]
```

```
FALSE
```

```
? PRINT EQUALP "NUME" FIRST [NUME]
```

```
TRUE
```

Semnul egal (=) poate fi folosit ca o operație infix echivalentă cu *EQUALP*.

Exemple :

```
? PRINT "NUME=FIRST" [NUME]
```

```
TRUE
```

```
? PRINT 2=3
```

```
FALSE
```

MEMBERP este un predicat care necesită doi parametri de intrare. Primul parametru de intrare poate fi orice obiect, iar al doilea trebuie să fie o listă. Rezultatul furnizat va fi *TRUE* dacă primul parametru de intrare este membru al celui de-al doilea parametru.

Exemple :

```
? PRINT MEMBERP "NUMELE [CARE ESTE NUMELE TAU?]  
TRUE
```

```
? PRINT MEMBERP [ESTE NUMELE] [CARE ESTE NUMELE  
TAU?]
```

```
FALSE
```

```
? PRINT MEMBERP [ESTE NUMELE] [CARE[ESTE NUMELE]  
TAU?]
```

```
TRUE
```

LESSP și *GREATERP* sînt predicate care necesită cîte doi parametri de intrare, ambele fiind numere. Rezultatul pentru *LESSP* este *TRUE* dacă primul parametru este numeric mai mic decît al doilea, iar rezultatul pentru *GREATERP* este *TRUE* dacă primul parametru este mai mare decît al doilea. În orice alt caz rezultatul va fi *FALSE*. Sînt permise și formele infix pentru *LESSP* (<) și *GREATERP* (>).

Definirea de noi predicate. Iată două exemple care arată cum se pot defini noi predicate :

A. *Definirea unei proceduri care indică dacă o literă este sau nu o vocală :*

```
TO VOCALA :LITERA
```

```
OUTPUT MEMBERP "LITERA [A E I O U]
```

```
END
```

```
? PRINT VOCALA "E
```

```
TRUE
```

```
? PRINT VOCALA "M
```

```
FALSE
```

B. *Definirea unei proceduri care indică dacă un număr este par :*

```
TO PAR :NUMAR
```

```
OUTPUT EQUALP REMAINDER :NUMAR 2 0
```

```
END
```

```
? PRINT PAR 5
```

```
FALSE
```

```
? PRINT PAR 118
```

```
TRUE
```

6.4. EVALUAREA CONDIȚIONALĂ

Principala utilizare a predicatelor este aceea de a evalua parametrul de intrare pentru procedura primitivă *IF*. Această procedură poate fi utilizată în diferite forme.

În primul rând *IF* poate fi folosită ca o comandă cu doi parametri de intrare. Primul parametru de intrare trebuie să fie ori cuvântul *TRUE*, ori cuvântul *FALSE*, iar al doilea parametru trebuie să fie o listă care să conțină instrucțiuni *LOGO*. Dacă primul parametru este *TRUE*, atunci efectul lui *IF* este acela de a evalua instrucțiunile conținute în cel de-al doilea parametru. Dacă primul parametru este *FALSE*, atunci această formă a lui *IF* nu are nici un efect.

Exemplu:

```
? IF EQUALP 12 4*3 [PRINT "CORECT]
CORECT
? IF EQUALP 4 2 [PRINT "INCORECT]
?
```

În *LOGO*, procedura primitivă *IF* are următoarea formă generală:

IF condiție [lista1 de instrucțiuni] [lista2 de instrucțiuni].

Deci, se observă că *IF* poate avea forma și cu 3 parametri de intrare. La întâlnirea acestei instrucțiuni, *LOGO* evaluează condiția. Dacă aceasta este adevărată (*TRUE*), atunci se execută lista1 de instrucțiuni, dacă nu (*FALSE*), se execută lista2 de instrucțiuni.

În *LOGO* există o comandă care termină evaluarea procedurii în care ea apare. Această comandă se numește *OUTPUT*. Practic ea transformă o procedură într-o operație. Iată o procedură cu care se poate calcula modulul unui număr prin intermediul comenzii *OUTPUT*:

```
TO MODUL :A
IF: A<0 [OUTPUT -1 * :A] [OUTPUT :A]
END
```

Oprirea unei proceduri. Uneori (de exemplu, în cazul excluderii unor cazuri extreme) apare necesitatea opririi unei proceduri. În acest scop se utilizează comanda *STOP*.

Această comandă nu are nici un parametru de intrare și se poate folosi numai în interiorul unei proceduri, nefiind permisă utilizarea ei imediat după apariția promptului *LOGO*. Efectul comenzii *STOP* este acela de a termina evaluarea procedurii în care este folosită, fără ca următoarele instrucțiuni din aceeași procedură să mai fie executate.

Trebuie notat faptul că *STOP* nu oprește toate procedurile active. De exemplu, dacă procedura *A* invocă procedura *B* și în procedura *B* există o comandă *STOP*, atunci procedura *A* va continua după punctul în care este invocată procedura *B*.

Amintim de asemenea faptul că și comanda *OUTPUT* oprește procedura care o invocă. Diferența constă în următorul fapt: dacă se scrie o operație care trebuie să furnizeze ceva (să aibe o ieșire), atunci se va utiliza *OUTPUT*; în cazul că se scrie o comandă care nu furnizează un rezultat, atunci se va utiliza *STOP*.

Exemplu de utilizare: dacă într-o procedură se întâlnește instrucțiunea *IF : L > 10 [STOP]*, în momentul în care variabila: *L* devine mai mare ca 10 procedura se va opri.

6.5. RECURSIA

Să presupunem că procedura *PATRAT* este definită astfel:

```
TO PATRAT
REPEAT 4 [FORWARD 80 LEFT 90]
PATRAT
END
```

Procedura se deosebește de alta obișnuită destinată definiției unui pătrat prin aceea că are în plus autoapelarea ei, devenind astfel o procedură recursivă. Această procedură va continua la nesfârșit, desenând pătrate peste același contur inițial dacă nu se intervine din exterior, prin acționarea simultană a tastelor *CAPS SHIFT* și *SPAGE* prin care se obține *STOP*.

Recursia reprezintă deci posibilitatea de a utiliza o procedură ca parte a propriei sale definiții. *PATRAT* este o procedură recursivă în forma simplă.

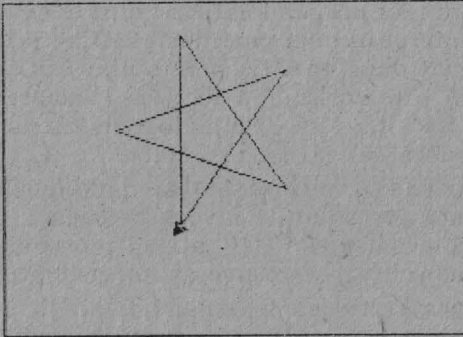


Fig. 6.4. Invocarea unei proceduri *POLI*

Să experimentăm următoarele proceduri grafice recursive:

```
TO POLI :LATURA :UNGHII
FORWARD :LATURA
RIGHT :UNGHII
POLI :LATURA :UNGHII
END
```

În fig. 6.4 poate fi urmărită forma generată prin invocarea procedurii *POLI* 80 144.

Procedura *POLISPI* va avea 3 parametri, folosindu-se și regula de *STOP*:

```
TO POLISPI LAT: :UNGHII :NR
IF :NR=0 [STOP]
FORWARD :LAT
RIGHT :UNGHII
POLISPI :LAT+1 :UNGHII :NR-1
END
```

Pot fi urmărite interesante spirale pe bază de pătrate, cu *POLISPI* 10 90 100 (vezi fig. 6.5a), pentagoane cu *POLISPI* 10 70 50 (vezi fig. 6.5b), triunghiuri cu *POLISPI* 10 120 80 (vezi fig. 6.5c) și stele cu *POLISPI* 10 140 50 (vezi fig. 6.5d).

Recursia la coadă. Cât timp rămâne activă fiecare procedură invocată necesită un anumit volum de memorie pentru a păstra în el anumite lucruri (de exemplu, variabilele locale). Deoarece procedura recursivă se poate autoinvoca de mai multe ori, această tehnică (recursivitatea) conduce la un

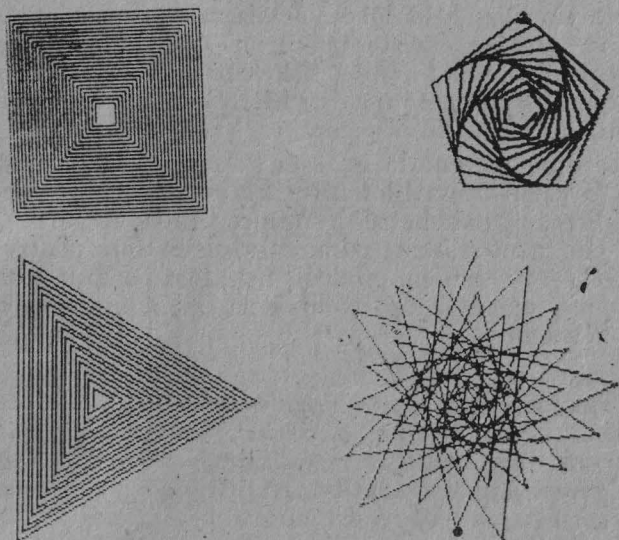


Fig. 6.5. Invocarea unor proceduri
POLISPI

consum mare de memorie. Dacă însă pasul recursiv este ultimul într-o procedură, atunci *LOGO* poate manipula acea procedură într-un mod special, prin care memoria este mai eficient utilizată, fără a se ieși în afara memoriei. Acest tip de procedură se numește *cu recursie la coadă*.

Operații cu recursie la coadă. În cazul unei operații care se autoapelează regula pentru recursia la coadă este puțin modificată, fără a fi suficient ca invocarea recursivă să se realizeze în ultima instrucțiune. Se va exemplifica acest lucru prin realizarea unei operații care furnizează ca rezultat factorialul unui număr:

```
TO FACT :N
IF :N=0 OUTPUT 1
OUTPUT :N * FACT :N-1
END
```

Se observă că se respectă regulile care definesc factorialul: $FACT(0) = 1$ și $FACT(N) = N * FACT(N - 1)$, adică $N! = N \times (N - 1)!$

De asemenea, calculul se realizează printr-o invocare recursivă care se găsește pe ultima instrucțiune. Limita calculatorului atunci când se folosește această procedură se găsește sub valoarea 35 (pentru PRINT FACT 35 se afișează mesajul „memorie insuficientă“).

Procedura prezentată nu este cu recursie la coadă, deoarece în cazul operațiilor invocarea recursivă trebuie utilizată direct ca parametru de intrare pentru operația *OUTPUT*. (În cazul studiat, parametrul de intrare pentru operația *OUTPUT* este un produs). Este însă posibilă transformarea unei operații fără recursie la coadă într-una cu recursie la coadă:

```
TO FACT :N
OUTPUT FACT1 :N 1 primul produs parțial
END
TO FACT1 :N :PROD
IF :N=0 [OUTPUT :PROD]
OUTPUT FACT1 (:N-1) (:N* :PROD)
END
```

În acest caz, recursia fiind la coadă, posibilitățile de calcul vor crește pînă la circa FACT 40, acestea depinzînd și de numărul de proceduri existente în memorie la momentul respectiv.

PRODUSE PROGRAM SPECIFICE
MICROCALCULATOARELOR

Răspîndirea și utilizarea microcalculatoarelor de tip PC pe scară largă se datoresc atît faptului că un microcalculator actual prezintă performanțe similare sau chiar superioare unui sistem de calcul din anii 60'-70', la un preț de circa 100-1 000 ori mai mic, cît și ușurinței în exploatare. (În acest sens, s-a adoptat termenul de *user friendly* — prietenos, familiar). De asemenea, este neîndoielnic că existența nenumăratelor programe de aplicații a jucat un rol determinant, acestea fiind considerate drept principalul factor al progresului înregistrat prin introducerea microcalculatoarelor în munca rutinieră, prelucrarea informațiilor și luarea pe această bază a deciziilor. Sarcini care pînă nu demult puteau fi îndeplinite numai manual se realizează acum mult mai repede. Are loc astfel o creștere a productivității muncii, precum și a calității produselor și serviciilor oferite.

Microcalculatoarele oferă moduri complet diferite de lucru, prin care rezultă o calitate superioară a rezultatelor față de cea obținută cu unelte manuale. Creșterea productivității muncii poate fi astfel măsurată atît în termeni cantitativi, cît și calitativi.

În legătură cu pachetele de programe disponibile pentru microcalculatoare este de remarcat faptul că ele nu au fost create ca urmare a adaptării și implementării unor programe de pe minicalculatoare sau sisteme de calcul, fiind în totalitate noi în ceea ce privește concepția și proiectarea. Acest fapt a avut mai multe cauze printre care: capacități de memorie internă foarte diferite; programele de aplicație pentru sisteme de calcul erau prea complexe și nu mai făceau față nevoilor tot mai mari ale economiilor naționale; pen-

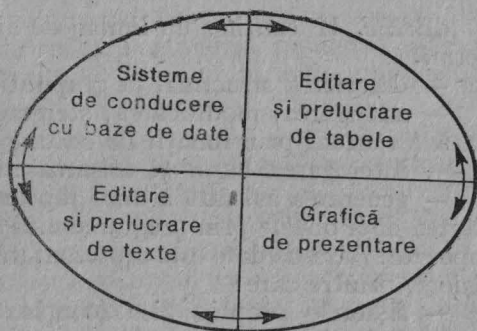
tru ca rezultatele să fie utilizabile în întreprinderi și birouri era nevoie, în cea mai mare parte, de prelucrare pe loturi.

Au rezultat astfel o serie întreagă de produse program specifice care au făcut posibilă o puternică descentralizare a prelucrărilor de date. Astfel, se au în vedere principalele tipuri de programe pentru gestiunea bazelor de date, editarea și prelucrarea de tabele, programe destinate calculului financiar-contabile, planificării, programe pentru analize economice complexe și conducerea proiectelor, sisteme de conducere și de luare a deciziilor, programe pentru editarea și prelucrarea de texte, programe pentru calcule tehnico-științifice în domeniul tehnic, cercetării științifice și statisticii economice, programe de simulare și modelare, programe pentru grafică economică, programe pentru realizarea de publicații etc.

Considerându-se faptul că microcalculatoarele de tip PC pe 16/32 biți (compatibile IBM PC, IBM PC XT și AT) vor reprezenta următorul pas și, probabil, vor domina aplicațiile în țara noastră în deceniul 1990-2000, se vor descrie principalele programe pentru aceste calculatoare. În sfârșit, deoarece în țările puternic dezvoltate industrial și-au făcut apariția microcalculatoare de tip PC pe 32 de biți (reprezentate în special prin noua familie de microcalculatoare IBM PS/2) se vor trece în revistă și câteva din produsele program existente în momentul de față și funcționabile pe aceste calculatoare, ca fiind de mare perspectivă în toate domeniile.

Dacă pînă la apariția și dezvoltarea microcalculatoarelor de tip PC pe 16 biți (1981-1982) pachetele de programe de aplicații se prezentau ca pachete de programe independente, după acest eveniment au început să devină accesibile *pachetele de programe integrate* (fig. 7.1). Prin integrarea software se înțelege facilitatea ca datele să poată fi transferate de la un dispozitiv (aplicație) software la altul. Primele pachete de programe integrate apărute se consideră a fi LOTUS 1-2-3 și Symphony. Acestea sînt utilizabile pe calculatoare IBM PC și, după cum se poate observa în figură, conțin programe pentru conducere cu baze de date, programe pentru editare și prelucrare de tabele financiar-contabile, editare și prelucrare de texte și programe pentru grafică economică.

Fig. 7.1. Integrarea software



Pe plan mondial se constată o mare dezvoltare a *pachetelor de programe integrate*, preconizându-se ca nu peste mult timp majoritatea programelor pentru realizarea de aplicații în economie să fie reprezentate de acest tip de programe. În plus, ele tind să înglobeze din ce în ce mai multe facilități și tipuri de aplicații. Datorită tuturor acestor considerente, conceptul de integrare software se va trata ca o problemă distinctă.

7.1. GESTIUNEA BAZELOR DE DATE

Un sistem de gestiune a bazelor de date (SGBD) oferă organizarea datelor în structuri complexe cu relații logice multiple între date, limbaje de descriere a acestor structuri de date (LDD) și de manipulare a lor (LMD) și are drept consecință o serie de avantaje în conducerea centralizată a oricărui proces economic. Astfel, datele pot fi standardizate, la ele pot avea acces complex și utilizatorii neprogramatori, se elimină redundanțele și incoerențele, se realizează un control unitar de restricții de acces etc.

dBASE II este un sistem de gestiune a bazelor de date relaționale, care rulează sub sistemul de operare CP/M pe microcalculatoare CUB/Z, JUNIOR, M118 și altele, înzestrate cu microprocesoare 8080 sau Z-80. A fost elaborat de firma Ashton-Tate (SUA), iar utilizarea este extrem de extinsă (peste un milion de utilizatori).

dBASE II conține un limbaj de dimensiuni reduse care permite:

- descrierea structurii (a grupului de tabele necesare);
- adăugarea, modificarea, ștergerea înregistrărilor din bază (eventual prin funcții de editare video);
- interogarea bazei și afișarea rezultatelor;
- generarea asistată a unor rapoarte simple, cu posibilitatea de a obține prin programe mai complexe orice fel de rapoarte. Baza de date este reprezentată fizic prin mai multe fișiere, printre care:

- fișierele propriu zise (conțin structura și datele);
- fișierele index (create cu ocazia declarării unor cîmpuri drept cheie unică); ele se utilizează pentru sortare logică și pentru acces rapid la înregistrări:

- fișiere în care sînt stocate programele, care — realizate cu ajutorul instrucțiunilor dBASE — pot fi foarte complexe;

- fișiere pentru variabilele cu care poate opera limbajul de interogare.

Limbajul oferit de dBASE conține deci limbajul de descriere a datelor (LDD) și cel de manipulare a datelor (LMD). El funcționează ca un interpretor, executîndu-se deci instrucțiune după instrucțiune. Permite și două tipuri de structuri:

- bucle de ciclare DO WHILE;
- blocuri executabile condiționat IF/DO CASE.

Sistemul dBASE II permite gruparea instrucțiunilor sale în fișiere de comenzi. Prin comanda DO se apelează și se lansează în execuție un asemenea program. Structurile DO WHILE, IF, DO CASE asigură scrierea în normele programării structurate.

Unul din avantajele lui dBASE II față de alte SGBD este deci acela că dBASE II nu apelează la alte limbaje în cadrul cărora să se insereze facilitățile limbajului de manipulare a datelor, LMD propriu. El oferă un mic limbaj cu care se execută toate funcțiile necesare utilizatorilor bazei de date.

Cu puțină experiență se pot aduce importante optimizări programelor scrise în dBASE II. Menționăm în conformitate cu asemenea tehnici:

a) Evitarea parcurgerii unui tabel de mai multe ori pentru a face operații diferite pe diverse coloane. Aceste

operații se pot executa deodată, în cadrul unei singure treceri prin fișier, prelucrând orizontal coloană după coloană, eventual folosind variabile sau fișiere special proiectate pentru totalizări, fără a utiliza comenzile dBASE de totalizare și contorizare. Se evită în acest mod accese inutile la dischete.

b) Apelarea unui fișier cu comenzi prin comanda DO aduce programul la începutul unei pagini de memorie de numai 1 Koctet, folosită de dBASE în acest scop. Dacă în cadrul programului, care poate fi mai lung de 1 Koctet, se scrie o structură de ciclare care depășește limita de 1 Koctet, atunci la execuție sistemul va proceda la nenumărate accese pentru a încărca când pagina anterioară, când pagina următoare, în scopul regăsirii limitelor buclei. Pentru a evita această situație se va urmări ca nici o buclă să nu „încalce” granița de 1 Koctet, fapt ce se poate realiza cu editorul de texte WS (Word Star), cu ajutorul căruia există posibilitatea de a elabora programe dBASE II.

Sistemul dBASE II poate citi fișiere COBOL, PASCAL, BASIC, C. De asemenea, are posibilitatea de a scrie fișiere utilizabile de către aceste compilatoare.

În concluzie, dBASE II beneficiază de simplitatea structurării logice a datelor într-o bază relațională în care toate fișierele sînt tabele omogene.

Produsul a evoluat însă sub sistemul MS-DOS pe calculatoare compatibile IBM PC. Față de dBASE II, dBASE III a extins la 10 numărul fișierelor bazei care pot fi deschise simultan; totodată permite existența permanentă în memorie a unor proceduri, scutind astfel apelul de pe disc cu comanda DO. Au fost extinse atît dimensiunile fizice ale bazei, cît și comenzile și funcțiile existente.

dBASE III PLUS permite accesul la rețele de PC-uri și extinde mai mult facilitățile dBASE.

În ultimul timp a apărut pe piața produselor software versiunea dBASE IV, care este net superioară celor precedente.

FOX-BASE este deja un compilator pentru limbajul dBASE, fiind astfel mult mai rapid.

Familiarizarea și lucrul cu dBASE II deschid drumul către utilizarea ulterioară a unor produse mai complexe. Dar chiar la dimensiunea actuală dBASE II permite scrie-

rea unor programe mari (modularizate eventual), care pot rezolva gestiunea economică de dimensiune medie a unor întreprinderi.

În tabelul 7.1 sînt prezentate comparativ cele mai utilizate pachete de programe destinate gestiunii bazelor de date pentru calculatoare personale evaluate.

Tabelul 7.1.

Sisteme de conducere cu baze de date relaționale

Nume pachet programe Facilități, performanțe	Q&A	Omnis Quartz	Data Ease	Paradox	Fox Base+
1	2	3	4	5	6
Introducere și validare date					
— operare prin comenzi	—	—	—	×	×
— operare prin liste de opțiuni	×	×	×	×	—
— validare date prin tabele „look up”	×	×	×	×	×
— verificare ordin de mărime date	×	×	×	×	×
— verificare dublare cheie index	×	×	×	×	×
— verificarea datei calendaristice	×	×	0	×	×
— verificarea introducerii orei	×	×	×	—	×
— număr formate per fișier	1	△	1	15	△
— număr ecrane/ferestre per format	10	12	16	15	△
Instrumente pentru dezvoltări					
— dicționar de date	—	×	—	—	—
— generator de aplicații	—	×	—	×	—
— limbaj procedural	—	×	0	×	×
— subrutine	—	×	—	×	×
— macroinstrucțiuni	×	×	—	×	×
— variabile	—	×	—	×	×
— modul „run-time”	—	×	—	×	×
Prelucrări și căutări în fișier					
— mai multe la unul	×	×	×	×	×
— unul la mai multe	—	×	×	×	×
— reactualizarea fișierelor multiple	—	×	×	×	×

Tabelul 7.1. (continuare)

1	2	3	4	5	6
— legături bidirecționale între tabele	—	×	×	×	×
— căutări condiționale	—	×	×	×	×
— căutări booleene	×	×	×	×	×
— căutări globale și reactuali- zări	×	×	0	×	×
Rapoarte					
— rapoarte multifişier	×	×	×	×	×
— ieşire rapoarte pe ecran	×	×	×	×	×
— ieşire rapoarte pe disc	×	×	×	×	×
— salvare formate rapoarte	×	×	×	×	×
— formare pentru poştă (eti- chete cu datele pentru fie- care corespondent)	×	×	×	×	×
— capete de tabel (la rapoar- te)	×	×	×	×	×
— rapoarte rezumat	×	×	×	×	×
— informații în josul paginii (la rapoarte)	×	×	×	×	—
— imprimarea datei	×	×	×	×	×
Capacitate					
— mărime fişier (în înregis- trări)	16 mili- oane	△	65535	2 mili- arde	1 mili- arde
— număr de cîmpuri pe înre- gistrare	2182	120	255	255	128
— număr de caractere pe înre- gistrare	16780	288000	4000	4000	4000
— număr fişiere pe bază de date	1	60	255	△	△
— număr indexări pe fişier	115	12	255	256	7
— număr cîmpuri pe index	1	120	1	255	100
— număr max. de fişiere de date deschise	△	60	32	△	10
— număr fişiere pe operații „Join”		12	△	△	10
— scriere/citire fişiere DIF	×	×	×	×	—
— scriere/citire fişiere ASCII	×	×	×	×	×
— cîmpuri de lungime varia- bilă	×	—	—	—	0
— cîmpuri multivaloare	—	—	×	—	—

× = Da; — = Nu; 0 = Deficitar; △ = Nelimitat

7.2. EDITARE ȘI PRELUCRARE DE TABELE

În esență, programele pentru editare și prelucrare de tabele (*spreadsheet programs*) calculează și afișează tabele de numere și nume. De aceea, ele sînt destinate analizei financiare și contabile, realizării unor situații și rapoarte de venituri și cheltuieli. În afară de aceste aplicații, prin facilitățile pe care le pun la dispoziție, aceste programe pot fi utilizate cu succes și pentru conducerea proiectelor, calculul tabelelor științifice și ingineresti, realizarea de modele economice, simulări și rezolvări de probleme.

Operarea se bazează pe o tabelă sau grilă, inițial vidă, compusă din celule (compartimente) organizate pe linii și coloane. Cu ajutorul unor comenzi simple, în celulele particulare ale tabelii se pot introduce date sau nume. De asemenea, există posibilitatea de a invoca expresii algebrice și formule care leagă o celulă de alta, o linie de alta sau o coloană de alta, astfel încît calculatorul să actualizeze rapid întreaga tabelă. Prin modificarea unuia sau mai multor parametri din celule (deci, pentru diferite situații), tabelele se pot reevalua și afișa într-un timp foarte scurt.

VU-CALC este un program pentru editare și prelucrare de tabele pentru calculatoare compatibile Sinclair ZX Spectrum. Tabela comportă 20 de linii marcate cu litere (începînd cu A) și 31 de coloane (numerotate de la 01 la 31), astfel încît fiecare celulă este definită în mod unic prin litera asociată liniei și prin numărul corespunzător coloanei. În partea superioară a ecranului sînt afișate două linii de comandă, iar la baza ecranului este plasată linia de introducere.

În oricare din etapele de folosire a programului se utilizează un cursor, reprezentat printr-un dreptunghi dispus în tabelă și care se poate deplasa, în vederea introducerii de date sau formule, în cele patru direcții (sus-jos, dreapta-stînga).

Programul acceptă patru tipuri de introduceri: text, date, formule și comenzi. Pentru introducerea unui text se poziționează cursorul în dreptul celulei la care se dorește plasarea textului, se acționează tasta „ și apoi se introduce

textul dorit. În vederea calculării unei valori cu o anumită formulă și plasării ei în tabelă, cursorul se poziționează în celula respectivă și se introduce formula dorită. Când formula apare în forma corectă, la baza ecranului se tastează ENTER. În mod automat, formula va fi evaluată și rezultatul obținut se va plasa în celula indicată de cursor. Formulele pot conține constante, referiri la numerele din alte celule și operatori aritmetici simpli (+, -, *, /). O formulă se poate aplica automat la mai multe celule.

Comenzile sînt introduse prin tastarea simbolului # și se referă la editarea, încărcarea, salvarea și tipărirea fișierelor. *Exemplu de comenzi*: #C (Compute) forțează recalcularea întregului tabel, în cazurile în care se modifică o formulă; #E (Edit) permite modificarea formulei dintr-o celulă sau înlocuirea cu altă formulă; #G (GO) deplasează cursorul într-o celulă specificată; #R (Repeat) permite repetarea conținutului unei celule în celulele dintr-un sector specificat; #S (Save) șterge ecranul și solicită utilizarea unui nume de fișier pentru salvarea tabelii pe caseta magnetică, etc.

MULTIPLAN este unul dintre cele mai utilizate programe de editare și prelucrare de tabele pentru calculatoare pe 8 hiți cu unități de discuri flexibile cu sistem de operare CP/M. Existența unor memorii externe cu acces direct (discuri flexibile), precum și memoria mai mare oferă posibilitatea lucrului cu o tabelă (grilă) mult mai mare, și anume: 63 de coloane și 255 de linii, permițînd analize financiar-contabile pentru un volum mult mai mare de date.

Ecranul comportă una sau mai multe ferestre pe tabel și o zonă de comenzi, mesaje și linii status. Mesajele explică acțiunea care va avea loc sau oferă explicații privind erorile, dacă acestea apar. Liniile status afișează coordonatele unei celule active, conținutul lor, procentajul de memorie rămasă la dispoziție, precum și numele tabelului. Pe tabel există în permanență marcată o celulă „activă”. Marcajul (cursorul) poate fi mutat în una din cele 4 direcții prin intermediul tastelor de direcții. Aceleași taste pot fi folosite și pentru vizualizarea conținutului ferestrei (*scroll*).

Oricînd este disponibil un *HELP* care explică comanda ce se execută, în momentul apelării lui oferind și alte informații suplimentare.

MULTIPLAN asigură următoarele facilități: introducere date, introducere text, introducere argumente pentru comenzi, editare texte sau formule, realizare de calcule, utilizare de funcții matematice, creare a unor capete de tabel, definirea sau ștergerea de nume pentru celule, modificare format numere, multiplicarea datelor sau formulelor mutare rînduri/coloane, salvare/încărcare fișiere, sortare alfabetică.

Sînt de asemenea puse la dispoziție numeroase formule: *AVERAGE* (media) *MAX*, *MIN*, *ABS*, *STDEV* (abaterea standard a valorilor), *COUNT*, *AND*, *OR*, *SUM*, *ATAN*, *COS*, *COLUMN* (numărul curent al coloanei), *FIXED* (n , m), *EXP*, *DOLLAR*, *IF*, *INDEX* (furnizează al n -lea element din vector), *INT*, *LEN*, *LN*, *LOG*, *LOOKUP*, *MID*, *MOD*, *NA*, *NOT*, *PI*, *REPT* (un text repetat de n ori), *ROUND*, *ROW* (rîndul curent), *SIGN*, *SIN*, *SQRT*, *TAN*, *TRUE*, *VALUE*.

Se acceptă următorii operatori în formule: +, -, ×, /, ↑, %, & (text concatenare), comparații < >.

Pentru alegerea comenzii se poate utiliza o listă de opțiuni. În acest scop este necesar să se selecteze o celulă activă, precum și o comandă (prin deplasarea cursorului spre un cuvînt-comandă și apoi *ENTER*, sau prin tastarea primei litere a unei comenzi), specificîndu-se parametrii comenzii. Tasta *CANCEL* se va folosi pentru reîntoarcerea la lista principală de opțiuni, *ENTER* pentru îndeplinirea comenzii, iar „?” pentru informații suplimentare.

Produsul 1-2-3 al firmei Lotus Corporation este un pachet de programe pentru editarea și prelucrarea de tabele pentru calculatoare compatibile IBM-PC, reprezentînd în prezent și cel mai comercializat pachet software pentru calculatoarele personale. Produsul menționat oferă un set de facilități mult mai numeroase față de *MULTIPLAN*, precum și o tabelă (grilă) de capacitate superioară. Astfel, tabelul conține 256 de coloane de 2 048 și rînduri (la prima versiune), adică un număr de 524 288 de celule care pot conține date, texte sau formule de calcul. Față de *MULTI-*

PLAN există posibilitatea ca informațiile să fie reprezentate și sub formă grafică, putîndu-se de asemenea organiza într-o bază de date. Utilizatorul poziționează cursorul pe ecran în dreptul unei celule, o examinează sau o încarcă cu date (formule sau funcții); el poate grupa mai multe celule într-o zonă care conține informații intercorelate în mod logic. Ca mod de operare, 1-2-3 beneficiază de două elemente: un meniu de comenzi și un sistem de ajutor în caz de nevoie (HELP), deosebit de dezvoltat. La fel ca la MULTIPLAN, selectarea comenzii se realizează prin poziționarea cursorului pe comanda dorită sau prin testarea inițialei acelei comenzi. Lista de comenzi, organizată în formă arborescentă, permite manevrarea programului atît de către o persoană fără cunoștințe de informatică, cît și de programatori. Aceștia pot dezvolta aplicații specifice în care au posibilitatea de a crea liste de opțiuni particulare (de exemplu, liste de opțiuni în limba maternă). Datele se salvează în fișiere pe disc, de unde se pot recupera ulterior, fiind incluse într-un tabel pe ecran.

Altă facilitare suplimentară față de MULTIPLAN o reprezintă posibilitatea utilizării de macroinstrucțiuni.

Recent a fost lansată versiunea 3 a acestui produs care, de la un tabel (grilă) în versiunile precedente, a mărit posibilitățile de calcul la 256 de tabele, legate sau nu între ele. De asemenea, s-a îmbunătățit modulul de prezentare grafică a datelor. Versiunea 3 funcționează atît sub sistem de operare MS-DOS, cît și sub OS/2.

Prezentăm în tabelul 7.2 o comparație între caracteristicile produsului 1-2-3 (versiunea 2) și Lucid 3-D (un alt pachet de programe de editare și prelucrare de tabele pentru microcalculatoare de tip PC evolute).

Tabelul 7.2.

Caracteristici ale principalelor pachete de programe pentru editare și prelucrare de tabele

Nume pachet programe Caracteristici, performanțe	1-2-3 (2.01)	Lucid 3-D
1	1	3
Introducere date și editare — citire/scriere fișiere ASCII	×	0

Tabelul 7.2. (continuare)

1	2	3
— citire/scriere fișiere, .WKS, .WK1	×	0
— citire/scriere fișiere .DBF (dBase)	×	—
— anularea efectului ultimei comenzi	—	0
— căutare globală și înlocuire	—	×
— marcare audiovizuală a celulei	—	×
— comenzi invizibile/ascunse	×	—
— fișiere legate	—	×
— dispozitiv de introducere date tip „șoarece”	—	×
— protecție cu parolă	×	—
Performanțe, capacitate		
— număr mediu de rinduri (linii)	8192	9999
— număr maxim de coloane	256	254
— număr maxim de ferestre pe ecran	2	9
— număr maxim de chei pentru sortare concurențială	2	2
— posibilitate de extindere a memoriei	×	—
— coprocesor matematic	×	×
— recalculare automată	—	×
— recalculare minimă	0	×
Raportări și grafică		
— număr maxim de caractere pe linie	240	255
— numerotare automată a paginii	×	×
— tipuri de reprezentări grafice (post-procesor grafic)	6	0
— posibilitate inserare text pe grafice	—	0
Macro/programare		
— mod de învățare macro	—	×
— instrucțiuni de tip IF ... THEN ... ELSE	×	×
— mod de lucru pas cu pas	×	×
— opțiune trasare celule	×	—
— posibilitate acces DOS din macroproceduri	—	×
— posibilități de acces la alte aplicații	×	×
— citire/scriere macroproceduri 1-2-3	×	×

× = Da, — = Nu; 0 = Deficitar

7.3. PROGRAME CARE REALIZEAZĂ GRAFICĂ DE PREZENTARE

Deoarece majoritatea datelor care sînt prezentate grafic îmbracă un caracter economic (statistici), un termen mai potrivit pentru programele cunoscute sub numele de programe de grafică de prezentare ar putea fi programe de grafică economică.

În general, aceste programe asigură relevarea datelor prin reprezentări clasice — diagrame cu bare, histogramme, diagrame sectorizate, grafice, organigrame etc. Productivitatea muncii poate crește mult (mai semnificativ decît prin folosirea de programe de editare și prelucrare de texte, în locul unor dactilografieri clasice), atunci cînd în locul realizării unor planșe desenate se va prefera o prezentare a datelor pe display-uri sau planșe realizate prin intermediul calculatorului cu primare sau dispozitive pentru desenat (plottere).

Calitatea prezentărilor va depinde de caracteristicile calculatorului și programelor de aplicație, precum și de cele ale monitorului folosit (rezoluția sa grafică).

Astfel, pot exista numeroase facilități care permit diverse reprezentări (adecvate datelor prezentate) — de exemplu, bare verticale sau orizontale, bare segmentate, bare grupate, bare sau sectoare în relief, hașurări pe arii, mai multe diagrame și grafice pe ecran, reprezentări grafice cu diverse elemente geometrice (dreptunghiuri, pătrate, cercuri, elipse, arce de cerc, modele), mărirea sau micșorarea diverselor elemente, inserare de texte în diagrame și pe ecran, desen e artistice de prezentare etc.

Un mare efect poate fi realizat prin utilizarea unor facilități care permit prezentarea în dinamică a unor date, rotiri și *flip*-uri ale unor elemente de pe ecran.

De o mare importanță pentru aceste programe este integrarea lor, în sensul de a putea utiliza (citi) date din alte programe de aplicații (tabele, baza de date) în scopul reprezentării lor grafice. De asemenea, importantă este și posibilitatea cuplării și utilizării unor echipamente periferice performante care să asigure realizarea unor planșe

de calitate, ca imprimante color, plottere specializate, paleta Polaroid, etc.

Programul *DIAHISTO* pentru calculatoare compatibile Sinclair ZX Spectrum asigură realizarea de diagrame cu bare verticale, histograme, diagrame cu bare segmentate și diagrame cu bare grupate, precum și imprimarea lor pe format A 4 cu ajutorul unei imprimante.

În tabelul 7.3 sînt ilustrate caracteristicile și performanțele unor programe de grafică de prezentare pentru microcalculatoare evolute (IBM PC XT, IBM PC AT, PS/2, Macintosh).

Tabelul 7.3.

Caracteristici și performanțe ale programelor de grafică economică

Nume pachet program Caracteristici, performanțe	Fre- eance Plus	Gem Presen- tation	Harvard Gra- phics	Micro- soft Chart	35 mm Express	Win- dows Graph
1	2	3	4	5	6	7
Diagrame cu bare („bargraph”)						
— bare verticale/orizon- tale	×	×	×	×	×	×
— număr maxim de date pe diagramă cu bare grupate	8	12	8	8000	60	△
— diagrame cu bare seg- mentate	×	×	×	×	×	×
— diagrame cu bare seg- mentate în relief (3D)	—	×	×	×	×	×
— pictograme	—	×	—	—	—	—
— diagrame cu bare/linii poligonale	—	×	×	×	—	×
— modificarea adîncimii/ spațiului între bare	—	—	×	×	—	×
Diagrame sectorizate („piecharts”)						
— număr maxim de sec- toare pe diagramă	16	20	12	16	12	△
— număr maxim de sec- toare scoase în evi- dență	16	20	12	16	12	—

Tabelul 7.3. (continuare)

1	2	3	4	5	6	7
— 2 sau 4 diagrame pe ecran	×	—	×	×	—	—
— diagrame proporționale cu sectoare de cerc	—	—	×	—	—	×
— diagrame cu sectoare de cerc în relief (3D)	—	×	×	×	×	×
— diagrame cu sectoare/bare	—	—	×	—	—	×
Diagrame cu linii (grafice)						
— diagrame cu grafice (poligonale)	×	×	×	×	×	×
— grafice pentru regresie (cu nor de puncte)	×	×	×	×	—	×
— număr maxim de tipuri de linii	8	1	4	6	1	5
— număr maxim de marcatori de linie	9	5	13	9	0	10
— număr maxim de puncte (date) pe linie	120	101	240	32000	60	△
— număr maxim de seturi de date pe linie	8	12	8	8000	12	△
— hașurare arii pe grafice	—	×	×	×	×	×
Alte tipuri de diagrame						
— diagrame cu cercuri	—	—	—	—	—	—
— diagrame de tip Gantt	—	—	—	—	—	—
— organigrame	×	×	×	×	×	×
— tabele	×	×	×	×	×	×
— diagrame cu text	×	×	×	×	×	×
Alte forme grafice						
— grafice cu linii/forme grafice	×	0	0	—	×	—
— număr maxim de modele	15	60	12	16	64	37
— elipse și arce de cerc	×	×	×	—	×	×
— dreptunghiuri normale/cu colțuri rotunjite	×	×	×	—	0	×

Tabelul 7.3.(continuare)

1	2	3	4	5	6	7
— alte elemente geometrice	0	—	×	—	×	—
— posibilitate de selec-tare toate elementele	×	×	—	0	×	—
— copiere/mutare ele-mente între fișiere	×	×	0	0	×	×
— rotație/flip pentru ele-mente	×	×	—	0	×	—
— anularea efectului ul-timei comenzi	×	×	×	—	×	×
— mărire, micșorare ele-mente	×	×	—	—	×	×
— gradații/grile pe dia-gramă (grafic)	×	×	—	—	×	×
Introducere date						
— citire fișiere ASCII	×	×	×	×	—	×
— citire fișiere .WKS, .WK1	×	×	×	×	×	×
— citire fișiere .PIC	×	—	—	—	—	—
— citire fișiere dBase	×	×	×	×	—	—
— citire fișiere Metafile	×	—	—	—	—	—
— citire fișiere DIF	×	—	—	×	—	×
Diverse facilități						
— tipărire cu diverse di-mensiuni și caracteris-tici	×	×	×	×	×	×
— posibilitate realizare microfilme	×	×	×	×	×	—
— facilități macro	×	—	×	—	—	—
Echipamente periferice						
— dispozitiv introducere date tip „șoarece”	×	×	×	×	×	×
— driver „post script”	0	×	×	0	×	×
— imprimantă HP laser	×	×	×	×	×	×
— ploter HP	×	×	×	×	×	×
— imprimantă color (Herox 4020)	×	×	×	×	×	×
— paletă Polaroid	×	0	×	×	×	×
— paletă Polaroid plus	—	—	×	—	×	×
— imprimantă cap matri-cial (Epson)	×	×	×	×	×	×

× = Da; — = Nu; 0 = Deficitar; Δ = Nelimitat.

7.4. PROGRAME DE EDITARE ȘI PRELUCRARE DE TEXTE

Marele avantaj al unui sistem de editare și prelucrare de texte (microcalculatoare specializate pentru editare și prelucrare de texte sau microcalculator de tip PC, împreună cu programul de editare și prelucrare de texte) constă în faptul că nu este necesară retipărirea întregului document în situația în care trebuie efectuate mici modificări sau corecții asupra corpului de text. Avantajele se referă la corecturi, la mutări de rânduri sau paragrafe în text, la realizarea mai multor copii cu diferențe minime între ele. Alte avantaje se referă la posibilitatea introducerii textului fără ca operatorul să fie atent la sfârșitul rândului (sistemul va realiza automat o respațiere între cuvinte, astfel încît ultimul cuvînt din rînd să nu fie împărțit, sau va realiza automat despărțirea în silabe a cuvîntului de la sfârșitul rîndului), la capacitatea de a ajusta lesne formatele documentelor, precum și de a corecta erorile de ortografie.

Tasword Two este un editor de texte pentru calculatoare compatibile Sinclair ZX Spectrum. Datorită capacității reduse a memoriei fișierului text, poate avea cel mult 320 de linii (cca 10 pagini) a cîte 64 de caractere pe linie. Pentru texte mai mari se vor realiza mai multe fișiere separate. Încărcarea unui fișier text maxim de pe caseta magnetică durează circa două minute.

Tastatura calculatorului este folosită pentru introducerea atît a caracterelor alfanumerice, cît și a comenzilor necesare editării salvării/încărcării pe/de pe suport magnetic a fișierului care conține textul. Procesorul operează pe un fișier text, care conține informația introdusă de la tastatură.

Ecranul TV conține o fereastră, în cadrul căreia se afișează 22 de linii, a cîte 64 de caractere. Cu ajutorul unor taste de comandă întregul fișier text poate fi deplasat în sus sau în jos, în cadrul ferestrei. Caracterele alfanumerice, afișate pe o linie a ferestrei, sînt generate prin program, fiind diferite de caracterele afișate în mod normal de cal-

culator. În principiu, în cadrul ferestrei, se pot afișa numai 32 caractere pe linie. O serie de cuvinte cheie de tipul TO; THEN; < > etc. indică utilizarea tastelor respective în vederea introducerii unei comenzi.

Tastele pentru comenzi devin efective în condițiile în care una din tastele de *SHIFT* (*CS* sau *SS*) este în prealabil activată. Programul dispune de două pagini de ajutor (*HELP*), care conțin sub formă o concisă descrierea semnificațiilor testelor de comandă, existînd posibilitatea de a fi apelate prin activarea simultană a tastelor *CS* și *1* (*EDIT*).

Comenzile se referă la deplasarea cursorului peste un cuvînt la stînga, dreapta, sus și jos, centrarea liniei, inserarea unei linii/unui caracter, deplasarea la începutul sau sfîrșitul textului încărcarea/salvarea/tipărirea textului, înlăturarea unei linii, defilare ecran (normală sau rapidă) în sus sau în jos, înlocuire/găsire text, marcarea, copierea sau deplasarea unui bloc de text etc.

Wordstar este un editor de texte care funcționează sub sistem de operare CP/M. Mulți ani, el a reprezentat cel mai vîndut produs program pentru microcalculatoare.

Wordstar lucrează numai în modul ecran: textul editat este afișat în permanență pe ecranul terminalului, poziția curentă fiind marcată prin cursor. Orice modificare a textelor este facilitată de afișarea în partea superioară a ecranului a sumarului comenzilor și de desfășurarea sub formă de dialog a funcțiilor de căutare, substituire, imprimare etc.

Inserarea textelor se realizează prin simpla tastare a textului dorit.

În cadrul unui paragraf, *Wordstar* trece automat la linia următoare și aliniază cuvintele la marginea din dreapta. Utilizatorul trebuie să acționeze toate comenzile *ENTER* doar la sfîrșit de paragraf.

Comenzile *Wordstar* sînt identificate prin secvențe de unul sau două caractere, din care primul este un caracter de control. Dacă după tastarea primului caracter utilizatorul face o scurtă pauză, programul afișează lista tuturor comenzilor care încep cu caracterul respectiv.

Textele editate sînt paginate automat, limitele de pagini fiind marcate pe ecran printr-o linie întreruptă. În acest scop, Wordstar folosește valori implicate ale unor parametri (număr rînduri pe pagină, marginile stînga și dreapta ale textului, spațiere etc.), care pot fi modificate de utilizator.

Fișierele folosite de Wordstar pentru memorarea textelor sînt fișiere standard CP/M. În general, Wordstar modifică conținutul unui fișier original, textul modificat fiind păstrat într-un fișier de lucru; la sfîrșitul editării, fișierul original primește tipul „BAK“, iar fișierul de lucru primește numele fișierului original. Prin inserarea într-un text a unor caractere de control se pot realiza îngroșări de caractere, sublinieri etc.

Invocarea unei funcții din lista inițială se face prin tastarea literei (poate fi majusculă sau minusculă) din dreptul funcției respective. Litera se va afișa în colțul din stînga-sus al ecranului. Urmează o scurtă descriere a comenzilor inițiale: *D* — editează un document, se afișează primul ecran și utilizatorul poate începe editarea lui; *N* — editează un text non-document (de exemplu, textul unui program; editarea decurge ca la comanda *D*, dar fără paginare și spațiere automate); *X* — Wordstar trece controlul sistemului de operare; *Y* — șterge un fișier, *O* — copiază un fișier; *E* — schimbă numele unui fișier; *R* — execută un program etc.

Wordstar este instalat și pe calculatoare compatibile IBM PC, funcționînd deci sub sisteme de operare MS-DOS. În această versiune, modul de operare este similar, fiind puse la dispoziție mai multe facilități, dintre care menționăm posibilitatea de lucru cu taste predefinite, posibilitatea eliminării caracterelor de control și vizualizarea pe ecran a formatului de tipărire etc.

În tabelul 7.4 sînt ilustrate caracteristicile principalelor pachete de programe de editare și prelucrare de texte pentru microcalculatoare de tip PC evaluate (IBM PC AT, PS/2, Macintosh).

Caracteristici ale principalelor pachete de programe de editare
și prelucrare de texte

Nume pachet programe Caracteristici, performanțe	Q&A Write	Nota bene	Xy Write III	Display Write
Caracteristici ecran:				
— număr de coloană	—	—	—	—
— număr de linie	×	×	×	×
— vizualizarea paginării	×	—	—	—
— scriere cu caractere îngroșate (bold)	×	×	×	×
— sublinieri	—	×	×	×
— caractere italice (scriere rotundă)	—	0	0	—
Formatare:				
— formatare condițională automată	×	×	×	—
— control fereastră	—	×	×	×
— despărțire automată în silabe	0	×	×	×
— text și grafică pe aceeași pagină	—	×	×	×
Căutări și înlocuiri				
— căutări cu ignorarea condițională a unor cazuri	×	×	×	×
— căutări caractere de control	×	×	×	—
— căutări cu expresii generice	×	×	×	—
Tipărire:				
— spațiere proporțională	×	×	×	×
— tipărire format document	×	×	×	×
Caracteristici speciale:				
— mod macro	×	×	×	×
— verificare scriere corectă (ortografic)	×	×	×	×
— trasări de linii	×	×	—	—
— inserare fișiere de grafică	×	×	×	—
— funcții matematice	×	×	×	×
— note de subsol	0	×	×	×
— număr maxim de ferestre	1	9	9	12
— facilități de poștă electronică	×	×	×	×
— crearea tabele de index	—	×	×	×
— modul telecomunicații	—	—	—	—
— anulare efect ultimă comandă	—	—	—	×
— anulare efect ultimă ștergere	×	×	×	×

× = Da; — = Nu; 0 = Limitat

7.5. POȘTA ELECTRONICĂ

Poșta electronică sau serviciu de mesagerie electronică (*electronic mail, message handling, messanging package system* sau *electronic data interchange*) este o facilitate software a unui sistem de calcul/mini/microcalculator, prin care un utilizator întocmește, transferă sau primește un mesaj către sau de la alt utilizator. Destinatarul poate fi un echipament local, dintr-un alt nod al unei rețele de calculatoare sau dintr-o altă rețea. Prin utilizarea acestor programe la nivelul utilizatorului sînt evidente facilitățile locale cu care se prelucrează mesajele de trimis sau mesajele recepționate: utilizatorul poate șterge, muta sau copia mesaje, avînd totodată posibilitatea de a acorda nume prescurtate unor adresați implicați sau des acceptați. Aceste facilități nu constituie însă partea centrală a programelor de poștă electronică, a căror esență este reprezentată de pregătirea mesajului pentru expediere, transferarea unui mesaj în rețea și recepționarea mesajului.

Un mesaj este întocmit de un utilizator și el îl predă sistemului de poștă electronică, indicînd o serie de servicii pe care le solicită: confirmare de primire, caracter urgent sau normal etc. Un mesaj poate conține orice fel de informații, documentele cu caracter economic prețindu-se foarte bine la astfel de servicii. Se pot expedia în acest fel facturi, comenzi de aprovizionare, dări de seamă statistice, formulare conținînd date solicitate de organele superioare. Un sistem de poștă electronică va garanta transferul în siguranță al mesajului.

Utilizarea microcalculatoarelor de tip PC favorizează dezvoltarea sistemelor de poștă electronică, grație posibilităților foarte comode de interacțiune cu utilizatorul, prin meniuri simple. Un exemplu: firma Retix (S.U.A.) a realizat un sistem de poștă electronică (dezvoltat cu produsul Windows), implementat pe o rețea locală de PC-uri și interconectat cu alte sisteme.

O serie de firme din domeniul informaticii (din țările dezvoltate industrial) au oferit de mai mulți ani servicii de poștă electronică, inițial la nivelul unei întreprinderi (în interiorul său), apoi între noduri, la distanță. De ase-

menea, în diverse țări au fost stabilite servicii de poștă electronică corelate cu rețelele publice de date.

Pentru sistemul de poștă electronică s-au elaborat o serie de recomandări și standarde cunoscute sub numele de X.400. Necesitatea definirii unor standarde în domeniul poștei electronice a rezultat din mai multe considerente ca:

- facilitarea schimburilor internaționale de mesaje efectuate între abonații la rețelele publice de date;

- necesitatea transferării de mesaje care au formate diverse;

- existența unor standarde care definesc interfața cu rețelele publicate de date; conectările internaționale între acestea; modelul de referință al interconectării sistemelor deschise. Funcțiile care realizează recomandările din seria X.400 sînt situate la nivelul de aplicație.

Un avantaj al poștei electronice îl constituie posibilitatea de comunicare între echipamente terminale foarte variate; telex, facsimil, Teletex, Videotex, voce, terminal. Astfel, un utilizator își pregătește mesajul pe echipamentul său local (de exemplu, un microcalculator de tip PC) și îl expediază spre un destinatar, indiferent de tipul de echipament pe care acesta îl are la dispoziție. Este sarcina sistemului de poștă electronică de a executa conversia necesară.

Conform unor calcule, utilitatea sistemelor de poștă electronică se reflectă în reducerea cu 50% a prețului față de schimbul de scrisori clasic și în posibilitatea de a transporta orice informație codificată în cadrul unui „plic” tip X.400, în mod rapid și sigur.

7.6. SISTEME AUTOMATE PENTRU REALIZAREA DE PUBLICAȚII

Sistemele automate pentru realizarea de publicații (*desktop publishing systems*) au o dată de apariție mai recentă, fiind legate intrinsec de microcalculatoare de tip PC evaluate. Unele dintre aceste sisteme provin din microcalculatoare PC înzestrate cu pachete de programe de

aplicație adecvate, în timp ce altele reprezintă microcalculatoare dedicate acestei aplicații. Un sistem automat pentru realizarea de publicații se bazează pe un microcalculator de tip PC evoluat său o stație de lucru, iar ca periferie minimă — o imprimantă de calitate (de exemplu laser HP) și un dispozitiv pentru introducere și digitizare imagini. Pachetele de programe specifice realizează editare și prelucrare de texte (cu un procesor de texte dezvoltat, cu ajutorul căruia se pot alege, de exemplu, diverse formate de litere, ca în tipografie), grafică (cu un procesor de grafică), putându-se combina texte, desene și imagini (fotografii digitizate), un sistem automat de aplicații realizând practic funcțiile unei tipografii (minitipografie).

PROGRAME POSIBILE

8.1. PROBLEME ... MATEMATICE

Ecuatie diofantică. Să se scrie un program general de rezolvare a unei ecuații diofantice

$$AX + BY = C,$$

cu $A, B, C \in \mathbb{N}$.

Soluția ecuației este reprezentată de perechi de numere întregi, X și Y , care verifică ecuația dată.

Dacă $C = 0$, ecuația este omogenă, iar soluția generală este $X = -BZ$, $Y = AZ$, unde Z este număr întreg oarecare, iar A și B sînt prime între ele (în caz contrar A și B se înlocuiesc prin cel mai mare divizor comun al lor).

Dacă $C \neq 0$ ecuația este neomogenă. Dacă (X_0, Y_0) este soluție particulară a ecuației omogene, iar (X_1, Y_1) este soluția a celei omogene, atunci și $(X_0 + X_1, Y_0 + Y_1)$ este soluție pentru ecuația neomogenă și, în plus, orice soluție se poate scrie sub această formă.

Pentru a găsi o soluție particulară a ecuației neomogene se pornește de la faptul că cel mai mare divizor comun al numerelor A și B se scrie sub forma $D = MA + NB$, cu M și N numere întregi; atunci, $X = MC/D$, $Y_0 = NC/D$ iar soluția generală a ecuației neomogene este:

$$X = X_0 + B/D \cdot Z, \quad Y = Y_0 - A/D \cdot Z,$$

unde Z este număr întreg oarecare.

Numerele M și N din relația $D = MA + NB$ se determină recursiv folosind ca referință algoritmul lui Euclid. Dacă se împarte restul R_{k-2} la restul R_{k-1} obținându-se câtul Q_k și restul R_k , atunci $R_{k-2} = Q_k R_{k-1} + R_k$, $k = 0, 1, 2, \dots$, unde $R_{-2} = A$, $R_{-1} = B$. De aici se obține $R_k = R_{k-2} - Q_k R_{k-1}$. Dacă $R_k = M_k A + N_k B$ (unde $M_{-2} = 1$, $N_{-2} = 0$, $M_{-1} = 1$, $N_{-1} = 0$), atunci din relațiile precedente rezultă:

$$M_k = M_{k-2} - Q_k M_{k-1}, \quad N_k = N_{k-2} - Q_k N_{k-1},$$

relații care pentru $k = 1, 2, 3, \dots$ permit determinarea numerelor M_k și N_k pînă cînd se obține ultimul rest diferit de zero.

Sistem. Să se scrie programul cu ajutorul căruia se află x^* și y^* ale sistemului de ecuații:

$$\begin{cases} 2x^2 - xy - 5x + 1 = 0, \\ x - y^2 + 1,6 = 0, \end{cases}$$

cu precizia $\varepsilon \leq 10^{-4}$, dacă se știe că punctul (x^*, y^*) aparține domeniului D ($3,5 \leq x^* \leq 3,6$; $2,2 \leq y^* \leq 2,3$).

Pentru calcularea rădăcinilor se va utiliza următoarea schemă iterativă:

Pasul 1. Inițializează $x_0 = 3,5$; $y_0 = 2,2$;

Pasul 2. Calculează

$$x_{n+1} = \sqrt{\frac{x_n (y_n + 5) - 1}{2}}, \quad y_{n+1} = \sqrt{x_n + 1,6};$$

Pasul 3. Verifică condițiile $|x_{n+1} - x_n| \leq \varepsilon$, $|y_{n+1} - y_n| \leq \varepsilon$. Dacă amîndouă sînt îndeplinite, atunci se trece la pasul 5, iar în caz contrar se continuă cu pasul 4;

Pasul 4. Se consideră $x_n \leftarrow x_{n+1}$, $y_n \leftarrow y_{n+1}$ și se trece la pasul 2;

Pasul 5. Valoarea rădăcinii se ia egală cu

$$x^* = x_{n+1}; \quad y^* = y_{n+1}.$$

Polinom $P(X) \in \mathbf{Z}_p$. Să se scrie un program de rezolvare a unei ecuații polinomiale cu coeficienți în corpul \mathbf{Z}_p al claselor de resturi modulo p , cu p număr prim.

Fie polinomul $P(x) = \sum_{i=1}^{101} a_i x^{101-i}$ de gradul 100 cu coeficienți în \mathbf{Z}_7 . Să reduc em gradul polinomului la unul de gradul șase, care să aibă aceleași soluții cu cel dat. Pentru aceasta din relația $x^p \equiv x \pmod{p}$, valabilă pentru orice $x \in N$, care satisface inecuația $0 \leq x < p$, rezultă:

$$x^{k(p-1)+1} \equiv x \pmod{p}, \quad k = 0, 1, 2, \dots$$

Reducerea gradului se face folosind această relație, care conduce în final la:

$$b_{95}x^6 + b_{96}x^5 + \dots + b_{100}x + b_{101} = 0,$$

unde $b_{101} = a_{101}$, $b_k = a_k + a_{k-6} + a_{k-12} + \dots$, cu $k = 95, 96, \dots, 100$.

Rădăcinile ecuației rămase le găsim printre numerele $0, 1, 2, \dots, p-1$, care se înlocuiesc succesiv în polinomul de gradul șase și se rețin acelea pentru care valoarea obținută este egală cu zero. Calculul valorii unui polinom într-un punct se face după schema:

$$(b_{95}x + b_{96})x + b_{97})x + b_{98})x + b_{99})x + b_{100})x + b_{101}.$$

Operații în corpul \mathbf{Z}_7 . Să scrie subrutine pentru tipărirea tabelelor de adunare, înmulțire și a inverselor elementelor din corpul \mathbf{Z}_7 .

Pentru tabela adunării, subrutina începe cu tipărirea pe prima linie a semnelui $+$ și a numerelor $0, 1, 2, \dots, n-1$, care reprezintă conținutul vectorului V .

Apoi, pentru fiecare $i \in \{0, 1, 2, \dots, n-1\}$ sînt tipărite pe cîte o linie elementul i și vectorul V . La fiecare pas, vectorul V se modifică prin deplasarea elementelor sale cu o poziție la stînga, prima componentă luînd locul ultimei.

Pentru scrierea unui rînd în tabela înmulțirii este folosit vectorul V cu n componente întregi. Pentru fiecare $i \in \{0, 1, 2, \dots, n-1\}$ elementele vectorului V primesc în ordine ca valori produsele dintre i și elementele $0, 1, 2, \dots, n-1$; valoarea produselor este înlocuită cu restul împărțirii lor prin n , deoarece se alege reprezentantul cuprins între 0 și $n-1$.

Pentru determinarea inversului lui i , unde $i \in \{1, 2, \dots, n-1\}$, se consideră produsul lui i cu fiecare dintre elementele $k \neq 0$ ale lui Z_n . În cazul în care există o valoare a lui k pentru care produsul este în clasa lui 1, valoarea lui k este înscrisă în componenta i a vectorului V , ca reprezentînd inversul lui i ; în caz contrar, $V(i)$ primește valoarea 0, marcînd astfel faptul că elementul i nu este inversabil în Z_n .

Ortonormare. Să se ortonormeze un sistem de vectori liniari independenți.

Pentru vectorii A și B , fiecare cu n componente se definește produsul lor scalar și norma prin relațiile:

$$(A, B) = \sum_{i=1}^n (a_i b_i), \quad \|A\| = \sqrt{(A, A)}.$$

Fie sistemul de vectori X_1, X_2, \dots, X_m , fiecare cu cîte n componente. Ortonormarea acestui sistem presupune existența unui alt sistem de vectori, Y_1, Y_2, \dots, Y_m , care să satisfacă relațiile:

$$(X_i, Y_j) = 0, \text{ pentru } i \neq j \text{ și } i, j \in \{1, 2, \dots, m\};$$

$$\|Y_i\| = 1, \text{ } i \in \{1, 2, \dots, m\}.$$

Procedeeul de ortonormare Gram-Schmidt folosește relațiile:

$$Y_1 = \frac{X_1}{\|X_1\|}, \quad Y_2 = \frac{X_2 - (X_2, Y_1) Y_1}{\|X_2 - (X_2, Y_1) Y_1\|}, \dots,$$

$$Y_k = \frac{X_k - \sum_{i=1}^k (X_k, Y_i) Y_i}{\left\| X_k - \sum_{i=1}^{k-1} (X_k, Y_i) Y_i \right\|}.$$

Problema comis-voiajorului. Un comis-voiajor pleacă dintr-un oraș, trebuie să viziteze un număr de orașe date și să se întoarcă în orașul de plecare cu minim de efort (de exemplu, în minimum de timp).

Enunțul matematic este următorul: fie $G = (X, \Gamma)$ un graf neorientat, în care oricare două vîrfuri sînt unite între ele printr-o muchie căreia i se asociază un cost strict pozitiv. Se cere să determinăm un ciclu care să înceapă dintr-un vîrf oarecare al grafului, să treacă exact o dată prin toate celelalte vîrfuri și să se întoarcă în vîrfurile inițial, ciclu care să îndeplinească în plus condiția că are un cost minim (costul unui lanț fiind definit ca suma costurilor atașate muchiilor componente).

Se consideră următoarea strategie: în situația că (v_0, v_1, \dots, v_r) este lanțul deja construit, atunci:

— dacă $\{v_0, v_1, \dots, v_k\} = X$ se adaugă muchia (v_k, v_0) și construcția ciclului este încheiată;

— dacă $\{v_0, v_1, \dots, v_k\} \neq X$, atunci se adaugă acea muchie (v_k, v_{k+1}) de lungime minimă și pentru care $v_{k+1} \notin \{v_0, v_1, \dots, v_k\}$.

Pentru graful din fig. 8.1 a, dacă se pleacă din vîrfurile $i = 1$ se obține ciclul din fig. 8.1 b, avînd costul 14. Acest ciclu nu este optimal deoarece, de exemplu, ciclul din fig. 8.1 c are costul 13, iar cel din fig. 8.1 d, 10.

Fie graful complet și simetric format din vîrfurile X_0, X_1, \dots, X_n . Pentru a găsi circuitul de valoare totală minimă care pleacă din X_0 se introduc variabilele bivalente x_{ij} determinate prin:

$$x_{ij} = \begin{cases} 1, & \text{dacă circuitul trece prin arcul } (i, j); \\ 0, & \text{în caz contrar.} \end{cases}$$

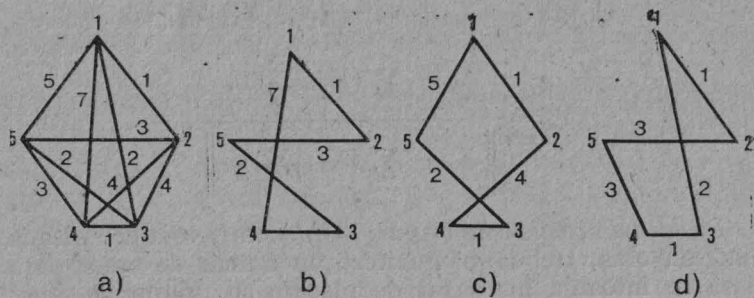


Fig. 8.1. Problema comis-voiajorului

Problema formulată este determinată prin sistemul de relații:

$$[\min] f = \sum_{i=0}^n \sum_{j=0}^n c_{ij} \cdot x_{ij}, \quad (8.1)$$

$$\sum_{i=0}^n x_{ij} = 1, \quad j = 1, 2, \dots, n, \quad (8.2)$$

$$\sum_{j=0}^n x_{ij} = 1, \quad i = 1, 2, \dots, n, \quad (8.3)$$

$$u_i - u_j + nx_{ij} \leq n - 1, \quad 1 \leq i \neq j \leq n, \quad (8.4)$$

$$u_i \in R, \quad i = 1, 2, \dots, n, \quad (8.5)$$

semnificația numerelor reale u urmînd a fi stabilită.

Fie o soluție care satisface relațiile (8.2) ÷ (8.4). Din primele două condiții se deduce că această soluție constă din unul sau mai multe circuite elementare. Dacă ar exista mai multe circuite, unul singur ar trece prin X_0 . Să alegem un circuit care nu trece prin X_0 și să notăm cu k , $1 < k < n$, numărul său de arce. Adunînd relațiile (8.4) corespunzătoare arcelor (i, j) ce aparțin acestui circuit (deci, cu $x_{ij} = 1$), diferențele $u_i - u_j$ se anulează și se ajunge la relația contradictorie $nk \leq (n - 1)k$. Să arătăm acum că pentru orice circuit hamiltonian care pleacă din X_0 se pot găsi numerele reale u_i pentru care (8.4) este realizată. Să alegem $u_i = r$, dacă X_i este extremitatea finală a celui de-al r -lea arc al circuitului, originea fiind considerată în X_0 ($r = 1, 2, \dots, n$). Este clar că $u_i - u_j \leq n - 1$ pentru orice arc (X_i, X_j) ; deci, condiția (8.4) este satisfăcută pentru $x_{ij} = 0$, iar pentru $x_{ij} = 1$ avem

$$u_i - u_j + nx_{ij} = r - (r + 1) + n = n - 1.$$

Iată deci modelul A.W. Tucker de rezolvare a problemei. Evident, pot exista „variante“ îmbunătățite ale acestuia, dar încercați să programați acest model.

Problema iepurilor de casă. Cîte perechi de iepuri de casă se nasc într-un an dintr-o singură pereche de iepuri? Pentru a afla cîte perechi se nasc într-un an, cineva a așezat cîteva

perechi de iepuri într-un loc îngrădit cu zid, știind că după o lună o pereche de iepuri aduce pe lume o altă pereche, iar iepurii încep să dea naștere la pui de la vârsta de o lună. Deoarece prima pereche dă descendenți în prima lună, perechea se dublează și, în această lună, se obțin două perechi, dintre care o pereche și numai prima, va avea descendenți și în luna următoare, astfel că în luna a doua vor fi trei perechi, dintre acestea în luna următoare două perechi vor avea descendenți, în așa fel încât în luna a treia se mai nasc două perechi de iepuri și numărul de perechi de iepuri în această lună este de cinci. Dintre acestea, în aceeași lună, vor avea urmași trei perechi, iar numărul de iepuri din luna a patra va fi opt. Dintre acestea cinci perechi vor da naștere la cinci perechi care adunate la cele opt perechi formează în luna a cincea treisprezece perechi ș.a.m.d.

Problema a fost formulată prin anul 1202, iar Leonardo Fibonacci (matematician italian, 1170-1250) a găsit legea numerică prin care se exprimă o însușire a materiei vii, și anume sub forma unui șir de numere întregi: 1, 1, 2, 3, 5, 8, 13, 21, 34, Aceste valori sînt date de funcția Fibonacci, $f : N \rightarrow N$,

$$f(n) = \begin{cases} 1, & \text{dacă } n = 0 \text{ sau } n = 1; \\ f(n-1) + f(n-2), & \text{în rest.} \end{cases}$$

Valorile acestei funcții se pot determina fără dificultăți pentru orice valoare a lui n , folosindu-se un calculator personal. Șirul de numere obținut este cunoscut și sub numele de *legea creșterilor organice*, deoarece prin el se exprimă dezvoltarea materiei vii, realizată prin compuneri care se însumează succesiv. Dintre exemple menționăm: distanțele dintre nodurile de creștere ale unei tulpini; dezvoltarea cochiliilor melcilor sau scoicilor; alungirea oaselor și a coarnelor animalelor etc.

Puteți determina, acum, cîte perechi de iepuri se nasc într-un an?

Dar să mai „depistăm“ o curiozitate. Se aplică repetat relația de recurență:

$$\begin{aligned}
 f(n) &= f(n-1) + f(n-2) = f(n-2) + f(n-3) + f(n-3) + \\
 &+ f(n-4) = f(n-3) + f(n-4) = 2f(n-4) + 2f(n-5) + \\
 &+ f(n-4) = 5f(n-4) + 3f(n-5).
 \end{aligned}$$

Deoarece $f(5) = 5$ rezultă că fiecare al cincilea număr este divizibil cu cinci.

Și încă o ... problemă: să se găsească volumul tetraedului ale cărui vîrfuri au respectiv coordonatele $(f(n), f(n+1), f(n+2))$, $(f(n+3), f(n+4), f(n+5))$, $(f(n+6), f(n+7), f(n+8))$ și $(f(n+9), f(n+10), f(n+11))$, unde $f(i)$ este al i -lea termen din șirul Fibonacci.

Deoarece $f(n) = f(n-1) + f(n-2)$ înseamnă că cele trei numere ale lui Fibonacci satisfac ecuația $Z = x + y$. Prin urmare, cele patru vîrfuri ale tetraedului sînt coplanare și, deci, volumul este nul. Mai mult, cele 12 coordonate nu trebuie să fie termeni consecutivi. Același rezultat rămîne valabil și în cazul cînd coordonatele fiecărui vîrf sînt numere ale lui Fibonacci consecutive.

Deținuții norocoși. Executînd prevederile unei amnistii parțiale, un gardian deschide pe rînd toate celulele închisorii. Pe urmă închide fiecare a doua celulă. Apoi, luînd celulele din trei în trei răsuțește cheia în broasca acestor celule, închizîndu-le pe cele deschise și deschizîndu-le pe cele închise. El continuă această operație, luînd celulele din n în n și răsucind cheia în broasca lor. Deținuții ale căror celule au rămas deschise după efectuarea tuturor operațiilor de acest fel sînt puși în libertate. Considerînd că celulele sînt așezate la rînd și că fiecare operație începe din dreptul primei celule să se determine care sînt norocoșii acestei „amnistii“.

Numărul m care indică de cîte ori a fost răsucită cheia în broasca celulei x este egal cu numărul divizorilor lui x . Dacă $x = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$, unde p_i sînt numere prime, atunci $m = (a_1 + 1)(a_2 + 1) \dots (a_k + 1)$. Dacă un număr a_i este impar, atunci m este par și celula corespunzătoare rămîne închisă. Dacă toate numerele a_i sînt pare, x este pătratul unui număr, m este impar și fericitul ocupant al celulei „pătratică“ va constata că celula lui a rămas deschisă.

Minimul lui $F(x)$. Să se determine minimul unei funcții, $F(x)$, de o singură variabilă.

Unul din algoritmi de rezolvare se bazează pe metoda de căutare aleatoare și este dat în continuare. În acest scop se consideră valoarea inițială, x_0 , și se calculează valoarea $F(x_0)$.

1. Se modifică variabila x cu valoarea pasului Δx , astfel încît

$$x_1 = x_0 + \Delta x$$

este noua valoare a lui x . Se calculează $F(x_1)$. Dacă $F(x_1) < F(x_0)$ se continuă incrementarea variabilei x cu Δx pentru a obține șirul x_2, x_3, \dots, x_n și se calculează corespunzător $F(x_2), F(x_3), \dots, F(x_n)$. Dacă în schimb $F(x_1) > F(x_0)$, variabila x se modifică în sens opus, adică se incrementează x cu $-\Delta x$.

2. Presupunînd că există un minim, la un anumit moment se va observa că valoarea lui F crește.

Fie deci

$$F(x_n) > F(x_{n-1}) \quad (8.6)$$

și

$$F(x_{n-1}) < F(x_{n-2}), \quad (8.7)$$

adică $F(x)$ are un minim între x_{n-2} și x_n . După calculul lui $F(x_n)$, cu condiția (8.6) îndeplinită, se trece la o căutare exactă a minimului, folosind un pas variabil determinat cu ajutorul relației de interpolare:

$$\Delta x_n = \frac{1}{2} \cdot \frac{(\Delta x_{n-1})^2 \Delta F(x_{n-2}) + (\Delta x_{n-2})^2 \Delta F(x_{n-1})}{(\Delta x_{n-1}) \Delta F(x_{n-2}) - (\Delta x_{n-2}) \Delta F(x_{n-1})} - \Delta x_{n-1} \quad (8.8)$$

unde Δx_{n-1} și Δx_{n-2} sînt pașii determinați la iterațiile anterioare, iar

$$\Delta F(x_{n-1}) = F(x_{n-1}) - F(x_n); \quad (8.9)$$

$$\Delta F(x_{n-2}) = F(x_{n-2}) - F(x_{n-1}). \quad (8.10)$$

3. Se va aplica de mai multe ori relația (8.8) pînă cînd $|F(x_i) - F(x_{i+1})| \leq \varepsilon$, cu ε valoare impusă. Valoarea $F(x_{i+1})$ este considerată drept minim.

În unele situații, parametrul x este limitat între două valori, x_{max} și x_{min} . Dacă în procesul de căutare a minimumului menționat la primul punct se ajunge la una din limitele date de x_{max} sau x_{min} fără a observa o creștere a lui F , se consideră că minimumul a fost atins la valoarea lui F corespunzătoare limitei (acesta nu reprezintă minimum matematic).

Trasarea traiectoriilor de fază. Comportarea dinamică a unui sistem poate fi interpretată prin analiza răspunsului acestuia la diferite semnale semnificative. Evidențierea răspunsului necesită integrarea ecuației diferențiale care descrie sistemul.

Fie ecuația diferențială omogenă:

$$a_2 \ddot{x} + a_1 \dot{x} + a_0 x = 0. \quad (8.11)$$

Ea poate fi exprimată sub forma sistemului de ecuații diferențiale:

$$\frac{dy}{dx} = y; \quad \frac{dy}{dt} = -\frac{a_1}{a_2} y - \frac{a_0}{a_2} x. \quad (8.12)$$

Prin eliminarea timpului din sistemul (8.12), se obține ecuația diferențială

$$\frac{dy}{dx} = \frac{-a_1 y - a_0 x}{a_2 y} = F(x, y), \quad (8.13)$$

care reprezintă, de asemenea, panta traiectoriilor sistemului descris de ecuația (8.11). Integrarea ecuației (8.13) conduce la expresia analitică a traiectoriilor de fază.

Se rezolvă sistemul (8.12) cu metoda Runge-Kutta de ordinul patru. Deoarece variabila t nu apare explicit în formule, atunci relațiile Runge-Kutta adoptă în acest caz forma:

$$x_{i+1} = x_i + (\Delta t) y_i + \frac{(\Delta t)^2}{6} (k_1 + k_2 + k_3),$$

$$y_{i+1} = y_i + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4),$$

în care:

$$k_1 = (\Delta t) f(x_i, y_i); \quad k_2 = (\Delta t) f\left(x_i + \frac{(\Delta t)}{2} y_i, y_i + \frac{k_1}{2}\right);$$

$$k_3 = (\Delta t) f\left(x_i + \frac{(\Delta t)}{2} y_i + \frac{(\Delta t)}{4} k_1, y_i + \frac{k_2}{2}\right);$$

$$k_4 = (\Delta t) f\left(x_i + (\Delta t) y_i + \frac{(\Delta t)}{2} k_2, y_i + k_3\right).$$

Se consideră că punctul (x_0, x_0) este cel din condițiile inițiale (la $t = 0$). Prin Δt s-a notat valoarea incrementată a timpului pentru care se calculează un nou punct al traiectoriei din cele n puncte.

Pentru rezolvarea sistemului (8.12) s-a apelat la metoda Runge-Kutta din motive de precizie și stabilitate.

8.2. PROBLEME ... CU OPERAȚII NUMERICE

Jocul numerelor. În cărțile de matematică, începînd chiar cu clasa întîi, întîlnim foarte multe operații cu numere întregi pozitive. Vă propunem un joc amuzant pe care îl pot practica două persoane, folosind două calculatoare (de buzunar sau personale) ori eventual hîrtie și creion. Să presupunem cazul a două calculatoare.

„Adversarii“ își înscriu, fiecare pe calculatorul său, cîte o cifră cuprinsă între 0 și 9, fără să știe unul ce a înscris altul. Prin tragere la sorți este decisă persoana care începe jocul. Să presupunem că sorții au decis asupra jucătorului A . Acesta anunță o nouă cifră, iar B va calcula diferența dintre cifra înscrisă de el și cea anunțată de A (dacă diferența este negativă se ia cu semn schimbat) și o va înscrie pe calculatorul său după cifra inițială. Este rîndul lui B să anunțe o cifră. Jucătorul A procedează la fel, calculînd diferența dintre ultima cifră înscrisă de el și cea anunțată de B (aceeași acțiune cînd diferența este negativă), dife-

rență pe care o va înscrie pe calculatorul său alături de celelalte cifre precedente scrise. Din nou A anunță un număr, iar B repetă operația cu această cifră și ultima înscrisă de el ș.a.m.d., pînă cînd fiecare jucător are pe ecranul calculatorului un număr compus din șase cifre. Cîștigă jucătorul care are suma celor șase cifre cea mai mare.

Pentru exemplificare presupunem că jucătorul A deschide jocul, iar numere alese inițial sînt 6 pentru A și 9 pentru B . Jucătorul A anunță 5, iar B calculează $9 - 5 = 4$, cifră pe care o înscrie alături de 9, obținînd 94. B anunță 6, iar A calculează $6 - 6 = 0$, pe care o înscrie alături de 6, obținînd 60. A anunță din nou, dar cifra 6, și B calculează $4 - 6 = -2$ și înscrie 2 alături de 94, obținînd 942. B anunță 8, iar A calculează $8 - 0 = 8$, înscrie 8 alături de 60, obținînd 608. Repetînd operațiile pentru numerele anunțate cînd de A , cînd de B (acestea fiind 8, 3, 2, 1, 8, 9) se vor obține în final numerele 942 644 pentru B și 608 545 pentru A . Cîștigător este jucătorul B , deoarece suma cifrelor numărului obținut de el ($9 + 4 + 2 + 6 + 4 + 4 = 29$) este mai mare decît suma cifrelor numărului lui A ($6 + 0 + 8 + 5 + 4 + 5 = 28$).

Jocul numerelor se poate modifica prim mărirea numărului cifrelor (în loc de șase cifre, propuneți-vă 15 sau 20 cifre). De asemenea, jocul poate fi practicat de mai multe persoane fiind aleși ordinea de joc (de exemplu, de la stînga la dreapta) și jucătorul care începe. În acest ultim caz, poate fi întocmit chiar un clasament al jucătorilor, după un anumit număr de jocuri.

Cuvinte și numere. Vă propunem un joc cu numere și litere. Pentru aceasta desenați o matrice pătratică cu 26 linii și 26 coloane. Pe prima linie treceți literele alfabetului în ordinea: $A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z$; pe linia a doua în ordinea: $B, C, D, \dots, X, Y, Z, A$; pe linia a treia în ordinea: $C, D, E, \dots, X, Y, Z, A, B$ ș.a.m.d., ultima linie fiind: Z, A, B, C, \dots, X, Y .

Jocul poate fi practicat de patru (sau mai mulți) parteneri, fiecare avînd cîte un calculator la dispoziție. Fiecare partener propune o frază compusă din 6-8 cuvinte. Studiul

fiecărei fraze constituie o rundă, învingător fiind acela ce va câștiga primul număr de cinci runde. Studiul frazei constă din următoarele: se ia fiecare literă din frază și i se determină una din cele 26 de coordonate posibile; de exemplu, litera *N* are coordonatele (1,14), (2, 13), (3, 14), ... (14, 1), (15, 26), ..., (26, 15); se determină produsul dintre abscisa și ordonată (pe exemplul literei *N*, cea mai mică valoare este $1 \times 14 = 14$, iar cea mai mare, $20 \times 21 = 420$); se alege una din aceste valori; se însumează valorile calculate corespunzătoare fiecărei litere din frază, obținând astfel ponderea frazei. Ponderea trebuie să fie mai mică decât 15 000. Câștigă un punct jucătorul care a obținut ponderea cea mai mare, inferioară însă lui 15 000. Dacă sînt doi sau mai mulți jucători care au obținut aceeași pondere maximă, fiecare va câștiga câte un punct la această rundă.

Câștigătorul jocului este cel care a obținut câștig în cinci runde. Se recomandă, ca strategie, să se ia la începutul frazei, numere mari și apoi să se manifeste o prudență mai mare.

Pentru a înlesni înțelegerea jocului să presupunem fraza NOI JUCĂM UN NOU JOC INTERESANT. Să analizăm numai primul cuvînt (NOI) și presupunem că deținem rolul jucătorului *A*. Vom alege pentru *N* coordonatele (8, 7), pentru *O* luăm (21, 21), iar pentru *I* — (16, 20) Atunci ponderea cuvîntului este $8 \times 7 + 21 \times 21 + 16 \times 20 = 56 + 441 + 320 = 817$. Jucătorul *B* poate alege *N* (12,3), *O* (23,19) și *I*(7,3), adică obține $12 \times 3 + 23 \times 19 + 7 \times 3 = 36 + 37 + 21 = 494$. Jucătorul *C* alege *N* (25, 16), *O* (1, 15), *I* (10, 26), adică obține $25 \times 16 + 1 \times 15 + 10 \times 26 = 400 + 15 + 260 = 675$, iar *D* dacă alege *N* (3, 12), *O* (21, 21) și *I* (20, 16) obține $3 \times 12 + 21 \times 21 + 20 \times 16 = 36 + 441 + 320 = 797$. La acest stadiu al jocului *A* conduce înaintea lui *D*, *C*, *B*.

Se poate propune și o variantă a acestui joc, ce constă în următoarele: frazele sînt extrase dintr-o carte și citite de un arbitru; fiecare jucător trebuie să procedeze la toate alegerile de coordonate și să efectueze calculele în maximum 10 minute; două greșeli de „ortografie“ elimină jucătorul respectiv din rundă.

O problemă de logistică. Vă propunem un joc în care fiecare partener simulează rolul unui comandant militar. La joc participă patru astfel de comandanți; fiecare dispune de un calculator personal și o „armată” formată din 100 pioni de aceeași culoare, pionii reprezentînd regimente, blindate, avioane sau nave.

Scopul jocului este de a obține patru grupe identice a câte 25 pioni din fiecare culoare.

La începutul jocului are loc tragerea la sorți a ordinii de joc și fie aceasta A, B, C, D . Jucătorul A alege un număr cuprins între 2 și 12, numit coeficient de schimb, c . Aceasta va oferi unuia din parteneri (B, C sau D) un număr oarecare de pioni de aceeași culoare, după rezultatele următoare: presupunem că A are 65 blindate și vrea să-i ofere lui C o parte din ele, coeficientul de schimb al lui A fiind 7; el poate da un număr de blindate egal cu unul din numerele: (a) $65 - 7 = 58$; (b) $56 \times 7/100 = 4$ (rezultatul se rotunjește la întregul superior; $7/100$ reprezintă 7%); (c) $65/7 = 9$ (rotunjire la număr întreg inferior); (d) fie chiar 7. Este rîndul lui B care — în mod asemănător — alege un număr cuprins între 2 și 12 și oferă unui partener un număr oarecare de pioni de aceeași culoare (blindate, avioane, nave sau regimente) după regula dată. Apoi, joacă C, D, A, \dots Jocul se termină cînd fiecare partener are cîte 25 regimente, nave, avioane și blindate.

Se recomandă să adoptați strategia ca la început să nu schimbați decît cantități mici.

O variantă a acestui joc constă în următoarele: dacă după un anumit număr de runde (o rundă este echivalentă cu un ciclu $ABCD$) „armatele” nu sînt egalizate, fiecare partener abandonează cîte cinci din pionii săi (la alegere), ca rezultat al uzurii materialelor. În acest caz, jocul se încheie în momentul cînd sînt 100 pioni abandonăți.

Observație. Dacă nu posedați 400 de pioni, atunci pe calculatorul dumneavoastră notați: r — regimentele, a — avioanele, n — navele și b — blindatele, reținînd pe ecran modificările de trupe corespunzătoare.

Număr magic. Vă propunem un nou joc la care participă trei jucători, fiecare utilizînd calculatorul său personal. La început, jucătorii A, B, C își aleg cîte un număr „magic”,

pe care îl introduc într-un plic, fără a ști ce număr și-au ales ceilalți (aceste trei numere sînt compuse din cîte cinci cifre). Ecranele celor trei calculatoare sînt vizibile tuturor jucătorilor. Fiecare își înscrie pe ecran un număr cuprins între 1 și 10.

Jocul începe într-o anumită ordine stabilită prin tragere la sorți; fie aceasta A, B, C . Partenerul A începe jocul prin alegerea unei operații aritmetice de bază ($+$ adunare, $-$ scădere, \times înmulțire și $:$ împărțire) și a unui număr din intervalul 10-999. Toți jucătorii execută operația anunțată de A între numărul de pe ecran și cel anunțat de A (dacă la operație de împărțire s-a obținut un număr real, el se va rotunji la întregul superior; de exemplu, 59,01 se rotunjește la 60). Rezultatele obținute se înscriu pe ecranele calculatoarelor. Jucătorul B își alege operația aritmetică ($+, -, \times, :$) și numărul din intervalul 10-999; toți jucătorii execută această operație între actualul număr de pe ecran și cel enunțat de B , înscriindu-și fiecare rezultatul pe ecran (atenție la rotunjirea rezultatului obținut prin împărțire). Este rîndul lui C să anunțe operația și numărul, jocul continuînd apoi în ordinea A, B, C, A, \dots . Dacă un jucător a obținut la un moment dat pe ecranul său un număr mai mare decît 999 999 sau mai mic decît $-999\,999$, atunci el este eliminat din joc, învingător în acest caz fiind cel care rămîne singur. Dacă însă pe parcursul jocului, un jucător a atins exact numărul său magic înscris în plic la începutul jocului, atunci el este declarat cîștigător (acesta este de fapt adevăratul cîștigător).

Pentru a înțelege mai bine strategia jocului, presupunem că inițial jucătorii și-au ales numerele de plecare 1, 2 și, respectiv, 3. Jucătorul A anunță operația $+$ (adunare) și numărul 400. Atunci ecranele celor trei parteneri conțin numerele 401, 402 respectiv 403. B anunță operația \times (înmulțire) și numărul 22. Acum ecranele vor conține $401 \times 22 = 8\,822$, $402 \times 22 = 8\,844$, $403 \times 22 = 8\,866$. Jucătorul C anunță operație $+$ (adunare) și numărul 277, iar numerele obținute pe ecrane sînt: $8\,822 + 277 = 9\,099$, $8\,844 + 277 = 9\,121$, $8\,866 + 277 = 9\,143$ și așa mai departe în ordinea A, B, C, A, \dots .

În desfășurarea jocului este bine să ghiciți numerele magice ale adversarilor, pentru a-i îndepărta cât mai mult. De asemenea, dacă doriți să eliminați din adversari alegeți la un pas oarecare numărul 999 dar, atenție, să nu vă „sinucideți“.

Jocul poate fi mai dinamic și plin de surprize dacă vă puteți alege și alte operații, cum ar fi calculul rădăcinii pătrate sau calculul logaritmului zecimal. În aceste cazuri, un criteriu de eliminare a unui jucător este ca numărul înscris pe ecranul său să fie nepozitiv.

TENDINȚE

9.1. MUTAȚII PRODUSE DE IBM PS/2

Dintre trăsăturile caracteristice ale familiei PS/2 menționăm: amplasarea pe placa principală a blocului de comandă a monitorului grafic și a blocului interfețelor, folosirea de circuite integrate proiectate la comandă, tehnologia montării la suprafață (TMS) și tehnica simplă, modulară de compunere. Folosirea de circuite VLSI și TMS a permis utilizarea unei surse de o putere mai mică și montarea unor ventilatoare mai mici (deci, mai puțin zgomotoase). Sursa se adaptează la tensiunea și frecvența rețelei de alimentare. Tastatura folosită este cea de XT. Se poate opta pentru un dispozitiv de introducere tip *mouse*.

Modelul 30 este oarecum un PC-XT modernizat. Pe placa principală sînt amplasate multe blocuri întîlnite pînă acum pe plăci suplimentare: varianta perfecționată a comenzilor grafice color — MCGA, interfețe (serială și paralelă), ceas/calendar și comanda stațiilor de discuri. Față de modelul 30, în modelele 50, 60 și 80 s-au folosit:

— o nouă magistrală multiprocesor, denumită **MICRO CHANNEL**;

— blocul de comandă a monitorului grafic (VGA), amplasat pe placa principală, care asigură parametri superiori nu numai față de CGA, ci și chiar față de EGA;

— acționarea discurilor flexibile cu o capacitate de 1,44 Mo, permițînd de asemenea citirea dischetelor de 720 Ko;

— comanda discurilor rigide cu un timp scurt de acces (cu excepția modelului 50), cu raportul 1 : 1 (în varianta AT raportul era 3 : 1).

Modelele 60 și 80 sînt echipate cu interfață ESDI (*Enhanced Small Device Interface*) care permite comunicarea cu discul rigid de șase ori mai rapidă decît varianta AT. Modelul 80 folosește același procesor ca și COMPAQ DESK-PRO 386. Are memoria operațională compusă din noile circuite de 1 Mo și magistrala MICRO CHANNEL în varianta de 32 biți. Puterea de prelucrare a modelului 80 este, conform IBM, de 3,5 ori mai mare decît AT.

Noile modele se impun înainte de toate prin posibilitățile grafice. Printre defecte se poate enumera încetinirea procesorului 80 286 în cazul modelului 50 : datorită memoriei lente a fost necesar ca la fiecare ciclu să se adauge un ciclu suplimentar de întîrziere (WAIT). În afară de aceasta, discul rigid are un ceas mediu de acces de 80 milisecunde, ceea ce constituie un regres față de modelul AT 3.

În familia calculatoarelor PS/2 s-au introdus trei noi rezolvări ale comenzilor grafice. Prima dintre ele (în modelul 30) poartă numele MGGA (*Multi Color Graphics Array*). La comandă, VGA poate fi montat și în modelul 30. Există și al treilea tip de comandă, denumită 8 514/A, pe o placă ce trebuie achiziționată separat.

MGGA se compune dintr-un circuit special produs la comandă, 64 Ko RAM cu dublu acces (*Dual-Ported*) și 16 Kb generator semne. Sistemul permite realizarea următoarelor noi moduri de lucru :

— text ; 80 coloane definiție 640 × 400 puncte, dimensiune semn 8 × 16 puncte, 16 culori alese din cele 266 144 posibile ;

— grafic ; definiție 320 × 200 puncte (256 culori alese), dimensiunile semnului 8 × 8 puncte.

Folosind o placă suplimentară împreună cu MCGA, se pot obține un text cu definiție de 720 puncte pe orizontală și dimensiunea semnului de 5 × 16 puncte. MCGA poate de asemenea emula modurile de lucru ale plăcii CGA, dar — fără o placă suplimentară — nu și EGA.

Elementul de bază al VGA este un circuit special, ce conține 12 750 porți. VGA realizează toate modurile stan-

dardului MCGA și, în plus, modurile EGA, precum și text 720×400 puncte, semn 9×16 , puncte, grafic 640×480 puncte și 16 culori.

Livrat la comandă, 8 514/A folosește priza suplimentară VIDEO și una din prizele magistralei MICRO CHANNEL (deci, nu poate fi folosit în modelul 30).

Placa 8 514/A blochează semnalele comenzii grafice de pe placa principală și le înlocuiește cu cele proprii. Este permisă astfel obținerea pe unul din noile monitoare IBM (8 514) a unei imagini de înaltă definiție (1024×768 puncte).

■ O placă suplimentară permite creșterea numărului de culori la 256, alese dintre cele 266 144 posibile. Placa 8 514/A (chiar și pe un ecran cu definiție mai mică) asigură în plus:

— definirea programată a formei și proporțiilor semnelor și scrisului;

— umplerea zonelor ecranului cu un anumit model grafic;

— deplasarea blocurilor grafice pe ecran.

Sistemele MCGA și VGA pot lucra cu oricare dintre noile patru monitoare: alb-negru (8 503), cu două culori de definiție medie (8 512 și 8 513) și color cu definiție înaltă. Acestea sînt monitoare analogice, respectiv semnalele purtătoare de informație video care permit o continuă (nu în salturi, ca în cazul monitoarelor digitale) schimbare a culorii sau nuanței. Fiecare monitor lucrează cu frecvență de selectare orizontală de 31,75 kHz și o frecvență de selectare a imaginii de 50 sau 70 Hz.

Banda de trecere este de 70 MHz. Monitorul 8 503 are diagonala ecranului de 31 cm. VGA și MCGA descoperă automat conectarea monitorului alb-negru și formează corespunzător semnalul video, trimițînd pe linia de semnal verde un semnal cvasianalogic care alege una din cele 64 de valori de codificare a gradului de gri. Monitorul are diagonala de 16" și poate lucra în toate standardele grafice ale familiei.

Sistemul BIOS în calculatoarele noii familii este aproape integral compatibil (din punctul de vedere al punctelor de intrare) cu sistemul BIOS din calculatoarele PC XT și AT,

adică programele anterioare de comunicare cu aparatura prin procedurile BIOS pot fi realizate și prin intermediul noilor calculatoare. De menționat că numai programele a căror funcționare corectă depinde de timpul în care sînt executate anumite proceduri nu vor funcționa corect datorită vitezei mai mari de lucru a calculatorului (8 MHz, față de 4,77 MHz).

Modelele 50, 60 și 80 au un sistem BIOS lărgit care se compune din CBIOS (BIOS compatibil), ce adresează 1MB memoria, și ABIOS lărgit) ce adresează 16 MB și permite multiaccesul.

În cadrul modelului 30, BIOS este amplasat în ROM, în două circuite 27 256, iar în modelele 50 și 60 — în patru circuite 27 256 și ocupă 128 KB. În fiecare calculator blocul ROM care conține sistemul BIOS are la adresa F000 = FFFF un bait care identifică modelul calculatorului.

IBM a schimbat anumite întreruperi BIOS fără mare importanță practică. De exemplu, actualmente sînt rezervate întreruperile OB și OC (comunicare), OD (imprimantă suplimentară), OF (imprimantă), 71 pînă la 74 și 76, 77 (IRQ 9, 10, 11, 12, 14, 15). Întreruperile F1-FF, pînă acum nefolosite, sînt alocate beneficiarului.

Pentru beneficiarul mediu schimbările programelor tipice IBM nu sînt vizibile. Ca argument că s-a păstrat compatibilitatea este faptul că ROM conține aceeași variantă pe casetă de BASIC VC1.10. Anumite programe din surse independente de IBM au nevoie de schimbări importante.

O caracteristică importantă a celor trei modele mai mari ale noii familii IBM este magistrala multiprocesor MICRO CAHNEL (MC), care se deosebește de magistrala folosită în modelele PC XT și AT ca standard mecanic și ca topografie, uneori și din punctul de vedere al caracterului semnalelor (al acelor care îndeplinesc funcții analogice).

La modelele 50 și 60 s-a folosit varianta de magistrală de 16 b, în timp ce la modelul 80, în 5 conectori s-a folosit varianta 16 biți, iar în cazul modelului 30 — varianta 32 biți. Nivelele logice ale tuturor semnalelor magistralei sînt conforme cu standardul TTL. Prin magistrală trec trei feluri de linii de alimentare: -12 V, 5 V, +12 V.

Topografia magistralei a fost proiectată cu accent deosebit pe eliminarea interferențelor electromagnetice. Din fiecare parte a conectorului magistralei, fiecare a cincea linie are potențialul maxim sau egal cu una din tensiunile de alimentare. O asemenea repartizare a semnalelor este foarte importantă când frecvențele sînt mari. Astfel, întregul sistem îndeplinește cerințele referitoare la cîmpurile parazite maxime conform normei S.U.A. (FCC pentru aparatele clasă B). MC este magistrală asincronă. Ca și în PC XT/AT s-au folosit linii separate pentru adrese și date. Standardul mecanic impune ca plăcile folosite să aibă dimensiuni 11,5" × 3".

Se folosește standardul așa-numitelor conectori-margine (?), ca în cazul PC XT-AT, care micșorează costurile, dar și fiabilitatea sistemului.

Elaborînd noua versiune a PC-DOS, firma Microsoft, a menținut compatibilitatea echipamentelor respective, fapt ce a limitat modificările de fond în noile variante ale sistemului. Pentru a învinge aceste limitări și a folosi procesoarele 80 286 și 80 386 s-a creat un nou sistem, OS/2 (Operating System/X).

OS/2 — sistem operațional multi-task pentru un singur beneficiar — permite executarea unui task în timpul real al microprocesorului 80 286 și mai multe în mod supravegheat, (*protected*). Deoarece setul de comenzi 80 386 reprezintă un set lărgit al lui 80 286, OS/2 va putea lucra și cu procesorul 80 386. Scopul principal al firmei Microsoft a fost crearea de programe pentru biroul automatizat. În acest sistem orice utilizator ar avea calculatorul propriu pe care ar executa simultan mai multe probleme și ar comunica prin rețea cu mai multe calculatoare. Noul sistem introduce o funcție cu totul nouă și este compatibil cu variantele anterioare MS-DOS. Aceasta înseamnă că programele care comunică cu hardul prin interfețele oferite de PC-DOS sau MS-DOS vor putea colabora cu OS/2 chiar dacă nu conțin proceduri dependente de tipul executării programului. Datorită vitezei mai mari a procesorului 80 286 vor trebui noi variante de programe dependente de timp (de exemplu, programe de comunicare). OS/2 poate de asemenea funcționa pe calculatoarele de pînă acum:

IBM cu 80 286, PC-AT și PC-XT 286. Însă, înainte de toate sarcina sa se referă la colaborarea cu noile calculatoare PS/2 (modelele 50, 60 și 80).

Sistemul OS/2 oferă două moduri de lucru: *Real* și *Protected*. În modul *Real* execută fără schimbări programele scrise pentru 80 286, iar în modul *Protected* sarcinii executate îi este distribuită o anumită zonă de memorie, iar sistemul nu are dreptul de a efectua operații IN-OUT; în cazul cererii de acces la altă zonă de memorie sau încercării de executare a unei operații IN-OUT se generează întreruperi și apoi trecerea în sistemul operațional.

În OS/2 în mod real (definit ca 3.X) pot lucra toate programele care colaborează cu variantele anterioare MS-DOS 3.1, 3.2 și 3.3 notată ca MS-DOS 3.XX. La crearea noilor programe este mai comod modul *Protected*, deoarece a fost dotat cu mecanisme perfecționate de colaborare cu programele utilitare (API — Application Programmer Interface) și cu instrumente mai bune pentru punerea în funcțiune a programelor.

Prelucrarea mai multor sarcini pe aceeași mașină este permisă de grupele de ecran (SCREEN GROUPS, SG). În calculatorul real funcționează simultan mai multe calculatoare virtuale și grupe de ecran (SG), unele dintre ele fiind conectate de utilizator. La ultima variantă de sistem, în acest scop se folosește o interfață grafică specială PM (*presentation manager*). În fiecare moment beneficiarul va avea acces la meniul din partea de jos a ecranului (*Drop-down MENU*). În scopul conectării cu o altă SG sînt suficiente fixarea cursorului pe poziția corespunzătoare a meniului cu ajutorul dispozitivului *mouse* și apăsarea butonului. Programul PM, care funcționează în mod *Protected*, va avea trăsături exterioare identice cu MICROSOFT WINDOWS, ce funcționează în mod *Real*. (PM este responsabil de organizarea introducerii informației pentru multe sarcini pe un singur monitor).

Structura dischetelor OS-2 este identică cu PC DOS-3, fapt ce permite folosirea în noul sistem a discurilor utilizate în calculatoarele PC cu aceleași limitări (de exemplu, capacitatea maximă 32 MB). S-au introdus totuși mecanisme care în variantele viitoare ale sistemului vor îndepărta

limitările respective. În OS-2, denumirea fiecărei dischete (*volume, name*) este foarte importantă. În sistemul multi-sarcină scrierea de către program pe altă dischetă decât cea corectă poate provoca multe complicații. În consecință, pe dischetă s-a introdus un câmp suplimentar unde — în afara denumirii date de utilizator — este scris un număr de 32 biți generați de sistem.

Înainte de folosirea în sistemul OS-2 a dischetei din sistemul DOS-3 trebuie neapărat dată instrucțiunea *LABEL* care provoacă atribuirea unui nume unic dischetei.

Una din grupele de ecran (SG) este cea reală. După alegerea acestei grupe utilizatorul intră în mod *Real* identic cu lucrul cu MS-DOS. Adevărata putere a sistemului OS-2 este încorporată totuși în modurile supravegheate realizate de celelalte SG.

În sistemul OS-2 interfața cu utilizatorul este aproape identică cu cea din sistemul DOS, însă cu totul alta este interfața cu programatorul (API). În locul întreruperilor de program sînt folosite apelurile de proceduri. Ele vor fi conectate doar în momentul încărcării programului în memorie sau chiar în timpul executării lui. Deci, apar conectări dinamice ale procedurilor.

Conectările dinamice au multe avantaje: permit conectare unitară cu toate serviciile sistemului, descoperă organizarea acestor servicii de nivel inferior și permit să se dea procedurilor denumiri legate de destinația lor. Parametrii sînt conferiți procedurilor prin stivă în convenția acceptată în PASCAL (folosită de multe compilatoare ale limbajului C).

Noul mod de apel al serviciilor sistemului permite memorarea unei importante părți a sistemului operațional în biblioteci pe disc. O parte a funcției consolidatorului este construită în sistemul operațional și colaborază cu funcțiile de organizare a memoriei. Consolidatorul (LINKER) construiește plicuri care sînt executate cu folosirea unui cod special (STUB CODE). Acest cod, în afară de instrucțiunile în cod mașină din program, conține denumiri și puncte de intrare în procedurile conectare dinamic. Se deosebesc două feluri de conectări dinamice: inițiale (PRELOAD)

și la comandă (LOAD ON DEMAND). Primele sînt consolidate în momentul încărcării programului, celelalte — cînd apare apelul procedurii de sistem.

Adesea merită să se împartă o sarcină în cîteva ordine executate simultan. De exemplu, în programe destinate editării de tabele (*spread sheet*), după calculele de actualizare a cîmpurilor vizibile, pe ecran se poate executa simultan calculul cîmpurilor ce sînt în afara ecranului și introduce noi date. OS/2 permite crearea în cadrul unei sarcini a unor astfel de execuții, denumite *threads*. Pentru fiecare sînt „private“ doar conținutul registrelor procesorului și stiva, celelalte fiind comune pentru întreaga sarcină. În consecință, dacă un *thread* deschide un plic, celelalte pot scrie sau citi din el. Este necesar ca diferitelor *threads* să le fie alocată o prioritate. *Thread* este observabilă doar prin sarcina din care provine, din care motiv nu poate fi îndepărtată sau nu i se poate lua prioritatea de către altă sarcină.

OS/2 permite instalarea monitoarelor — respectiv, a programelor speciale — conectate cu programe de comandă a instalației. Monitorul poate adăuga, îndepărta sau schimba semne transmise de programul de control al instalației. La fiecare program de comandă se pot conecta multe monitoare, obținîndu-se un lanț de programe care modifică funcționarea aceluia program.

Păstrarea compatibilității cu DOS 3.XX a constituit unul din scopurile principale ale proiectanților OS/2. Se pare că jumătate din fonduri au fost afectate pentru asigurarea compatibilității. Aceasta este de fapt tendința mondială. Păstrarea compatibilității nu s-a reflectat însă asupra posibilităților sistemului. Actualmente, creatorul de programe utilitare are posibilitatea de a opta pentru trei medii de programare: microsoft, Windows, DOS 3.30, OS/2-*protected mode*.

Ne putem aștepta ca în curînd să apară doar programe corelate cu OS/2, deoarece asigură posibilitatea așteptării lor la sistemul de operare ce este creat special pentru 80386. Astăzi se crede că rezolvarea optimă este Family API, deoarece permite construirea de programe care colaborează atît cu DOS 3.XX. cît și cu OS/2.

9.2. UN CONTRACT ÎNTRE STRATEGII

Categoric, IBM nu are o existență liniștită în domeniul calculatoarelor personale, acesta fiind, de fapt, singurul segment din piața tehnicii de calcul a cărui dominare nu este întotdeauna concludentă (fig. 9.1). Antecedentele sînt cunoscute : deși a reușit să detroneze rapid pe Apple la începutul deceniului (cînd a lansat PC-ul său original), după o (relativ) scurtă perioadă de supremație absolută a început să piardă tot mai multe procente de piață în favoarea producătorilor de calculatoare compatibile, devenind astfel prizonierul propriului său standard. Mai mult decît atît, Apple își stabilizează permanent propria sa linie de calculatoare personale Macintosh, avînd proprii săi utilizatori fideli și amenințînd cu o replică standardul tradițional. În aceste condiții, IBM a lansat familia PS/2, dar de undeva din Texas apăruse Compaq, care devine brusc cel mai puternic concurent al IBM pe piața calculatoarelor personale, fapt cu atît mai inedit, cu cît IBM nu a mai cedat niciodată un domeniu pe care îl cucerise. Cum

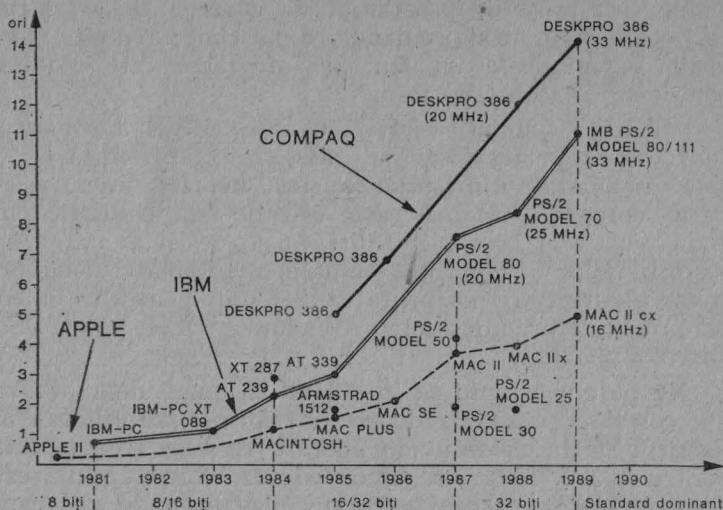


Fig. 9.1. Trei linii de dezvoltare — IBM, APPLE, COMPAQ

a fost posibil acest lucru și care sînt factorii care l-au determinat? Cum a putut o firmă care nici măcar nu există cînd IBM a introdus calculatoarele sale personale să devină o concurență așa serioasă?

În anul 1987, deși vînzările de calculatoare personale IBM erau mult mai mari decît cele ale firmei Compaq, veniturile sale au sporit foarte lent, în timp ce veniturile firmei texane înregistrau creșteri spectaculoase. De fapt, în anul 1987 Compaq a vîndut mai multe calculatoare personale bazate pe microprocesoare 80386 decît IBM și acest fapt a continuat și în anul următor. Astfel, *Compaq Deskpro 386* a devenit primul calculator personal non-IBM, care s-a impus ca lider în categoria calculatoarelor de birou (*desktop*). Acest fapt este în mod particular semnificativ, deoarece pînă în prezent toate bătăliile pierdute de „Big Blue“ în domenii particulare restrînse nu au reușit să fie recîștigate.

O dată cu lansarea familiei PS/2 prăpastia care separa pe cei doi mari rivali părea să dispară și astfel Compaq, care dominase sectorul calculatoarelor pe 32 de biți, se găsește în decembrie 1988 într-o poziție critică față de IBM. Apare însă o gravă penurie de calculatoare IBM rezultată, după unii analiști, atît din creșterea cererii, cît și din dificultățile întîmpinate la pregătirea fabricației. Partea deținută de IBM pe piața calculatoarelor cu 80386 scade de la 38% la 26%, în timp ce procentul deținut de Compaq crește de la 40% la 49%.

În 1989, IBM și Compaq își propun amîndouă să recîștige preferințele utilizatorilor de PC-uri. În consecință, concurența devine tot mai acerbă, iar faptul că sînt anunțate calculatoarele cu performanțe similare face ca tensiunea provocată de luptă să mai crească cu cîteva grade.

Iată cîteva din schimburile de lovituri ale celor două firme în anul 1989.

Compaq lansează un nou model 286 cu 12 MHz, cu care încearcă să acapareze și o parte din piața calculatoarelor bazate pe microprocesoare 80286, piață pe care IBM, cu tradiționalele AT-uri precum și cu PS/2 model 30 (în special modelul 30286 lansat în toamna lui 1988, prin el IBM arătînd că nu dorește să părăsească piața AT-urilor), era lider.

În consecință, IBM răspunde prin lansarea unui calculator portabil cu 386, precum și o mașină pe 386—33 MHz. Ca o ripostă Compaq lansează în aprilie 1989, propriile mașini de 33 MHz.

După opinia utilizatorilor, aceste anunțuri ilustrează perfect fazele diferite ale luptei care se duce între cei doi giganți pentru statutul de lider în inovarea tehnologică.

În conformitate cu părerea unor observatori, rivalitatea IBM-Compaq are și un caracter puțin curios, în sensul că cu cât Compaq se dezvoltă mai mult, cu atât are tendința de a semăna cu IBM. Iată ce declară Brian Roemmele, președintele unei societăți de proiectare tehnică, Ariel Computer: „Înainte Compaq propunea o alternativă efectivă fenomenului IBM. A cumpăra un Compaq era nu numai o decizie normală din punct de vedere tehnic, dar și un veritabil angajament politic. Astăzi, în schimb, prin prețurile sale și prin baza instalată, Compaq pare să adopte politica de întreprindere pe care înainte o contesta”. După Dan Howel, inginer analist la Texaco, însă „Compaq rămîne aceeași. Politica de marketing este poate mai conservatoare, dar deviza nu este *noi sau nimeni altcineva*, ca a IBM-ului”.

Un alt fenomen, consecință a concurenței acerbe, este reprezentat de presiunea enormă care determină o punere în vânzare foarte rapidă a calculatoarelor, iar acest lucru creează deseori neplăceri. Ca urmare, IBM a anunțat că există probleme tehnice pentru modelul 70/A 21 și a suspendat temporar fabricația acestuia. Un comerciant de calculatoare IBM a declarat că 4 din cele 6 modele primite de 70/A 21 erau inutilizabile încă de la sosire. Este destul de neobișnuit din partea IBM să livreze un produs avînd probleme așa de grave, știut fiind faptul că de obicei acesta este supus unor teste complete. Astfel, cu toate sumele uriașe cheltuielile pentru promovarea liniei PS/2 și atenționarea asupra proprietății arhitecturii MICRO CHANNEL, IBM nu a reușit să acapareze nici măcar un segment din piața ce aparține firmei Compaq. Mai mult decît atât, Compaq era aceea care continua să cîștige părți din piața IBM. Ce a făcut deci Compaq și nu a reușit IBM să facă?

Răspunsul la această întrebare dezvăluie modul în care IBM a conceput însăși piața de calculatoare personale și

sugerează, de asemenea, că prin aceasta îi va fi dificil să-și mențină poziția de cel mai mare vânzător de calculatoare personale din lume. Primul lucru care se poate spune despre succesul lui Compaq este acela că, în mod neîndoielnic nu s-a bazat pe prețuri mai scăzute. Dimpotrivă, prețurile Compaq sînt, în mod tipic, mai mari comparativ cu acelea ale celorlalți competitori. Cumpărătorii care sînt interesați în primul rînd de preț pot achiziționa calculatoare de la alți producători de astfel de echipamente la prețuri mult mai accesibile.

Doi au fost însă cei mai importanți factori care au determinat succesele obținute de Compaq — calitatea și performanța produselor oferite, sistemele Compaq bucurîndu-se de o mare reputație. De exemplu, recentul Deskpro 386/20 al firmei Compaq depășește în performanță vîrfurile liniei PS/2 și anume modelul 80/111, cu toate aprecierile la adresa presupusei sale superiorități tehnologice.

În timp ce performanțele superioare ale recentelor modele Deskpro 286 comparate cu modelele PS/2 50 și 60 erau datorate, în primul rînd, vitezei mai mari a ceasului (12 MHz, față de 10 MHz pentru sistemele IBM), calculatorul Deskpro 386/20 depășește modelul 80-111, la aceeași viteză a ceasului, în pofida existenței magistralei M.C.A.

Numai performanțele superioare ale Compaq-ului nu explică însă suficient succesul firmei. Calculatoarele altor producători oferă deseori performanțe superioare, de multe ori la prețuri mai scăzute.

Alt factor în succesul competițional al firmei Compaq îl reprezintă constanta sa abilitate în livrarea de produse pentru o piață ce se suprapune peste cea a IBM. De la început, strategia Compaq a fost aceea de a oferi facilități pe care IBM nu le punea la dispoziția beneficiarilor. Sistemul inițial, Compaq Portable, este un exemplu în acest sens. Deși sistemele create de Adam Osborne au precedat echipamentele Compaq, ele se bazau pe microprocesoare Z-80 și sistemul de operare CP/M. Compaq a oferit, în schimb, compatibilitatea IBM PC într-un sistem portabil și de calitate, și aceasta cu mult înaintea IBM. Mai recent, Deskpro 386 a precedat cu luni lansarea primului model IBM bazat

pe 386 (PS/2 Model 80). În timp ce IBM era gata pentru introducerea primei versiuni a modelului 80 (pe 16 MHz), Compaq comercializa deja sistemul Deskpro 386/20 (pe 20 MHz). În mod similar, chiar înainte de anunțarea modelelor IBM bazate pe 286 (model 50 și 60) la 10 MHz, Compaq trecuse la modelele 286 pe 12 MHz (Deskpro 286). Calculatorul transportabil Compaq 386 este un alt exemplu al consecvenței abilități a firmei de a se afla înaintea IBM în ceea ce privește oferirea de facilități reale utilizatorilor.

Succesul firmei Compaq a fost facilitat și de faptul că firma respectivă a rămas credincioasă vechiului standard pe care IBM l-a creat o dată cu lansarea calculatoarelor IBM PC, XT și AT. Inovațiile care au menținut facilitățile oferite de Compaq în fața celor IBM au reprezentat, de fapt, modele mai reușite de implementare a standardului pe care IBM l-a creat în mod neintenționat.

Probabil că cea mai semnificativă diferență între strategiile IBM și Compaq se va găsi, totuși, în măsura în care cele două firme au livrat o arhitectură cu adevărat deschisă. Dacă se observă documentația pusă la dispoziție se constată un contrast evident. Specificațiile detaliate și listinurile BIOS-ului din primele referințe tehnice ale calculatoarelor IBM PC nu sînt astăzi decît istorie. În prezent, în timp ce politica de documentație a IBM este de a realiza scurte manuale, Compaq livrează două volume pentru ghidul tehnic de referință al lui Compaq Deskpro 386/20, care conțin atît schemele electrice, cît și descrieri detaliate ale funcțiilor BIOS. Astfel, documentația Compaq îmbunătățește standardele din documentația IBM. De asemenea, documentația Compaq se diferențiază și față de cea oferită de majoritatea comercianților de calculatoare personale, care — din cauza prețului scăzut — oferă un minim (dacă nu și un inadecvat) set de documentație.

9.3. STAȚII DE LUCRU INGINEREȘTI

Stațiile de lucru ingineresti reprezintă un domeniu relativ nou care a apărut după cel al calculatoarelor perso-

nale din care s-a desprins. În categoria stațiilor de lucru ingineresti sînt cuprinse stațiile grafice, echipamentele pentru proiectarea asistată de calculator (CAD), fabricația asistată de calculator (CAM), ingineria asistată de calculator (CAE) etc. O dată cu dezvoltarea calculatoarelor personale de mare performanță, stațiile de lucru ingineresti cunosc și ele un progres substanțial. De remarcat faptul că în anul 1986 vânzările de stații de lucru ingineresti au depășit pe cele ale calculatoarelor personale.

Piața stațiilor de lucru ingineresti a fost creată în anul 1981 de firma Apollo, care a și dominat apoi această piață, cu un procent de 30-40% din vânzări. În anul 1984, și-a făcut însă apariția firma SUN, care în perioada 1985/86 a preluat conducerea în acest domeniu.

O stație de lucru bazată în primul rînd pe tehnologiile de vîrf cuprinde:

- unitate centrală pe 32 de biți (sau mai mult);
- memorie internă de 2-4 Mo și, de obicei, suport de memorie virtuală;
- display integral cu o rezoluție de minim 640×480 pixeli;
- interfețe pentru rețea (de obicei Ethernet);
- sistem de operare multitasking (nu este neapărat necesar un sistem de operare multiutilizator, deși unele stații prezintă acest tip de sistem).

O dimensiune mare a registrelor memoriei interne, precum și un spațiu de adresare suplimentar (eventual virtual) sînt necesare pentru rularea unor aplicații tipice ingineresti. Rezoluția înaltă depinde direct de magistrala microprocesorului și permite o interfață cu utilizatorul prin ferestre, ceea ce are ca urmare creșterea productivității muncii la utilizarea stației. Facilitatea integrării într-o rețea este prima armă împotriva formării unor „insule de prelucrare informațională”, și în sfîrșit, sistemul de operare multitasking leagă toate aceste elemente împreună.

Exemple tipice de mașini care respectă definiția dată sînt stațiile care aparțin liniei VAX a firmei Digital Equipment, precum și formele cele mai utilizate ale producătorilor

tradiționali, ca SUN și Apollo. De remarcat faptul că există mai multe firme (InterAct, InterPro) unite sub numele de Intergraph, care produc stații pe bază de VAX sau MicroVax; în anul 1988 acestea dețineau împreună circa 29% din piața stațiilor de lucru.

Firma DEC a intrat pe piața stațiilor de lucru grafice în anul 1984, cu prima stație VAX pentru satisfacerea necesităților impuse de aplicațiile CAD/CAM/CAE. Deși cu performanțe ceva mai scăzute și costuri mai ridicate, linia de stații VAX a câștigat teren în mod continuu, devenind a treia putere după SUN și Apollo. În anii 1985/86, modelele VAX station II și VAX GPX reprezentau unele dintre cele mai căutate produse pe piața mondială, oferind, în afara facilităților grafice specifice domeniului, și celelalte programe aplicative ce funcționau sub sistemul de operare VMS (după unele estimări, DEC a livrat în 1985 echipamente CAD/CAM pentru circa 1/3 din piața mondială). În acea perioadă, avîndu-se în vedere faptul că producătorii declarau mărirea capacității și performanțelor grafice, se prognoza pentru stațiile de lucru cu grafică în 3 D o creștere de 60% pentru 1988. Dar Charles D. Wise, software manager pentru VAX/VMS, a prevăzut un conflict între stațiile VAX și calculatoarele VAX multiutilizator: „DEC are un interes vast la mini și supermini și, din această cauză, nu are de gînd să facă o afacere din stațiile de lucru care ar reduce vânzările pentru VAX 8000”. Urmarea a fost o scădere simțitoare a procentului din viața stațiilor de lucru deținută de DEC. Astfel la sfîrșitul anului 1987 piața stațiilor de lucru ingineresti se prezenta astfel:

Sun Microsystems	33%
Apollo	18%
DEC	3%
HP	2%

Iată și prognoza Dataguest la acea vreme în ceea ce privește performanțele stațiilor:

1987	7-10 MIPS;
1988	16-20 MIPS;
1989	30-40 MIPS.

Un exemplu de stație DEC la nivelul anilor 1987/88 îl reprezintă VAX 3200 cu performanțe de 10 MIPS.

De remarcat faptul că pentru primii trei producători de stații rata de creștere a vânzărilor este dublă față de restul (ca grup). Per total, rata de creștere pe piața stațiilor a fost în 1988 mult sub cea prevăzută, și anume 30%, rezultat care se datora parțial operației familiei de calculatoare PS/2.

Pe piața stațiilor de lucru au apărut și producătorii mai puțin tradiționali; Hewlett Packard care a venit din domeniul minicalculatoarelor, precum și Tektronix, din domeniul terminalelor, sînt două exemple în acest sens. De mai mulți ani Hewlett Packard joacă un anumit rol pe piața stațiilor de lucru. Deși a vîndut un număr mare de unități, unele din acestea nu sînt conforme cu definiția dată anterior stațiilor. Astfel, cu toate că a comercializat cu rezultate bune seria 9000 (care se potrivește standardului, performanțelor și prețului), Hewlett Packard rămîne mai mult un producător de echipamente care fac parte din categoria sistemelor la cheie.

Tektronix a început să fabrice a doua sa serie de stații de lucru. Prima încercare (seriile 61 XX și 62 XXs) a constituit o semireușită, prezentînd cîteva probleme de producție atît cu circuitul National 32000, cît și cu software-ul. Concepția firmei este că piața tradițională de terminale grafice se va transforma rapid în piață de stații de lucru. Date fiind mărimea și forța sa financiară, Tektronix rămîne o firmă a cărei evoluție pe piața stațiilor de lucru trebuie urmărită.

Alt producător este desigur IBM. Însă singurul său produs care se potrivește definiției date este stația 6150 PCRT, cu toate că și PS/2 Model 80 cu OS/2 și Model 55 cu 80386 SX pot fi considerate adevărate stații de lucru. Deși a îmbunătățit stația 6150 PCRT, aceasta este departe de a reprezenta un succes răsunător. Stațiile PCRT sînt declarate de firmă echipamente profesionale pe 32 de biți, cu sistem IBM/AIX, prețul lor fiind destul de mare, mai ales datorită faptului că sînt realizate cu microprocesoare

IBM originale. Iată și membrii stației 6150 PCRT, împreună cu prețurile corespunzătoare anilor 1988/89:

IBM PC RT Model 1 (<i>desktop</i>)	13 000 dolari
IBM PC RT Model 2 (<i>floor-standing</i>)	40 000 dolari
IBM PC RT 130 (<i>desktop</i>)	23 000 dolari
IBM PC RT 135 (<i>floor-standing</i>)	30 000 dolari
IBM PC B 35 (+IBM 5 080 Graphics)	32 000 dolari

Cu toată această gamă variată, PS/2 Model 80 și mult discutatul Model 90 se încadrează mai bine în definiția dată stațiilor de lucru. Dată fiind baza instalată de PC-uri, aceste mașini pot determina ca firma IBM să devină un producător foarte serios de stații. De asemenea, calculatorul personal MAC II cu A/UX poate aduce și firma Apple în lumea stațiilor de lucru.

Liderul stațiilor de lucru ingineresti, SUN, și-a întinerit toată gama de produse, introducând — în anul 1989 — 10 noi modele pe linia mașinilor cu microprocesor 68000, precum și cele cu arhitectură cu set redus de instrucțiuni (RISC). Iată două din produsele lansate, împreună cu caracteristicile lor:

- SUN D 3/80, cu microprocesor 68030 la 20 MHz, memorie internă de 4 Mo și sistem de operare UNIX sau MS/DOS (performanțe: 3 MIPS);

- SUN D 3/400, cu microprocesor 68030 la 33 MHz, memorie internă de 8-12 Mo (performanțe: 7 MIPS).

De remarcat că prețul este comparabil cu cel al unui calculator personal MacIntosh II sau PS/2 Model 70.

Deși în mod tradițional stațiile de lucru au fost utilizate mai mult în aplicații ingineresti, un mare număr de stații sînt folosite în mod curent în alte domenii: controlul proceselor, analiza calității, gestiunea stocurilor etc.

În aceste cazuri posibilitățile de multitasking și interfețele grafice sînt dominate de controlul de timp real. Domeniul publicațiilor, în particular influențat puternic de revoluția electronică, este un consumator tot mai mare de stații de lucru. Întrucît prețul stațiilor tinde să scadă

mult, ele vor înlocui calculatoarele personale ca resurse desktop pentru cei care au posibilități financiare mai mari. De exemplu, tradiționala piață a automatizării birourilor (biroticii), dominată acum de PC-uri, va fi caracterizată în curînd de echipamente orientate pe interfețe iconice, cu meniuri controlate și ferestre etc., oferind astfel o productivitate de grup mai bună și mai puține posibilități de erori. Pentru întreprinderile medii, puterea de rețea a stațiilor poate rezolva problemele de achiziționare a informațiilor. Pentru manageri stațiile pot reprezenta o adevărată revoluție, în sensul utilizării unor aplicații precum *sisteme de raportare on line*, care nu necesită probleme deosebite de aptitudini și experiență în utilizarea tastaturii.

Altă aplicație însemnată a stațiilor o reprezintă așa-numita „vizualizare a software-ului“. Există o direcție relativ nouă de dezvoltare provenită din prezentări grafice, deosebit de utilă în cercetarea științifică fundamentală pentru realizarea de modele în domenii diverse — dinamica fluidelor, prognoza meteorologică, fizica nucleară etc. Vizualizarea software este utilizată pentru a memora și prelucra volume mari de informații produse de sisteme de prelucrare a imaginii și supercalculatoare. Stațiile de lucru prezintă un prilej unic, datorat posibilităților grafice și de cuplare în rețele, în vederea controlului în timp real al acestor resurse.

Piața de stații de lucru se caracterizează printr-o rată de creștere de două ori mai mare față de cea a întregii industrie de calculatoare. Primii doi mari producători în domeniul tehnicii de calcul și informaticii mondiale, IBM și DEC, recunosc importanța strategică a acestor calculatoare.

Raportul preț/performanță al stațiilor de lucru este în scădere (fig. 9.2) datorită atât micșorării prețului, cât și creșterii performanțelor.

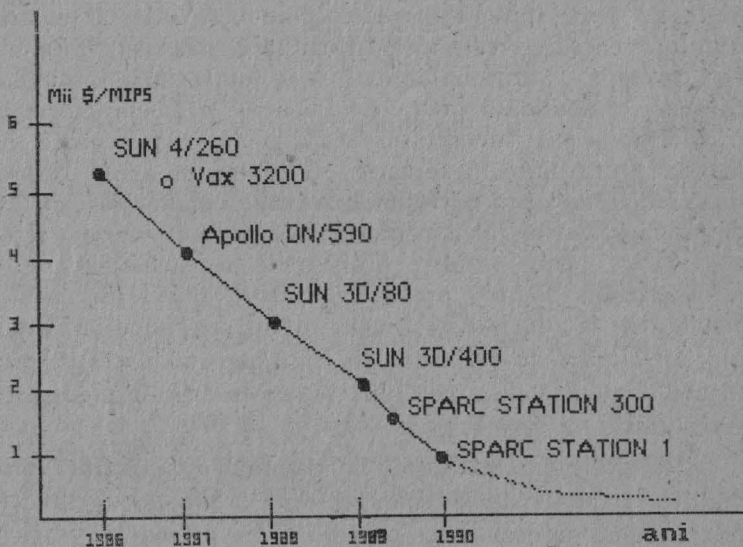


Fig. 9.2. Raportul cost/performanță

9.4. SECURITATEA INFORMAȚIILOR PE PC

Tot mai multe fenomene apărute în ultimii ani, o dată cu proliferarea extraordinară a calculatoarelor personale, arată că, neîndoielnic, securitatea informațiilor nu mai poate fi privită în mod superficial, ci ca o necesitate.

În trecut, informațiile erau stocate pe sisteme de calcul de capacitate mare, apoi, în anii '70-'80 și-au făcut apariția minicalculatoarele, care au permis o descentralizare a informațiilor la un prim nivel. La începutul deceniului '80, apariția microcalculatoarelor a amplificat procesul. În prezent, rețelele sînt cele care duc fenomenul mai departe. Atît proliferarea micro-urilor, cît și multiplicarea rețelilor și a interconectărilor includ astfel și numeroase amenințări și riscuri. Care sînt aceste riscuri?

Generalizarea în întreprinderi a terminalelor inteligente pe bază de PC/PS oferă fiecărui utilizator o productivitate

sporită, lăsînd însă cîmp liber actelor de piraterie în domeniul informațiilor. În consecință vulnerabilitatea sistemelor informatice crește.

Proliferarea microcalculatoarelor din ce în ce mai performante permite oricărei persoane să „navigheze“ prin rețele și să pătrundă astfel în diverse sisteme informatice. Iar această activitate se poate desfășura chiar și la domiciliu. Aceste fenomene au apărut la începutul deceniului trecut, cînd studenți sau elevi au reușit în joacă (?) să pătrundă în cele mai păzite sisteme informatice (Pentagon, NASA, CIA). Fără îndoială, aceste fapte au permis să se înțeleagă că orice persoană inițiată în informatică, posedînd unelte adecvate și, în special, voință și multă răbdare, reprezintă un posibil intrus în sisteme de securitate cu reputație de inviolabilitate. Unele din aceste persoane au devenit, datorită experienței și „competențelor“ acumulate, cunsultați ai comisiilor de experți care se ocupă de securitatea informațiilor. De asemenea, au căpătat un nume distinct — *hackers* (pirăți), devenind astfel o categorie recunoscută din punct de vedere social.

Importanța din ce în ce mai mare a rețelelor, tot mai interconectate între ele, oferă unui eventual intrus posibilități dintre cele mai extinse. Iată cîteva exemple în acest sens.

În 1987, un cercetător american a observat o mică eroare la factura privitoare la utilizarea unui calculator din laboratorul în care lucra. În urma investigației întreprinse a constatat că suma corespunde facturii unui necunoscut, care se servește de calculatorul său pentru a pătrunde în rețeaua militară Milnet. După cîteva săptămîni, intrusul a fost reperat. S-a constatat că forțase intrarea în 450 calculatoare, în 40 de reprize, reușind să pătrundă la Pentagon, NASA, laboratorul nuclear din Los Alamos, etc. După o lungă anchetă care a depășit granițele S.U.A., desfășurîndu-se și în Europa, au fost implicate 10 persoane bănuite de spionaj. Grație posibilității de a „naviga“ pe rețele, ele au „vizitat“ cele mai protejate calculatoare din S.U.A. și Europa.

În decembrie 1987, rețeaua internă de calculatoare a firmei IBM a suferit un atac al unui „virus“, a cărui origine

se afla într-o telegramă de felicitare expediată prin rețea de către un utilizator acreditat. Felicitarea făcea să apară pe ecran un pom de Crăciun, dar, în realitate, ascundea un adevărat „cal troian“, care — în timpul afișajului — consulta fișierele calculatorului. Apoi, înarmat cu numele persoanelor care au schimbat mesaje cu acest utilizator, trimitea copii ale telegramei (dar și ale programului „cal troian“) la toți utilizatorii recenzați. Pentru a se debarasa de „virus“, IBM a trebuit să închidă sectoare întregi ale rețelei.

În decursul anului 1987 s-au înregistrat în întreprinderile din Franța circa 30 000 de accidente informatice deci, practic, aproape 100 zilnic. Pierderile cauzate au fost evaluate de societățile de asigurări la 8 miliarde de franci, ceea ce a reprezentat o creștere cu 8% față de anul precedent. Periodic se organizează congrese (cum este, de exemplu, Securicom) care atenționează întreprinderile ce dispun de sisteme informatice în vederea protejării acestora. În general, atît concluziile acestor congrese, cît și ale unor specialiști și instituții specializate arată că securitatea informațiilor ține, în primul rînd, de o stare de spirit, iar întreprinderile și conducerile lor trebuie să analizeze metodic riscurile la care se expun instituțiile respective, mijloacele de care dispun și mediul în care își desfășoară activitatea, în vederea reducerii riscurilor, neexistînd un model unic de securitate informatică.

Sindromul „fortăreței“ este aplicat de fiecare întreprindere în felul său propriu, dar cu toate că 85% din cei care răspund de securitatea informațiilor manifestă preocupări în ceea ce privește salvarea datelor, iar 75% dispun de control, se constată că în peste 25% din accidentele informatice, astfel de situații nu au fost prevăzute.

În conformitate cu opinia a numeroși experți, importanța strategică a informaticii în viața întreprinderii este subestimată, iar 30% din cadrele de conducere afirmă că securitatea este insuficientă. Fenomenul este însă mult mai complex, deoarece securitatea și evaluarea riscurilor nu reprezintă numai o problemă strict informatică, fiind necesar a se lua în considerare toți factorii implicați: spa-

tiul global al întreprinderii, nevoile tehnice, organizarea funcțională, elementul uman etc.

Din argumentele invocate rezultă că este necesar studiul tipurilor de pierderi. În acest sens, analizele efectuate au impus următoarele tipuri :

● *Pierderile datorate echipamentelor* se referă la costul reparațiilor sau al înlocuirilor echipamentelor care au fost defectate sau sustrase. În legătură cu acest aspect, trebuie subliniat faptul că accidentele informatice nu au numai cauze materiale. Cîteodată pot exista și motive legate de factorul uman — erori de supraveghere, de transfer de date sau programe, interpretare sau utilizare, exploatare, concepție, realizare de aplicații etc. Unele din acestea pot deveni veritabile amenințări pentru societate, cum a fost cazul unei aplicații în medicină în S.U.A., care a provocat cîteva victime datorită faptului că nu au fost verificate cu atenție toate ramurile posibile ale programului.

● Alte *pierderi* pot fi cele *financiare* sau *de clienți*. Este vorba de dispariția de bunuri financiare, în special în domeniul contabilității.

● În sfîrșit, cele mai „la modă“ *pierderi* se referă la „*bombele logice*“ și *virusi*, care se răspîndesc foarte repede.

În unele întreprinderi mici (care sînt și foarte numeroase) postul informatic este de obicei unic. Aceeași persoană se ocupă atît cu exploatarea, cît și cu sistemul, rețeaua, administrarea datelor și securitatea lor. Există, deci, un pericol de divulgare sau de piraterie a informațiilor.

Sustragerile de echipamente nu reprezintă evenimente rare, așa cum s-ar putea crede la prima vedere, ci ele se referă în special la obiecte mai puțin voluminoase — microcalculatoare, imprimante, modemuri etc. În acest caz, consecințele accidentelor informatice pot fi atît pierderi materiale, cît și legate de întreruperea activității. Dacă echipamentele pierdute pot fi lesne înlocuite, mai grave sînt pierderile informațiilor înregistrate pe discurile dure. În ceea ce privește deturnările, posibilitățile sînt extrem de variate, mergînd de la manipulări de fișiere sau programe, pînă la modificarea întregului sistem. În cea mai

mare parte aceste deturnări sînt funcționale: se realizează exploatari ilicite, fraude prin controale logice și de programe. Dacă pirateria de programe poate fi pusă la adăpost prin legea dreptului de autor, nu același lucru se poate realiza în legătură cu deturnarea informațiilor. Se practică în mod curent traficul cu listinguri, suporturi magnetice etc., iar întreprinderile cele mai expuse riscurilor sînt cele care realizează exporturi și cele cu tehnologie înaltă.

Iată etapele identificate în cadrul unui accident informatic într-o întreprindere:

— *etapa 1*: este bransat un ansamblu de proceduri, au loc ștergeri de date de pe suporti magnetici etc. Direcționarea se face de la locul respectiv sau de la distanță;

— *etapa 2*: este creat un eveniment de exploatare care duce la modificarea succesivă a salvărilor. Fișierele sînt distruse;

— *etapa 3*: este cea în care, de obicei, se intervine prin acționarea procedurilor de „redemărare”. În privința informațiilor, totul depinde de valabilitatea lor și de posibilitatea de a fi reconstituite, pornindu-se de la documente;

— *etapa 4*: se realizează sincronizarea ansamblului de „redemărare”. Procesul este îndelungat și progresiv existînd posibilitatea ca perioada de sterilitate a unui sistem informatic să dureze — după caz — între 3 și 9 luni.

Organizarea, protecția, supravegherea și redondanța sînt expresii cheie care privesc o securitate fiabilă. Pentru aceasta există diverse soluții care pot fi simple sau complicate, totale sau parțiale. Important este să se stabilească o schemă de securitate omogenă, dar și adaptată la întreprindere, la obiectul și mediul său. De exemplu, este evident că natura protecției pentru un mediu informatizat care asigură contabilitatea unui centru de fabricație nu va semăna cu cel al unei bănci. Mai întîi este necesară protecția echipamentelor, operație care se poate realiza cu ajutorul detectoarelor de fum, sistemelor cu extinctoare etc. În același timp vor trebui protejate și programele. Salvările, afirmă specialiștii, vor fi depuse într-un local care prezintă condiții bune de conservare și vor fi plasate în dulapuri protejate

la incendii. Controlul asupra acceselor, fișierelor și programelor poate necesita, în unele cazuri, protecții foarte elaborate (programe specializate). De asemenea, este necesar controlul parametrilor însărcinați cu identificarea utilizatorilor și terminalelor în funcție de parolă. Controlul accesului permite asigurarea confidențialității conținutului fișierelor și depistarea programelor care pot accesa fișierele. În ceea ce privește controlul tranzacțiilor, se pot folosi coduri de acces.

MEMENTO BASIC-80

A.1. Elemente de bază

Alfabetul limbajului este compus din litere, cifre și caractere speciale: virgulă (,), punct și virgulă (;), +, -, *, /, ^ sau ↑ sau ** =, apostroful și parantezele.

Constantele sînt valori ce nu se schimbă în timpul execuției unui program. Ele pot fi numerice sau de tip șir. Cele numerice au următoarele moduri de reprezentare:

- întregi: toate numerele întregi cuprinse între -32768 și +32767;
- virgulă fixă: numerele reale (cu punct zecimal) pozitive sau negative;
- virgulă mobilă: numerele reale pozitive sau negative (mantisa) urmate de exponent D sau E și un număr întreg pozitiv sau negativ (exponentul).

O constantă de tip șir este o secvență de pînă la 255 caractere incluse între apostrofuri. Caracterele pot fi litere, cifre sau simboluri.

Constantele π și e sînt definite în general intern și pot fi utilizate prin simbolurile PI și EE.

Variabilele sînt nume utilizate pentru a reprezenta valori în programele BASIC. Ca și în cazul constantelor există două tipuri de variabile: numerice și de tip șir. Variabilele sînt definite obișnuit: primul caracter din nume este neapărat literă, iar cel de la sfîrșit indică faptul variabilei astfel:

- \$ — variabilă de tip șir de caractere;
- % — variabilă numerică întreagă;
- ## — variabilă numerică dublă precizie;
- ! — sau orice alt caracter diferit de precedentele — variabilă numerică simplă precizie.

O variabilă nu poate avea ca nume un cuvînt rezervat de instrucțiunile BASIC. Numele care începe cu prefixul FN se așteaptă a fi o variabilă declarată funcție (vezi enunțul DEF FN).

Expresia aritmetică poate fi o variabilă/constantă numerică, un operator sau o combinație între constante, variabile și operatori care produce o singură valoare numerică. Operatorii numerici realizează operații numerice sau logice și au ca rezultat o valoare numerică. Ei se clasifică în: aritmetici, relaționali, logici; funcții.

Operatorii aritmetici realizează operațiile aritmetice-uzuale în ordinea matematică standard: **, *, /, +, - cu semnificația: exponențiere, înmulțire, împărțire, adunare și, respectiv, scădere.

Operatorii relaționali compară două valori care pot fi atât de tip șir cît și numerice, iar rezultatul nu poate fi decît adevărat sau fals. Operatorii sînt: = pentru egalitate, < pentru mai mic, <= sau = < pentru mai mic sau egal, > pentru mai mare, >= sau = > pentru mai mare sau egal, >< sau <> pentru inegalitate.

Cînd operatorii aritmetici și relaționali se combină într-o expresie, mai întîi se execută operațiile aritmetice.

Operatorii logici execută operații logice sau booleene asupra valorilor numerice, combinînd valori adevărat-fals și atribuie un rezultat la rîndul lui adevărat sau fals. Operatorii logici sînt: NOT (complement logic), AND (conjunție), OR (disjunție), XOR (sau exclusiv) EQV (echivalență), IMP (implicație).

Expresia șir poate fi o constantă șir, o variabilă șir sau o combinație a acestora prin utilizarea operatorilor pentru a produce o singură valoare șir. Cele două categorii de operatori sînt: concatenarea (realizată cu ajutorul simbolului "+") și funcția.

Funcția numerică este utilizată pentru a apela o operație predeterminată ce urmează a fi executată pe unul sau mai mulți operanzi. Funcțiile pot fi încorporate în sistem (SQR, SIN, COS etc.) sau definite de utilizator prin instrucțiunea DEF FN.

Funcția șir este identică cu cea numerică, deosebirea fiind aceea că rezultatul primeia este o valoare numerică.

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

A.2. Moduri de lucru

Comunicația cu limbajul BASIC se poate realiza în două moduri:
a) *modul direct*. În acest caz limbajul execută o comandă imediat ce a fost introdusă. Instrucțiunea introdusă nu are număr de linie și ea nu este memorată după afișarea unui rezultat. Acest mod de lucru este avantajos pentru depanarea programelor și pentru unele calcule rapide.

b) *modul program (indirect)*. Liniile introduse alcătuiesc un program, fiecare avînd un număr de linie (etichetă). Programul astfel memorat poate fi rulat cu comanda RUN.

A3. Instrucțiuni, comenzi și funcții

O linie BASIC are următorul format:

n instrucțiune₁ [:instrucțiune₂] ... ['comentariu]

unde n reprezintă numărul de linie și este un număr întreg, pozitiv, cu valori cuprinse între 0 și 65 529. El servește la indicarea ordinii de introducere a liniilor, la referire în cadrul salturilor și la editare.

instrucțiune₁, ... este orice instrucțiune BASIC;

'comentariu' definește un text ce joacă rol de comentariu; el apare la sfârșitul liniei după caracterul ' (apostrof).

Pe o linie se pot introduce mai multe instrucțiuni separate între ele prin caracterul ":". O linie de program nu poate avea mai mult de 255 de caractere (inclusiv numărul de linie).

Comenzile operează asupra programelor și de aceea pot fi introduse în modul direct. Instrucțiunile însă urmează un algoritm în cadrul unui program și de aceea ele pot fi introduse în modul indirect, deci ca parte a unui program. În versiunile cele mai noi instrucțiunile și comenzile pot fi introduse atât în modul direct cit și în modul program.

A3.1. Variabile BASIC

Variabilele limbajului BASIC ordonate alfabetic au următoarele descrieri și roluri:

DATE\$ inițializează data calendaristică sub forma $v\$ = \text{DATE\$}$, unde $v\$$ este un șir de 10 caractere sub forma $11-zz-aaaa$ cu semnificația cunoscută;

ERDEV\$ variabilă ce este utilizată sub forma $v\$ = \text{ERDEV\$}$ și furnizează codul de eroare al unei erori de periferic;

ERDEV\$ variabilă ce este utilizată sub forma $v\$ = \text{ERDEV \$}$ și indică numele dispozitivului la care s-a semnalat eroarea;

ERR variabila este utilizată sub forma $v = \text{ERR}$ și conține codul ultimei erori;

ERL variabila $v = \text{ERL}$ indică numărul liniei de program în care a fost semnalată eroarea;

INKEY\$ variabila $v\$ = \text{INKEY\$}$ introduce un singur caracter de la tastatură.

A3.2. Comenzi BASIC

AUTO [n], [[p]] generează automat numere de linie începînd cu n și pasul p (prin lipsă se asumă $p = 10$);

BLOAD (s_f) [d] încarcă fișierul cu numele s_f în spațiul indicat de d ;

BSAVE (s_f), e , l salvează orice porțiune de memorie cu lungimea l octeți care începe de la un deplasament dat de expresia întregă, e , față de segmentul definit de un **DEF SEG** anterior, salvarea făcîndu-se în fișierul cu numele s_f ;

CHDIR „ e_s ” schimbă directorul curent cu cel dat de expresia șir e_s ;

- CLEAR** [n] [m] inițializează cu zero toate variabilele numerice și cu caracterul NULL toate variabilele șir, anulează variabilele comune, închide fișierele și eliberează zonele tampon. Opțional se poate specifica prin n numărul de octeți pe care îi va adresa limbajul precum și mărimea m a stivei;
- CONT** reia (continuă) execuția programului după ce s-a tastat CTRL/BREAK ori s-au executat enunțurile STOP sau END, din punctul de intrerupere;
- DELETE** [l_1] [$-l_2$] șterge liniile programului de la aceea care începe cu l_1 (sau de la început dacă l_1 este omis) până la cea cu numărul l_2 (sau până la sfârșit, dacă l_2 este omis);
- EDIT** n afișează pe ecran linia de program cu numărul n ;
- END** semnalează terminarea execuției programului;
- FILES** [s_f] afișează numele fișierelor din catalogul curent sau specificat explicit prin s_f ;
- KILL** s_f șterge fișierul cu numele s_f ;
- LCOPY** [n] descarcă conținutul ecranului pe imprimanta explicită;
- LIST** [l_1] [$-l_2$] [s_f] listează fișierul cu numele s_f (sau programul sursă, cînd s_f este omis) între liniile de numere (implicit, prima linie) și l_2 (implicit, ultima linie);
- LLIST** [l_1] [$-l_2$] listează la imprimanta de 132 caractere per rînd programul sursă curent cuprins între l_1 și l_2 (valorile implicite și cele anterioare);
- LOAD** s_f [R] încarcă în memorie programul cu numele s_f după ce au fost închise toate fișierele active, șterse toate variabilele și liniile programului curent, exceptînd cazul opțiunii R cînd fișierele sînt păstrate deschise și are loc în același timp și executarea programului;
- MERGE** (s_f) interclasează programul de pe disc cu numele dat de specificatorul s_f cu programul din memorie;
- MKDIR** e_s specifică, prin expresia șir e_s , noul director ce urmează a fi creat;
- NAME** s_{f1} AS s_{f2} schimbă numele fișierului din s_{f1} în s_{f2} ;
- NEW** șterge programul curent din memorie;
- RESET** închide toate fișierele din program;
- RMDIR** e_s șterge directorul specificat de expresia șir e_s ;
- RUN** [l , s_f] lansează în execuție programul curent din memorie începînd cu linia l sau încarcă în memorie și execută programul ce are specificatorul de fișier s_f ;
- SAVE** s_f $\begin{bmatrix} A \\ P \end{bmatrix}$ salvează pe disc programul [cu numele s_f în format ASCII (dacă se specifică A) sau în format binar (dacă se specifică protejarea prin P)];

SYSTEM închide automat toate fişierele deschise şi restituie controlul sistemului de operare;

TRON trasează execuţia instrucţiunilor programului prin afişarea numărului de secvenţă;

TROFF opreşte trasarea programului.

A3.3. Instrucţiuni BASIC

Limbajul posedă următoarele tipuri de instrucţiuni: declarative, de intrare/ieşire, de lucru (asupra variabilelor, constantelor numerice, şirurilor de caractere), definire şi lucru cu subprograme. O caracterizare sumară a acestora este dată în continuare. Astfel,

BEEP activează soneria;

CALL $a[(p_1[, p_2] \dots)]$ permite transferul controlului subrutinei externe aflate la adresa a , opţional transmişându-se şi parametrii $p_1, p_2 \dots$;

CIRCLE [**STEP**] $(x, y), r, [c[, s, f [, a]]]$ trasează un (arc de) cerc sau o elipsă cu centrul în punctul de coordonate (x, y) şi raza r , calculate în elemente de imagine (pixeli), eventual între unghiurile s şi f (exprimate în radiani) cu culoare c (expresie întreagă 0-3) şi cu un raport de aspect a (implicit 5/6);

CLOSE $[[\#]nf_1 [, [\#]nf_2] \dots]$ încheie activitatea de I/E cu un fişier sau periferic specificat prin numele nf ; dacă nu specifică nici un fişier atunci se vor încheia toate fişierele sau perifericele;

CLS şterge tot ecranul;

COM $(n) \left\{ \begin{array}{l} \text{ON} \\ \text{OFF} \\ \text{STOP} \end{array} \right\}$ permite luarea unei acţiuni pentru canalul cu numărul n (expresie întreagă 1-4) astfel: în varianta **ON** se activează întreruperea datorată unui eveniment pe canalul n , în varianta **OFF** întreruperea se dezactivează iar în varianta **STOP** se suspendă;

COMMON l_{var} permite plasarea variabilelor din lista l_{var} într-o zonă comună;

DATA c_1, c_2, \dots, c_n defineşte şi memorează constantele c_i numerice şi de tip şir;

DEF FN $a[(v_1, v_2 \dots v_n)] = e$ defineşte funcţia proprie cu numele a de argumente v_1, v_2, \dots, v_n şi expresia e ; tipul expresiei (numerică sau de tip şir) trebuie să coincidă cu tipul declarat; instrucţiunea se va executa înaintea apelării funcţiei pe care o defineşte; nu sînt suportate funcţii recursive iar **DEF FN** nu este permisă decît în modul program;

DEF $\left\{ \begin{array}{l} \text{INT} \\ \text{DBL} \\ \text{SNG} \\ \text{STR} \end{array} \right\} l_{var}$ declară tipul variabilelor ca fiind întreg (**INT**)

real dublă precizie (**DBL**), real simplă precizie (**SNG**) respectiv şir (**STR**); lista l_{var} conţine numai litera de început l a variabilelor sau intervalul $l_1 - l_2$ în care se află litera de început a numelui;

- DELETE** $\left\{ \begin{matrix} n \\ m-p \end{matrix} \right\}$ șterge linia cu numărul n sau liniile cu numerele cuprinse între m și p ;
- DIM** $a [\$] (n_1 [, n_2] \dots)$ permite dimensionarea tabloului cu numele a fiecare dimensiune n_k aparținând domeniului [1, 255];
- DRAW** e_s desenează un obiect specificat prin expresia șir e_s ;
- END** termină execuția programului, închide fișierele și dă controlul nivelului de comandă;
- ERASE** l_{masive} elimină (șterge) din memorie matricele specificate în lista l_{masive} ;
- FOR...NEXT** permite definirea unui ciclu astfel:
- FOR** $i = e_1$ **TO** e_2 [**STEP** e_3]
- s
- NEXT** i
- secvența s se execută de $\lceil (e_2 - e_1) / e_3 \rceil + 1$ ori, i este variabila ciclului și are valoarea inițială e_1 , pasul e_3 , și valoarea finală e_2 ; dacă e_3 este omis se asumă $e_3 = 1$;
- GET** $[\#] n_f [, n]$ citește înregistrarea cu numărul n din fișierul cu numărul n_f ; dacă n se omite, în tamponul de memorie se va citi următoarea înregistrare;
- GOSUB** n indică un salt (control) la linia program cu numărul n de unde se presupune că începe o subrutină; revenirea din subrutină la prima instrucțiune de după **GOSUB** se face la întilnirea instrucțiunii **RETURN**;
- GO TO** n execută un salt (transfer) necondiționat la instrucțiunea cu numărul de linie n ;
- IF** $c [,]$ **THEN** s_1 [**ELSE** s_2]
- IF** $c [,]$ **GO TO** $n [[,]$ **ELSE** s_3] execută salt condiționat; dacă expresia c este adevărată se execută clauza **THEN** sau **GO TO**, iar în caz contrar se execută clauza **ELSE** (dacă este prezentă) și se continuă cu următorul număr de linie ce conține o instrucțiune executabilă; expresia c poate fi orice expresie numerică; secvențele s_1, s_2, s_3 sînt secvențe de instrucțiuni BASIC (separate prin două puncte), sau un număr de linie pentru salt, n este numărul de linie al unei instrucțiuni activate prin execuția clauzei **GO TO**; instrucțiunile **IF-THEN-ELSE** pot fi imbricate oricît, singura restricție constînd în lungimea limitată a liniei;
- INPUT** $[;]$ [*'mesaj'* ;] v_1, v_2, \dots, v_n permite introducerea dinamică a datelor de la terminal pentru variabilele v_1, v_2, \dots, v_n ;
- INPUT** $\# n, v_1, v_2, \dots, v_n$ permite citirea datelor din fișierul secvențial n (desemnat printr-un **OPEN** anterior) și introducerea lor în variabilele din listă;
- IOCTL** $[\#] n, e_s$ trimite unui șir de control specificat de expresia șir e_s către fișierul cu numărul n ;
- IOCTL** $\$ ([\#] n)$ restituie utilizatorului un șir de date de pe perifericul cu numărul n ;

- KEY $\left\{ \begin{array}{l} \text{ON} \\ \text{OFF} \\ \text{LIST} \end{array} \right\}$ afișează (ON) și respectiv șterge (OFF) primele șase caractere ale celor 10 chei; dacă se pune KEY LIST se vor lista toate valorile cheilor utilizând 15 caractere pentru fiecare cheie:
- [LET] $v = e$ atribuie variabilei v valoarea expresiei sau șirului de caractere e ; tipul expresiei (numerice sau de tip șir) trebuie să concorde cu cel al variabilei;
- LINE INPUT $[;] ["mesaj"] ; v_s$ permite recepționarea în variabila șir v_r a maximum 254 caractere introduse de la tastatură, cu afișarea eventuală a unui "mesaj";
- LINE INPUT $\# n, v_s$ citește din fișierul secvențial cu numărul n un șir de caractere ce se vor memora în variabila șir v_s ;
- LPRINT $[l_e] [;]$ permite tipărirea valorilor expresiilor din lista de expresii l_e ;
- LPRINT USING $f; l_e$ tipărește valorile expresiilor din listă l_e conform formatului definit de f ;
- LSET $v_s = e_s$ mută date din expresia șir e_s în variabila șir v_s , datele fiind aliniate la stînga;
- ON ERROR GO TO n transferă controlul la linia cu numărul n , linie de la care începe o rutină de tratare a întreruperilor;
- ON e GOSUB n_1, n_2, \dots, n_k execută, în funcție de valoarea expresiei e , salt la una din cele k subrutine ce au ca număr de linie de început pe n_k ; dacă valoarea lui e este în afară mulțimii $\{1, 2, \dots, k\}$ se va executa următoarea instrucțiune; fiecare subrutină posedă obligatoriu instrucțiunea RETURN;
- ON e GO TO n_1, n_2, \dots, n_k execută salt la instrucțiunea cu numărul de linie n_k dacă expresia e are valoarea k ; în rest se execută următoarea instrucțiune;
- ON PLAY (n) GOSUB l generează continuu muzică în modul fundamental în timpul execuției programului; $n \in \{1, 2, 3, \dots\}$ indică notele care trebuie detectate, iar l este numărul de linie pentru instrucțiunea PLAY;
- ON TIMER (n) GOSUB l execută salt la instrucțiunea cu numărul l după scurgerea a n secunde;
- OPEN s_f [FOR mod_1] AS $[\#] n_f$ [LEN = l_r]
- OPEN $mod_2, [\#] n_f, l_f [l_r]$ instrucțiunile permit deschiderea fișierului cu numărul n_f și numele l_f ; dacă fișierul este exploatat aleator se poate indica și lungimea înregistrării l_r ; mod_1 și mod_2 pot fi: INPUT și respectiv I pentru acces secvențial la intrare, OUTPUT și respectiv O pentru acces secvențial la ieșire, APPEND și respectiv A pentru modul secvențial de ieșire în care fișierul este poziționat la sfîrșitul șirului de date în momentul deschiderii acestuia; mod_2 mai poate fi R cînd specifică acces aleator;
- OPTION BASE (n) declară că limita inferioară a indicilor tablourilor este n , sau $n \in \{0; 1\}$;

- OUT** n, m trimite un octet de valoare m la postul cu numărul n ;
- PLAY** e_s execută melodia reprezentată de caracterele din expresia șir e_s ;
- PLAY** (a) furnizează numărul de note din tamponul melodiei de fond;
 a este argument fictiv;
- PLAY** $\left\{ \begin{array}{l} \text{OFF} \\ \text{ON} \\ \text{STOP} \end{array} \right\}$ inhibă (OFF), activează (ON) respectiv suspendă (STOP) intreruperea provocată de atingerea unei limite în conținutul zonei tampon de melodii de fond;
- POKE** n, m înscrie octetul m (expresie numerică) la o adresă dată de deplasarea n față de segmentul curent definit de un enunț; DEF SEG anterior;
- PRESET** [STEP] (x, y) [c] va desena pe ecran, cu o culoare corespunzătoare numărului c , un element de imagine dat de coordonatele absolute (x, y) sau relative (x, y) dacă opțiunea STEP este prezentă;
- PRINT** [e_1, e_2, \dots, e_n] editează date sau rezultate intermediare/finale, unde $e_i, i = 1, 2, \dots, n$ sînt variabile, constante, expresii; dacă lista este omisă se trece la următoarea linie pe terminal;
- PRINT USING** $f; e_1, e_2, \dots, e_n$ permite afișarea datelor e_i (variabile, constante, expresii) după formatul f ;
- PRINT** # n, e_1, e_2, \dots, e_n
- PRINT** # $n, \text{USING } f; e_1, e_2, \dots, e_n$ scriu date secvențiale în fișierul cu numărul n ; în rest semnificația argumentelor este ca la precedentele instrucțiuni **PRINT**;
- PSET** (x, y) [c] desenează un punct la poziția specificată de coordonatele (x, y) cu o culoare dată de atributul c ;
- PUT** [#] n [m] scrie o înregistrare în fișierul aleator cu numărul n ; m este numărul înregistrării ce urmează a fi scrisă;
- PUT** (x, y), v [a] desenează imagini pe o arie specificată a ecranului; (x, y) sînt coordonatele colțului din stînga sus a ecranului unde are loc transferul imaginii stocate în masivul/vectorul de octeți v potrivit modului stabilit de acțiunea a ce are valorile PSET (transfer punct cu punct cu respectarea culorii), PRESET (transfer punct cu punct cu reversul imaginii), AND (se efectuează operația logică "și" între elementul existent și cel transmis), OR (operația de supraimprimare a celor două imagini), XOR (operația "SAU exclusiv" între cele două imagini);
- RANDOMIZE** [e] inițializează generatorul de numere aleatoare cu "sămînța" dată de expresia e ;
- READ** v_1, v_2, \dots, v_n permite introducerea datelor c_i dintr-o instrucțiune **DATA** corespondentă în variabilele v_i ;
- REM** c inserează o linie de comentariu c ;
- RENUM** [n] [, [m] [, p]] renumerează liniile de program începînd cu vechea linie m (va avea noul număr n) și cu pasul p ; dacă m este

omis se asumă prima linie din program iar dacă p este lipsă atunci se ia valoarea implicită 10;

RESTORE [n] (re)stabilește poziția curentă într-un bloc de date (definit de enunțuri **DATA**) la subblocul n ; implicit, se asumă primul bloc **DATA**;

RSET $v_s = e_s$ atribuie variabilei șir v_s , aliniat la dreapta, conținutul expresiei șir e_s ;

SOUND f, d generează sunet de frecvență f (exprimată în Hertz) pe o durată d (exprimată în tacte);

STOP întrerupe execuția programului în curs, revenindu-se la nivelul comandă după afișarea unui mesaj;

TIME $\$ = e_s$ stabilește ceasul curent la valoarea dată de expresia șir e_s care poate fi dată sub una din formele: "hh", "hh:mm", "HH:mm:ss";

TIMER $\left\{ \begin{array}{l} \text{OFF} \\ \text{ON} \\ \text{STOP} \end{array} \right\}$ dezactivează (**OFF**), activează (**ON**) și, respectiv suspendă (**STOP**), întreruperea provocată de evenimentul ce constă din trecerea unui interval de timp (care se află în atenția lui **ON TIMER**(n) **GOSUB**);

SWAP v_1, v_2 schimbă între ele valoarea variabilelor de același tip v_1 și v_2 ;

VIEW [[**SCREEN**] $(x_1, y_2) - (x_2, y_2)$ [c, f]]] definește o fereastră din ecran la care face apel instrucțiunea **WINDOW**; (x_1, y_1) și (x_2, y_2) sînt coordonatele stînga-sus respectiv dreapta-jos ale ferestrei definite care se va umple cu o culoare definită prin atributul c ; fereastra are margine dacă frontiera f are o valoare nenulă; opțiunea **SCREEN** arată că coordonatele se vor calcula relativ la întregul ecran iar în caz contrar sînt relative la fereastră;

VIEW PRINT [l_1 TO l_2] fixează limitele unei ferestre de text stabilite între numerele de linie l_1 și l_2 ;

WAIT p, n [m] suspendă execuția programului pînă ce un port p elaborează un șir de biți specificat; datelor citite de la port l_1 se aplică operația XOR cu expresia întregă m , apoi operația AND cu n ; dacă m se omite, se ia implicit zero;

WHILE...WEND definește un ciclu sub forma

WHILE c

S

WEND

instrucțiunile din secvența s executindu-se cît timp condiția c este adevărată;

WIDTH LPRINT n stabilește lungimea liniei de imprimantă la n caractere;

WINDOW [[**SCREEN**] $(x_1, y_1) - (x_2, y_2)$] definește dimensiunile (în coordonate universale) ferestrei prin precizarea colțului din stînga-sus

de coordonate (x_1, y_1) și colțului din dreapta jos de coordonate (x_2, y_2) ; cu opțiunea **SCREEN** are loc intervertirea axelor X și Y ;

WRITE $[v_1, v_2, \dots, v_n]$ permite afișarea conținutului variabilelor v_i pe ecran; dacă lista este omisă, instrucțiunea afișează o linie de spații;

WRITE $\#n, v_1, v_2, \dots, v_n$ are același efect ca și precedentă, dar cu seriere în fișierul cu numărul n .

A 3.4. Funcții BASIC

În limbajul BASIC există funcții implicite care trebuie numai apelate pentru a prelucra expresii matematice sau de tip șir. De aceea ele vor fi folosite sub formele $y = f(x)$ și, respectiv, $y\$ = f\$(x\$)$. Funcțiile definite de utilizator sînt construite prin instrucțiunea **DEF FN**.

Funcțiile intrinseci ale limbajului sînt:

ABS(e) calculează valoarea absolută a expresiei numerice e ;

ASC(e_s) determină valoarea numerică în cod ASCII a primului caracter din expresia șir e_s ;

ATN(e) găsește valoarea, în intervalul $(-\pi/2, \pi/2)$, a arcului a cărui tangentă este expresia numerică e ;

CDLB(e) convertește expresia numerică e din simplă precizie în dublă precizie;

CHR $\$(n)$ convertește caracterul n din cod ASCII în caracterul său echivalent;

CINT(x) convertește argumentul x într-un întreg;

COS(x) calculează cosinus de x (x exprimat în radiani);

CSNG(x) convertește argumentul x în simplă precizie;

CSRLIN citește coordonata verticală a cursorului;

CVI(c_2) convertește variabila șir de doi octeți c_2 într-un întreg;

CVS(c_4) convertește variabila șir de patru octeți c_4 într-un număr simplă precizie;

CVD(c_8) convertește variabila șir de opt octeți c_8 într-un număr dublă precizie;

EOF (s_f) indică condiția de sfîrșit fișier pentru fișierul s_f

EXP(x) calculează e la puterea x

FIELD $[\#n, l_1 \text{ AS } v_1 [l_2 \text{ AS } v_2] \dots]$ alocă spațiu pentru variabilele din fișierul cu numărul n ; v_1, v_2, \dots sînt variabile utilizate în accesarea fișierului iar l_1, l_2, \dots specifică numărul de caractere alocat variabilelor corespunzătoare;

FIX(x) trunchiază x la un întreg; pentru x negativ nu este evaluat următorul număr mai mic;

FRE determină numărul de octeți neutilizați din spațiul de date;

HEX $\$(e)$ evaluează expresia zecimală e și o convertește în hexazecimal;

INP(n) evaluează octetul citit de la portul n ;

INSTR ($[n,]a\$,b\$\$$) caută prima apariție a șirului $b\$\$$ în șirul $a\$\$$ începînd cu poziția a n -a sau cu primul caracter cînd n este omis.

Rezultatul căutării este un număr ce arată poziția de la care începe $b\%$ și în $a\%$;

INT(x) determină cel mai mare întreg care este mai mic sau egal cu x ;

LEFT $\%$ (e_s, n) extrage cele mai din stînga n caractere din expresia șir e_s ;

LEN(e_s) determină numărul de caractere din expresia șir e_s ;

LOG(x) calculează logaritmul natural din x ($x > 0$);

LPOS(n) determină poziția curentă a capului imprimantei ($n = 0, 1, 2, 3$) în timpul tipăririi;

MID $\%$ (e_s, n, m) evaluează, din șirul e_s , un șir de lungime m caractere începînd cu al n -lea. Dacă m este omis sau în dreapta sint mai puțin de m caractere atunci se vor evalua cele mai din dreapta caractere din e_s începînd cu caracterul n ;

MKI $\%$ (e), **MKS** $\%$ (e_s, p), **MKD** $\%$ (e_d, p) convertește expresia întregă e , expresia simplă precizie $e_s.p$ și respectiv expresia dublă precizie $e_d.p$ într-un șir de doi, patru respectiv opt octeți;

OCT $\%$ (e) calculează valoarea octală a expresiei numerice e ;

PEEK(n) citește un octet de la locația de memorie dată de valoarea numărului n ; **PEEK** este complementara lui **POKE**;

PMAP(x, n) transformă: abscisa universală x în coordonată fizică dacă $n = 1$; ordonata universală y în coordonată fizică dacă $n = 2$; abscisa fizică x în coordonată universală dacă $n = 3$; ordonata fizică y în coordonată universală dacă $n = 4$;

POINT (x, y) evaluează atributul unui punct de coordonate (x, y) de pe ecran;

POS(n) evaluează poziția din rînd a cursorului, n fiind argument fals;

RIGHT $\%$ ($a\%$, n) atribuie unei variabile șir cele mai din dreapta n caractere ale șirului $a\%$;

RND(e) calculează un număr aleator în intervalul (0,1) ținînd seama de „sămînța” specificată implicit (sau explicit printr-un enunț **RANDOMIZE** anterior) și potrivit caracteristicilor opțiunii e ;

SCREEN (r, c [, e]) atribuie unei variabile codul ASCII corespunzător caracterului din rîndul r și coloana c . Parametrul e este o expresie aritmetică evaluată la adevărat sau fals. Dacă e este adevărat atunci se atribuie culoarea caracterului de coordonate (r, c) iar dacă este fals se furnizează codul ASCII al caracterului (r, c) (opțiune implicită);

SGN(x) se restituie 1, 0, -1 după cum valoarea argumentului x este strict pozitivă, nulă sau negativă;

SIN(x) calculează sinus de x (x exprimat în radiani);

SPACE $\%$ (n) furnizează un șir de spații de lungime n ;

SPC(n) inserează în linia de ieșire asociată unor enunțuri **PRINT**, **LPRINT**, **PRINT** un număr de spații egal cu n ;

SQR(x) calculează rădăcina pătratică a lui x ($x \geq 0$);

STR(e) determină reprezentarea valorii expresiei numerice e sub forma unui șir de caractere;

- TAN**(x) atribuie unei variabile valoarea tangentei (x este dat în radiani);
- TIME**\$ atribuie unei variabile și valoarea curentă a timpului sub forma $hh:mm:ss$;
- TIMER** furnizează numărul de secunde scurse de la reinițializarea sistemului;
- USR** [n] (v) apelează subrutina (scrisă în limbaj mașină) indicată de argumentul v (expresie numerică sau variabilă și);
- VAL** (e_s) convertește expresia și e_s a unui număr în valoarea sa numerică ignorând spațiile, stopurile de tabulare și caracterele <F>.

BIBLIOGRAFIE

1. ALLISON, A., *Clones VS IBM, Buyer Beware*, Mini-Micro Systems, march, 1987.
2. ALLISON, A., *Powerful New Machines Find a Niche*, Mini-Micro Systems, oct., 1987.
3. ALSOP, S., *Stop that Bus*, PC World, july, 1988.
4. ARCHIBALD, P., *The Foremost US Company in the Data Processing Industry*, Datamation, june, 1982
5. ARCHIBALD, P., VERITY, J., *The Top 10 in Mainframes*, Datamation, june, 1985.
6. BARON, N. *Sun offer workstation performance at PC prices*, Byte, may, 1989.
7. BUNELL, D., *The Big Chilling Effect*, PC World, july, 1988.
8. CHOSSAING, Ph., *La fin du cinéma mut*, PC Informatique, 73, mars, 1989.
9. CHRISTOMAN, A.M., *Data base Management Enhonces Abilities of Integrated CAD/CAM*, Computer Technology, dec., 1983.
10. DAVIDOVICIU, A., *Generația a cincea de calculatoare*, Simpozion Academia Română, noiembrie 1984.
11. DIAMANDI, I., CĂLINESCU, C., *Dialog cu viitorul*, Editura Științifică și Enciclopedică, București, 1988.
12. ELMER, Ph.; DeWITT, A., *"Virus" Epidemia Strikes Terron with Computer World*, Time, sept., 1988.
13. PERTIG, R.T., *The Software Revolution: Trends, Players, Market Dynamic in Personal Computer Software*, Elsevier Science Publishing Co, Inc., 1985.

14. HART, JEFF, *Workstation Domination*, System International, may, 1988.
15. HONAN, P., *Avoiding Virus Hystoria*, Personal Computing, may, 1989.
16. KIRKLEY, J.L., *Midrange Systems*, Datamation, april, 1986.
17. LANDRY, R., *Beyoud the Speed Trap*, PC World, july, 1988.
18. MARCHAND, CR., *La sécurité informatique du PC*, PC Informatique, avril, 1989.
19. MOTO-OKA, TAHRY, *Les ordinateurs de cinquième génération*, La Recherche, 154, avril, 1984.
20. RHUE, FR., *Le Combat des géants*, PC Informatique, [73, mars, 1989.
21. RING, T., *Weighing Lip the RISC factor*, System International, may, 1988.
22. SCANNELL, T., *IBM Digital Square off to Capture "Work Sharing" Market*, Mini-Micro Systems, january, 1987.
23. SCANNELL, T., *IBM's PS/2 won't stall computing systems makers*, Mini-Micro Systems, oct., 1987.
24. THOMPSON, J.R., *The Supercomputer of Cray Research*, Elsevier Science Publishers, 1986.
25. TYPROWICZ, A.C., *Distributed Graphics Workstation Provide a Reliable Design Tool*, Computer Tehnology, 1983.
26. WOLFE, AL., *Software engineers will face the challenge of writing programs. Tehnology outlook*, Electronics, oct., 1986.
27. ZACHMANN, W., *IBM vs Company*, PC World, july, 1988.
28. * * * *Caracteristicile microcalculatoarelor multiutilizator*, Mini-Micro Systems, aug., 1987.
29. * * * *Impactul noului sistem de operare pentru calculatoare personale OS/2, 1987—1991*, Mini-Micro Systems, oct., 1987.
30. * * * *Introducing the HP 3000 series P30 supermini*
31. * * * *Minisupercomputers, hardware project*, Mini-Micro Systems, oct., 1987.
32. * * * *Mini/micros Survey*, Datamation, nov., 1987.
33. * * * *Mini/micros Survey*, Datamation, nov., 1988.
34. * * * *Minisupercomputers, hardware project*, Mini-Micro Systems, oct., 1987.

35. * * * *Overseas market report Japan, West-Germany, United Kingdom, France, Italy*, Electronics, ian., 1986, 1987, 1988, 1989.
36. * * * *PC Magazine*, 8, 7, apr., 1989.
37. * * * *PC Informatique*, 73, mars, 1989.
38. * * * *US Market Raport*, Electronics, ian. 1986, 1987, 1988, 1989.
39. * * * *What's new Systems*, Byte, nov., 1986.

Redactor : Locot.-col. AL. MIHALCEA
Tehnoredactor : D. BODEA

Bun de tipar : 02.07.1991. Apărut : 1991.
Coli de tipar : 13. B 10290.



Tiparul executat sub comanda nr. 526
la I.P. „Filaret“, str. Fabrica de chibrituri
nr. 9—11. București
România

EDITURA MILITARĂ

ISBN 973-32-0178-2

Lei 130

