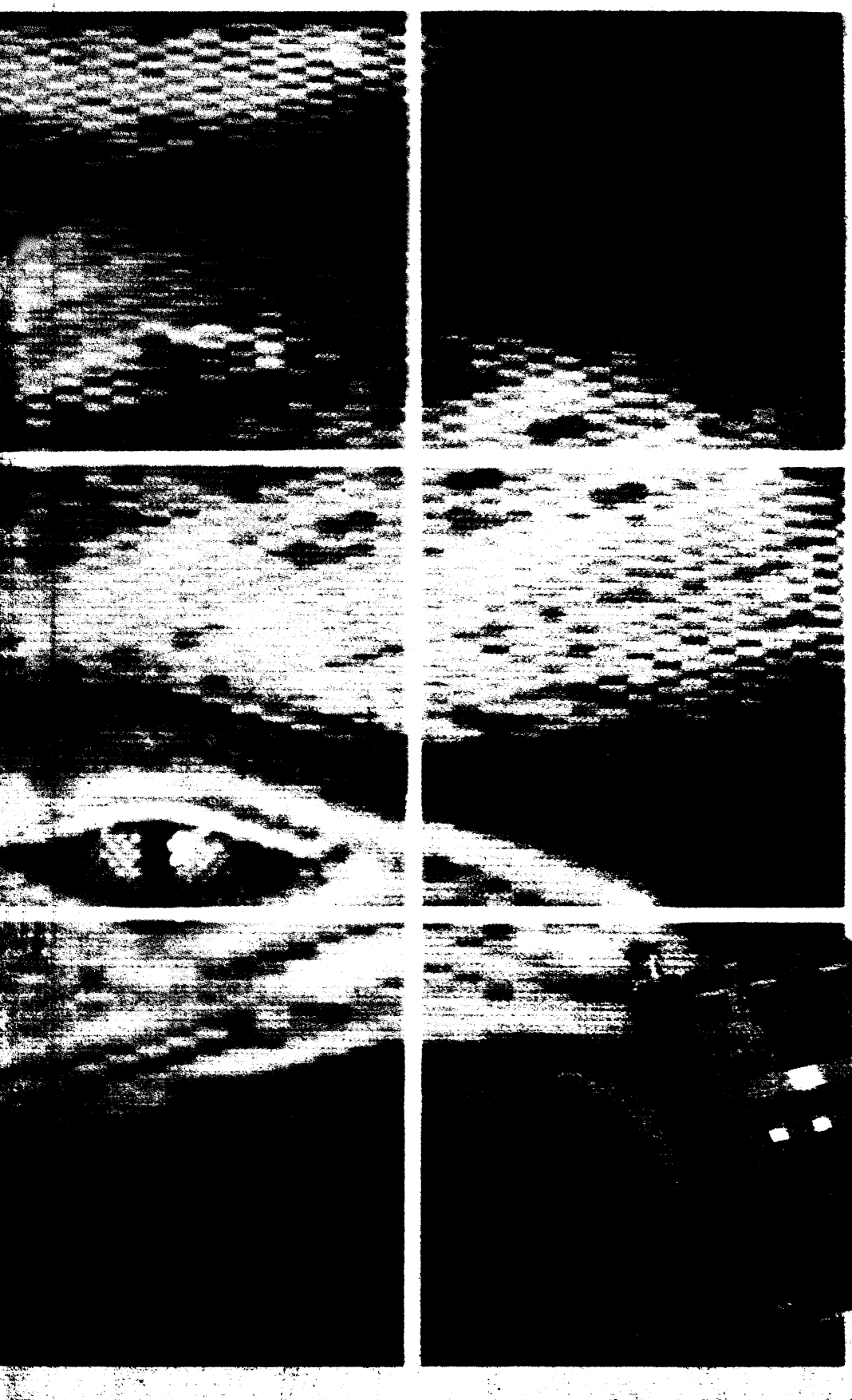
PC magazin



Revistă independentă de informatică editată de
ADISAN

ANUL II, Nr. 2/1991

- Limbajul PROLOG
- Reţele neuronale(II)
- Incursiune în ORACLE
- Totul despre memorie
- "Spargerea" programelor
- A fost odata "UNIX"
- Prezentări SOFTWARE
- Noutăţi

PLOTTERS

CULTURA INFORMATICĂ

INFORMATICA FĂRĂ PROFESOR

WINDOWS

HOME COMPUTER

PERSPECTIVE

MARKET SOFT/HARD

STOP

PC-MAGAZIN

ANUL II NR.2/1991

FONDATORI
Adrian Negru
Alexandru Babin

DIRECTOR
Alexandru Babin

REDACTOR COORDONATOR
Mircea Ciobotaru

COLABORATORI
Prof.univ.dr. Cornel Popa
conf.dr.ing. Irina Athanasiu
conf.dr.ing. Eugenia Kalisz
Ing. Sorin Deaconu
Ing. Ionuţ Stoica
Roxana Bicleanu
Marius Sturzoiu
Virgil Ţeculescu
Ing. Cureleţ-Bălan Gheorghe
Ion Paraschiv
dr. Tiberiu Spircu
Ion Muşlea
Bogdan Georgescu
Bogdan Iordănescu
Dobrilă Mirel

GRAFICĂ
Octavian Penda
Lucian Mişcă
Luminiţa Ciupitu

ADRESA REDACŢIEI
Bucureşti, str. Ion Mincu nr.11
sect.1
Tel: 17.74.30

Tiparul a fost executat la
Tipografia UNIVERSUL

INFORMATICĂ

C.N.I.-C.P.I.
BAZA DE INSTRUIRE ÎN INFORMATICĂ
VATRA-DORNEI

5975
str. GEORGE COŞBUC nr. 6, tel. 988/73534

ORGANIZEAZĂ PERMANENT

CURSURI

CALIFICARE, FORMARE, SPECIALIZARE,

PERFECŢIONARE

PENTRU PERSOANELE CU STUDII MEDII

PE MINI-MICROCALCULATOARE (8-16 BIŢI)

— Formare operatori echipamente culegere, transmitere, prelucrare primară a datelor.

— Formare analişti-programatori asistenţi.

— Operatori iniţiere în informatică, lucrări secretariat şi dactilografie.

— Perfecţionare pe microcalculatoare (prelucrare de texte şi documentaţii).

— Programare în limbaj BASIC.

— Programare în limbaj dBASE.

— Utilizare microcalculatoare compatibile IBM-PC.

La cursuri pot participa şi persoanele cu studii superioare.

Cursurile se organizează la sediul bazei (cu asigurarea cazării) şi la BENEFICIAR cu şi fără scoatere din producţie atît pentru persoanele trimise de unităţile de stat, cît şi pentru persoanele particulare.

Informaţii la telefon: 988/73534.

BAZELE LOGICE ALE INTELIGENŢEI ARTIFICIALE(II)

Prof.univ.dr. Cornel Popa

6. Derivare şi respingere rezolutivă

6D1. Fie S un set de clauze şi k o clauză oarecare. Spunem că din S *derivă rezolutiv* k şi scriem $S \models_{res} k$, dacă şi numai dacă, putem construi o sevenţă de clauze:

$D=(k_1,k_2,...k_n)$

care satisface condiţiile:

a) $k_n=k$

b) Pentru orice $1 \leq i \leq n, k_i \in S$ sau

c) dacă $k_i \notin S$, atunci există în D, k_j şi k_m cu $j<i$ şi $m<i$ şi există un l astfel încît $k_i=res_l(k_j,k_m)$.

Cu alte cuvinte, D este un şir de clauze din S sau obţinute prin rezoluţia din S al cărui ultim element k_n coincide cu clauza de derivat k. Relaţia $\models_{res}$ introduce o relaţie de ordine, punînd la început elemente din S şi apoi elemente din S sau obţinute prin rezoluţie din S.

A construi o derivare rezolutivă pentru o clauză k dintr-un set de clauze S este ceva analog cu a construi un text demonstrativ pentru o teoremă k dintr-un set de axiome S. Şi într-un caz şi într-altul se procedează strict formal, prin aplicarea unor reguli de derivare. Deosebirea rezidă doar în faptul că elementele lui S nu sînt tautologii sau formule identic adevărate, ci doar formule adevărate şi analog k nu este o tautologie ci tot o formulă adevărată.

6D2. Respingerea rezolutivă este o derivare rezolutivă încheiată prin obţinerea clauzei vide, i.e a clauzei fără nici un literal, notată prin □ sau NiL.

Obţinerea clauzei vide dintr-un set S de clauze atestă irealizabilitatea lui S.

Demonstrarea unei teoreme prin respingere rezolutivă se face astfel: 1) se obţin clauzele din premisele raţionamentului; 2) se obţin clauzele din negaţia concluziei raţionamentului; 3) se dezvoltă, prin aplicarea iterată a principiului rezoluţiei, o respingere rezolutivă.

Aplicarea principiului rezoluţiei la un set de clause se face, de regulă, după anumite criterii şi reguli suplimentare care definesc o strategie rezolutivă. Despre strategiile rezolutive cu alt prilej.

Acum ne putem întoarce la construirea unei derivări rezolutive sau a unei respingeri rezolutive pentru clauzele raţionamentului **R1**.

Demonstrarea validităţii raţionamentului **R1** se reduce astfel la o derivare rezolutivă de forma:

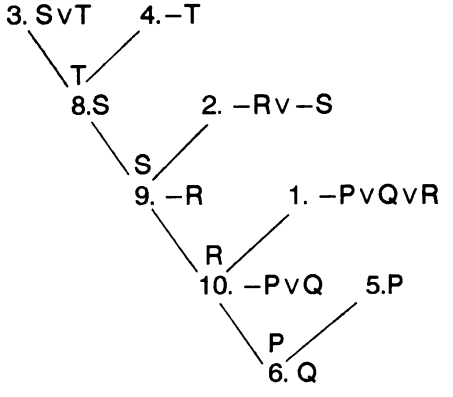
$$\underset{-P \vee Q \vee R,}{1} \quad \underset{-R \vee -S,}{2} \quad \underset{S \vee T,}{3} \quad \underset{-T,}{4} \quad \underset{P \models_{res} Q}{5} \quad 6 \qquad (d1)$$

unde $\{.^1..^5\}$ conţine clauzele provenite din premisele lui **R1**, iar Q reprezintă concluzia lui **R1** sau la o *respingere rezolutivă* de forma:

$$\underset{-P \vee Q \vee R,}{1} \quad \underset{-R \vee -S,}{2} \quad \underset{S \vee T,}{3} \quad \underset{-T,}{4} \quad \underset{P,}{5} \quad \underset{-Q \models_{res} \square}{7} \qquad (d2)$$

De remarcat că (d2) conţine în setul de clauze premise pe −Q, negaţia concluziei. Respingerea rezolutivă este un raţionament prin reducere la absurd în care postulăm premisele şi negaţia concluziei.

Derivăm mai întîi, din clauzele 1−5, clauza 6, conform lui (d1).



Clauza 3 conţine literalul T, iar clauza 4 pe −T. Rezoluţia în T din clauzele 3 şi 4 duce la clauza 8. Clauzei 8 îi asociem pe 2 pentru că 2 conţine pe −S; obţinem făcînd o rezoluţie în S pe 9. Şi aşa mai departe, pînă îl obţinem pe Q, clauza de derivat.

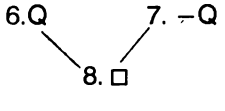
Secvenţa de clauze D definită în **6D1,** pentru derivarea făcută este

$$D1 = \underset{S \vee T,}{3} \quad \underset{-T,}{4} \quad \underset{S,}{8} \quad \underset{-R \vee -S,}{2} \quad \underset{-R,}{9} \quad \underset{-P \vee Q \vee R,}{1} \quad \underset{-P \vee Q,}{10} \quad \underset{P,}{5} \quad \underset{Q}{6}$$

Clauzele 3,4,2,1,5 aparţin lui S1, setul de premise; clauza 6 este clauza k ce trebuie derivată, iar clauzele 8, 9, 19 şi 6 sînt obţinute prin derivarea rezolutivă din clauze anterioare lor în şirul D1.

Derivarea de mai sus este o *derivare liniară,* în care orice clauză rezolvent apare ca o clauză părinte în inferenţa rezolutivă următoare.

Pentru a aplica o respingere rezolutivă conform relaţiei (d2), prelungim derivarea rezolutivă anterioară obţinînd drept rezolvent din: 6 şi 7 pe □. (Clauza 7 este negaţia concluziei)



Evident, nici derivarea, nici respingerea rezolutivă nu sînt demersuri unice.

Putem acum conchide că raţionamentul **R1** este valid. Nu ne-am pronunţat încă asupra celorlalte: **R2, R3** şi **R4**.

Să recapitulăm. Am formalizat raţionamentul **R1**. Formulele obţinute le-am adus la forma clauzală disjunctivă. Apoi din clauzele obţinute din premise am derivat rezolutiv clauza obţinută din concluzie, conform demersului (d1) sau din clauzele obţinute din premise şi din negaţia concluziei am derivat clauza vidă, conform demersului (d2). În ambele cazuri am folosit de mai multe ori principiul rezoluţiei, dovedit a fi, prin teorema rezolventului, o

schemă validă de inferenţă, asemenea legii tranzitivităţii sau regulii detaşării i.e. *modus ponens.*

Pentru formalizarea lui **R1** ne-au fost suficiente resursele logicii propoziţiilor. Formalizarea celorlalte raţionamente **R2-R4** nu va mai fi posibilă în limitele logicii propoziţiilor.

7. Formalizarea unor raţionamente în limbajul logicii predicatelor

Să ne întoarcem la raţionamentele **R2**, **R3** şi **R4**. În acestea apar unele particule lexicale ca "toţi", "cineva", "cei care...", particulele anunţă ideea de cuantificator universal sau existenţial. În consecinţă, ele pot fi formalizate numai în logica predicatelor.

Cititorul va fi izbit de caracterul manifest fals al premiselor din **R2** în timp ce va admite, probabil, concluzia acestuia.

Experienţa comună sau bunul simţ, ne îndeamnă să considerăm un raţionament ce are una sau două premise false, drept invalid.

Aceasta este însă o capcană în care nu trebuie să cădem cu uşurinţă. O implicaţie materială poate fi adevărată pe temeiul că antecedentul ei este fals sau pe temeiul că urmarea sau consecventul ei este adevărat. Sub raport semantic, **R2** satisface ambele condiţii.

În **R2** intervin predicatele piatră (x), naşte-pui-vii (x), capră (x). Toate pot fi luate ca predicate monadice, de un singur argument. Singurul cuantificator care apare este cel universal. În consecinţă, obţinem:

1. $\forall x$ (piatră (x)$\supset$naşte-pui-vii (x))
2. $\forall y$ (capră (y)$\supset$piatră (y)) ($R2^0$)

---

3. $\forall z$ (capră (z)$\supset$naşte-pui-vii(z))

În **R3** intervin patru predicate şi o constantă individuală. Predicatele sînt: se teme (x,y), iubeşte (x,y), este şef (x,y), este păzit (x). Constanta individuală este numele propriu "Mafia".

Din predicatul binar se-teme (x,y) i.e. "x se teme de y" trebuie să obţinem predicatul monadic "x se teme" sau "x este un temător" Pentru ca x să devină temător trebuie să existe un y astfel încît x să se teamă de y.

se-teme(x) = $\exists y$ (se-teme(x,y)

În acelaşi timp, din predicatul iubeşte (x,y) trebuie să obţinem predicatul "x nu este iubit de toţi".Obţinem mai întîi din iubeşte (x,y), predicatul monadic "y este iubit de toţi".

iubit-de-toţi(y) = $\forall x$ (iubeşte(x,y))

De aici obţinem:

non-iubit-de-toţi(y) = $-\forall x$ (iubeşte(x,y))

Predicatul se-teme-de-cineva(x) poate fi, la rîndul său, obţinut prin predicatul binar se-teme(x,y).

se-teme-de-cineva(x) = $\exists y$ (se-teme (x,y))

În sfîrşit, din predicatul şef(x,y) se poate obţine şef al Mafiei(x) prin înlocuirea variabilei y prin numele propriu Mafia.

şef-al-Mafiei(x) = şef(x, Mafia)

Putem acum asambla toate aceste predicate în patru formule ce descriu premisele şi concluzia raţionamentului **R3.**

Adoptăm convenţia de a utiliza pentru fiecare premisă variabilele individuale distincte:

1. $\forall u(\exists t$(se-teme(u,t))$\supset -\forall v$(iubeşte(v,u)))
2. $\forall y$(păzit (y)$\supset\exists z$(se-teme(y,z)))
3. $\forall x$(şef(x,Mafia)$\supset$păzit(x)) ($R3^0$)

---

4. $\forall w$(şef(w,Mafia)$\supset -\forall s$(iubeşte(s,w)))

Trecem la formalizarea raţionamentului **R4.**

În **R4** intervin constantele individuale Ion şi Ana, predicatul monadic fericit(x) şi din nou predicatul binar iubeşte(x,y).

Din predicatul binar iubeşte(x,y), trebuie să obţinem predicatul monadic este-iubit-de-cineva(y):

este iubit-de-cineva(y) = $\forall x$(iubeşte (x,y))

Putem, acum trece la redarea completă a lui **R4** în limbajul logicii predicatelor:

1. $\exists x$(iubeşte(x,Ion))$\supset$fericit(Ion)
2. $\forall y$(iubeşte(y,Ana)$\supset$ iubeşte (Ana,y)
3. iubeşte (Ion, Ana) ($R4^0$)

---

4. fericit (Ion)

($R2^0$)-($R4^0$) reprezintă formalizarea în limbajul logicii predicatelor a raţionamentelor **R2-R4** propuse în 1.

Dacă dorim să decidem asupra validităţii raţionamentelor **R2-R4** prin metoda rezoluţiei, atunci acestea trebuie aduse la o formă clauzală.

Aducerea la o formă clauzală presupune transpunerea lor, mai întîi, într-o formă normală prenexă şi apoi într-o formă normală Skolem şi, în sfîrşit, prin aplicarea unor legi de distributivitate, atingerea formei clauzale la care se aplică metoda rezoluţiei.

Obţinerea formelor normale prenexe se face prin utilizarea unor legi ale logicii predicatelor.

8. Legi ale logicii predicatelor

Fie un domeniu de interpretare D=$\{d_1,d_2...\}$ în care iau valori variabilele individuale x,y,z. Atunci

$\forall x P(x) = P(d_1)\&P(d_2)\&...$ 8.1.

$\exists x P(x) = P(d_1) \vee P(d_2) \vee ...$ 8.2.

$\forall x P(x)$ afirmă, aşadar, că toţi indivizii dintr-un domeniu D au proprietatea P; $\exists x P(x)$ afirmă că cel puţin un individ din D are proprietatea P.

$\forall x(P(x)\&R(x)) = \forall x P(x)\&\forall x R(x)$ (8.3).

Dacă toţi indivizii din D au proprietăţile P şi R, atunci toţi au proprietatea P şi toţi au proprietatea R şi invers.

Aceeaşi formulă transcrisă cu cuantificatorul existenţial nu mai este adevărată. Din faptul că $\exists x P(x)$ şi $\exists x R(x)$ nu decurge în mod necesar că $\exists x(P(x)\&R(x))$. Dacă există un campion al ţării la săritura în înălţime şi există un campion al ţării la box categoria grea, nu rezultă că există un individ ce este simultan campion la săritura în înălţime şi la box categoria grea. Formula:

$\exists x P(x)\&\exists x R(x)\supset\exists x(P(x)\&R(x))$

nu e lege logică. În schimb, conversa ei este adevărată. Dacă cineva este blond şi înalt, atunci este blond şi este înalt.

$\exists x(P(x)\&R(x))\supset\exists x P(x)\&xR(x)$

Formula 8.3 descrie o lege de distributivitate a cuantificatorului universal faţă de conjuncţie.

În mod analog, cuantificatorul existenţial este distributiv faţă de disjuncţie.

$\exists x(P(x) \vee R(x)) \equiv \exists xP(x) \vee \exists xR(x)$ (8.4)

A exista cineva care este P sau R este totuna cu a exista cineva care este P sau a exista cineva care este R.

Transcrierea formulei (8.4) cu cuantificator universal în loc de cel existenţial nu genèrează o formulă adevărată.

Dacă vom considera drept domeniu D mulţimea oamenilor şi vom interpreta predicatul P drept a fi bărbat iar pe R a fi femeie, atunci implicaţia de la stînga la dreapta a formulei

$\forall x(P(x) \vee R(x)) \equiv \forall xP(x) \vee \forall xR(x)$

nu este adevărată. Din faptul că orice fiinţă umană este bărbat sau este femeie nu rezultă nicidecum că toţi oamenii sînt bărbaţi sau toţi oamenii sînt femei.

În schimb, implicaţia de la dreapta la stînga a formulei de mai sus este adevărată.

Vor fi, de asemenea, adevărate următoarele două formule de eliminare a negaţiei din faţa cuantificatorilor:

$-\forall xP(x) \equiv \exists x-P(x)$ (8.5)

$-\exists xP(x) \equiv \forall x-P(x)$ (8.6)

Dacă este *fals că toţi* indivizii din D au proprietatea P, *atunci există cel puţin un individ din D care nu* are proprietatea P şi, invers, dacă *unul nu are* proprietatea incriminată, atunci *nu mai e adevărat că* toţi o au.

Este uşor de atestat şi cea de a doua formă i.e. (8.6). Dacă nu există nici un individ care să sară la înălţime 3 m, atunci despre fiecare dintre individ sau despre toţi se poate spune că nu sare 3 m la înălţime şi invers dacă nici unul nu sare 3m la înălţime, atunci este fals că există cineva care să sară 3m la înălţime (se înţelege, fără prăjină !).

Legile (8.5) şi (8.6) pot fi sintetizate în regula: *Orice cuantificator prefixat de negaţie poate fi înlocuit prin dualul său urmat de negaţie.*

Formulăm acum legea extinderii fictive a domeniului unui cuantificator.

Convenim să notăm prin Q*x* pe $\forall x$ sau $\exists x$ şi prin $*$ pe & sau $\vee$.

Fie A o formulă în care nu apare variabila *x*. Atunci va avea loc echivalenţa:

$A * Qx P(x) \equiv Qx(A * P(x))$ (8.7)

Cuantificatorul Q*x* din primul termen a trecut înaintea lui A în cel de al doilea, pe temeiul că A nu conţine variabila *x* şi, în consecinţă, Q*x* va lega aceleaşi apariţii din P(*x*).

Dacă ţinem seama că Q*x* ţine locul lui $\forall x$ sau $\exists x$ iar $*$ ţine locul lui & sau $\vee$, atunci din (8.7) obţinem lesne pentru echivalenţe distincte, ca exemplificări ale lui 8.7. De exemplu:

$A\& \exists xP(x) \equiv \exists x(A\&P(x)$

se obţine din (8.7) prin înlocuirea lui $*$ prin & şi a lui Q*x* prin $\exists x$.

Dacă într-o formulă închisă (care nu are variabile libere), aceiaşi variabilă individuală apare sub cuantificatori diferiţi şi ca argument pentru semne predicative distincte, putem înlocui apariţiile variabilei în cauză într-un cuantificator sau într-altul şi predicatele corespunzătoare printr-o altă variabilă.

$Q_1xP(x) * Q_2xR(x) \equiv Q_1x\ P(x) * Q_2yR(y)$ (8.8).

Păstrăm în (8.8) convenţiile introduse mai sus pentru Q şi $*$.

Schema (8.8) poate fi, între altele, instanţiată ca:

$\forall xP(x)\ \&\ \exists xR(x) \equiv \forall xP(x)\ \&\ \exists yR(y)$

Legile introduse mai sus servesc ca reguli sau instrumente formale cu ajutorul cărora putem aduce orice formulă din logica predicatelor la o formă standard numită *formă normală prenexă* în care cuantificatorii, dacă există, apar la începutul formulei.

Cu ajutorul legilor exprimate la (8.3) – (8.8) orice formulă din logica predicatelor de ordinul întîi poate fi adusă la o formulă echivalentă cu ea în formă prenexă.

9. Forme normale în logica predicatelor

Vom vorbi despre trei feluri de forme normale: prenexe, Skolem şi forme clauzale.

Orice formulă din logica predicatelor poate fi adusă la o formă normală prenexă echivalentă cu ea. O formă normală prenexă are forma:

$\exists x_1 ... \exists x_n \forall y_1 ... \forall y_m \exists z_1 ... \exists z_l\ F(x_1,...,x_n, y_1,..,y_m, z_1,...z_l)$ (9.1.)

Formula 9.1. are un *prefix* alcătuit de cuantificatorii $\exists x_1 ... \forall y_1 ... \exists z_1 ... \exists z_r$ şi o matrice alcătuită din $F(x_1,...,z_l)$. Evident, prefixul poate conţine mai multe succesiuni de cuantificatori existenţiali şi universali decît s-a ilustrat în schema 9.1.

Revenim la raţionamentele **R2**, **R3** şi **R4** sau mai exact la formulele din logica predicatelor ce corespund acestor raţionamente, $R2^0$, $R3^0$ şi $T4^0$. $R2^0$ este deja la o formă normală prenexă.

$R3^0$ are în formă normală prenexă numai premisa 3, celelalte trei formule, 1, 2 şi 4 nefiind în forma cerută. Formula 1:

1. $\forall u(\exists t$ (se-teme $(u,t)) \supset -\forall v$(iubeşte$(v,u)))$
1.2. $\forall u(-\exists t$ se-teme (u,t) $\vee -\ \forall v$ (iubeşte $(v,u)))$
1.3. $\forall u(\forall t-$(se-teme $(u,t)) \vee \exists v-$(iubeşte $(v,u)))$
1.4. $\forall u \forall t\ \exists v(-$(se-teme $(u,t)) \vee -$ (iubeşte $(v,u)))$

Trecerea de la 1 la 1.2 s-a făcut pe baza echivalenţei $A \supset B \equiv -A \vee B$; trecerea de la 1.2 la 1.3 s-a făcut pe baza legilor (8.5) şi (8.6) de eliminare a cuantificatorilor negaţi; în sfîrşit, trecerea de la 1.3. la 1.4 s-a făcut pe baza legii extinderii fictive a domeniului unui cuantificator, i.e. a legii (8.7).

Aducem la formă normală prenexă premisa 2 din $R3^0$.

2. $\forall y$(păzit$(y) \supset \exists z$(se-teme $(y,z)))$
2.1. $\forall y(-$(păzit $(y)) \vee \exists z$(se-teme $(y,z)))$
2.2. $\forall y\ \exists z\ (-$(păzit $(y)) \vee$ se-teme $(y,z))$

2.2.este f.n. prenexă a lui 2 din $R3^0$. S-a aplicat pentru a ajunge la această formă $A \supset B \equiv -A \vee B$ şi legea extinderii fictive a domeniului unui cuantificator (8.7).

Aplicăm acelaşi tratament concluziei din $R3^0$.

4. $\forall w$(şef $(w$,Mafia$) \supset -\forall s$ (iubeşte $(s,w)))$
4.1. $\forall w(-$şef$(w$,Mafia$) \vee -\forall s$ (iubeşte $(s,w)))$
4.2. $\forall w(-$şef $(w$,Mafia$) \vee \exists s-$(iubeşte $(s,w)))$
4.3. $\forall w\ \exists s\ (-$şef $(w$, Mafia$) \vee -$(iubeşte $(s,w)))$

S-au aplicat în ordine legile eliminării implicaţiei, ne-

gaţiei unui cuantificator universal şi a extinderii fictive a domeniului unui cuantificator.

Formulele 1.4, 2.2, 3 şi 4.3 reprezintă f.n. prenexă pentru $R3^0$.

Aducem la f.n. prenexă setul de formule $R4^0$ ce reprezintă expresia formalizată a raţionamentului **R4** propus în 1.

Considerăm prima premisă din ($R4^0$):

1. $\exists x$ (iubeşte (x, Ion)) $\supset$fericit (Ion).
1.1. $-\exists x$ (iubeşte (x,Ion) $\vee$ fericit (Ion)
1.2. $\forall x-$(iubeşte (x,Ion)) $\vee$ fericit (Ion))
1.3. $\forall x$(−(iubeşte (x,Ion) $\vee$ fericit (Ion))

În trecerea de la 1 la 1-3 s-au aplicat în ordine legile excluderii implicaţiei, excluderii cuantificatorului existenţial şi a extinderii fictive a domeniului unui cuantificator, (8.7).

Formula 2 din $R4^0$ este la o f.n. prenexa dar nu este la o formă clauzală.

2.1. $\forall y$(−(iubeşte (y,Ana)) $\vee$ iubeşte (Ana,y))

3 şi 4 din $R4^0$ sînt deja la o formă clauzală; ele sînt chiar clauze unitare.

Toate premizele din $R3^0$ şi $R4^0$ au fost astfel aduse la forme normale prenexe. ■

Limbajul PROLOG

Ion Muşlea

SCHELETUL DE BAZĂ A UNUI SOFTWARE DE ANALIZA A SITUAŢIILOR IVITE ÎN CADRUL UNEI PARTIDE DE GO

1. Introducere

În numerele precedente revista **PC-MAGAZIN** a publicat mai multe articole privind atît programarea logică, cît şi diverse posibilităţi de folosire ale limbajului **PROLOG** . Avînd în vedere cunoştinţele dobîndite pînă acum de cititor am decis prezentarea în continuare a unui program, de dimensiuni mai mari în care se folosesc mai multe facilităţi ale **TURBO PROLOG**-ului (baze de date interne, grafică,etc.),urmînd ca după acest "hop" să continuăm cu prezentarea sistematică a tuturor categoriilor de predicate oferite de produsul Borland, versiunea 2.0.

Acest articol are două scopuri: pe de o parte cel anunţat în titlu, iar pe de altă parte evidenţierea folosirii diverselor resurse şi tehnici specifice programării în **PROLOG**.

Deci pentru iubitorii jocului de **GO** programul prezentat poate fi interesant atît din punct de vedere al programării în **PROLOG**, cît şi ca o bază ce poate fi perfecţionată şi extinsă spre un program veritabil de jucat **GO**.

Pentru cei neiniţiaţi în **GO** va rămîne doar scopul didactic de aprofundare a cunoştinţelor despre **PROLOG**; în ajutorul lor va veni al doilea capitol al articolului în care se vor explica pe scurt acele reguli ale jocului de **GO** pe care le foloseşte programul. Avînd în vedere faptul că se spune că "pe cît de simple sînt regulile, pe atît de dificil e jocul", cititorul articolului poate aborda fără FRICĂ SCURTA PREZENTARE CE URMEAZĂ.

2. Cîteva reguli din jocul de GO

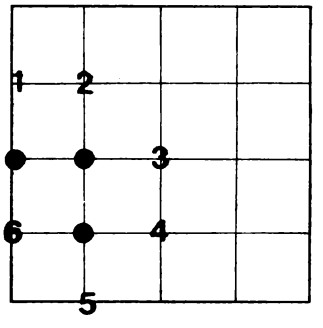
GO-ul se practică pe tablă pătrată, avînd de obicei un caroiaj de 19*19. La începutul partidei tabla e goală. Piesele se aşează în intersecţiile caroiajului, inclusiv pe marginile tablei — nu în interiorul "pătrăţelelor" — şi nu se mai mută în timpul jocului. Se va folosi termenul de "MUTARE" în sensul generic de plasare a unei piese pe tablă. Jucătorii (în număr de doi) mută pe rînd, scopul fiecăruia fiind de a închide cît mai multe puncte ale caroiajului în contururi formate din piese proprii şi folosind eventual şi marginile tablei drept graniţă. În timpul partidei se capturează prin încercuire piese adverse, al căror număr se adaugă la cel al punctelor încercuite, contribuind la punctajul fiecărui jucător. Va cîştiga jucătorul ce realizează un punctaj superior.

Punctele (libere) ale caroiajului ce sînt adiacente unei piese se numesc LIBERTĂŢI ale acelei piese. Deci o piesă solitară aflată la mijlocul tablei poate avea cel mult 4 libertăţi, la margine 3, iar pe colţ 2. Un grup de piese adiacente pe liniile şi coloanele tablei se comportă solidar, însumîndu-şi libertăţile. Un grup de piese încercuit de adversar astfel încît nu mai are nici o libertate se va numi CAPTURAT şi se va elimina de tablă.

Printr-un ochi se înţelege un grup de libertăţi INTERIOARE unui grup de piese. Pentru ca un grup de piese să fie VIU (VIABIL) el trebuie să aibă cel puţin doi ochi separaţi (neadiacenţi). Acest lucru e adevărat în virtutea faptului că în **GO** e interzisă SINUCIDEREA (adică plasarea unei piese pe tablă de aşa manieră încît să nu aibă nici o libertate), cu excepţia cazului în care efectul imediat al mutării duce la capturarea unor grupuri adverse. Deci în ideea existenţei a doi ochi neadiacenţi, capturarea este imposibilă prin faptul că mutarea în interiorul unui ochi va duce la sinucidere fără a se obţine vreo captură (căci mai există cel puţin încă o libertate interioară în cadrul celuilalt ochi).

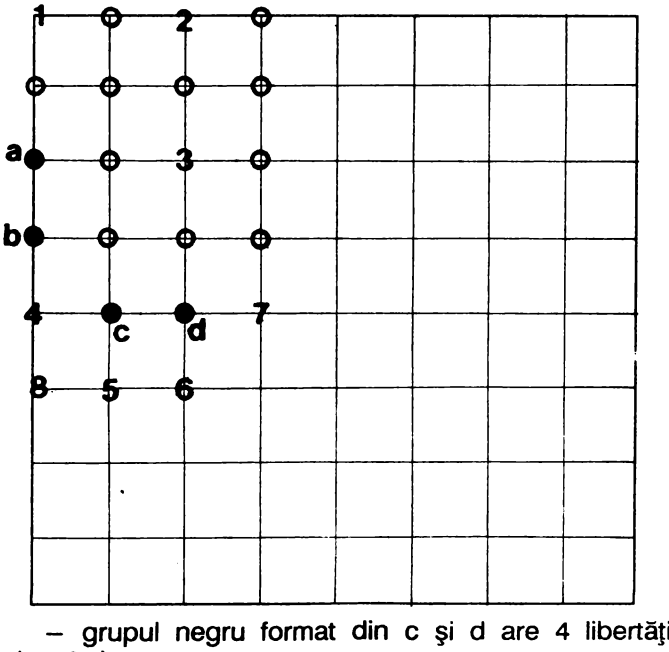
Exemple:

– grupul are 6 libertăţi;



– grupul alb are 3 libertăţi interioare (1,2,3);

– grupul negru format din a şi b are o singură libertate (4);



– grupul negru format din c şi d are 4 libertăţi (4,5,6,7);

– dacă albul mută în poziţia 4 capturează a şi b;

– dacă negrul mută în poziţia 4, va obţine un singur grup a, b,c, d (cu patru libertăţi: 5,6,7,8).

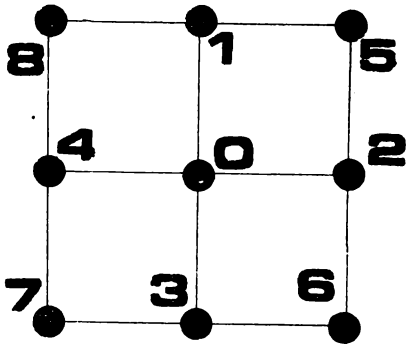
3. PROGRAMUL PROLOG

3.1. Noţiuni introductive

Translatînd acum noţiunile de **GO** într-un limbaj mai "informatic", vom înlocui noţiunea de "ADIACENŢĂ" cu cea de "VECINĂTATE DE 4", astfel încît piesele unui grup vor fi în relaţia de VECIN4 (adică o piesă poate avea vecine — adiacente — altele 4). Vom mai introduce şi noţiunea de "VECIN DE 8", care e caracterizată prin faptul că o piesă poate avea cel mult 8 piese cu care e în relaţia de VECIN8 (adică cele 4 de tip "VECIN DE 4" şi cele 4 aflate pe "diagonală").

Exemplu:

Piesa 0 este în relaţia VECIN4 cu piesele 1-4 şi în relaţia VECIN8 cu piesele 1-8.



Practic acest program are drept scop înregistrarea unei partide de **GO** a 2 parteneri de aşa manieră încît să asigure atît un "menaj intern" (nu permite mutări eronate, asigură eliminarea grupurilor capturate, etc.) cît şi ţinerea la zi a grupurilor de pe tablă. Totodată se asigură şi determinarea numărului de ochi ai fiecărui grup.

Programul e conceput de aşa manieră încît să poată fi extins printr-o analiză riguroasă a ochilor (falşi sau nu), rezolvînd astfel problema viabilităţii grupurilor de piese de pe tablă.

Piesele de pe tablă vor fi păstrate foarte simplu şi asigurînd un acces rapid, după cum urmează:

— două liste (cîte una pentru fiecare culoare) conţinînd grupurile formate din piese ce se află în relaţia VECIN4 (grupurile fiind păstrate sub formă de liste rezultă o structură de tip lista de liste);

— două liste (cîte una pentru fiecare culoare) conţinînd grupurile formate din piese ce se află în relaţia VECIN8;

— o bază de date de tip intern (se foloseşte pentru verificări rapide şi pentru o eventuală salvare a unei poziţii de pe tablă sau a unei partide).

3.2. Structura datelor şi lista predicatelor

```
domains
cda_meniu=char
xc,yc,col = integer
o_piesa = coord(xc,yc)
grup_piese = o_piesa*   /* GRUP DE PIESE */
L_grps = grup_piese*    /* LISTA DE GRUPURI */

database
piesa(xc,yc,col)

predicates

go
tabla
horiz(integer)
vertic(integer)
play(L_grps,L_grps,L_grps,L_grps,
read_ans(L_grps,L_grps,L_grps,L_grps,cda_meniu)
verify(L_grps,L_grps,L_grps,L_grps,xc,yc,col,
L_grps,L_grps,L_grps,L_grps)
pune_piesa(integer,integer,integer)
pe_tabla(xc,yc)

adauga_la_grupuri(L_grps,L_grps,xc,yc,col,
L_grps,L_grps,integer)
cauta_grup_apartinator(L_grpsmxc,yc,L_grps,
grup_piese,L_grps,integer)
exista_vecin_în_grup(grup_piese,xc,yc,integer)
vecin(xc,yc,xc,yc,integer)
concateneaza(grup_piese,grup_piese,grup_piese)

vizualizeaza_grupuri(L_grps,col)
marcheaza_grup(grup_piese,col)
marcheaza(xc,yc,col)

refa_grup(grup_piese,col)
translateaza(grup_piese,integer)
gaseste_Xmin(grup_piese,integer,integer)
gaseste_Xmax(grup_piese,integer,integer)
gaseste_Ymin(grup_piese,integer,integer)
gaseste_Ymax(grup_piese,integer,integer)
situatia(xc,yc,xc,yc,integer)
max_şi_ajust(integer,integer,integer,xc,yc,xc,yc,
integer)
corectie_inf(integer,integer)
corectie_sup(integer,integer)
ajustare_coordonate(grup_piese,xc,yc,grup_piese,
grup_piese)
det_nr_ochiuri(grup_piese,integer,integer)
inverseaza_grup(grup_piese,grup_piese,integer)
genereaza(integer,integer,integer,grup_piese,
grup_piese,grup_piese)
membru(grup_piese, o_piesă)
form_grupuri(grup_piese,L_grps,L_grps)
numara_grupuri(L_grps,integer,integer)
tip_nr_ochi(integer)

capturi(L_grps,L_grps,L_grps,L_grps,L_grps,
L_grps,L_grps,L_grps,integer)
captureaza(L_grps,L_grps,L_grps,L_grps,L_grps,
col)
pierdut(grup_piese,integer,col)
libertati(grup_piese,integer,col)
cauta_grup8(L_grps,grup_piese,L_grps,L_grps)
elim(grup_piese,grup_piese,grup_piese,grup_piese)
conc_11(L_grps,L_grps,L_grps)
refa_L8(L_grps,grup_piese,L_grps)
caut_grup_apar(L_grps,xc,yc,L_grps)
verifica(xc,yc,grup_piese,col)
```

3.3. Predicatele

În funcţie de rolul lor, subrutinele se pot împărţi în patru clase (după cum se ocupă de probleme introductive, de adăugarea noii piese la grupurile existente, de efectuarea capturilor sau de analiza grupurilor).

3.3.1. Probleme introductive

```
/* GO este "THE GOAL" — ŢELUL */
go:—          detectgraph (Gd, Gm),
              initgraph(Gd,Gm,_,_,""),
              makewindow (1,7,7,"",1,65,5,14),
                    /* fereastra folosită ca meniu */
              tablă,
              play ([],[],[],[]).

/* TABLĂ va desena o tablă 19*19 */
tablă:—       horiz(19),
              vertic(19).

/* HORIZ va trasa liniile orizontale */
horiz(0):—    !.
horiz(X):—    X2=200—10*(21—X)+5,
              line(129,X2,488,X2),
              X1=X—1,
              horiz(X1).

/* VERTIC va trasa liniile verticale */
vertic(0):—   !
vertic(X):—   X2=525-20*(21-X)+3,
              line(X2,5,X2,184),
              X1=X—1
              vertic(X1).

/* PLAY afişează meniul principal */
play(L_albe8,L_negre8,L_albe4,L_negre4):—
              write("Movement"),
              n1,
              write("Analyser"),
              n1,
              write("Quit"),
              readchar(AnsW),
              n1,n1,n1,n1,n1,
              read_ans(L_albe8,L_negre8,L_albe4,
              L_negre4,Answ).
/* READ_ANS gestionează meniul principal */
read_ans(L_a8,L_n8,L_a4,L_n4,'m'): /* URMĂTOAREA
                                      MUTARE */
              write("colour"),
              readint(Cul),
              write("x"),
              readint(Xcoord),
              write("y"),
              readint (Ycoord),
              verify(L_a8,L_n8,L_a4,L_n4,Xcoord,
              Ycoord,Cul,New_a8,New_n8,New_a4,
              New_n40),
              capturi(New_a8,New_n8,New_a4,
              New_n4,New_8a,New_8n,
              New_4a,New_4n,Cul),
              play(New_8a,New_8n,New-4a,New_4n).
read_ans(L_a8,L_n8,L_a4,L_n4,'a'):— /* ANALIZA
                                    GRUPURILOR */
              vizualizează_grupuri(L_a8,0),
              vizualizează_grupuri(L_n8,1),
              play(L_a8,L_n8,L_a4,L_n4).
read_ans(_,_,_,_,'q'):—closegraph.    /* QUIT */
read_ans(L_a8,L_n8,L_a4,L_n4,_):—
              play(L_a8,L_n8,L_a4,L_n4). /* DEFAULT */
```

3.3.2. Adăugarea noii piese la grupurile de piese existente

Pentru a realiza acest lucru se începe prin verificarea corectitudinii mutării (coordonatele să fie în domeniul cerut şi în acea poziţie să nu existe vreo altă piesă). Dacă această condiţie e îndeplinită se va căuta în lista grupurilor de culoarea respectivă grupul sau grupurile ce au cel puţin o piesă care să fie în relaţia VECIN4 cu noua piesă. În caz că vom avea un singur grup piesa se va adăuga la acesta.În caz că nu vom găsi vreun grup de acest gen se va forma un nou grup ce va avea drept unic element noua piesă. În cazul în care piesa va avea vecini de tip 4 în mai multe grupuri toate acestea vor fi concatenate într- un singur grup la care se va adăuga şi noua piesă, iar vechile grupuri vor fi eliminate din listă.

```
/* VERIFY verifică dacă mutarea e corectă */
verify(L_a8,L_n8,L_a4,L_n4,X,Y,_,New_a8,New_n8,
              New_a4,NeW_n4):—
              /* Mutare greşită: în acea poziţie există
               o piesă */
              piesa(X,Y,_),
              write("Bad movement"),
              write("Try again !!"),
              readchar (_),
              n1,
              New_a8=L_a8,
              New_n8=L_n8,
              New_a4=L_a4,
              New_n4=L-n4.
verify(L_a8,L_n8,L_a4,L_n4,X,Y,_,New_a8,New_n8,
              New_a4,New_n4):—
              /* Mutare greşită: coordonatele piesei
              nu aparţin tablei */
              not(pe_tablă(X,Y)),
              write("Bad movement"),
              write("Try again !!"),
              readchar(_),
              n1,
              New_a8=L_a8,
              New_n8=L_n8,
              New_n4=L_n4.
verify(L_a8,L_n8,L_a4,L_n4,X,Y,C,New_a8,New_n8,
              New_n4):—
              /* Mutare posibil de executat */
              adaugă_la_grupuri(L_a8,L_n8,X,Y,C,
              Noua_la8,Noua_ln8,0),
              adaugă_la_grupuri(L_a4,L_n4,X,Y,C,
              Noua_la4,Noua_ln4,1),
              New_a8=Noua_la8,
              New_n8=Noua_la4,
              NeW_n4=Noua_ln4,
              pune_piesa(X,Y,C),
              asserta(piesa(X,Y,C)).
```

NOTĂ

Într-o versiune ulterioară ar trebui testată şi condiţia neacceptării unei sinucideri decît în cazul în care duce la o captură.

```
/* PE_TABLĂ verifică apartenenţa coord.la domeniul
                                          [0,18] */
              pe_tablă(X,Y,):-X=0,X  ,Y=0,Y

/* PUNE_PIESA va desena o piesă pe tabla de joc */
pune_piesa(Row,Col,0):—
              /* desenează piesa albă */
              Xcoord=128+Row*20,
```

```
                Ycoord=5+Co1*10,
                setfillsryle(0,1),
                fillellipse(Xcoord, Ycoord,8,4),
                ellipse(Xcoord,Ycoord,0,360,8,4),
                setfillstyle(1,1).
pune_piesa(Row,Col,1):-
                /* desenează piesa neagră */
                Xcoord=128+Row*20,
                Ycoord=5+Col*10,
                fillellipse(Xcoord, Ycoord,8,4).

/* ADAUGĂ_LA_GRUPURI adaugă noua piesă la lista cu grupurile formate deja */

adaugă_la_grupuri(L_a,L_n,X,Y,O,Noua_la,Noua_ln,0):-/* primul 0
indică faptul că piesa e albă, iar ultimul 0
                indică căutarea după relaţia VECIN8 */
                caută_grup_aparţinător(L_a,X,Y,[],[],
                Lista_actuală,O),
                Noua_la=Lista actuală,
                Noua_ln=L_n.

adaugă_la grupuri(L_a,L_n,X,Y,1,Noua_la,Noua_ln,0):-
/* parametrul 1 indică faptul că piesa e neagră, iar
parametrul 0 indică căutarea după relaţia VECIN8 */
                caută_grup_apartinator(L_n,X,Y,[],[],
                Lista_actuala,0),
                Noua+ln=Lista_actuala,
                Noua_la=L_a.

adauga_la_grupuri(L_a,L_n,X,Y,O,Noua_la,Noua_ln,1):-/* parametrul 0 indică faptul că piesa e albă, iar parametrul 0 indică căutarea după relaţia VECIN4 */
                cauta_grup_apartinator(L_a,X,Y,[],[],
                Lista_actuala,1),
                Noua_la=Lista_actuala,
                Noua_ln=L_n.

adauga_la_grupuri(L_a,L_n,X,Y,1,Noua_la,Noua_ln,1):-/* primul 1
indică faptul că piesa e neagră, iar al
        doilea 1 indică căutarea după relaţia VECIN4 */
                caută_grup_apartinator(L_n,X,Y,[],[],
                Lista_actuala,1),
                Noua_ln=Lista_actuala,
                Noua_la=L_a.

/* CAUTA_GRUP_APARTINATOR verifică dacă piesa
                        aparţine unui grup existent */
cauta_grup_apartinator([],X,Y,L_beg,Noul_grup,
                Lista_actuala,_):-
                Lista_actuala=[[coord(X,Y)|Noul_grup]
                |L_begî
cauta_grup_apartinator([GRUP:Rest],X,Y,L_bg,
                Noul_grup,Lista_actuala,TipVecin):-
                exista_vecin_in_grup(GRUP,X,Y,TipVecin),
                concatenează(Noul_grup,Grup,G_conc),
                cauta_grup_apartinator(Rest,X,Y,L_bg,
                G_conc,Lista_actuala,TipVecin).
cauta_grup_apartinator([Head|TAI[],X,Y,
                L_beg,Noul_grup,L_actuala,TipVecin):-
                cauta_grup_apartinator(TAIL,X,Y,[Head
                :L_beg],Noul_grup,L_actuala,TipVecin).

/* CONCATENEAZĂ două liste, returnînd răspunsul
                                        în a treia */
concatenează([],L,Final_list):-
                Final_list+L.
concateneaza([Head|Tail],L2,Final_list):-
concateneaza(Tail,[Head|L2],Final_list).

/* EXISTA_VECIN_IN_GRUP(Grup,X,Y) verifică dacă
                        (X,Y) are un vecin în Grup */
                exista_vecin_in_grup([],_,_,_):-fail.
exista_vecin_in_grup([coord(Xe,Ye)|_],X,Y,TipVecin):-
                vecin(X,Y,Xe,Ye,TipVecin).
exista_vecin_in_grup([|Rest],X,Y, TipVecin):-
                exista_vecin_in_grup(Rest,X,Y, TipVecin).

/* VECIN testează dacă două piese sînt vecine */
vecin(XE,Ye,X,Y,O):-
                Ye=Y,
                Xe=X-1
                or
                Ye=Y,
                Xe=X+1
                or
                Ye=Y+1,
                Xe=X
                or
                Ye=Y-1,
                Xe=X
                or
                Ye+Y-1,
                Xe=X-1
                or
                Ye=Y+1,
                Xe=X+1
                or
                Ye+Y+1,
                Xe+X-1
                or
                Ye=Y-1,
                Xe=X+1.
vecin(Xe,Ye,,X,Y,1):-
                Ye=Y,
                Xe=X-1
                or
                Ye=Y
                Xe=X+1
                or
                Ye=Y+1,
                Xe=X
                or
                Ye=Y-1,
                Xe=X
```

3.3.3. Analiza grupurilor

Pe moment această parte a programului se rezumă la determinarea numărului de ochi al fiecărui grup de pe tablă. De notat faptul că realizarea s-a făcut de o manieră care să permită o cît mai eficientă continuare a programului în scopul testării viabilităţii grupurilor.

Tehnica de lucru este următoarea: din lista de grupuri se vor extrage pe rînd toate grupurile care vor fi analizate după cum urmează.În primul rînd se va efectua o încadrare a grupului într- un pătrat de dimensiuni minime de aşa manieră încît să se respecte situaţia de pe tablă (adică dacă vreo piesă a grupului e pe o margine sau într-un colţ, aceeaşi piesă să fie în acelaşi gen de poziţie şi în "tabla miniaturală", iar dacă nici o piesă a grupului nu este poziţionată pe vreo margine sau colţ, situaţia să se reflecte identic şi pe "noua tablă"). O dată obţinut acest deziderat nu mai rămîne de realizat decît un "negativ" al imaginii obţinute anterior prin încadrarea grupului şi ignorarea tuturor celorlalte piese de pe tablă (în sensul că toate poziţiile caroiajului neocupate de piese se vor considera drept piese de aceeaşi culoare iar cele ce conţineau piese se vor considera drept goale) în care să formăm grupuri de piese caracterizate de relaţia VECIN4. Dacă vom obţine N grupuri întotdeauna unul din ele va reprezenta "restul tablei", iar celelalte N-1 vor fi ochiurile grupului.

```
/* VIZUALIZEAZA_GRUPURI va vizualiza succesiv toate grupurile de
o culoare printr-un semn distinctiv */
vizualizeaza_grupuri([],_).
vizualizeaza_grupuri([Grup|Rest],Culoare):-
                marcheaza_grup(Grup,Culoare),
                translateaza(Grup,_),
                readchar(_),n1,
                refa_grup(Grup,Culoare),
                vizualizeaza_grupuri(Rest,Culoare).

/* MARCHEAZA_GRUP va marca un grup de piese */
marcheaza_grup([],_).
marcheaza_grup([coord(X,Y)|Rest],Culoare):-
                marcheazaX,Y,Culoare),
                marcheaza_grup(rest,Culoare).

/* MARCHEAZA va desemna piesele ce aparţin
                                aceluiaşi grup */
marcheaza(Row,Col,0):-
                Xcoord=128+Row*20,
                Ycoord=5+Col*10,
                X1=Xcoord-2,
                X2=Xcoord+2,
                Y1=Ycoord-2,
                Y2=Ycoord+2,
                line(X1,Ycoord,X2,Ycoord),
                line(X1,Y1,X2,Y2),
                X3=Xcoord-2,
                X4=Xcoord+2,
                Y3=Ycoord+2,
                Y4=Ycoord-2,
                line(Xcoord,Y3,Xcoord,Y4),
                line(X3,Y3,X4,Y4).
marcheaza(Row,Col,1):-
                setcolor(0),
                Xcoord=128+Row*20,
                Ycoord=5+Co1*10,
                X1=Xcoord-2,
                X2=Xcoord+2,
                Y1=Ycoord-2,
                Y2=Ycoord+2,
                line(X1,Ycoord,X2,Ycoord),
                line(X1,Y1,X2,Y2),
                X3=Xcoord-2,
                X4=Xcoord+2,
                Y3=Ycoord+2,
                Y4=Ycoord-2,
                line(X3,Y3,X4,Y4),
                line(Xcoord,Y3,Xcoord,Y4),
                setcolor(1).

/* REFA_GRUP elimină marcajele efectuate asupra
                                unui grup */
refa_grup([],_).

refa_grup([coord(X,Y)|Rest],Culoare):-
                pune_piesa(X,Y,Culoare),
                refa_grup(Rest,Culoare).

/* TRANSLATEAZĂ va realiza încadrarea unui grup
                        într-o "tablă minimală" */
translateaza(Grup,Nr0):-
                gaseste_Xmin(Grup,20,Xm),
                gaseste_Xmax(Grup_1,XM1),
                gaseste_Ymin(Grup,20,Ym),
                gaseste_Ymax(Grup,-1,YM1),
                Nr1=XM1-Xm+1,
                Nr2=YM1-Ym+1,
                situatia(Xm,Ym,XM1,YM1,Tip),
                max_si_ajust(Nr1,Nr2,NrLinii,Xm,Ym,Xa,
                Ya,Tip),
                ajustare_coordonate(Grup,Xa,Ya,[],
                Grup_ajustat),
                Dim=NrLinii-1,
                det_nr_ochiuri(Grup_ajustat,Dim,Nr0).

/*'GASESTE_XMIN va determina MIN pe coord X */
gaseste_Xmin([],Val,Xm):-
                corectie-inf(Val,Val1),
                Xm=Val1.
gaseste_Xmin([coord(X,_)|Rest],Val,Xm):-
                Val<=X,
                gaseste_Xmin(Rest,Val,Xm).
gaseste_Xmin([coord(X,-)|Rest],Val,Xm):-
                Val>X,
                gaseste_Xmin(Rest,X,Xm).

/* GASESTE_YMIN va determina MIN pe coord Y */
gaseste_Ymin([],Val,Ym):-
                corectie_inf(Val,Val1),
                Ym=Val1.
gaseste_Ymin([coord(_,Y)|Rest],Val,Ym):-
                Val<=Y,
                gaseste_Ymin(Rest,Val,Ym).
gaseste_Ymin([coord(_,Y)|Rest],Val,Ym):-
                gaseste_Ymin(rest,Y,Ym).

/* GASESTE_XMAX va determina MAX pe coord X */
gaseste_Xmax({},Val,XM):-
                corectie_sup(Val,Val1),
                XM=Val1.
gaseste_Xmax([coord(X,_)|Rest],Val,XM):-
                Val>=X,
                gaseste_Xmax(Rest,Val,XM).
gaseste_Xmax([coord(X,_)|Rest],Val,XM):-
                Val<X,
                gaseste_Xmax(Rest,X,XM).

/* GASESTE_YMAX va determina MAX pe coord Y */
gaseste_Ymax([],Val,YM):-
                corectie_sup(Val,Val1),
                YM=Val1.
gaseste_Ymax([coord(_,Y)|Rest],Val,YM):-
                Val>=Y,
                gaseste_Ymax(Rest,Val,YM).
gaseste_Ymax([coord(_,Y)|Rest],Val,YM):-
                Val<Y,
                gaseste_Ymax(Rest,Y,YM).

/* CORECTIE_INF lasă spaţiu între piesă şi margine
                        (dacă nu e pe margine) */
corectie_inf(Init,Final):-
                Init=0,
                Final=Init.
corectie_inf(Init,Final):-
                Init<>0,
                Final=Init-1.    /* la scădere va da 1,
                                deci nu e pe margine */

/* CORECTIE_SUP lasă spaţiu între piesă şi margine
                        (dacă nu e pe margine) */
corectie_sup(Init,Final):-
                Init=18,
                Final=Init.
corectie_sup(Init,Final):-
                Init18,
                Final=Init+1.      /* la scădere va da 1,
                                deci nu e pe margine */

/* SITUAŢIA recunoaşte dacă un grup e de:
   — centru(return 0);
   — margine(return1-stînga/sus10-dreapta11-jos);
   — colţ(return2-stînga-jos,3-dreapta-jos,
     4-dreapta- sus,5-stînga-sus). */
```

(Continuarea articolului în numărul viitor.)

REŢELE DE CALCULATOARE(II)

Ing. Sorin Deaconu

Introducere

Continuînd seria de articole referitoare la reţelele de calculatoare, prezentăm în acest număr cîteva caracteristici generale ale reţelelor locale.

În modul cel mai general o reţea locală de calculatoare (LAN) permite împărţirea unor resurse comune (hard discuri, imprimante, linii de comunicaţie) între mai multe calculatoare interconectate pe o arie restrînsă. Aceasta permite, de exemplu, conectarea unei maşini mai slab dotate din punct de vedere hardware (fără unităţi de disc) în reţea, mărindu-se astfel performanţele acesteia.

În cadrul reţelelor locale, o importanţă majoră o au reţelele de PC-uri. Din punct de vedere conceptual există două tipuri de reţele de PC-uri:

a) Reţele *"peer-to-peer"*, în care nu există calculatoare cu funcţii speciale în reţea. Acest tip permite fiecărui utilizator conectat în reţea să aibă acces la resursele oricărui alt utilizator (prin resursele unui utilizator se înţeleg anumite resurse ale calculatorului pe care acesta le accesează, cum ar fi unităţi de disc, imprimante, etc.). De exemplu, într-o reţea "peer- to-peer" cu doi utilizatori A şi B, utilizatorul A are acces la resursele utilizatorului B şi reciproc.

Reţelele *"peer-to-peer"* sînt relativ uşor de implementat şi de utilizat prezentînd în schimb două dezavantaje majore: numărul redus de utilizatori şi, mai ales, performanţe scăzute ale reţelei.

b) Reţele bazate pe strategia *"client-server"*. În acest tip de reţele unul sau mai multe calculatoare funcţionează ca "file server" sau "network server", deosebindu-se de celelalte calculatoare din reţea. În rîndurile ce urmează vom face o comparaţie între acest tip şi reţelele de tip *"peer-to-peer"*, din care va rezulta superioritatea reţelelor *"client-server"*.

În reţelele de tip *"peer-to-peer"* reţeaua există ca "oaspete" sau ca un task în calculatorul gazdă (rulînd sub un anumit sistem de operare, de exemplu MS-DOS). Toate limitările datorate sistemului de operare din maşina gazdă (de exemplu mărimea volumelor de disc, structura fişierelor, lipsa posibilităţii de lucru multitasking, lipsa unui sistem de securitate a informaţiilor vehiculate sau memorate în mediul de reţea, etc.) vor afecta performanţele reţelei.

În reţelele *"client-server"* există, după cum am arătat, calculatóare setate ca "file server" pe care se rulează un sistem de operare numit *Network Operating System* (NOS). Acest sistem de operare gestionează într-un mod propriu resursele maşinii (ale file *server*-ului), folosind, de exemplu, o structură proprie de formatare a discului, o structură proprie a fişierelor şi a metodelor de acces la fişiere, precum şi un ansamblu de facilităţi pentru creşterea vitezei şi a fiabilităţii reţelei. Dintre acestea am aminti gestionarea optimă a acceselor la hard discurile file serverului (elevator seeking), fiabilizarea hard discurilor (cuplarea a două discuri pe un canal al controller-ului de hard disk - *disk mirroring,* sau a cîte unui disc pe cîte un canal - *disk duplexing*) şi un sistem de securitate al reţelei organizat pe mai multe nivele.

Din punctul de vedere al numărului de utilizatori care pot lucra simultan, reţelele *"client-server"* sînt net superioare. De exemplu, în reţeaua NetWare a firmei Novell, versiunea 3.1, numărul maxim de staţii este de 1024.

Topologii de reţele locale

Un PC se cuplează într-o reţea de calculatoare prin introducerea unei plăci de interfaţă (*Network Interface Card* - NIC) într-unul din sloturile de extensie liber (de 8 sau 16 biţi, în funcţie de specificaţiile hardware ale plăcii). Cuplarea la mediul de comunicaţie (în general, cablu coaxial) se face prin mufe BNC, conectori de tip T, conectori Cheapernet etc.

În funcţie de modul de conectare a calculatoarelor în reţea se definesc tipurile de topologii de reţele locale.

În prezent există trei tipuri majore de topologii de LAN:

— topologie de tip *star* (stea)
— topologie de tip *ring* (inel)
— topologie de tip *bus* (magistrală)

a) Topologia *star* presupune file serverul plasat în centrul "stelei", fiecare staţie de lucru (*work station*-WS) fiind conectată la file server printr-un cablu separat (fig.1).

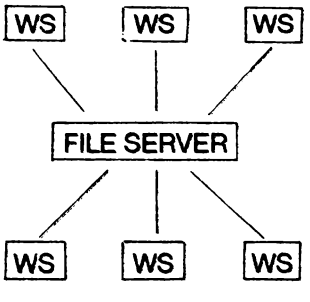


Figura 1.

Un exemplu de astfel de reţea este reţeaua S-Net a firmei Novell.

b) Topologia *ring* presupune o schemă de conectare în care staţiile de lucru şi file serverul formează un inel complet (fig.2).

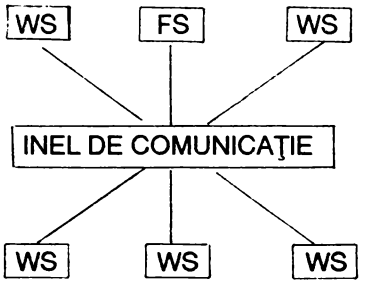


Figura 2.

Exemplu de reţea cu topologie ring: **IBM-Token Ring**

c) Topologia *bus* utilizează un singur cablu care conectează atît file serverul cît şi staţiile de lucru (fig.3).

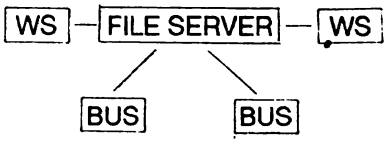


Figura 3.

Exemple de reţele cu topologie **bus**:*Ethernet, Omninet, Vlan*

Standarde de reţea

În prezent, în lumea reţelelor locale există trei standarde majore, fiecare standard corespunzînd unei topologii din cele prezentate anterior.

a) Standardul **ARCnet** utilizează topologia star, permitînd realizarea de reţele cu performanţe medii şi cost scăzut.

b) Standardul **Token Ring** utilizează topologia ring. Este dezvoltat în principal de IBM, permitînd performanţe foarte bune şi o mare conectivitate. La variantele cele mai recente s-a ajuns la o viteză de 16 Mbps.

c) Standardul **Ethernet**, utilizează topologia bus. Este tipul de reţea locală cel mai utilizat în lume, permiţînd performanţe excelente şi posibilităţi foarte bune de interconectare.

Metode de transmisie

Două metode se folosesc astăzi pentru vehicularea informaţiilor în cadrul reţelei locale:

— metoda "*token passing*"

— metoda *CSMA/CD*

a) Metoda "*token passing*" utilizează un *token* ("semn") electronic care este trecut de la o staţie la alta de-a lungul inelului. În momentul iniţial este pus in circulaţie, de către o staţie "master", un *token* liber (*token free*). O staţie nu poate transmite pachetul de date pe care-l are de transmis dacă nu este în posesia acestui *token free*. În momentul interceptării *token*-ului *free*, staţia care doreşte să transmită marchează *token*-ul ca fiind ocupat (*token busy*), după care transmite pachetul de date. La staţia destinaţie, după preluarea pachetului de date, se marchează *token*-ul *busy* ca fiind *token free*, procesul reluîndu- se. Prin creşterea numărului de *token*-uri *free* puse în circulaţie se poate mări viteza de transmisie, de la valoarea de 2 Mbps pînă la 16Mbps.

Acest proces se aseamănă cu urmatoarea situaţie: Să ne imaginăm un grup de persoane strînse în jurul unei mese rotunde.Între aceste persoane circulă un microfon, în sensul acelor de ceasornic. Pentru ca o persoană să poată lua cuvîntul e necesar ca ea să fie în posesia microfonului. În momentul în care persoana respectivă are microfonul, ea poate întreprinde două acţiuni: să ia cuvîntul, urmînd să transmită microfonul persoanei următoare după ce a terminat luarea de cuvînt sau, în cazul în care nu are nimic de spus, să predea direct microfonul. În toate cazurile este necesară respectarea sensului de mişcare al microfonului.

Reţelele bazate pe acest tip de metoda de transmisie au o comportare bună şi la încărcări mari.

b) Metoda *CSMA/CD* (*Carrier Sense Multiple Access/Collision Detection* — detecţia purtătoarei, acces multiplu, detecţia coliziunilor) permite accesul concurent al tuturor staţiilor la mediul de comunicaţie. Acestea se află în permanentă "ascultare" a mediului (detecţia purtătoarei). În momentul în care o staţie are un mesaj de transmis, ea aşteaptă pîna cînd mediul de comunicaţie e liber, moment în care începe transmisia. Dacă o altă staţie a început în acelaşi moment transmisia propriului său mesaj, ambele staţii intrate în conflict semnalează o coliziune. Datele sînt compromise, fiind ignorate. După un interval de timp aleator ambele staţii încearcă o retransmisie a datelor. Aceste retransmisii se fac de un număr stabilit de ori (n), semnalîndu-se eroare la transmisie dacă numărul de retransmisii pentru acelaşi mesaj depăşeşte valoarea n.

Performanţele acestui tip de reţea scad cu creşterea încărcării, datorită creşterii numărului de coliziuni.

Sisteme de operare în reţea (Network Operating Systems NOS)

Aşa cum am arătat, pe file server-ul unei reţele locale se rulează un software special care constituie sistemul sau de operare.

În acest moment, cele mai importante NOS pentru reţele locale comercializate sînt următoarele:

— *NetWare*, al firmei Novell, cu un procent de 65% din piaţa mondială

— *3PlusOpen*, al firmei 3COM, rulînd sub OS/2, cu un procent de 10-15%

— *Vines,* al firmei Banyan, care rulează ca task sub sistemul de operare Unix şi oferă o cale convenabilă de implementare a reţelelor generale (WAN)

— *DECnet*, al firmei Digital Equipment Corp. (DEC), proiectat ca o reţea Ethernet pentru cuplarea minicalculatoarelor DEC VAX în reţea locală de PC-uri, asigurînd funcţionarea ca *file server.*

Reţeaua NetWare a firmei Novell

NetWare a fost proiectat de Novell ca sistem de operare rulînd pe servere în reţele locale de PC-uri. După cum am arătat, această firmă deţine 65% din piaţa mondială de reţele locale. Vom enumera mai jos trei din motivele pentru care firma deţine această supremaţie:

1. Arhitectura deschisă din punct de vedere hardware. Plăcile de reţea (NIC) folosite pot fi de o mare diversitate şi de la mai multe firme producătoare.

Poate fi folosit ca server aproape orice tip de calculator 286/386, iar ca staţii de lucru orice PC (compatibil IBM, COMPAQ, OLIVETTI, Macintosh etc.). De asemenea pot fi folosite ca file servere şi maşini DEC VAX, urmînd ca în această "familie" să intre şi marea majoritate a maşinilor Unix.

2.. Performanţe excelente ale reţelei, prin folosirea unei mari game de tehnici dintre care enumerăm:

— *turbo FAT* (memorarea File Allocation Table pentru fişierele mai des folosite)

— *elevator seeking* (optimizarea cererilor de acces la hard discurile file server-ului)

— *split reads* (împărţirea cererilor de citire între două discuri "mirrored")

3. Standardizarea reţelei. Novell şi-a stabilit propriile standarde de reţele locale, suportînd în plus orice standard recunoscut de NetWare. De exemplu este suportat standardul NETBIOS pîna în momentul în care acesta va deveni singurul standard de LAN de PC-uri, precum şi "gateway"-uri cu standarde ca SNA, TCP/IP, sistemul de fişiere Macintosh, etc.

NetWare cuprinde cîteva game de reţele locale:

a) *ELS (Entry Level System)*, o reţea cu cîţiva utilizatori (4-8);

b) *Advanced NetWare* 286, o reţea cu performanţe excelente şi cu un număr mare de utilizatori;

c) *SFT NetWare*, asigurînd în plus faţa de precedenta servicii de System Fault Tolerance;

d) *NetWare 386*, cel mai bun produs de pînă acum al firmei Novell;

e) *NetWare for VMS*, asigurînd servicii de PC LAN pentru minicalculatoare de tip DEC VAX rulînd sub VAX/VMS;

f) *Portable Netware,* rulînd pe maşini Unix.

Se studiază, pentru viitor, realizarea de soluţii şi pentru cuplarea calculatoarelor de tip "*mainframe*" într-o reţea de PC- uri.

Din punct de vedere hardware, într-o reţea NetWare există trei componente majore:

I. *Network Interface Card* (NIC), ce formează interfaţa dintre staţia de lucru şi restul reţelei.

II. *Network Cable* (cablul de reţea), la care sînt ataşate toate calculatoarele reţelei

III. *File server*-ul, care asigură managementul resurselor comune ale reţelei sub controlul NOS

În prezent NetWare prevede soluţii pentru cele trei tipuri de reţele prezentate anterior (ARCnet, Ethernet şi Token Ring)

De asemenea, sînt prevăzute facilităţi de extensie a reţelei locale, cum ar fi *bridge*-uri (interne sau externe) şi *gateway.*

Un loc aparte îl ocupă folosirea imprimantelor ataşate reţelei, existînd un "manager" al acestor servicii în reţea ("printer server").

Într-un articol viitor vom prezenta principalele caracteristici ale reţelei Advanced NetWare 286, ca o exemplificare a conceptelor discutate în acest articol. ■

REŢELE NEURONALE(II)

Ing. Ion Stoica

1. MEMORII CLASICE. CARACTERUL DE LOCALITATE

În calculatoarele moderne informaţia este păstrată în circuite electronice speciale numite memorii. În general, o dată memorată se poate citi specificînd adresa locaţiei din memorie unde se găseşte. Fiecărei adrese îi corespunde o singură locaţie de memorie de unde se citeşte, respectiv se scrie data corespunzătoare locaţiei adresate.

Exemplu:Să considerăm o memorie cu 8 locaţii,fiecare locaţie conţinînd 4 biţi (fig. 1). Pentru a putea adresa fiecare locaţie sînt suficienţi 3 biţi de adresă ($2^3 = 8$) notaţi A2, A1 si A0. Pentru fiecare adresă distinctă este selectată o singură locaţie de memorie la ieşire obţinîndu-se valoarea memorată în locaţia respectivă. Astfel, pentru adresa 100 (A2 = 1, A1 = 0 si A3 = 0) la ieşire se obţine valoarea 1111.

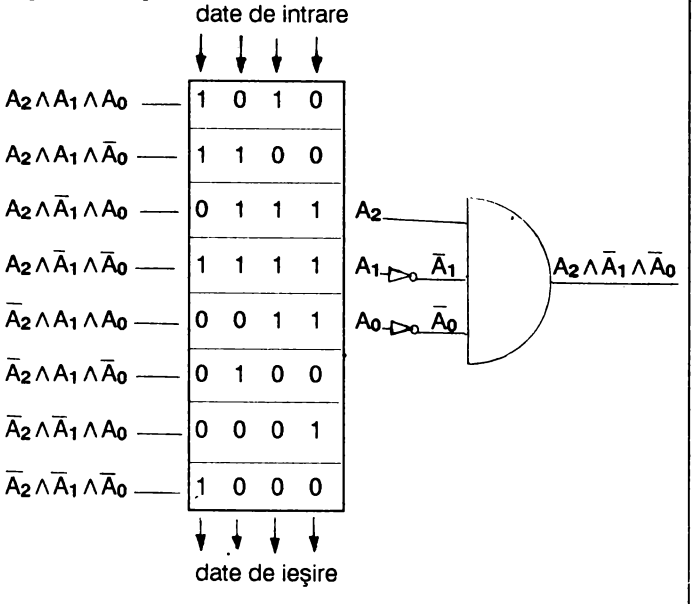


figura 1

Evident, funcţiile logice care asigură selectarea unei locaţii pot fi implementate cu ajutorul porţilor logice (ŞI, INVERSOR). În cazul memoriilor reale toată această logică este integrată pe acelaşi circuit şi poartă denumirea de decodificator. De obicei, un decodificator are n intrări şi 2^n ieşiri, fiecare ieşire fiind activată pentru o singură combinaţie de biţi de intrare. În figura 2 este prezentat un decodificator cu 3 intrări şi 8 ieşiri.

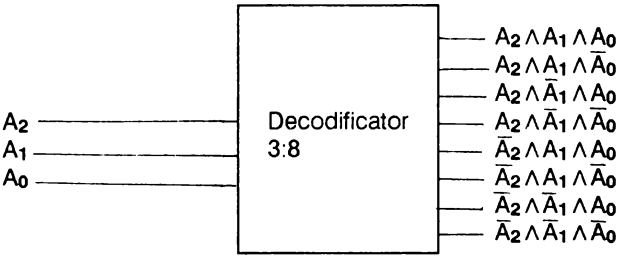


figura 2

Un alt tip de memorie întîlnită în calculatoarele clasice este memoria adresată prin conţinut numită şi *memorie asociativă*. La acest tip de memorie decodificatorul este înlocuit cu o memorie de construcţie specială care în fiecare locaţie conţine o "cheie" (figura 3). Fiecărei "chei" îi corespunde o locaţie din memoria principală în care se găseşte informaţia utilă. La aplicarea unei combinaţii de biţi la intrarea memoriei asociative, această combinaţie este comparată cu "cheile" memorate. Dacă se găseşte cheia atunci la ieşire se obţine informaţia din locaţia de memorie principală, corespunzătoare "cheii" de intrare.

Exemplu: Dacă la memoria prezentată în figura 3, se aplică "cheia" 1011, atunci la ieşire se obţine 1111.

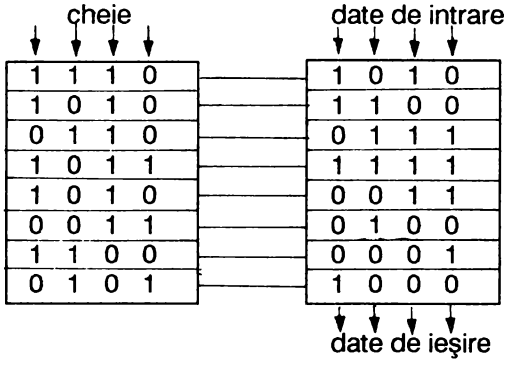


figura 3

Printr-o construcţie mai complexă este posibil să se facă căutarea după o "cheie" incompletă. Comparaţia se consideră satisfăcută pentru acea "cheie" care este cea mai "apropiată" de "cheia" de intrare (aici noţiunea de "apropiat" este vagă. Ea va fi clarificată mai tîrziu în acest articol prin definirea distanţei Hamming.)

Conceptul de memorie asociativă se regăseşte într-un sens mai larg în cazul *memoriei biologice*. Pe de altă parte, între modelele de memorie întîlnite în calculatoarele clasice (prezentate succint mai sus) şi *memoria biologică* există diferenţe fundamentale. Astfel, în timp ce memoriile clasice au un caracter de localitate (fiecare informaţie este conţinută într-o locaţie), în cazul reţelelor neuronale informaţia se regăseşte disipată în întreaga structură. Memorarea unei informaţii se reflectă în modificarea ponderilor asociate conexiunilor dintre noduri.

Pe de altă parte, în timp ce în calculatoarele clasice memoriile sînt digitale (ca de altfel toate circuitele componente) în cazul memoriilor biologice funcţionarea este analogică.

Dacă reţeaua neuronală este privită ca o "cutie neagră" atunci ea poate fi asimilată cu o memorie asocia-

tivă care pentru fiecare şablon ("pattern") de intrare oferă la ieşire o informaţie completă (eventual restaurată) asupra şablonului de intrare.

Exemplu: O persoană cunoscută poate fi identificată pe baza unor informaţii incomplete (aceste informaţii au rolul unei "chei" ca în cazul memoriei asociative) de exemplu după ochi, siluetă, mers etc. Odată recunoscută, acestei persoane i se asociază toate informaţiile deţinute despre aceasta.

2. Reţeaua HOPFIELD

2.1. Descriere

Acest tip de reţea a fost conceput de către J.J. Hopfield şi în cursul timpului a cunoscut o serie de îmbunătăţiri. În acest articol se descrie modelul iniţial propus de Hopfield, care funcţioneaza ca memorie asociativă.

Reţeaua, prezentată în *figura* 4, este compusă dintr-un singur nivel cu N noduri (neuroni). Fiecare neuron comunică cu exteriorul prin intermediul unei conexiuni de intrare şi al unei conexiuni de ieşire. În plus, ieşirea unui nod este conectată la intrarea celorlalte noduri (obs: ieşirea unui nod nu este legată la intrarea aceluiaşi nod). O astfel de reţea, la care ieşirile sînt legate direct sau indirect la intrări este o reţea cu conexiune inversă (feed-back).

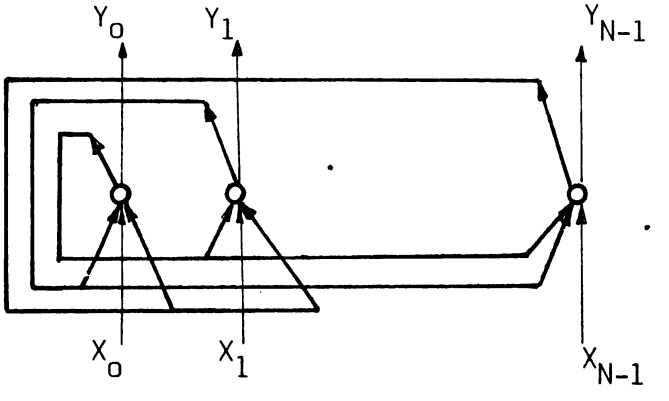


figura 4

În general acest tip de reţea este folosit cu intrări şi ieşiri binare. Aceste intrări pot fi 0, 1 sau -1, 1. În cazul de faţă va fi preferat cel de-al doilea grup de valori.

Exemplu: Intrările unei reţele de tip Hopfield pot fi intensităţile asociate pixelilor pe un ecran monocrom cu două nivele de gri (alb, negru). Astfel, pentru nivelul de "negru" intrarea corespunzătoare este −1 iar pentru nivelul de "alb" 1.

Acest tip de reţeà poate fi utilizat, dacă este cazul, şi pentru intrări nebinare. În această situaţie intrările se pot discretiza în prealabil.

Exemplu: Pentru un monitor monocrom cu 256 de nivele de gri valoarea intensităţii în fiecare punct poate fi convertită în 8 biţi ($2^8 = 256$). În acest mod pentru fiecare pixel corespund 8 intrări.

Fiecare nod din reţeaua Hopfield calculează ieşirea prin intermediul unei funcţii prag. Reamintim modul de definire al acestei funcţii:

$$(1)\quad F_h(x) = \begin{cases} -1, & \text{pentru } x < 0 \\ 0, & \text{pentru } x = 0 \\ 1, & \text{pentru } x > 1 \end{cases}$$

2.2. Funcţionare

Ne propunem să memorăm cu ajutorul unei reţele neuronale de tip Hopfield M şabloane cu o lungime de N biţi fiecare, apoi la prezentarea unui şablon necunoscut să obţinem la ieşire cel mai "apropiat" şablon faţa de cel prezentat. Pentru a memora M şabloane într-o astfel de reţea este suficient să se iniţializeze ponderile asociate conexiunilor după cum urmează:

$$(2)\quad T_{ij} = \begin{cases} \sum_{s=0}^{M-1} x_i(s) * x_j(s), & i <> j \\ 0, & i = j,\ 0<=i, j<=N-1;\ 0<=s<=M; \end{cases}$$

unde:

– T_{ij} reprezintă valoarea ponderii asociată conexiunii care leagă ieşirea nodului i cu intrarea nodului j. Pentru i = j, $T_{ij} = 0$, deci nu există conexiune între ieşirea şi intrarea aceluiaşi nod.

– $x_i(s)$ reprezintă valoarea corespunzătoare intrării i la prezentarea şablonului s. Evident $x_i(s)$ va avea valoarea 1 sau -1.

Aşa cum s-a subliniat, informaţia memorată într-o reţea neuronală nu are un caracter de localitate ci ea se distribuie în întreaga structură. În cazul de faţa memorarea unui nou şablon duce la modificarea ponderilor asociate tuturor conexiunilor (ecuaţia 2).

Odată memorate cele M şabloane, reţeaua poate fi interogată în cazul unui şablon necunoscut anterior. Algoritmul care calculează răspunsul reţelei Hopfield comportă 2 paşi:

Pasul 1. Iniţializarea ieşirilor nodurilor cu valorile prezentate la intrările reţelei:

$$(3)\quad Y_i(0) = x_i,\ 0<=i<=N-1$$

unde $Y_i(0)$ este ieşirea nodului i la momentul de timp iniţial, iar x_i este valoarea corespunzătoare intrării nodului i pentru şablonul necunoscut.

Pasul 2. Calcularea iterativă a valorilor ieşirilor reţelei pînă la convergentă:

$$(4)\quad Y_i(t+1) = F_h\left(\sum_{j=0}^{N-1} T_{ji} * Y_j(t)\right),\quad 0<=i<=N-1$$

unde $Y_i(t)$ reprezintă evident ieşirea nodului i la momentul de timp *t*. Procesul specificat in ecuaţia 4 se repetă pînă în momentul în care toate ieşirile reţelei rămîn neschimbate, adică $Y_i(t+1) = Y_i(t)$ cu $0<=i<=N-1$. Hopfield a demonstrat că reţeaua este convergentă cînd ponderile sînt simetrice ($T_{ij} = T_{ji}$, in cazul nostru T_{ij}

$= x_i(s)*x_j(s) = x_j(s)*x_i(s) = T_{ji}$) și cînd ieșirile sînt actualizate în mod asincron folosind ecuațiile prezentate mai sus.

Ieșirea, astfel obținută, reprezintă exemplarul care "aproximează" cel mai bine șablonul de intrare. Această ieșire poate fi interpretată în două moduri:

– dacă se consideră că șablonul de intrare prezentat este un exemplar incomplet din cele memorate atunci, șablonul de ieșire reprezintă restaurarea șablonului de intrare.

Exemplu: Dacă în rețea se memorează imaginea cifrelor de pe un ecran atunci la prezentarea unei imagini neclare (perturbate) a unei cifre la ieșirea rețelei se obține imaginea clară (restaurată) a cifrei prezentate la intrare.

– se poate considera că, fiecare exemplar memorat reprezintă o clasă și în acest caz la prezentarea unui exemplar necunoscut la intrare, rețeaua decide cărei clase ii aparține acesta.

Rețeaua Hopfield are două limitări majore cînd este folosită ca memorie adresată prin conținut:

– numărul de șabloane memorate este foarte limitat. Dacă sînt memorate prea multe pattern-uri atunci în procesul de recunoaștere rețeaua poate converge către un șablon diferit de toate șabloanele memorate. Limita numărului de șabloane memorate pentru care se poate asigura o funcționare optimă a rețelei este de 0.15*N, und N este numărul de noduri. Astfel pentru a clasifica în 10 clase sînt necesare aproximativ 70 de noduri și 5000 de conexiuni.

– un șablon memorat este instabil dacă are multi biți în comun cu un altul. Această deficiență poate fi înlăturată folosind anumite procedee de ortogonalizare pentru "pattern"- urile de intrare.

3. Rețeaua HAMMING

3.1. Descriere

Această rețea are ca principiu de funcționare calculul distanței Hamming. Distanța Hamming măsoară "gradul de diferență" a două șabloane și este egală cu numărul de biți diferiți între cele două șabloane. Valoarea maximă a distanței Hamming este egală cu numărul de biți ai șablonului iar valoarea minimă este evident 0 (în cazul în care cele două șabloane comparate sînt identice).

Exemplu: Pentru două "pattern"-uri S1 = 1010 și S2 = 1011 distanța Hamming notată prin d(S1, S2) are valoare 1 deaorece S1 diferă de S2 prin al patrulea bit. Dacă luam S3 = 1100, atunci vom avea d(S1, S3) = 2 si d(S2, S3) = 3.

Rețeaua Hamming implementează, după cum vom arata, calculul acestei distanțe. Această rețea (figura 5) își propune să clasifice "pattern"-uri cu o lungime de N biți în M clase. În acest sop rețeaua Hamming are N intrări, două nivele a cîte M noduri fiecare și M ieșiri (o ieșire corespunzătoare ficărei clase). Primul nivel al rețelei asigură implementarea calculului distanței Hamming iar nivelul superior selectează clasa pentru care distanța Hamming dintre aceasta și șablonul de intrare este minimă.

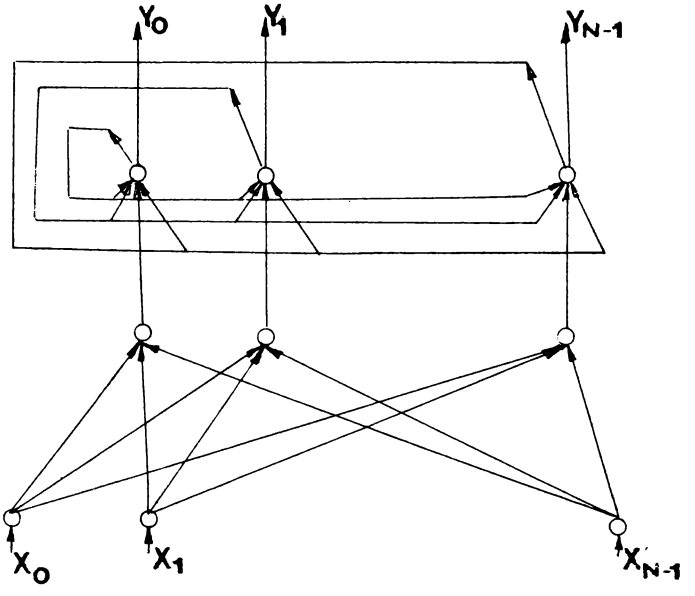


figura 5

Fiecare nod din acest tip de rețea calculează ieșirea după o funcție de tip rampă:

$$(5)\quad F_t(x) = \begin{cases} 0, & \text{pentru } x<0 \\ x, & \text{pentru } 0<=x<=\alpha \\ \alpha, & \text{pentru } \alpha>1 \end{cases}$$

3.2. Funcționare

Procesul de învățare constă (ca și în cazul rețelei Hopfield) în inițializarea ponderilor asociate conexiunilor după ecuațiile:

$$W_{ij} = x_i(j)/2,\ \theta_j = -N/2,\ 0<=i<=N-1;\quad 0<=j<=M-1$$

$$(6)\quad T_{kl} = \begin{cases} 1, & k = l \\ -\varepsilon, & k<>l,\ \varepsilon<1/M \end{cases}$$

$$0<=k, l<=M-1$$

unde:

– W_{ij} reprezintă ponderea asociată conexiunii între intrarea i si nodul j din nivelul intermediar.

– $x_i(j)$ reprezintă valoarea intrării i corespunzătoare șablonului j.

– j reprezintă valoarea pragului asociat neuronului din nivelul j.

– T_{kl} reprezintă ponderea asociată conexiunii între ieșirea nodului k și intrarea nodului l din nivelul superior. Se observă că în acest caz există o conexiune între ieșirea și intrarea aceluiași nod (spre deosebire de rețeaua Hopfield) cu o pondere unitară.

– ε reprezintă un factor de inhibiție asociat conexiunilor între două noduri diferite aparținind nivelului superior.

Exemplu: Considerăm nivelul inferior al unei rețele

Hamming cu 4 intrări şi 2 noduri (*figura* 6). Această reţea este capabilă să clasifice şabloane cu o lungime de 4 biţi în 2 clase. Fie "pattern"-urile corespunzătoare celor două clase S1=1010, respectiv S2=1011. Ca şi în cazul reţelei Hopfield se aleg valorile binare de 1 şi -1 pentru intrările reţelei. Astfel, pentru şablonul S1 se va aplica la intrare combinaţia 1, -1, 1, - 1, iar pentru S2 combinaţia 1, -1, 1, 1. În *figura* 6 sînt prezentate valorile ponderilor asociate conexiunilor din reţeaua inferioară.

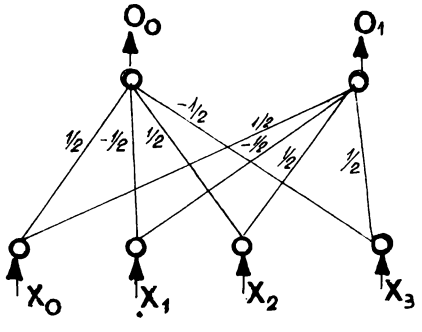


figura 6

Operaţia de clasificare a unui şablon se desfăşoara şi în acest caz în doi paşi.

Pasul 1.Iniţializarea ieşirilor nodurilor în funcţie de valorile prezentate la intrările reţelei:

$$Y_j(0) = F_t\left(\sum_{i=0}^{N-1} W_{ij} * x_j - \theta_j\right), \quad 0 <= j <= M-1 \tag{7}$$

unde:

– $Y_j(0)$ este ieşirea nodului j din reţeaua superioară la momentul iniţial.

De notat că ieşirea nodului j aparţinînd nivelului inferior este egală cu diferenţa dintre numărul de intrări (N) şi distanţa Hamming calculată între şablonul reprezentativ al clasei j şi şablonul aplicat la intrare. Pentru aceasta este necesar ca funcţia Ft să nu se satureze pe domeniul valorilor de intrare. Aceasta se întîmplă dacă coeficientul de saturaţie este $\alpha >= N$.

Exemplu: Fie şablonul de intrare S3 = 1100. Valorile aplicate la intrare vor fi 1, 1, -1, -1. Folosind formula de însumare ponderată a intrărilor în nodurile 0 si 1, obţinem:

$$I_0 = \sum_{i=0}^{3} (W_{i0} * x_i - \theta_i) = 0 + 4/2 = 2 = 4 - 2 = N - d(S1, S3);$$

$$I_1 = \sum_{i=0}^{3} (W_{i1} * x_i - \theta_i) = -1 + 4/2 = 1 = 4 - 3 = N - d(S2, S3);$$

Aplicînd funcţia de tip rampă (ecuaţia 5) şi ţinînd seama de condiţia de ne-saturaţie obţinem ieşirile nodurilor studiate:

$O_0 = F_t(I_0) = I_0$, respectiv $O_1 = F_t(I_1) = I_1$

Deci, la ieşirea nivelului inferior într-o reţea Hamming se obţine o valoare maximă la nodul corespunzător clasei pentru care distanţa Hamming dintre şablonul care reprezintă această clasă şi şablonul de la intrare este minimă. Deoarece distanţa Hamming este pozitivă şi mai mică ca N, domeniul de variaţie a ieşirilor nodurilor de pe nivelul inferior este cuprins între 0 si N.

La pasul 1 valoarea de ieşire a nodurilor de pe nivelul superior sînt iniţializate cu valorile de ieşire ale nodurilor de pe nivelul inferior.

Pasul 2. Calcularea iterativă a valorilor ieşirilor reţelei pînă la convergenţă:

$$Y_i(t+1) = F_t\left(Y_i(t) - \varepsilon \sum_{i<>k} Y_k(t)\right), \quad 0<=i, k<=M-1 \tag{8}$$

unde $Y_i(t)$ reprezintă ieşirea nodului i la momentul de timp t. În această etapă şablonul de intrare este înlăturat iteraţia desfăşurîndu-se în reţeaua superioară cu ieşirile iniţializáte la pasul 2. De fapt,ecuatia 6 reprezintă calculul ieşirilor nodurilor din reţeaua superioară la momentul t + 1 pe baza sumei ponderate a intrărilor care au valorile de la ieşirea nodurilor de la momentul t:

$$Y_i(t+1) = F_t\left(T_{ii} * Y_i(t) + \sum_{i<>k} T_{ki} * Y_k(t)\right) = F_t\left(Y_i(t) - \sum_{i<>k} \varepsilon * Y_k(i)\right) = F_t\left(Y_i(t) - \varepsilon \sum_{i<>k} Y_k(t)\right) \tag{9}$$

Iteraţia se desfăşoară pînă în momentul în care o singură ieşire rămîne pozitivă celelalte fiind nule. S-a demonstrat ca acest tip de reţea este întotdeauna convergent pentru 1/M. În plus în urma experimentelor a rezultat o convergenţă destul de rapidă. Astfel pentru o reţea cu 1000 de intrări şi 100 de noduri sînt necesare sub 10 iteraţii pentru a clasifica un şablon necunoscut.

Reţeaua Hamming are o serie de avantaje comparate cu reţeaua Hopfield cînd aceasta este folosită în operaţii de clasificare. Acest tip de reţea necesită mai puţine conexiuni decît o reţea de tip Hopfield similară. Astfel pentru a clasifica şabloane cu 100 de biţi în 10 clase o reţea de tip Hamming necesită 1100 de conexiuni în timp ce o reţea de tip Hopfield necesită aproximativ 10000 de conexiuni. Pe de altă parte reţeaua Hopfield obţine rezultate în clasificare mai bune decît o reţea Hamming pentru "pattern"-uri ai căror biţi au fost modficaţi aleator.

BIBLIOGRAFIE

J.J. Hopfield, *"Neural networks and physical systems with emergent collective computational abilities"*, Proceeding National Academic Science, 1982.

A. F. Murray and **A. V. W. Smith**, *"Asynchronous VLSI Neural Networks Using Pulse-Stream Arithmetic"*, IEEE Journal of Solid-States Circuits, 1988

R. P. Lippmann, *"An Introduction to Computing with Neural Nets"*, IEEE ASSP MAGAZINE, 1987

LIMBAJUL C (VI)

Conf.dr.ing. Irina Athanasiu

Limbajul **C** permite utilizarea unor expresii puţin mai ciudate decît alte limbaje de programare. Una dintre situaţiile mai "dificile" de care se loveşte un începător, mai ales dacă are o experienţă semnificativă în alte limbaje de programare, constă din modul de interpretare (evaluare) pentru o expresie de forma:

```
(float) a / b + c-- - ++d
```

Înţelegerea unei astfel de notaţii presupune atît înţelegerea tipurilor de date cu care lucrează limbajul **C** cît şi a modului de acţiune a operatorilor care apar în instrucţiuni şi a modului în care limbajul **C** tratează priorităţile operatorilor si proprietăţile de asociativitate ale acestora.

În limbajul **C,** ca în orice limbaj de programare, fiecare operator are o prioritate pe baza căreia se fixează ordinea relativă de evaluare a elementelor unei expresii. De asemenea, pentru operatorii cu aceeaşi prioritate este definită o regulă de asociativitate. Astfel, pentru expresia:

```
a + b * c
```

se evaluează întîi produsul şi apoi suma deoarece operatorul de înmultire are o prioritate mai mare decît operatorul de adunare. Pentru a schimba ordinea de evaluare se pot utiliza paranteze. Adică, pentru expresia:

```
(a + b) * c
```

se va evalua întîi suma şi apoi produsul. Pentru expresia :

```
a + b * c + d
```

ordinea de evaluare este aceeaşi cu cea din expresia:

```
(a + (b * c)) + d
```

deoarece pentru operaţia de adunare se consideră regula de asociativitate stînga.Pînă aici textul anterior se aplică majorităţii limbajelor de programare. În limbajul **C** lucrurile sint mai delicate. Astfel, conform "bibliei **C**" (*The C programming language,* Brien W. Kernigham, Dennis M. Ritchie), într-o expresie de forma:

```
f() operator g()
```

nu este precizată ordinea în care sînt evaluate apelurile funcţiilor *f* si *g*. Datorită acestui fapt efectele laterale pe care poate să le producă unul dintre aceste apeluri poi să influenţeze datele asupra cărora operează cealaltă funcţie. Ceea ce este mai grav, datorită acestei libertăţi în contexte diferite aceeaşi expresie se poate evalua în mod diferit. Să considerăm de exemplu următoarele programe:

```
/*
   eval1.c
   program pentru studiul evaluarii expresiilor
*/

#include <stdio.h>
static char a[] = "AB";
static int i;
void main()
{
 for(i = 0; a[i++] = a[i] ,i < 1 ; )
   ;
 printf("\n%s",a );
}
```

şi

```
/*
   eval2.c
   program pentru studiul evaluarii expresiilor
*/
void main()
{
 char a[] = "AB";
 int i;
 for(i = 0; a[i++] = a[i] ,i < 1 ; )
   ;
 printf("\n%s",a );
}
```

compilînd şi executînd aceste programe utilizînd compilatorul **TURBO C++** se vor obţine efecte diferite pentru cele două programe deşi ar trebui să fie echivalente ca efect. Ceea ce conduce la diferenţe este modul în care se execută instrucţiunea:

```
a[i++] := a[i]
```

Dacă se urmareşte codul generat de către compilator pentru cele două exemple se va constata că în primul caz secvenţa de operaţii corespunzătoare execuţiei instrucţiunii este:

```
determină adresa elementului a[i]
salvează în registrul AL valoarea a[i]
salvează în registrul BX adresa elementului a[i]
actualizează i
memorează valoarea din registrul AL la adresa
                indicată de registrul BX
```

În al doilea caz se va executa secvenţa de operaţii:

```
salvează în registrul BX adresa elementului a[i]
actualizează i
salvează în registrul AL valoarea a[i]
/* i a fost deja modificat */
memorează valoarea din registrul AL la adresa
        indicată de registrul BX
```

Se poate considera că am găsit o eroare în compilatorul **TURBO C++** ?. Încă o dată conform "bibliei **C**", nu ! deoarece operatorul de atribuire (=) este un operator binar ca oricare altul şi nimeni nu stabileşte ordinea de evaluare a operanzilor săi.

Revenind la discuţia iniţială în tabelul 1 sînt prezentaţi operatorii limbajului **C** în ordinea descrescătoare a priorităţii şi se specifică pentru fiecare operator regula de asociativitate.

operatori	unar/binar	asociativitate
() [] −> .	unar .	stinga
! ~ ++ −− − (tip) * & sizeof	unar	dreapta
* / %	binar	stinga
+ −	binar	stinga
<< >>	binar	stinga
< <= > >=	binar	stinga
== !=	binar	stinga
&	binar	stinga
^	binar	stinga
|	binar	stinga
&&	binar	stinga
||	binar	stinga
? :	binar	dreapta
= += −= , etc	binar	dreapta
,	binar	stinga

Tabel 1

Un test util de cunoaştere a limbajului **C** constă acum în parcurgerea tabelului şi specificarea semnificaţiei fiecărui operator conţinut în tabel. De exemplu, ce inseamna operatorul ^ , dar operatorul , (care este diferit de caracterul virgulă utilizat pentru separarea elementelor în cadrul listelor) ?.

Dacă executăm următorul program:

```
/*
    print1.c
   program pentru ilustrarea proprietatilor
   de asociativitate a operatorilor
*/
#include <stdio.h>
void main()
{
  int a = 1, b = 2, c = 3, d = 4;

  a += b += c += d;
  printf("\na = %d ", a);
  printf("b = %d ", b);
  printf("c = %d ", c);
  printf("d = %d \n", d);

  a = 11 % 7 % 5;
  b = 11 % (7 % 5);

  printf("\na = %d ", a);
  printf("b = %d ", b);

}
```

Ca efect, se va afişa textul :

```
a = 10 b = 9 c = 7 d = 4
a = 4 b = 1
```

Analizînd rezultatele obţinute se verifică faptul că regula de evaluare conform proprietăţilor de asociativitate este pentru operatorul += de la dreapta la stînga în timp ce pentru operatorul % (restul împărţirii întregi) este de la stînga la dreapta. Deci instrucţiunea:

```
a += b += c += d
```

este echivalentă cu secvenţa de instrucţiuni:

```
c = c + d;
b = b + c;
a = a + b;
```

În timp ce expresia:

```
11 % 7 % 5
```

este echivalentă cu expresia:

```
(11 % 7) % 5
```

care are o valoare diferită de expresia:

```
11 % (7 % 5)
```

Să analizăm acum programul **priorit2.c :**

```
/*
   priorit2.c
   program pentru ilustrarea prioritatilor
   operatorilor in limbajul C
*/
#include <stdio.h>
void main()                                    ⇒
```

```
{
 int a = 2, b = 3, c = 6, d = 1, s, t;
 s = a * b == c * d;
 t = a * (b == c) * d;
 printf("\n s = %i  t = %i", s, t);
}
```

Ca rezultat se va afişa:

s = 1 t = 0

deoarece expresia:

a * b == c * d

este echivalentă cu:

(a * b) == (c * d)

deoarece operatorul de înmulţire este prioritar faţa de operatorul de comparaţie ==.

Să considerăm şi programul urmator :

```
/*
    print3.c
    program pentru ilustarea proprietatilor
    de asociativitate a operatorilor
*/
#include <stdio.h>
void main()
{
 printf ("\n 3.5 / 2 = %5f", 3.5 / 2);
 printf ("\n (int) 3.5 / 2 = %5f", (int) 3.5 / 2);
 printf ("\n (int) 3.5 / 2 = %5d", (int) 3.5 / 2);
 printf ("\n (float) (int) 3.5 / 2 = %5f",
            (float) (int) 3.5 / 2);
}
```

Executînd acest program, utilizînd compilatorul **TURBO C++**, se obţine următoarea secvenţă de afişări:

```
3.5 / 2 = 1.750000
(int) 3.5 / 2 = 0.000000
(int) 3.5 / 2 =     1
(float) (int) 3.5 / 2 = 1.500000
```

Prima linie afişează un rezultat de tip float în urma împărţirii dintre o valoare de tip float (3.5) şi o valoare de tip int (2). A doua linie afişează un rezultat greşit deoarece s-a încercat afişarea unui rezultat de tip int utilizînd un format de tip float. Rezultatul de tip int se obţine prin împărţirea a două valori de tip int. Prima valoare se obţine prin conversia valorii de tip float (3.5) la o valoare de tip int prin aplicarea operatorului de conversie (int). Ultima linie demonstrează faptul că asociativitatea operatorilor de conversie este de dreapta. Adică întîi se face conversia la tipul int pentru operatorul (int) şi apoi conversia la tipul float pentru operatorul float.

Pentru a ilustra un alt aspect al modului de evaluare a expresiilor să considerăm programul **print4.c** :

```
        /*
        print4.c
        program pentru ilustarea proprietatilor
        de asociativitate a operatorilor
        */
        #include <stdio.h>
        int a, b;
        void main()
        {
/* 1 */ a = 1;
/* 2 */ b = (a = 2, a++, a + 5);
/* 3 */ printf("\n b = %i", b);
/* 4 */ a = 1;
/* 5 */ printf("\n b = %i", (a = 2, a++, a + 5));
/* 6 */ b = 1;
/* 7 */ printf("\n b++ = %i b = %i", b++, b);
/* 8 */ b = 1;
/* 9 */ printf("\n b = %i b++ = %i", b, b++);
/* 10 */ b = 1;
/* 11 */ printf("\n ++b = %i b = %i", ++b, b);
/* 12 */ b = 1;
/* 13 */ printf("\n b = %i ++b = %i", b, ++b);
        }
```

Ca efect al execuţiei acestui program se va afişa textul :

```
b = 8
b = 8
b++ = 1 b = 1
b = 2 b++ = 1
++b = 2 b = 1
b = 2 ++b = 2
```

Se observă că în lista de instrucţiuni separate de operatorul , se utilizează regula de asociativitate stînga, adică instrucţiunile se vor executa în ordine de la stînga la dreapta, rezultatul obţinut fiind reprezentat de valoarea ultimului rezultat.

Să analizăm puţin însă ce se întîmplă in instrucţiunile de la 6 la 13. Comparînd efectul instrucţiunilor din liniile 7, 9, 11, 13 se observă o altă particularitate a limbajului C, referitoare la ordinea în care se evaluează argumentele funcţiilor. Şi anume se observă că evaluarea argumentelor (şi implicit introducerea lor în stivă) se face de la dreapta la stînga. Corespunzător, pentru linia 7 de exemplu, actualizarea valorii variabilei b se face dupa introducerea in stivă a valorilor celor doua argumente în timp ce în linia 9 actualizarea variabilei b se face dupa ce s-a introdus în stivă valoarea ultimului argument şi înainte de introducerea în stivă a primului argument.

Putem acum să indicăm modul în care se evaluează expresia complicată pe care am prezentat-o la început, rescriind-o într-o formă parantezată:

((((float) a)/b) + (c--)) - (++d) ■

EDITOARE ŞI FONTURI (V)

Ion Paraschiv şi Tiberiu Spircu

Întrerupem pentru moment prezentarea editorului **MatTex** pentru a insera cîteva informaţii ce se doresc a fi adresate utilizatorilor limbajului **Turbo C**. După cum se ştie, firma de soft Borland a pus la dispoziţia lor o bibliotecă grafică destul de bogată, anexă a programului,

`bgidemo.c`

(a se vedea PC-Magazin, nr.2, pag.60-65, la care ne vom referi curent prin **PCM-2**). Această bibliotecă, pe lîngă rutinele grafice necesare, conţine şi un număr de 4 fişiere-font:

`gothic.chr, litt.chr, triple,chr.`

Vom încerca să prezentăm modul de folosire, precum şi structura internă a acestor fonturi.

În programul sursă `bgidemo.c`. întîlnim (vezi **PCM-2**) directiva:

```
# include  graphics.h
```

Ce vom găsi important în fişierul - anexă `graphics.h`?

În primul rînd, o listă a denumirilor de identificare a fonturilor existente:

```
enum font - names {
DEFAULT-FONT =0 /* FONTUL STANDARD
                                8X8*/
TRIPLEX-FONT =1
SMALL-FONT =2
SANS-SERIF-FONT = 3
GOTHIC-FONT = 4
};
```

Urmează definirea unei constante:

```
# define USER-CHAR-SIZE )
```

Apoi, o structură ce caracterizează tipul textului, prin fontul folosit, direcţia de scriere, mărimea caracterului:

```
struct textsettingstype {
int font;
int direction;
int charsize;
int horiz;
int vert;
} ;
```

În fine, lista funcţiilor ce "prelucrează" texte sau fonturi:

```
  void far _Cdecl gettextsettings (struct
textsettingstype, far * texttypesinfo);
  int far _Cdecl installuserfont (char
far * name);
  void far _Cdecl settextstyle (int font,
int direction, int charsize);
  void far _Cdecl setusercharsize (int
multx, int divx, int multy, int divy);
  int _Cdecl register bgifont (void
(*font) (void));
```

(Aceste funcţii au rolul de a "agăţa" fonturile; ele nu pot fi chemate direct):

```
  void  _Cdecl triplex_font (void);
  void  _Cdecl small_font (void);
  void  _Cdecl sansserif_font (void);
  void  _Cdecl gothic_font (void);
```

Să revenim la programul `bgidemo.c`. Vom întîlni în el, în continuare, o listă de denumiri ale fonturilor existente, denumiri ce sînt folosite doar pentru "demonstraţie" (vezi **PCM-2**, funcţia `Main Window`):

```
  char * Fonts [ ] ={
  "DefaultFont", "Triplex Font", "Small
Font", "SansSerifFont", "Gothic Font"
   } ;
```

Apoi, în cadrul prototipurilor, vom întîlni funcţiile:

```
  void TextDump (void);
  void TextDemo (void);
```

după care, se intră în corpul funcţiei principale. Aceasta va avea forma (a se compara cu **PCM-2**):

```
  int main ( )
  {
  Initial ( );
  ....
  TextDump ( );
  TextDems ( );
  ...
  closegraph ( );
  return (0);
  }
```

Dintre cele două funcţii prototip noi, vom prezenta doar pe prima. Ea are ca efect afişarea tuturor caracterelor "tipăribile" din fiecare font disponibil.

```
  void TextDump ( )
  {
  static int CGASizes [ ] = {
  1, 3,7, 3,3 ;};
  static int NormSizes [ ] = {
  1, 4, 7,4, 4 ;};
```

```
  char buffer [ 80 ];
  int font, ch, wwidth, lwidth, size;
  struct viewporttype vp;
  for (font=0; font ; ++font)
                              /* pt.fişier.font */
  sprintf (buffer, "% s Caracterele fon-
tului:, Fonts [font]);
  Main Window (buffer);
                    /* afişează numele fontului */
  getviewsettings (&vp);
  settextjustify (LEFT_TEXT, TOP_TEXT);
  moreto (2,3);
  buffer [1] = '\0';
  wwidth = vp.right-vp. left;
                           /* lăţimea ferestrei */
  lwidth = textwidth ("H");
                    /* lăţimea medie a semnului*/
  if (font == DEFAULT_FONT){
    changetextstyle (font, HORIZ_DIR,1);
    ch = 0;
    while (ch  256){
                    /* pentru fiecare caracter */
       buffer[0] = ch;
       outtext (buffer);
                     /* este trimis pe ecran */
       if ((getx() + lwidth)
          moveto (2, gety () + textheight
          ("H")+3);
          ++ ch:     /* următorul caracter */
    }
  }
  else {
     size = (MaxY Â)? CGASizes[font]
     : Norm Sizes [font] ;
   changetextstyle(font,HORIZ_DIR, size);
     ch= '!';        /* începe cu primul tipăribil */
    while(ch<27){ /* pentru fiecare caracter tipăribil */
         buffer[0] = ch;
         onttext (buffer);
                              /* trimis pe ecran */
         if (ewidth+getx ()  wwidth)
              moveto (2, gety () +tex-
theight
           ("H") +3);
           ++ch;          /* următorul caracter */
     }
  }
  Pause ();
      }
  }
```

Un fişier-font este format din două părţi: prima informativă, urmată de cea efectivă. (Vom exemplifica prin fontul `gothic.chr`.)

Partea informativă ocupă, de regulă, primii 128 bytes şi este formată din:

— semnalul de recunoaştere (8 bytes):

`50 48 08 08 42 47 49 20`

— mesaj explicativ (denumirea, versiunea, data creării, autorul), terminat cu:

`OD OA OO 1A`

— pointer spre începutul părţii efective (2 bytes):

`80 00`

— denumirea codificată a fontului (4 bytes), de exemplu

`47 4F 54 48 (GOTH)`

— lungimea părţii efective (2 bytes, interpretaţi ca număr întreg) de exemplu:

`FO 20`

Partea efectivă a fontului este formată din patru zone. Prima zonă conţine caracteristici generale, anume:

— un byte semnal

`2B`

— un byte special, de exemplu

`6 A`

— 2 bytes nuli:

`00 00`

— 1 byte ce conţine LOWCHR, în cazul nostru aceasta este 21:

`15`

(amintim că HIGCHR = 127)

— 2 bytes ce conţin un pointer spre începutul zonei a 4-a, în cazul nostru:

`4E 01`

— 1 byte nul:

— 1 byte ce conţine ASCMAX, în cazul nostru:

`15`

— 1 byte nul:

— 1 byte ce conţine - DESMAX, în cazul nostru acesta este -7:

`F9`

— 5 bytes nuli.

Zona a doua este o zonă de pointeri. Fiecărui caracter LOWCHR îi corespund cîte 2 bytes, în care se află pointerul spre descrierea vectorială a caracterului. Astfel, în exemplul nostru caracterului 33 (!) îi corespunde pointerul

`38 00`

deci descrierea semnului ! începe la adresa

0080h + 014 Eh + 0038h

A treia zonă este rezervată lăţimilor efective. Anume, fiecărui caracter ≥ LOWCHR îi corespunde cîte 1 byte, în care se află ACTWID-ul caracterului. Deci această zonă este formată din HIGCHR-LOWCHR +1 bytes.

În fine, zona a patra conţine descrierile vectoriale ale caracterelor. O astfel de descriere este formată din secvenţe de grupe, fiecare grupă (de cîte 2 bytes) corespunzînd unui nod (punct). De la punctul curent ne putem deplasa spre noul punct (avînd coordonatele x, y) cu "peniţa jos", în care caz acestui nou punct îi corespunde grupa:

80 + x 80 + y

respectiv cu "peniţa sus" în care caz grupa este:

80 + x y

Fiecare descriere se termină cu 2 bytes nuli

Iată, de exemplu, (pe 26 noduri = 52 + 2 bytes) descrierea semnului ! în fontul GOTHIC:

Utilitar

Pachetul Vcache 5 al firmei Golden Bow Systems optimizează viteza unui PC pe care rulează sistemul de operare MS-DOS pentru afişare +300%; pentru citirea discurilor, cu limita a 12 discuri sau 12 partiţii +800%; pentru execuţia programelor +300%. Se înlocuieşte driverul Smartdriv.Sys. Ocupă în modul rezident 9 ko.

Limbaje

Borland lansează, la un preţ foarte mic, versiunea 6, a limbajului orientat pe obiect Turbo Pascal. Produsul se distinge printr-un mediu de dezvoltare integrat care oferă programatorului o productivitate sporit datorită ferestrelor multiple, mouse- ului ca şi unui editor multifişier îmbunătăţit. Produsul conţine şi un asamblor integrat care oferă accesul complet la simbolii Pascal. Versiunea profesională completă conţine: compilator Turbo Drive, Turbo Debugger, Turbo Profiler, Turbo Assembler.

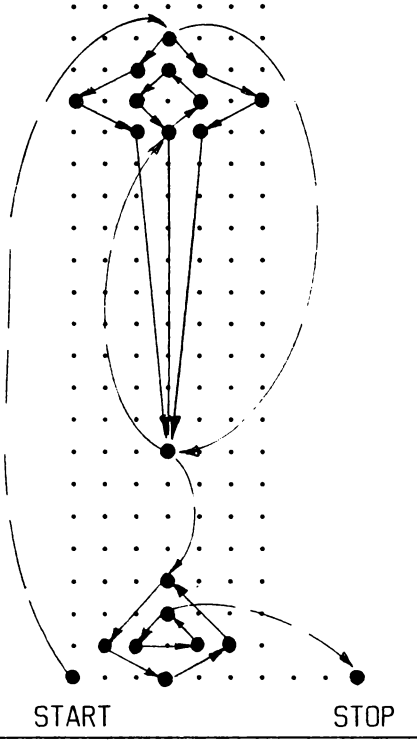
Calcul

Pachetul Matlab permite efectuarea tuturor calculelor matematice sub formă literală şi înlesneşte punerea în pagină pentru tipărirea adecvată. Datorită unui număr de biblioteci disponibile, Matlab poate fi folosit în domenii particulare cum ar fi calculul spectral, chimie, estimarea şi identificarea funcţiilor de transfer ale unor sisteme, filtrarea numerică. Este disponibil şi pentru MacIntosh.

Conversie de formate

Pachetul Software Bridge al societăţii Echange asigură conversia între peste 30 de procesoare de text diferite, cu recunoaşterea automată a tipului documentului sursă. Este disponibil şi pe MacIntosh. Mai mult, fiind compatibil cu pachetul Apple File Exchange se pot efectua transferuri de documente PC/MacIntosh utilizînd programele de comunicaţie disponibile. ■

83	15	82	94	80	93	82	92
83	87	83	12	84	93	83	94
82	93	83	92	83	87	83	15
84	94	86	93	84	92	83	87
83	03	81	81	83	80	85	81
83	83	83	02	82	81	84	81
83	82	89	00	00	00		



INIŢIERE ÎN PROGRAMARE - LIMBAJUL PASCAL (II)

conf.dr.ing. Irina Athanasiu
conf.dr.ing. Eugenia Kalisz

Sperăm că aţi reuşit să executaţi cu succes toate programele prezentate în episodul trecut şi chiar aţi încercat să experimentaţi idei personale. Dacă aţi făcut-o poate că v-aţi pus problema — de ce să utilizăm calculatorul pentru a afişa, chiar şi într-o formă elegantă, nişte valori cunoscute — constante. Răspunsul este simplu: din motive didactice.

În realitate însă calculatorul este utilizat pentru a prelucra date care nu sînt toate cunoscute înainte de elaborarea şi execuţia programului. Elementele care trebuie să fie cunoscute în momentul elaborării programului sînt tipul şi semnificaţia datelor şi a rezultatelor. Cu alte cuvinte este necesară o specificare riguroasă a "resurselor" accesibile şi a "obiectivului" urmărit. Numai în acest mod poate să fie elaborată rapid şi eficient "strategia" de rezolvare a problemei sub forma unui algoritm.

Variabile simple

În exemplele pe care le-am prezentat pînă acum au apărut o serie de valori numerice. Ori de cîte ori vom executa programele respective, aceste valori nu se vor schimba (atîta timp cît nu schimbăm programele), valorile respective apărînd în clar sub forma unor constante. Într-un program real avem însă nevoie şi de valori care să se poată modifica pentru ca programul să realizeze într-adevar o prelucrare. În acest scop au fost inventate variabilele.

O variabilă este reprezentată de un nume căruia îi este asociat la un moment dat o anumită valoare. Tipul acestei valori reprezintă "tipul variabilei". Într-un program Pascal trebuie declarat explicit tipul tuturor variabilelor utilizate. Operaţiile efectuate asupra variabilelor pot determina schimbarea valorilor asociate acestora, dar nu si a tipului lor.

Să considerăm următorul program:

```
program variabile_simple;
var i, j : integer;
    x : real;
    caracter : char;
    b1, b2 : boolean;
begin
    x:= 3;
    i := 2;
    j := i;
    x:= j;
    caracter := 'A';
    b1 := true;
    b2 := false;
{ afisare rezultate }
    writeln(i);
    writeln(x);
    writeln(caracter);
    writeln(b1);
    writeln(b2)
end.
```

Ca rezultat al execuţiei acestui program se obţine :

```
 2
 2.0000000000E+00
 A
 TRUE
 FALSE
```

Urmărind programul variabile_simple se observă că a apărut o componentă nouă între titlul programului şi cuvîntul begin care precede corpul programului (nu este ultima componentă posibilă în această zonă). Această componentă realizează declararea tipului variabilelor utilizate în program. Începutul său este desemnat cu ajutorul cuvîntului cheie "var". În program au fost definite mai multe variabile: două de tip *integer*, una de tip *real*, una de tip *char* şi două de tip *boolean*.

În program a fost ilustrat unul din modurile în care o variabilă poate să işi modifice valoarea şi anume prin atribuire. Instrucţiunea de atribuire are forma generală:

```
variabila := expresie
```

Prelucrarea descrisă de această intrucţiune se realizează în două etape:

- calculul valorii expresiei din membrul drept;
- asocierea valorii astfel obţinute variabilei din membrul stîng.

Modul în care în programul variabile_simple au fost afişate valorile variabilelor nu este foarte elegant deoarece interpretarea valorilor afişate nu se poate face decît urmărind textul programului. Din acest motiv este de preferat soluţia prezentată în programul **variabila_simpla1**:

```
program variabila_simpla1;
var i, j : integer;
    x : real;
    caracter : char;

begin
    x := 3;
    i := 2;
    j := i;
    x := j;
    caracter := 'A';
{ afişare rezultate }
    writeln('Rezultatele programului sint : ');
    writeln;
    write('i = ',i,', x = ', x:6:2,', caracter = ',caracter);
end.
```

Pe ecran se va obţine:

Rezultatele programului sint:

i = 2, x = 2.00, caracter = A

Ar fi bine să încercaţi să urmăriţi atent modul în care au fost utilizate virgulele şi blancurile în cadrul textelor care se afişează, astfel încît să se prezinte cît mai clar informaţiile ce compun rezultatul.

Să mai încercăm să rezolvăm o problemă concretă şi foarte simplă: efectuarea operaţiei de schimb valutar. în acest caz datele problemei sînt reprezentate de suma schimbată şi de cursul monedei respective faţă de moneda în care se face conversia. Rezultatul este reprezentat de suma ce constituie echivalentul în noua moneda. Valoarea rezultatului se calculează evident printr-o simplă înmulţire între suma schimbată şi cursul valutar. De ce tip sînt valorile acestor date? De obicei cursul valutar este o valoare de tip *real,* dar suma schimbată poate să fie o valoare de tip *real* sau de tip *întreg* (în acest ultim caz înseamnă că pentru schimb nu se admit subdiviziuni monetare şi că suma care se schimbă nu este mai mare decît cel mai mare numar întreg reprezentabil în calculator). Suma obţinută este evident de tip *real.*

Strategia de rezolvare a acestei probleme este reprezentată de următorul algoritm:

citeşte curs valutar
citeşte suma schimbată
calculează suma obţinută
afişează rezultatul

şi este realizată de către următorul program:

```
program schimb;
var
   curs_valutar, suma_schimbata, suma_obtinuta
   : real;

begin
   readln(curs_valutar);
   readln(suma_schimbata);
   suma_obtinuta := suma_schimbata * curs_valutar;
   writeln('Suma_obtinuta este ', suma_obtinuta:4:2);
end.
```

Se observă că toate cele trei variabile utilizate au fost declarate de tip *real.* Sperăm că aţi observat utilizarea caracterului "_" ca element de legatură între cuvintele care formează un nume de variabilă. Evident, în locul numelui "suma_schimbata" era posibil să utilizăm orice alt nume (de exemplu sumaschimbata sau xw3), dar alegerea făcută este motivată de dorinţa de a obţine programe care să fie cît mai uşor de citit şi înţeles.

Un alt element de noutate în acest program este reprezentat de către instrucţiunile de citire de forma:

readln(variabila)

Aceste instrucţiuni au următorul efect:

– se citeşte de la tastatură valoarea introdusă de către utilizator (introducerea valorii se încheie prin apăsarea tastei RETURN);
– valoarea citită se atribuie variabilei specificate în instrucţiune;
– cursorul afişat pe ecran este poziţionat la începutul următoarei linii.

Citirea se poate face şi fără trecere la linia urmatoare, printr-o instrucţiune:

read(variabila)

Dacă aţi lansat în execuţie programul "schimb" poate aţi sezizat faptul că sînteţi aşteptat să introduceţi datele problemei după care se va afişa rezultatul calculat pe baza celor doua valori. De exemplu dacă se doreşte calculul sumei ce se obţine pentru un curs valutar de 19.88 şi o sumă schimbată de 25 se va obţine pe ecran următoarea imagine:

```
19.88
25
Suma obtinuta este 497.00
```

În acest caz nu mai este clar care este partea scrisă de utilizator şi care este partea afişată de către calculator. Pentru aceleaşi valori, dacă utilizatorul uită ordinea în care trebuie să facă introducerea datelor, poate să aibă loc următoarea execuţie a programului **schimb**:

```
25
19.88
Suma obtinuta este 497.00
```

Se observă că în acest caz rezultatul nu este afectat (datorită proprietăţii de comutativitate a operaţiei de înmulţire). Există însă situaţii în care ordinea de introducere a valorii datelor este deosebit de importantă şi trebuie săfie respectată cu stricteţe. Pentru a realiza acest deziderat este bine ca programul să indice utilizatorului care este informaţia aşteptată pentru fiecare citire în parte. Modificînd programul nostru în această idee, se obţine:

```
program schimb_dialog;
var
   curs_valutar, suma_schimbata, suma_obtinuta
   : real;

begin
   write('Cursul de schimb : ') ;
   readln(curs_valutar);
   write('Suma schimbata : ') ;
   readln(suma_schimbata);
   suma_obtinuta := suma_schimbata*curs_valutar;
   writeln('Suma obtinuta este ', suma_obtinuta:4:2);
end.
```

Dacă se execută acest program pentru un curs valutar de 3.7942 şi o suma de schimbat de 50 se va obţine următoarea imagine pe ecran:

```
Cursul de schimb : 3.7942
Suma schimbata : 50
```

Suma obtinuta este 189.71

Să considerăm şi o altă variantă de program pentru aceeaşi problemă:

```
program schimb_valutar;
var
    curs_valutar : real ;
    suma_schimbata : integer ;
begin
    write('Cursul de schimb : ') ;
    readln(curs_valutar) ;
    write('Suma schimbata : ') ;
    readln(suma_schimbata);
    writeln('Suma obtinuta este ',
          suma_schimbata * curs_valutar : 4 : 2)
end.
```

În acest program s-au făcut cîteva modificări. Astfel a fost modificat tipul variabilei suma_schimbata, care a fost declarată de tip *integer* şi s-a renunţat la utilizarea unei variabile pentru memorarea rezultatului. Deci o primă observaţie este că în instrucţiunea de tipărire pot să apară şi expresii al căror rezultat va fi afişat (evident un lucru asemănător nu se poate realiza pentru operaţiile de citire). Dacă se execută programul cu aceleaşi date ca şi în cazul anterior efectul obţinut va fi acelasi. Să presupunem însă că executăm programul şi în loc de valoarea 50 introducem valoarea 50. (deci o valoare de tip *real*). Atunci, ca efect al execuţiei, pe ecran vom obţine:

```
Cursul de schimb : 3.7942
Suma schimbata  : 50.

I/O error 10, PC=2DE6
Program aborted

Searching
9 lines

Run-time error position found. Press <ESC>
```

La apăsarea tastei ESC se intră în regimul de editare a textului programului, cursorul fiind poziţionat pe instrucţiunea de citire a sumei schimbate. În acest mod utilizatorul este informat care dintre instrucţiunile de citire sau scriere (I/O) nu s-a executat corect. Erorile din această categorie pot să fie prevenite prin afişarea tipului (eventual chiar a domeniului) în care trebuie să se încadreze datele furnizate de către utilizator. Să considerăm de exemplu următoarea variantă de program:

```
program schimb_valutar;
var
   curs_valutar : real ;
   suma_schimbata : integer ;
begin
   write('Cursul de schimb  (real) : ') ;
   readln(curs_valutar) ;
   write('Suma schimbata (intreg) : ') ;
   readln(suma_schimbata);
   writeln('Suma obtinuta este ',
         suma_schimbata * curs_valutar : 4 : 2)
end.
```

Iată şi un exemplu de execuţie a acestui program:

```
Cursul de schimb  (real) : 3.7942
Suma schimbata (intreg) : 50
Suma obtinuta este 189.71
```

Cîte ceva despre expresii în limbajul Pascal

Înainte de a prezenta instrucţiunile utilizate în limbajul Pascal este util să aprofundăm puţin discuţia referitoare la expresii. Să considerăm de exemplu expresia:

```
21 - 3 * 9 div - 6 div 2
```

Se observă că în această expresie intervin operatorii -, * şi div, iar operanzii sînt reprezentaţi de constante. Valoarea expresiei depinde atît de valorile operanzilor, cît şi de ordinea în care se execută diferitele operaţii. Dacă alegem o expresie mai simplă:

```
21 - 3 * 9
```

pare evident faptul că rezultatul este obţinut efectuînd întîi înmulţirea şi apoi scăderea. Adică, expresia anterioară are acelaşi rezultat cu expresia:

```
21 - (3 * 9)
```

Deci evaluarea se face ca şi cum operaţia de înmulţire este "mai importantă" decît operaţia de scădere. Utilizînd termenii obişnuiţi în descrierea limbajelor de programare spunem că operatorul de înmulţire este prioritar faţă de operatorul de scădere. Să considerăm şi expresia:

```
2 * -3
```

În acest caz semnul "-" apare ca operator unar şi va fi utilizat în evaluare înainte de operatorul de înmulţire. În limbajul Pascal ordinea de prioritate a operatorilor este următoarea:

```
operatori unari: -
operatori binari: * / div mod { operatori multiplicativi }
                  + -         { operatori aditivi }
```

(operatorii care apar pe aceeaşi linie au aceeaşi prioritate).

Să considerăm acum expresia:

```
4 + 2 + 3
```

Evident, indiferent dacă evaluarea se face în ordinea (4 + 2) + 3 sau în ordinea 4 + (2 + 3) rezultatul este

acelaşi datorită proprietăţii de asociativitate a operaţiei de adunare. Ce se întîmplă însă în cazul expresiei:

12 div 4 div 3

În acest caz expresiile (12 div 4) div 3 si 12 div (4 div 3) au valori diferite. Cu alte cuvinte, operaţia de împărţire nu este asociativă. Conform definiţiei limbajului **Pascal** în cazul operatorilor cu aceeaşi prioritate evaluarea se face de la stinga la dreapta sau, altfel spus, operatorii au asociativitate stinga. Corespunzător, expresia

21 − 3 * 9 div −6 div 2

are aceeaşi valoare ca şi:

21 − (((3 * 9) div (−6)) div 2).

Dacă am deschis discuţia referitoare la expresii numerice în limbajul **Pascal** poate că merită să discutăm şi despre expresii care conţin operanzi şi operatori corespunzători altor tipuri de date.

Pentru datele de tip *boolean* se utilizează următoarele operaţii specifice: *not*, *and*, or si *xor* (ultimul operator nu există în limbajul **Pascal** standard fiind o extensie care apare în **TURBO Pascal**). Operatorul *not* aplicat valorii *false* produce rezultatul *true*, iar aplicat valorii *true* produce rezultatul *false*. Pentru a explica operatorii *and, or, xor*, să analizăm următoarele fraze:

a. Mă duc la film dacă plouă sau rulează un film bun
b. Mă duc la film dacă plouă şi rulează un film bun
c. Mă duc la film dacă plouă şi nu rulează un film bun sau nu plouă şi rulează un film bun

Se observă că în cazul în care plouă şi rulează un film prost cel care a enunţat frazele anterioare se va duce la film in cazurile **a** si **c**. Dacă nu plouă şi rulează un film prost nu se va merge la film în nici un caz. Dacă plouă şi rulează un film bun se va merge la film în cazurile **a** şi **b**.

Să identificăm şi situaţiile în care nu se merge la film pentru fiecare caz în parte:

a. nu plouă şi nu rulează un film bun
b. nu plouă sau nu rulează un film bun
c. nu plouă şi nu rulează un film bun sau plouă şi rulează un film bun !

Condiţiile "plouă" şi "rulează un film bun" au fost legate în cazul **a**. printr-o operaţie de tip "şi logic" (*and*), în cazul **b** printr-o operaţie de tip "sau logic" (*or*), iar în cazul **c** prin operaţia "sau exclusiv" (*xor*). Definiţiile acestor operatori sînt următoarele:

false and false este false false or false este false
false and true este false false or true este true
true and false este false true or false este true
true and true este true true or true este true

false xor false este false
false xor true este true
true xor false este true
true xor true este false

Să considerăm acum o expresie în care apar valori şi operatori de tip *boolean*:

false or *not false* and *false*

Şi în acest caz pentru a determina valoarea expresiei trebuie să precizăm care sînt priorităţile operatorilor şi care este ordinea de evaluare în cazul operatorilor care au aceeaşi prioritate. Conform definiţiei limbajului operatorul *not* fiind un operator unar este cel mai prioritar dintre operatorii de tip *boolean*. Operatorul *and* are aceeaşi prioritate cu operatorii aritmetici: *, /, div şi *mod*. Operatorii *or* şi *xor* au aceeaşi prioritate cu operatorii aritmetici binari: + si −. Rezultă că expresia considerată anterior se evaluează ca şi expresia:

false or ((*not false*) and *false*)

Asupra datelor de tip elementar (standard) − *integer, real, byte, boolean şi char* − se pot aplica şi operatori de comparaţie (operatori relaţionali):

< mai mic
<= mai mic sau egal
= egal
<> diferit
>= mai mare sau egal
> mai mare

Rezultatul unei expresii de forma:

operand1 operator operand2

cu operand1, operand2 de tip elementar şi operator de tip relaţional este o valoare de tip *boolean*. Dacă rezultatul unor expresii de forma:

2 < 3

'a' <= 'b'

0 > 3

este clar (*true* pentru primele două şi *false* pentru a treia, se pune însă problema care este valoarea unei expresii de forma:

false < *true*

Cînd am discutat despre modul de reprezentare în calculator a datelor de tip *boolean* s-a precizat faptul că indiferent de implementarea concretă se respectă regula *false* < *true*. Rezultă deci că expresia anterioară are valoarea *true*. ■

Incursiune în ORACLE

Roxana Bicleanu

Creatorii de soft se întrec în a ne oferi modalităţi cît mai eficiente pentru stocarea, întreţinerea şi vizualizarea informaţiei. Obţinerea rapidă de informaţii complexe a devenit indispensabilă procesului decizional. Pe această cale calculatoarele, ca instrumente de gestiune a volumelor mari de date, au pătruns în toate domeniile vieţii economico-sociale şi ca o consecinţă, a crescut interesul specialiştilor în a uşura cît mai mult rezolvarea problemelor care decurg de aici.

Aşa se face că în ultima vreme s-au dezvoltat o mulţime de sisteme de gestiune a bazelor de date (SGBD) pe toate tipurile de calculatoare. Majoritatea lor au în comun folosirea modelului relaţional în conceperea bazei de date.

De ce s-a ales relaţionalul?

Primul criteriu a fost simplitatea.

Imaginarea informaţiei ca fiind stocată în tabele este la îndemîna oricui. Este simplă şi pentru cei care au studiat cu ani în urmă şi au programat mult în Cobol sau Fortran, şi pentru cei care se familiarizează cu calculatorul de pe băncile şcolii.

Simplitate decurge din faptul că toată informaţia cuprinsă în tabelă se poate vizualiza în orice moment. Şi aceasta pentru că atît obiectele, cît şi legăturile dintre ele se descriu folosind aceleaşi mijloace: *tabele.*

Tabela care descrie un obiect are în componenţă coloane, fiecare mapîndu-se pe un atribut al obiectului.

Tabela care descrie o legătură alătură acele coloane din tabelele de bază a căror combinaţie identifică rîndul în mod unic (cheile unice).

Subliniem acest aspect pentru a diferenţia modelul relaţional de cele reţea sau ierarhic, unde legăturile între obiecte se realizau prin pointeri, care necesitau o tratare diferită de cea a atributelor obiectului.

Al doilea criteriu care a dus la răspîndirea modelului relaţional a fost constatarea că, pentru baze mici şi medii (mii de rînduri în tabele), timpul de răspuns pentru prelucrarea şi regăsirea informaţiei este comparabil cu cel obţinut în SGBD-urile reţea şi ierarhic.

S-a constatat, de asemenea, că majoritatea aplicaţiilor utilizînd bazè de date nu depăşeau dimensiunile medii. Acesta a fost un aspect important în dezvoltarea softurilor de baze de date pe PC- uri.

Alături de DBASE, unul din cele mai răspîndite SGBD-uri în lume, dar şi în centrele noastre de calcul este **ORACLE.**

Oracle şi ORACLE

Firma ne atrage atenţia asupra felului cum ortografiem acest cuvînt.

Oracle este numele firmei care a creat, lansat şi care dezvoltă în continuare produsul al cărui nume este **ORACLE.**

Apoi trebuie făcută diferenţa între **SGBD ORACLE** şi **sistemul ORACLE.**

SGBD ORACLE este produsul central. El include un nucleu şi mai multe utilitare.

Funcţiile nucleului sînt:

— supervizează definirea şi memorarea informaţiei
— controlează şi limitează accesul concurenţial la date
— permite salvarea şi restaurarea informaţiei
— interpretează SQL

La rîndul lor, *utilitarele* permit:

— transferul de informaţie din bază către şi dinspre fişiere **ORACLE**
— încărcarea în bază a datelor din fişiere sistem
— gestiunea ecranului
— optimizarea cererilor SQL

Sistemul ORACLE include toate produsele firmei Oracle instalate la un moment dat pe un calculator. În afară de SQL*Plus prin intermediul căruia utilizatorul are acces la instrucţiunile SQL şi care în acest fel devine obligatoriu alături de SGBD, linia de produse **Oracle** mai include:

— SQL * Calc
— SQL * Forms
— SQL * Graph
— SQL * Menu
— SQL * QMX
— SQL * Report Writer
— Interfeţe cu limbajele ADA, C, Cobal, Fortran, PL/1

SQL * Plus — este conceput ca o interfaţă prietenoasă cu utilizatorul, care să-i uşureze formularea cererilor, obţinerea de informaţii despre contextul de lucru şi formatarea rezultatelor; este foarte util în obţinerea rapidă de informaţii.

SQL * Calc — combină facilităţile lucrului cu baze de date relaţionale cu cele folosite în foile de calcul.

SQL * Forms — este un utilitar pentru lucrul cu ecrane formatate, el poate fi folosit atît la introducerea datelor cu verificarea condiţiilor de integritate a bazei, cît şi la regăsirea informaţiei.

SQL * Graph — îmbogăţeşte posibilităţile de reprezentare a informaţiei prin introducerea graficelor şi, pentru coloanele rapoartelor create în SQL * Plus, a atributelor de culoare.

SQL * Menu — permite definirea de meniuri arborescente pentru aplicaţii.

SQL * QMX — reprezintă o interfaţă care permite formularea cererilor în limbajul Query by Example.

SQL * Report Whiter — este folosit pentru obţine-

rea unei game foarte largi de rapoarte de ieşire.

Interfeţele cu limbajele evoluate permit accesul la informaţiile stocate în bază şi prelucrarea lor cu mijloacele specifice limbajelor procedurale; se poate suplini în acest fel lipsa unora din utilitarele menţionate mai sus.

Funcţie de configuraţia hard pe care se lucrează, există soft pentru lucrul cu **ORACLE** în reţea.Ce ne sugerează această impresionantă enumerare?

Faptul că firma **Oracle** ne pune la dispoziţie o varietate extraordinară de produse. Practic, problema compatibilităţii între softuri dispare: putem obţine tot ce dorim doar cu componente **ORACLE.**

În plus, autorii insistă asupra portabilităţii şi compatibilităţii SGBD-ului; principiile învăţate o dată pot fi utilizate apoi în diferite contexte hard şi soft.

Sigur că, pe de altă parte, efortul pentru asimilarea modului de lucru cu toate aceste componente nu este neglijabil, fiecare în parte oferind nenumărate căi de rezolvare a unei probleme, dintre care trebuie alese soluţiile cele mai bune.

Punctul central al acestui edificiu îl constituie limbajul SQL(*Sructured Query Language*). Firmei **Oracle** îi aparţine prima implementare a acestui limbaj disponibilă utilizatorilor, în 1979.

Standardul ANSI datează din 1986.

Majoritatea SGBD-urilor relaţionale suportă azi forme de SQL.

LIMBAJUL SQL

Să începem prin a enumera calităţile cu care limbajul a cucerit piaţa:

— limbajul este comun tuturor tipurilor de utilizatori, de la administratorul sistemului pînă la utilizatorii neinformaticieni şi constă din cîteva instrucţiuni de bază care pot fi asimilate foarte repede

— acelaşi limbaj este folosit pentru toate operaţiile specifice lucrului cu obiectele bazei: creare, interogare, actualizare, control.

— programele scrise în standardul **SQL** rămîn valabile chiar la schimbarea calculatorului sau softului cu care se lucrează.

Limbajul **SQL** trebuie privit în strînsă legătură cu teoria bazelor de date relaţionale. El va oferi soluţii pentru rezolvarea operaţiilor specifice acestor baze: *selecţia* şi *join*-ul.

Selecţia este operaţia care permite constituirea de grupuri (seturi) de rînduri (înregistrări) care răspund aceloraşi condiţii.

Join-ul este operaţia care permite selectarea de atribute (coloane) din mai multe tabele.

Instrumentele cu care se realizează aceste operaţii sînt comenzile, operatorii şi funcţiile.

Comenzile se grupează în trei categorii:

— comenzi de definire a obiectelor bazei (comenzi de tipul: CREATE, ALTER, DROP);

— comenzi de manipulare (SELECT, INSERT, UPDATE, DELETE);

— comenzi de control(GRANT).

Operatorii utilizaţi sînt şi ei de mai multe tipuri:

— operatori aritmetici (+, -, etc.)

— operatori şir (operatorul de concatenare)

— operatori de comparaţie (=, <, >, IN, BETWEEN, IS [NOT] NULL, etc.)

— operatori logici (NOT, AND, OR)

— operatori set (UNION, INTERSECT, MINUS)

— alţi operatori (DISTINCT, PRIOR, etc.)

Funcţiile pot fi clasificate în:

— funcţii care acţionează la nivelul fiecărui rînd selectat într-un grup (DECODE, LPAD, REPLACE, etc.)

— funcţii de grup (MAX, MIN, SUM, AVG, etc.)

— funcţii de conversie (TO-CHAR, TO-NUMBER, etc.)

— funcţii care acţionează asupra tipului de dată DATE

Menţionăm în plus posibilitatea folosirii unei game largi de formate.

Forţa pe care o dobîndeşte limbajul prin combinarea tuturor acestor elemente este uimitoare.

Condiţia care trebuie respectată pentru a avea acces la toate facilităţile oferite de limbaj este proiectarea corectă, în spirit relaţional a bazei.

Asupra paşilor care trebuie urmaţi în proiectarea modelului logic vom reveni într-unul din numerele viitoare cînd vom comenta realizarea unei aplicaţii.Pînă atunci să amintim că între cele trei modele de reprezentare a bazelor de date nu există bariere de netrecut şi că în **SQL** există implementată posibilitatea de navigare pe un arbore atît în preordine cît şi în postordine.

SQL — *LIMBAJ NEPROCEDURAL*

Ne oprim asupra acestei probleme pentru că nu mulţi dintre noi au avut ocazia să programeze în limbaje neprocedurale.

Numim neprocedurale limbajele în care nu putem stabili cu exactitate ordinea în care se efectuează operaţiile. În cazul **SQL**-ului, aceasta cade în sarcina optimizatorului, parte a nucleului, care alege soluţia cea mai bună.

Prelucrările nu se mai referă la înregistrări singulare, ci la grupuri de înregistrări. Sarcina programatorului este aceea de a exprima cu claritate operaţiile dorite şi condiţiile de constituire a grupurilor.

Ca o consecinţă a caracterului neprocedural, din **SQL** au dispărut instrucţiunile de realizare a ciclurilor, precum şi structurile de tipul IF THEN ELSE.

Ne putem descurca fără ele?

În foarte multe situaţii răspunsul este afirmativ.

Ciclurile erau folosite în primul rînd pentru a repeta aceeaşi prelucrare pentru mai multe înregistrări ale unui fişier. Această necesitate dispare aici, întrucît sînt prelucrate automat toate înregistrările selectate.

O situaţie mai complicată apare atunci cînd avem nevoie de mai multe cicluri imbricate, sau cînd dorim execuţia aceleiaşi secvenţe modificînd unul sau mai mulţi parametri. Deşi în **SQL** nu putem rezolva această problemă, utilitarele ne oferă soluţii în multe cazuri (ne referim la **SQL * Forms** şi la **SQL * Report Writer**). Şi bineînţeles că întotdeauna rămîne la dispoziţia programatorului calea interfeţei cu limbajele procedurale, care oferă performanţe de timp foarte bune.

Dacă reuşim să ne menţinem foar în sfera **SQL**-ului,

efortul depus pentru obţinerea rezultatelor este mult redus.

Structura IF THEN ELSE o simulăm prin folosirea operatorilor şi funcţiilor.

EXPRIMAREA CERERII

Problema centrală a lucrului în **SQL** rămîne felul în care cerem numai informaţiile de care avem nevoie la un moment dat; operaţiile principale pe care le efectuăm asupra bazei sînt, aşa cum am arătat mai devreme, selecţia şi join-ul.

Comanda folosită pentru regăsirea informaţiei este SELECT.

Deci, SELECTăm dintr-o tabelă coloane reale sau expresii. În mod obligatoriu trebuie să ştim despre aceste coloane cum se numesc şi cărei tabele aparţin. Scriem

```
SELECT nume-coloană/expresie, ...
FROM nume-tabelă;
```

Încheierea formulării cererii înseamnă şi intrarea ei în execuţie; rezultatul SELECT-ului apare pe ecran. În cazul nostru este vorba despre toate rîndurile unei tabele, dar numai coloanele cerute.

Dorim mai puţine rînduri? Enunţăm condiţiile care restricţionează numărul de rînduri în clauza WHERE.

Vrem ca ele să apară într-o ordine anumită?

Încheiem prin menţionarea clauzei ORDER BY urmată de cheile de sort.

Dacă dorim să dispunem de informaţii memorate în mai multe tabele (operaţia de join), menţionăm toate coloanele, indiferent de tabelă, avînd grijă să calificăm numele coloanelor care nu sînt unice cu denumirea tabelei din care provin, apoi enumerăm toate aceste tabele în clauza FROM. Dacă ne oprim aici, ne apare pe ecran rezultatul produsului cartezian între rîndurile tuturor tabelelor. Pentru a limita numărul de rînduri incluse în set, condiţiile restrictive le înscriem, fireşte, în clauza WHERE. Scriem deci

```
SELECT [nume-tabelă.]nume coloană/expresie,...
FROM nume-tabelă ...
[ WHERE condiţii]
[ ORDER BY cheie-sort 1, ...] ;
```

Pentru a obţine informaţii totalizatoare referitoare la o tabelă aplicăm asupra numelor coloanelor funcţii de grup.

Dacă dorim ca aceste rezultate să se refere nu la ansamblul tabelei, ci la grupuri identificate prin valorile uneia sau mai multor coloane, folosim clauza GROUP BY urmată de numele acestor coloane.

Procedee similare se folosesc pentru constituirea setului de rînduri în cazul altor comenzi de manipulare a datelor (UPDATE, DELETE, INSERT). ■

Nota Redacţiei

Referitor la articolul "Despre ecrane şi memorarea lor", vă semnalăm cîteva erori de redactare care s-au strecurat în textul articolului şi, în acelaşi timp, vă cerem cuvenitele scuze.

- în programul din coloana a doua(pagina 38), la linia 1030, se va citi:
1030 POKE 50000+i, PEEK(16384+i) în loc de: ....,PEEK(16348+i)
- în acelaşi loc, la linia 1040, se va citi:
1030 NEXT i: CLS : STOP în loc de : 1040 CLS : STOP (altfel apare eroarea elementară că bucla deschisă în linia 1020 prin FOR... nu se închide prin NEXT);
- în schema programului în cod maşină din chenarul prim de la pagina 39, se va citi:
....9 237 176 LDIR în loc de: .....9 237 176 .DIR
- atît în programul de la pagina 39(titlu şi linia 9999), cît şi la COMENZI BASIC (penultimul rînd din pagina 39), cuvîntul SCREEN este urmat de simbolul de şir **$**, respectiv **SCREEN$** ;
- în programul în chenar de la pagina 39, la linia 40, se va citi:
40 ... PRINT AT 11,3; "Programul BASIC trebuie sa contina comanda"; ... în loc de: "Programul BASIC trebuie sa continuam comanda"
- în acelaşi program, la linia 60, se va citi:
60 ... LET h=255−j*27: ... în loc de: LET h+255−j*27;
- în tabelul de la pagina 40, în rîndul 0 al liniei 0. coloana 31, se va citi: **16415** în loc de: 16515.
- în prima coloană de la pagina 38, rîndul 10 de jos, se va citi:**"Partea grafică este cuprinsă între adresele 16384 şi 22527 şi este împărţită în 3 sectoare de cîte 8 linii...."** în loc de: "Partea grafică este cuprinsă în 3 sectoare de cîte 8 linii...."
- în coloana a doua de la pagina 39, primul rînd sub chenarul programului în cod maşină, se va citi:
"BASIC (cu PLOT, DRAW şi CIRCLE). După desenare sau după încărcare de pe bandă..." în loc de:
"BASIC (cu PLOT, DRAW şi CIRCLE). După desenare de pe bandă..."

TOTUL DESPRE MEMORIE(I)

Marius Sturzoiu

Atunci cînd IBM PC şi-a început marşul triumfal, se părea că un RAM de 640 K va fi suficient pentru toate aplicaţiile. Foarte curînd bariera s-a prăbuşit. La început a fost dorinţa de a se stoca în RAM cît mai multe date; au cerut-o 123 a lui Lotus, Oracle şi alţii. Mai tîrziu a beneficiat şi DOS 4.1. Apoi a venit Windows 3.0 care a instalat programe dincolo de barieră.

Tipuri de memorie

Dacă ar fi să facem o clasificare simplificată a RAM putem vorbi de 4 tipuri de memorie:

— *memoria convenţională.* Aceasta se întinde de la 0- 640 K. Este zona în care se execută toate aplicaţiile sub MS-DOS. Pînă la versiunea 4.01 sistemul de operare MS-DOS nu permite accesul direct la o zonă de memorie peste 1 Mo. Restricţia provine de la aceea că sistemul de operare DOS a fost conceput pe un PC echipat cu microprocesorul 8088. Acesta avea magistrala de memorie de 20 biţi, permiţînd adresarea în intervalul 0 - ($2^{20}-1$) adică 1Mo. S-a hotărît atunci ca zona 0-9FFFFH640ko) să fie destinată aplicaţiilor— deci RAM, iar zona A0000H- FFFFFH (384 Ko) să fie destinată sistemului — deci ROM, plăci de memorie, etc.

În funcţie de versiunea de DOS utilizată, de comenzile de configurare ale sistemului, de eventualele programe rezidente RAM-ul rămas la dispoziţia utilizatorului este în cel mai bun caz aproximativ 560 ko.

— *memoria înaltă* (high memory). Este zona cuprinsă între AOOOOH-FFFFFH, din considerentele anterioare.

— *memoria extinsă* (extended memory). Este memoria de dincolo de 1 M. Acest spaţiu de memorie nu există pentru microprocesoarele 8088/8086. Pentru 80286 (are 24 biţi adresă, deci adresează 16 Mo), 80386SX (idem), 80386 (are 32 biţi adresă, deci adresează 4 Go), acest spaţiu poate fi adresat în anumite condiţii. Astfel, aceste procesoare au fost concepute să funcţioneze compatibil 100% cu 8086/8088, pentru a putea rula imensa cantitate de programe scrise pînă la apariţia lor. De aceea ele lucrează în două moduri diferite: modul real, în care se comportă exact ca 8086 şi modul protejat, în care se profită de toate facilităţile lor.

Din păcate la 80286 trecerea mod real-mod protejat se face uşor; în sens invers numai prin reiniţializarea microprocesorului, timp în care sistemul de întreruperi este blocat. Cu începere de la 80386, trecerea se face şi invers fără dificultăţi.

Chiar eliminată limitarea de natură fizică, apare limitarea impusă de sistemul de operare MS-DOS. Pînă la versiunea 4.01 inclusiv, acesta se execută doar în mod real. Singura utilizare a memoriei extinse este stocarea de date sub forma unui disc RAM(vdisk.sys,Ramdrive.sys), a unui disc "cache" (Smartdrv.sys) sau a unor buffer-e (comanda Buffers=...). Şi nu uitaţi, nu folosiţi memoria extinsă pe o configuraţie XT.

Apariţia lui Windows 3.0 a modificat această situaţie. Utilizînd driverul HIMEM.SYS, instalat în CONFIG.SYS care are la bază protocolul XMS (eXtended Memory Specification), aplicaţiile adaptate lui Window 3.0 se execută în memoria extinsă, pentru microprocesoarele 80286, 80386, 80486. Aplicaţiile DOS lansate sub Windows 3.0 se vor executa în memoria convenţională pentru un PC echipat cu 80286; pentru PC echipat cu 80386 se pot executa inclusiv în memoria extinsă.

Nu numai Windows 3.0 are acces la memoria extinsă. Tehnica de acces foloseşte un soft specializat — un DOS Extender. Şi 1-2-3/3 a lui Lotus a folosit această metodă. Dar asupra acestor chestiuni vom reveni.

— *memoria expandată, paginată* (Expanded memory). Conceptul a apărut în 1985 cînd Lotus şi Intel au dezvoltat specificaţia soft-hard EMS 3.2 (Expanded Memory Specification — specificaţia de memorie extinsă). Ulterior, la aceasta a aderat şi Microsoft apărînd specificaţia LIM (Lotus Intel Microsoft). Pentru a funcţiona, specificaţia necesită: plăci memorie special concepute (pe modelul Intel Above Board) şi un driver care se instalează în CONFIG.SYS (de obicei EMM.SYS).

Norma EMS 3.2 se bazează pe conceptul de pagină. Memoria paginată se prezintă sub forma unui "pool" de pagini de 16 Ko care nu au iniţial nici-o adresă. Pentru a putea fi adresate aceste pagini se copiază într-o zonă de memorie cu adresa sub 1Mo, accesibilă DOS. Conform normei, copierea se face într-o fereastră de 64 Ko rezervată, cu adresă fixă, fixată de switch-urile de pe placa de memorie expandată. Dacă se doreşte accesul la o altă zonă din memoria expandată (specificaţia admiţînd un maximum de 8 Mo) intră în acţiune două mecanisme:

— întîi are loc salvarea conţinutului actual al ferestrei în memoria expandată (page swapping).

— apoi are loc transferul noilor 4 pagini selectate.

Evident procesul are loc transparent pentru utilizator, fiind lăsate în seama driverului de memorie instalat în CONFIG.SYS şi a plăcii de memorie expandată.

Norma respectivă a reprezentat un progres. Ea nu a produs însă o evoluţie sensibilă. În fapt, limitarea ferestrei la 64 K a făcut ca acest mecanism să poată fi utilizat doar pentru stocarea de date, înţelegînd prin aceasta şi eventualele programe în aşteptare.

Deci execuţia programelor trebuia să se facă tot în memoria convenţională.

Pasul următor l-a marcat apariţia normei EEMS (*Enhanced Expanded Memory Specification*), o evoluţie a normei anterioare cu care, de altfel, este compatibilă. Ea a fost definită cu începere din 1987 de Lotus, Intel, Microsoft şi Ast şi e cunoscută sub numele: LIM- EMS 4.0.

Noua normă permite utilizarea unor ferestre mai mari; deci în loc de o fereastră de 64 K pot fi utilizate mai multe, de dimensiuni cuprinse între 16 Ko şi 1 Mo. Adresa ferestrei poate fi definită şi modificată prin program. Norma suportă 32 Mo memorie expandată. De data aceasta memoria paginată poate primi cod executabil. Windows 2.10, de exemplu, foloseşte această zonă pentru a permite executarea simultană a mai multor aplicaţii.

În concluzie deci, referindu-ne la cele două tipuri mai speciale de memorie putem afirma:

— *memoria expandată (paginată)*, sub forma unor plăci de memorie şi a unui driver în CONFIG.SYS poate fi instalată şi utilizată pe maşini echipate cu 8088, 80286, 80386 etc. Ea va fi adresată sub 1 Mo. Memoria extinsă, sub forma unor plăci de memorie şi a unui driver în CONFIG.SYS poate fi utilizată pe maşini echipate cu 80 286, 80 386 etc. Ea va fi adresată peste 1 Mo. De notat că folosind drivere adecvate (EMM 386.SYS din pachetul Windows 3.0, QEMM.SYS al firmei Quarterdeck) memoria extinsă poate fi configurată şi folosită ca memorie expandată.

— o placă de memorie expandată necesită regiştri de control a memoriei şi o MMU (Memory Management Unit) pentru a controla aceşti regiştri. O placă de memorie extinsă nu are astfel de necesităţi.

— o placă de memorie expandată poate fi folosită ca placă de memorie extinsă. Nu şi invers, cu excepţia folosirii unui driver din cele amintite mai sus.

Microprocesorul 80386 rezolvă definitiv problema (chiar şi un 80386SX). Dispunînd de MMU internă el poate folosi memoria extinsă ca memorie expandată. ■

Probleme specifice bazelor de date documentare

Ţeculescu Virgil

Să presupunem că aveţi la dispoziţie o vastă bibliotecă şi doriţi să vă împrospătaţi cunoştinţele într-un anumit domeniu (ex: economia de piaţă).

Cum procedaţi?

Este incomod să tot căutaţi printre fişele bibliotecii diferite cărţi şi reviste în care parcă ar exista ceva informaţii de acest gen. Poate dura ore bune şi efort de concentrare.

Dacă aveţi însă la dispoziţie un computer, problema dumneavoastră e ca şi rezolvată; împreună cu utilitarul **CDS/ISIS**, în cîteva secunde veţi afla cărţile şi articolele ce tratează acest domeniu.

Acest utilitar vă pune la dispoziţie o metodă de interogare a bazei de date biblioteca, după anumite cuvinte cheie. Dacă am avut grijă ca printre aceste cuvinte cheie să se afle şi economia de piaţă, vor fi afişate toate publicaţiile în care acest domeniu a fost atins.

Domeniul de activitate al informaticii documentare porneşte tocmai de la corelaţia dintre creşterea volumului de date stocate şi necesitatea reducerii timpului de acces la informaţie.

Baza de date documentară este o bază de date a cărei informaţie utilă o reprezintă tocmai datele referitoare la publicaţii (titluri de cărţi sau articole, locul unde pot fi găsite, precum şi alte informaţii de interes). Ca orice bază de date, cea documentară are sarcina de a permite utilizatorilor interogări pe diferite cîmpuri (autor, domeniu, etc.) pentru accesarea informaţiei dorite.

Un program orientat spre proiectarea şi interogarea bazelor de date documentare, disponibil pe calculatoare personale, este cum am amintit **CDS/ISIS.** Un astfel de program specializat este simplu de utilizat, însă lasă un spaţiu mic de "manevră" pentru proiectant.

Din punct de vedere al proiectării , baza de date documentară implică unele diferenţe faţă de cele nespecializate. Vom avea şi aici fişiere cu formate de introducere şi de afişare a datelor, tabele de definire a cîmpurilor, tehnici de regăsire, etc.

Tipul informaţiei vehiculate impune însă şi existenţa unor tehnici particulare cum ar fi: interogarea prin dicţionar şi prezenţa fişierelor de istoric, a dicţionarului vid sau de tehnici de indexare.

Problematica generală a bazelor de date nepreocupîndu-ne, în cadrul acestui articol vom detalia în continuare tehnicile particulare de mai sus.

Să revenim la publicaţiile noastre alegînd un exemplu didactic, care să răspundă la întrebarea: "Ce interesează la o carte ?".

— titlul	A	I1
— lista autorilor	A	I2
— cota cărţii (cod pentru regăsire)	A	
— data apariţiei	A	
— editura	A	
— lista descriptorilor		I3
— rezumatul cărţii	A	

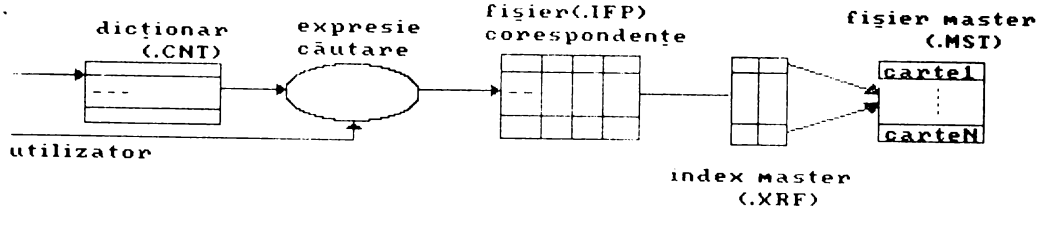
Deci am ales un fişier master cu şapte cîmpuri, fiecare înregistrare conţinînd informaţii ce caracterizează o carte.

Ca scurtă explicaţie, lista de descriptori reprezintă un set de cuvinte-cheie (expresii) care descriu schematic conţinutul cărţii.

În tabelul de mai sus am notat cu 'I' cîmpurile necesare la regăsirea informaţiei (cîmpuri posibil cunoscute de utilizator la interogare), iar cu 'A' pe cele ce interesează şi vor fi afişate. Cîmpurile notate cu 'I' vor fi tratate ş introduse într-un fişier special (dicţionar) ordonat alfabetic în vederea accesării rapide. De menţionat că dicţionarul este memorat în forma arborescentă (B-tree). Deci dicţionarul va "condensa" informaţiile mai multor cîmpuri. Acesta va fi vizualizat la cerere de utilizator, putîndu-se selecta de pe ecran autorii sau domeniile de interes. De remarcat că la o singură interpretare se pot selecta mai mulţi termeni ai dicţionarului, fiind disponibile mai multe tehnici cum ar fi AND, OR şi NOT.

Să presupunem termenii selectaţi de pe ecran a, b şi c, cărora le vor corespunde listele de publicaţii A, B şi C. Tehnicile vor avea ca rezultat operaţiuni de intersecţie, reuniune respectiv scădere a mulţimilor A, B şi C.

Evident, folosirea dicţionarului pentru interogarea bazei de date nu este obligatorie, aceasta putîndu-se realiza explicit.



Exemplu:

Dorim găsirea cărţilor (articolelor) în care este atins domeniul "economia de piaţă" şi "privatizare".

Avem la dispoziţie două metode:

1. *interogarea prin dicţionar:*

Navigăm un dicţionar, selectăm sintagma "economia

de piaţă", selectăm şi cuvîntul "privatizare" şi lansăm cererea de interogare a bazei de date. Vom obţine toate publicaţiile în care ambele domenii sînt tratate.

2. *interogarea explicită:*

Editâm o linie astfel: <economia de piaţă.* privatizare> , '*' are rolul de AND logic. Lansînd cererea de interogare vom obţine acelaşi rezultat ca şi în cazul precedent.

Obs.: Putem căuta toate publicaţiile în care apar noţiuni de economie. Interogarea se va face astfel:<economie$> , deci rădăcina, urmată de '$', furnizînd toate cărţile în care apar informaţii legate de orice tip de economie (nu doar cea de piaţă).

Analizînd în continuare cîmpurile ce formează dicţionarul, se poate constata neomogenitatea acestora, ceea ce implică tratarea lor diferită. Deci există mai multe tehnici de indexare, care se aplică diferit asupra cîmpurilor fişierului master.

Această corespondenţă este conţinută într-un fişier special, fişierul tehnicilor de indexare.

Concret, la exemplul nostru aceste tehnici sînt:

- preluare text dintre delimitatori (**I3**);
- preluare precedată de descompunerea pe subcîmpuri (**I2**);
- preluare precedată de descompunerea cuvînt cu cuvînt a textului.

Detaliind mecanismul descompunerii titlului cărţii (**I1**), vom constata introducerea în dicţionar şi a numeroase conjuncţii, prepoziţii şi a altor cuvinte nedorite. Pentru eliminarea acestui neajuns s-a introdus mecanismul "dicţionarului vid", care constă în căutarea unui cuvînt într-o listă existentă de cuvinte de legătură. Vor fi trecute în dicţionarul de interogare numai cele ce nu au fost găsite în dicţionarul vid.

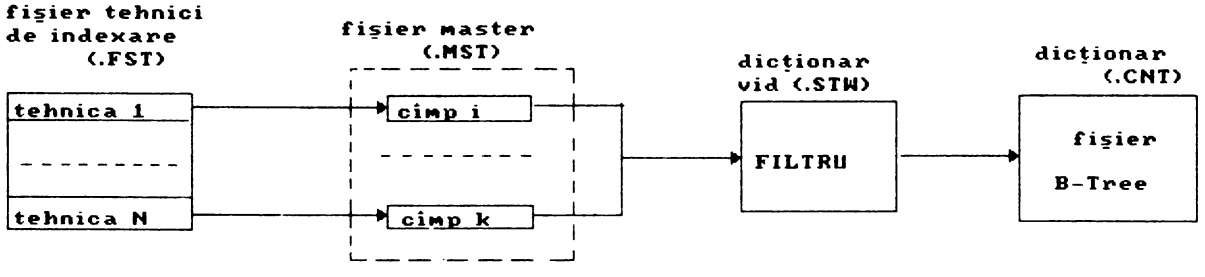


fig.2. Construcţia dicţionarului (CDS/ISIS)

O problemă deosebită a acestor programe destinate proiectării bazelor de date documentare o reprezintă alocarea spaţiului de memorie. Se poate uşor observa din analiza exemplului fişierului de mai sus diferenţa între necesităţile de spaţiu de la carte la carte. Unele liste de descriptori sau rezumate pot fi foarte scurte şi altele foarte lungi, iar numărul autorilor poate fi şi el diferit.

Apare deci necesitatea compactării zonei de memorie, pentru a preveni un consum exagerat de spaţiu, care se realizează prin memorarea lungimii fiecărui cîmp de dimensiune variabilă.

Recapitulînd, mergem la bibliotecă, deschidem computerul, apelăm ISIS, vizionăm dicţionarul autorilor şi descriptorilor, selectăm termenii de interes, interogăm baza de date şi căutăm în bibliotecă publicaţiile listate. ■

STOP NEWS

Paradox SQL Link

SQL (*Structured Query Language* — pronunţaţi sequel) este standardul în manipularea datelor pentru SGBD relaţionale. Utilizînd **Paradox SQL Link** veţi putea (atenţie, doar din interiorul **Paradox**) avea acces la datele stocate pe o bază de date SQL amplasată pe un server, într-o arhitectură client/server.

Paradox Engine

Este o bibliotecă funcţii C şi proceduri Pascal care pot fi apelate din programe C sau Pascal. Acestea constituie o API (*Applications Programming Interface*) care oferă utilizatorului nucleul **Paradox** de manipulare a datelor, inclusiv în medii multiutilizator.

LIMBAJUL MAŞINĂ Z80 PE CALCULATOARELE COMPATIBILE SPECTRUM (III)

Bogdan Georgescu
Bogdan Iordănescu

În continuarea incursiunii noastre în lumea codului maşină, ne vom ocupa acum de sistemul de input-output al **SPECTRUM**-ului. În configuraţie standard, dispozitivele de intrare-ieşire sînt: tastatura, ecranul, difuzorul intern şi casetofonul.

Afişarea pe ecran se face prin intermediul memoriei video. Această memorie este accesată de către sincrogenerator, care transmite datele, convertite din forma binară în cea analogică, la monitor. Activitatea sincrogeneratorului nu poate fi controlată prin software, ea poate fi cel mult observată. Datorită întreruperilor generate microprocesorului.

Comunicaţia cu celelalte periferice se realizează prin intermediul unui port de intrare şi al unuia de ieşire, care împreună sînt denumite în mod convenţional "portul de I/O de adresă #FE". Vom vedea în continuare de ce această denumire este mai mult o convenţie de limbaj decît o denumire exactă.

Avem în codul maşină **Z80** următoarele instrucţiuni de I/O:

— IN A, (n); OUT (n), A, unde n este un număr pe un octet.

— IN r, (C); OUT (C), r, unde r este unul dintre regiştrii: A,B,C,D,E,H,L.

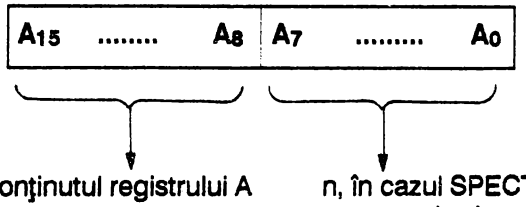
— INI sau IND; OUTI sau OUTD

— INIR sau INDR; OTIR sau OTDR

Vom explica în acest număr comportarea portului #FE ca port de input. Să vedem întîi, dintr-un punct de vedere mai apropiat de hardware, cum face microprocesorul **Z80** dacă trebuie să citească date din exterior.

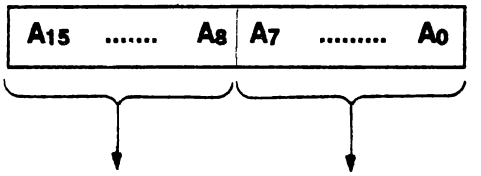
Fiecare periferic se caracterizează printr-o anumită adresă. Deci, pentru a-l putea accesa, microprocesorul trebuie să pună pe magistrala de adrese adresa corespspunzătoare dispozitivului respectiv. Magistrala codifică prin semnale electrice un număr binar pe 16 biţi, format astfel:

— în cazul instrucţiunii IN A, (n):



conţinutul registrului A — n, în cazul SPECTRUM-ului standard, n= #FE

— în cazul instrucţiunii IN r, (C):



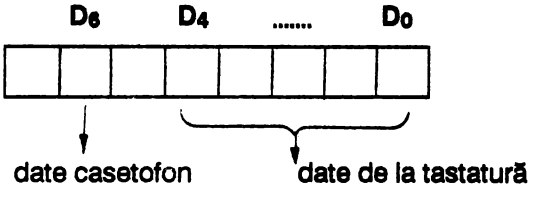
conţinutul registrului B — conţinutul registrului C, care în cazul nostru trebuie să fie #FE

— în cazul instrucţiunilor repetitive, comportarea este identică, dar pe liniile A8-A15 se pune conţinutul registrului B decrementat.

Protocolul de I/O este, în realitate, puţin mai complicat, dar ceea ce ne interesează este că microprocesorul citeşte, de pe magistrala de date, informaţiile transmise de periferic, în registrul A la IN A, (n), în registrul r la IN r, (C) sau în locaţia de memorie adresată de registrul dublu HL la instrucţiunile INI, IND, INIR şi INDR.

Din această scurtă caracterizare reiese faptul că adresa unui port de input-output este un număr pe 16 biţi, deci se pot accesa pînă la 65 536 porturi.

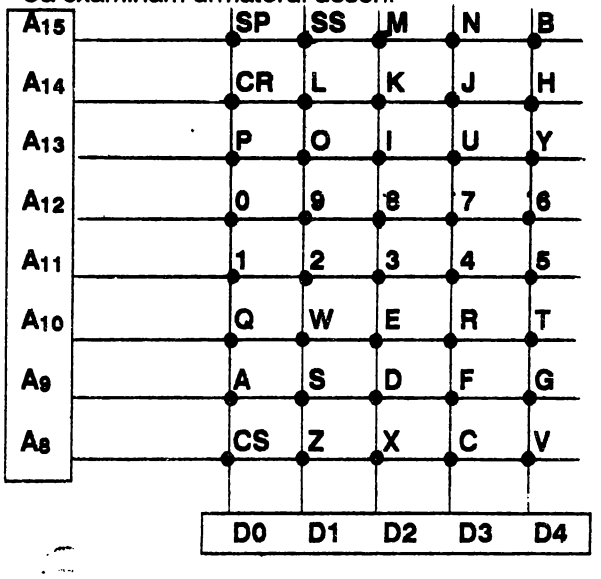
Revenind la **SPECTRUM**, în care, evident, nu se folosesc 65 536 porturi, să vedem cum este organizată informaţia care vine pe magistrala de date, sub forma unui număr de 8 biţi:



date casetofon — date de la tastatură

Se pune întrebarea: cum se poate citi starea a 40 de taste prin intermediul a numai 5 biţi?

Să examinăm următorul desen:



Probabil v-aţi închipuit deja o reţea de fire (o matrice) în care fiecare intersecţie corespunde unei taste. Observaţi că liniile sînt conectate la partea superioară a magis-

tralei de adrese, iar coloanele sînt conectate la partea magistralei de date care corespunde tastaturii în desenul anterior.

În mod normal, liniile nu sînt conectate cu coloanele. Cînd o tastă este apăsată, linia şi coloana corespunzătoare acesteia se pun în contact, deci starea coloanei devine aceeaşi cu starea liniei. În mod normal, liniile magistralei de date sînt menţinute la starea "1" logic. Dacă o linie este pe "0" logic şi una dintastele de pe această linie este apăsată, coloana corespunzătoare ei va trece şi ea în starea "0". Deci, dacă toate liniile, în afară de una sînt pe "1" logic, la magistrala de date se va citi starea tastelor de pe linia pusă pe "0", cu observaţia că o tastă este apăsată cînt bitul corespunzător pe coloană este pe "0". Dar mai bine să dăm un exemplu: să presupunem că vrem să testăm dacă tasta corespunzătoare cifrei 3 este apăsată. Va trebui să plasăm pe A8-A15 următorul număr în binar:

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
1	1	1	1	0	1	1	1

↓

corespunzător liniei de taste: 1,2,3,4,5.

Să presupunem că starea magistralei de date este:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
X	X	X	1	0	1	1	0

nu ne interesează

Conform celor spuse anterior, rezultă că tasta 3 nu este apăsată, în schimb sînt apăsate tastele 1 şi 4. Dacă starea magistralei de date este:

D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	1	0	0	1	1

atunci înseamnă că tasta 3 este apăsată, în rest fiind apăsată doar tasta 4.

Considerăm că acest exemplu a fost suficient de elocvent şi acum să trecem de la teorie la practică. Programul în limbaj de asamblare care testează tasta 3 este:

```
        LD      A,#F7     ; binar 11110111
        IN      A,(#FE)   ; citeşte portul
        AND     #04       ; testează bitul 2 din A
        JR      Z,APASAT  ; dacă bitul este 0, salt la
                          ; rutina corespunzătoare
NU      ----------
--------  --------
APASAT  ----------
--------  --------
```

Pentru a reţine mai uşor ce bit trebuie resetat pentru a selecta una din liniile de 5 taste, să le reprezentăm aşa cum sînt ele dispuse în mod normal:

	D0	D1	D2	D3	D4	D4	D3	D2	D1	D0	
3 →	1	2	3	4	5	6	7	8	9	0	← 4
2 →	Q	W	E	R	T	Y	U	I	O	P	← 5
1 →	A	S	D	F	G	H	J	K	L	CR	← 6
0 →	CS	Z	X	C	V	B	N	M	SS	SP	← 7

În dreptul fiecărei linii am scris bitul ce trebuie resetat, iar deasupra corespondenţa dintre biţii citiţi de pe magistrala de date şi taste.

Din mica schemă pe care am făcut-o se poate observa că se pot citi mai multe rînduri de taste concomitent, dar fără a şti exact care din rînduri conţine tasta apăsată. De exemplu, punînd A15 şi A11 pe 0, dacă la citirea portului vom avea D2 pe 0, nu vom putea şti sigur care din tastele 3 sau M au fost apăsate, dar sigur una din ele a fost apăsată.

O altă observaţie ar fi cu privire la denumirea "portul #FE". Tot din schemă rezultă că, de fapt, sînt 8 porturi cu adresele obţinute aşa cum am arătat, iar combinaţiile de rînduri citite în acelaşi timp ar reprezenta combinaţii între aceste porturi.

A mai rămas un singur lucru de spus, şi anume rolul bitului D6 de pe magistrala de date. Indiferent care din liniile A8-A15 este pe 0, atunci cînd se face citire de la un port cu #FE pe liniile A0 - A7, bitul D6 al magistralei de date va conţine starea bitului corespunzător casetofonului. Acest bit este folosit în general pentru citirea programelor de pe casetă.

Vom prezenta în continuare un program care rulează în întreruperi şi foloseşte citirea tastaturii pentru a mişca pe partea de sus a ecranului ... veţi vedea ce ! Din moment ce se folosesc întreruperile, programul va rula în paralel cu orice alt program **BASIC** sau chiar cu un program în cod maşină, evident cu anumite restricţii de memorie, temporizare etc. În subrutina de tratare a întreruperii, pe lîngă citirea tastaturii prin metoda prezentată anterior, se citeşte tastatura şi prin apelarea rutinei din ROM, de la adresa #02BF, care este indispensabilă rulării BASIC-ului.

```
        ORG     #FF00   ; subrutina de iniţializări începe
INIT    LD      HL,#F000; la #FF00
        LD      B,C
```

```
        LD      A, #F1      ; Se creează tabela de întreruperi
INIT0   LD      (HL), A     ; între adresele #F000 şi F100 inclu-
        INC     HL          ;siv, conţinînd valoarea #F1
        DJNZ    INIT0
        LD      (HL), A
        LD      IX, DATED   ; Se creează la adresa "DATES" o
        LD      DE,3        ; bază de date pentru orientarea spre
        LD      C,32        ; stînga a maşinii, folosind datele
INIT1   LD      A, (IX+0)   ; de la adresa "DATED" unde este
        LD      H, (IX+1)   ; stocată orientarea spre dreapta
        LD      L, (IX+2)
        LD      B, 24
INIT2   SLA     L
        RL      H
        RL      A
        RR      (IX +96)
        RR      (IX+97)
        RR      (IX +98)
        DJNZ    INIT2
        ADD     IX, DE
        DEC     C
        JR      NZ, INIT1
        LD      A,#F0       ; se trece în modul 2 de întreruperi
        LD      I,A
        DI
        IM      2
        EI                  ; Urmează, de la adresa #F101,
        RET                 ; datele pentru orientarea la dreapta
        ORG     #F101
DATED   DEFB    0,4,0,0,4,0,1,4,0,1,130,0, 1,130,0,63,241,128,110
        DEFB    191,240,255,191,248
        DEFB    224,188,28,206,121,204,145,50,36,100,160,148,
                46,253
        DEFB    208,36,132,144,17,2,32,14,1,192,255,191,248,
                238,189
        DEFB    220,209,122,44,170,181,84,100,180,148,42,253,
                80,17
        DEFB    2,32,14,1,192,110,191,240,255,191,240,241,190,60
        DEFB    228,252,156,174,181,212,100,180,148,17,122,32
        DEFB    14,1,192
DATES   DEFS    96          ; alocă spaţiu pentru DATES
POZ     DEFB    0           ; iniţializează locaţiile POZ, ROT şi
ROT     DEFB    4           ; ADR folosite în program
ADR     DEFW    #400A
        ORG     #F1F1       ; la adresa #F1F1 este subrutina
        DI                  ; pentru tratarea întreruperilor
        PUSH    HL
        PUSH    DE
        PUSH    BC
        PUSH    AF
        PUSH    IX
        CALL    MASINA      ;Se apelează subrutinele pentru
        CALL    #02BF       ;mişcarea maşinii şi pentru
        POP     IX          ;citirea tastaturii prin ROM
        POP     AF
        POP     BC
        POP     DE
        POP     HL
        EI
        RET
BUFFER  DEFS    64          ; alocă spaţiu pentru buffer
MASINA  LD      A, #FE      ; IX conţine adresa POZ, pentru
        LD      IX, POZ     ; a putea fi folosit ca index
        IN      A, (#FE)
        AND     #01         ; Testează apăsarea simultană a taste-
        JP      NZ, DESEN;lor CS+9 (stînga) sau CS+SS
        LD      A, #EF      ;(dreapta), dacă nici una aceste
        IN      A, (#FE)    ;combinaţii nu este întîlnită
        AND     #02         ; desenează maşina nemişcată
        JR      Z,STG
        LD      A, #7F
        IN      A, (#FE)
        AND     #02
        JP      NZ, DESEN
DRT     CALL    CONTOR      ; cheamă contor temporizare
        JR      NC, DRT0    ; dacă imaginea este la dreapta,
        RES     7, (IX+0)   ;sare la DRT0. Dacă nu, se pozi -
        LD      A, (ROT)    ; ţionează indicatorii pentru poziţia la
        AND     #0F         ;dreapta (întoarcere) şi desenează
        JP      NZ, DESEN
        LD      (IX+1), 4
        DEC     (IX+2)
        JP      DESEN
DRT0    BIT     2, (IX+1)   ; stabileşte indicatorii pentru poziţia
        JR      NZ, DRT1    ; următoare spre dreapta
        INC     (IX+1)
        JP      DESEN
DRT1    LD      A, (ADR)    ; Se incrementează adresa de afişaj,
        CP      #1B         ; numai dacă nu se iese din limite
        JP      Z,DESEN
        INC     (IX+2)
        LD      (IX+1), 1
        JP      DESEN
STG     CALL    CONTOR
        JR      C, STG0
        SET     7, (IX+0)   ; la fel ca la dreapta, dar se
        BIT     2, (IX+1)   ; poziţionează indicatorii pentru
        JP      Z,DESEN     ; la stînga
        LD      (IX+1),0
        INC     (IX+2)
        JP      DESEN
STG0    LD      A, (ROT)
        CP      0
        JR      Z,STG1
        DEC     (IX+1)
        JP      DESEN
STG1    LD      A, (ADR)
        CP      1
        JP      Z, DESEN
        DEC     (IX+2)
```

```
        LD      (IX+1), #03
DESEN   LD      A, (POZ)
        LD      HL, DATED
        SLA     A
        JR      NC, DES0
        LD      HL, DATES
DES0    LD      DE, BUFFER
        CALL    DES1
        PUSH    HL
        LD      A, (POZ)
        AND     #03
        JR      Z, DES3
        LD      D,4
        RR      A
        JR      C,DES2
        LD      B,8
DES2    LD      DE, BUFFER+31
        LD      HL, BUFFER+30
        PUSH    BC
        LD      BC,31
        LDDR
        POP     BC
        DJNZ    DES2
DES3    POP     HL
        LD      A, (POZ)
        AND     #03
        JR      Z, DES4
        LD      BC,24
        ADD     HL,BC
        RR      A
        JR      C, DES4
        ADD     HL, BC
DES4    LD      DE, BUFFER+32
        CALL    DES1
        LD      A, (ROT)
        SLA     A
        AND     #0F
        JR      Z, DES7
        LD      C,A
DES5    LD      B,64
        LD      HL, BUFFER
DES6    RR      (HL)
        INC     HL
        DJNZ    DES6
        DEC     C
        JR      NZ, DES5
DES7    LD      HL, (ADR)
        LD      DE, BUFFER
        CALL    DES8
        LD      HL, (ADR)
        LD      DE,32
        ADD     HL, DE
        LD      DE, BUFFER+32
DES8    LD      B,8
        EX      DE, HL
DES9    PUSH    DE
        PUSH    BC
        LD      BC,4
        LDIR
        POP     BC
        POP     DE
        INC     D
        DJNZ    DES9
        RET
CONT    DEFB    3
CONTOR  LD      A, (CONT)
        DEC     A
        AND     #03
        LD      (CONT), A
        JR      Z,CONT0
        POP     HL
        JP      DESEN
CONT0   LD      A, (POZ)
        INC     A
        AND     #83
        LD      (POZ), A
        SLA     A
        RET
DES1    XOR     A
        LD      B,8
DES 10  PUSH    BC
        LD      BC,3
        LDIR
        LD      (DE), A
        INC     DE
        POP     BC
        DJNZ    DES10
        RET
```

Vă propunem să descifraţi singuri subrutina de "DE-SEN-are". Pentru aceasta, iată caracteristicile modului de stocare a datelor: maşina se reprezintă pe ecran pe spaţiul a 6 caractere, 3 pe orizontală şi 2 pe verticală. În memorie stocarea este realizată pe linii, adică prima linie de biţi a celor 3 caractere este urmată de a doua etc. (este ca şi cum am avea un "caracter" mare, cu fiecare linie de lungime 3 octeţi, 24 de biţi). Cele 3 caractere de sus nu se modifică, ele sînt doar translatate pe verticală, iar pentru partea de jos avem 3 poziţii intermediare.

Indicatorul "POZ" specifică pe primii 2 biţi poziţia intermediară a maşinii (specifică partea de jos care se afişează) şi este incrementat modulo 4, iar ultimul bit specifică orientarea (0 - dreapta, 1-stînga). Indicatorul "ROT" specifică pe primii 3 biţi numărul de "shift-uri" spre dreapta înmulţit cu 2 (la fiecare mişcare maşina se deplasează cu 2 biţi). Valorile pot fi 0, 1, 2, 3, pentru stînga şi 1,2,3,4 pentru dreapta.

Locaţia "ADR" specifică adresa de afişare pe ecran.

Succes!

■

"SPARGEREA" PROGRAMELOR

Bogdan Georgescu
Bogdan Iordănescu

Cam ce-ar putea însemna un asemenea titlu?Iată o scurtă explicaţie: ne propunem să abordăm un alt mod de a cîştiga experienţă în programarea pe SPECTRUM, şi anume însuşirea experienţei altora. Ne vom transforma deci (numai pentru un timp ...) în "piraţi" şi vom încerca să vedem care sînt posibilităţile de a "sparge" protecţia unui program utilitar sau a unui joc, mai ales cînd autorul programului a făcut tot ce a ştiut mai bine pentru a ne împiedica să o facem. Tocmai datorită varietăţii protecţiilor, a pătrunde în "interiorul" unui program protejat înseamnă de fapt a pătrunde în modul de gîndire al celui care l-a scris. A sparge un program este trea- bă de imaginaţie, dar necesită şi o cunoaştere veritabilă în programare.

Ar fi cazul să trecem la lucruri concrete. Ideea de bază ar fi de a urmări pas cu pas rularea programului, adică de a face un fel de simulare controlată a acestuia.

În ordinea încărcării, primul bloc al unui program este, de obicei, un bloc de BASIC, mai mult sau mai puţin protejat, care poate conţine şi cod maşină sub diferite forme. Ne vom ocupa de acest bloc din toate punctele de vedere.

De obicei, BASIC-ul foloseşte la trecerea din interpretor la nivelul codului maşină, deci listarea urmăreşte stabilirea eventualilor parametri care se transmit codului, a modului în care se încarcă blocurile de cod (dacă acest lucru se face cu LOAD" "CODE), cît şi a adresei de lansare.

O metodă de încărcare şi listare a BASIC-ului este cea prin intermediul programelor utilitare (COPY 86/M cu opţiunea "B", SPION etc.), dar după părerea noastră, controlul integral asupra a ceea ce se întîmplă în program nu poate fi obţinut decît prin scrierea unor rutine proprii.

O metodă mai puţin sigură, utilă doar în cazurile fericite cînd BASIC-ul nu este protejat prea puternic, este încărcarea cu LOAD"", după care programul se poate opri cu BREAK. Dacă la BREAK programul se blochează, reacţionează ciudat sau deloc, se poate încerca încărcarea lui cu MERGE"". Această încercare poate eşua dacă programul este protejat împotriva încărcării cu această instrucţiune (vom explica cum se poate face aceasta). În cazul încărcării cu LOAD, există riscul ca înainte de a se apăsa BREAK să fi intrat în funcţiune o rutină care să fi modificat sau mascat părţi din program. Dacă acest lucru nu se observă la timp, se poate rata totul.

Dacă după cîteva jonglerii cu caracterele de control, "intuiţia" vă spune că, totuşi, e prea simplu ca să fie adevărat, vă recomandăm o metodă de privire a BASIC-ului prin perspectiva codului maşină şi a controlului asupra memoriei, care duce la rezultate sigure.

Pentru aceasta este necesar să aveţi în memorie utilitarele GENS3M21 şi MONS3M21. Să încărcăm GENS-ul la adresa 40 000 şi MONS-ul la 55 000, după ce am introdus în prealabil CLEAR 39 999.

Acum să scriem şi să asamblăm următoarea rutină în GENS:

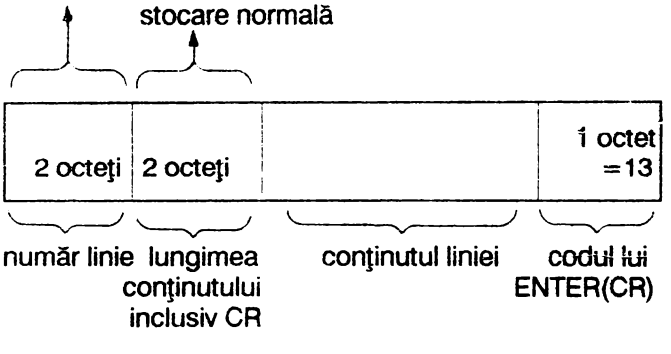
```
        ORG     54 900    ; de exemplu!
HEADER  LD      IX, 30 000; header-ul se încarcă
        LD      DE, 17    ;la adresa 30 000 şi
        SCF               ;are lungimea 17 baiţi
        XOR     A
        CALL    #0556
BLOC    LD      IX, 30100 ; programul propriu-zis se
        LD      DE, lung  ;încarcă la adresa 30 100
        SCF               ;şi are lungimea determinată
        LD      A, #FF    ; din header sau cu un copier
        CALL    #0556
        RET
```

Această rutină încarcă întîi header-ul BASIC-ului, din care se pot afla următoarele date:

— octetul 00: tipul fişierului(în cazul BASIC, conţine 0)
— octeţii 01 - 0A: numele porogramului
— octeţii 0B-0C: lungime program + variabile
— octeţii 0D-0E: linia de autolansare a programului
— octeţii 0F-10: lungime BASIC fără variabile

Partea de program cu numele (*label*-ul) "BLOC" încarcă fişierul BASIC ca pe un fişier headerless. Dacă se cunoaşte modul de stocare în memorie a BASIC-ului şi variabilelor, se poate realiza un alt fel de citire a acestuia, fără a apela la interpretor, deci avînd control deplin asupra situaţiei (în general protecţiile la BASIC folosesc particularităţile rutinei de listare din ROM).

Adresa de început a programului BASIC se ia din variabila de sistem PROG, la adresa 23635-23636 şi în mod normal (la SPECTRUM-ul fără extensii) este 23 755. După această adresă se stochează programul BASIC format din linii aranjate una după alta în ordinea crescătoare a numerelor de linie. Structura unei linii BASIC în memorie este următoarea:



De reţinut că numărul de linie este stocat în ordine inversă faţă de celelalte constante întregi pe 16 biţi (adică octetul cel mai semnificativ este primul).

Pentru a vedea cum este stocat conţinutul liniei, să introducem următorul program BASIC care să listeze conţinutul memoriei:

```
10 FOR N=23 755 TO 30 000
20 IF PEEK N>1 THEN PRINT N, PEEK N; TAB 22;
   CHR$, N PEEK N:NEXT M
30 PRINT N, PEEK N: NEXT N
```

Linia 20 exclude posibilitatea opririi listing-ului în cazul întîlnirii unor caractere neprintabile (cu coduri mai mici de 32).

Iată rezultatul rulării acestui program:

23 755	0	
23 756	10	
23 757	27	
23 758	0	
23 759	235	FOR
23 760	78	N
23 761	6	=
23 762	50	2
23 763	5	3
23 764	55	7
23 765	53	5
23 766	53	5
23 767	14	
23 768	0	
23 769	0	
23 770	203	THEN
23 771	92	/
23 772	0	
23 773	204	TO
23 774	5	3
23 775	48	0
23 776	48	0
23 777	48	0
23 778	48	0
23 779	14	
23 780	0	
23 781	0	
23 782	48	0
23 783	117	4
23 784	0	
23 785	13	

.....

Observăm acum, de fapt, cum arată în memorie prima linie a programului anterior. Se observă modul de stocare a numărului de linie (10) şi a lungimii liniei (27).

Constantele numerice care apar în linie sînt stocate întîi ca şir (fiecare cifră este stocată ca un caracter), pentru a fi listate uşor, apoi urmează caracterul de control 14 şi reprezentarea în virgulă mobilă, pe 5 octeţi, care este folosită de interpretor la rularea BASIC-ului. Numărul 203 de la adresa 23 770 nu reprezintă codul cuvîntului cheie "THEN" ci face parte din reprezentarea floating-point a numărului 23 755.

Acestea fiind spuse, să revenim la spargerea jocurilor şi să aplicăm cunoştinţele despre BASIC în cazul jocului "SPLITTING".

Acest joc este compus din 3 blocuri:

— primul bloc de BASIC, cu lungimea de 150 octeţi şi autolansare la linia 10

— al doilea bloc de BASIC, cu lungimea de 735 octeţi şi autolansare la linia 0

— al treilea bloc, care este un fişier headerless, de lungime 49 152

Aceste date se cunosc de la copierea programului. Neavînd nimic în memorie, încercăm să încărcăm primul „bloc cu LOAD"". La BREAK, observăm că nu se întîmplă nimic grav, dar ecranul rămîne alb, ceea ce înseamnă o protecţie de culoare. Cu instrucţiunea "INK 0" problema este rezolvată şi se poate citi BASIC-ul, care conţine doar un mesaj care ne anunţă că BORDER-ul rămîne negru pe durata încărcării programului, instrucţiunea "INK 7" care a cauzat "albirea" ecranului şi instrucţiunea LOAD"", ceea ce înseamnă că primul bloc nu are nici o importanţă, deoarece este şters la încărcarea următorului.

Faptul că BORDER-ul rămîne negru la încărcarea programului, ca şi faptul că blocul de cod este headerless ne indică faptul că programul foloseşte o rutină proprie de încărcare. Inexistenţa unui alt bloc de cod arată că rutina în cod maşină este inserată în al doilea BASIC. Şansele ca această rutină să poată fi oprită cu BREAK sînt minime, deci încercăm instrucţiunea MERGE. Vedem cu această ocazie cam cum acţionează o protecţie la MERGE, şi anume blocarea sistemului. Această protecţie este realizată de altfel foarte simplu: se scrie BASIC-ul într-o singură linie şi în locaţiile care conţin lungimea liniei se încarcă valoarea 0. Cum rutina de MERGE din ROM foloseşte lungimea fiecărei linii pentru ordonarea programului, se înţelege de ce apare blocajul. Această modificare nu afectează rularea programului, în el neexistînd instrucţiuni de salt etc. Se mai poate proceda şi altfel pentru a încărca BASIC-ul: se încarcă într-un copier (COPY 86/M), se dezactivează linia de autorun (cu opţiunea R, în COPY 86/M), se salvează pe casetă şi apoi se încarcă cu LOAD, fără ca programul să mai intre în execuţie. În cazul SPLITTING-ului, veţi avea o surpriză neplăcută dacă veţi încerca această metodă.Vom arăta imediat de ce.

Să aplicăm varianta prezentată anterior. Vom avea BASIC-ul încărcat la adresa 30100. Cu modificări ale intervalului de adrese în programul BASIC de listare a memoriei (ciclul FOR- NEXT de la 30 100 la 40 000, deoarece limita superioară nu ne interesează) vom putea vedea cum arată acest BASIC.

Observăm că numărul de linie este 0, lungimea liniei este tot 0 (...), după care urmează caracterele de control 17, urmat de 1 şi 16, urmat de 1, ceea ce înseamnă PAPER 1 şi INK 1. Urmează RANDOMIZE USR 24070, care este prima instrucţiune BASIC executată. Rezultă că tot ce urmează ca instrucţiuni BASIC (şi care ocupă, probabil 24 070 - 23 760 = 310 octeţi) sînt pentru "derută". S-ar putea ca în acest interval să fie şi instrucţiuni de cod maşină, dar asta vom vedea în continuare. Listînd mai departe, observăm caracterul de control 9, care mută cursorul cu un caracter la stînga, şi repetat va cauza "acoperirea" instrucţiunii "RANDOMIZE". Urmează schimbarea culorii în vizibil (17,1, apoi 16,6 şi 19,1 adică PAPER1, INK 6 şi BRIGHT1) şi un mesaj-rugăminte din partea autorului: "PLEASE DON'T STEAL THIS COPY...", care apoi ne face "favoarea" de a ne indica un RANDOMIZE USR 25 000, pentru a ne "uşura" munca. Iată de ce

PARADOX 3.5

Paradox 3.5 este, aşa cum arată şi numele, un paradox: un SGBD(sistem de gestiune a bazelor de date), relaţional(deci ceva sofisticat), uşor de învăţat şi utilizat. El permite: **1.** organizarea informaţiei în tabele(relaţii), datele fiind alfanumerice, numerice, currency(se referă doar la sume de bani), dată; **2.** o tabelă poate conţine pînă la 2 milioane de înregistrări, cu pînă la 4000 de caractere fiecare: 255 cîmpuri cu pînă la 255 caractere fiecare; 3. informaţia stocată poate fi schimbată, regăsită în orice formă dorită, incluzînd efectuarea de calcule asupra ei, precum şi reprezentări grafice. **Paradox** implementează strălucit QBE(*Querry by Exemple*). Se dă pur şi simplu un exmplu de răspuns dorit şi se obţin toate răspunsurile adecvate. De notat că într-o singură"querry" pot fi legate pînă la 24 tabele. Fiecare tabelă poate avea ataşate pînă la 15 formate speciale de formatare a datelor şi tot pînă la 15 forme diferite de aşezare a lor în rapoarte. Bineînţeles, atît formulele cît şi rapoartele pot conţine informaţii din tabele diferite. **4. Paradox** este un produs cu precădere vizual. Totul, începînd cu proiectarea rapoartelor, formatelor, QBE-urilor şi terminînd cu utilizarea pe scară largă a meniurilor, se produce pe ecran. 5. **Paradox** poate fi folosit pe o reţea locală la fel de uşor ca de către un singur utilizator(fără a se utiliza comenzi specisale). Se asigură blocarea automată a fişierului sau înregistrării. **6.** Se pot importa/exporta date din/către alte programe:123, dBase,Reflex. Se asigură o legătură strînsă cu Quattro Pro 2.0. **7.** Se pot memora secvenţe de taste care ulterior pot fi executate. **8.** Accesul la tabele poate fi controlat printr-un mecanism de codificare, asigurîndu-se protecţia datelor. **9.** Utilizînd PAL(*Paradox Application Lan*guage), limbaj structurat, se creează un mediu de programare permiţînd dezvoltarea de aplicaţii.

Quattro Pro

Q.P., Versiunea 2.0 vă oferă:

— legături dinamice între 64 foi de calcul diferite, aflate în memorie sau pe hard disc.

— o capacitate de calcul sporită. Datorită tehnologiei VROOM (Virtual Real Time Object Oriented Memory Manager) Q.P. poate încărca şi utiliza foi de calcul conţinînd o mare cantitate de date.

— o grafică deosebită. Puteţi alege între mai multe reprezentări grafice suprafeţe, linii, XY, diagrame, inclusiv tridimensionale.

O bibliotecă de simboluri grafice vă va permite o prezentare deosebită a graficelor obţinute. Fiecare obiect poate fi deplasat, colorat, redimensionat. Există şi posibilitatea de a crea o înlănţuire de imagini, realizînd un "show" de diapozitive. În aceeaşi direcţie s-a prevăzut importul (exportul) unor noi tipuri de fişiere grafice: EPS (Encapsulated PortScript), PCX, CGM (primele două formate pot fi exportate, ultimul numai importat). Veţi putea alege din 9 fonte diferite. Dispuneţi de asemenea de o funcţie de pre-vizualizare, adevărat WYS/WIG.

— funcţia de "goalseek", permiţînd determinarea valorii unor celule care sînt cuprinse într-o formulă pornind de la rezultatul dorit.

— cuplarea strînsă cu un alt produs remarcabil al firmei Borland: Paradox 3.5. Utilizatorii lui Paradox 3.5 care dispun de 2 Mo de memorie şi de o maşină pe bază de 80286 pot lansa în execuţie Q.P. din Paradox 3.5. Astfel o tabelă Paradox va putea fi folosită de Q.P. fără a mai fi încărcată în prealabil în foaie. Şi dacă în plus dispuneţi şi de accesul la o bază de date amplasată pe un mini sau pe un server, Paradox 3.5 şi Paradox SQL Link permiţîndu-vă accesul prin intermediul cererilor SQL, veţi putea asigura o prezentare deosebită a analizelor dumneavoastră utilizînd facilităţile lui Q.P.

listarea obişnuită nu prea era recomandabilă. Mai departe, de la adresa 30 226 pînă la 30 414 sînt octeţi 0 (NOP) care ne liniştesc în privinţa acestei zone.

Nu trebuie să vă deruteze adresa 24 070. Ea este valabilă în cazul cînd programul ar fi fost încărcat de la 23 755. În acest caz, trebuie considerat "deplasamentul relativ", care este de 24 070-23 755=315, deci adresa reală va fi 30 100 + 315= 30 415.

În continuare apelăm MONS-ul şi dezasamblăm de la adresa 30 415, unde vom avea următoarele instrucţiuni.

```
LD      HL, 24 090
LD      DE, 41 320
LD      BC, 400
LDIR
JP      41 320
NOP
....
```

ceea ce arată că se transferă un bloc de 400 octeţi la adresa 41320 şi se predă controlul acestui bloc.

Dar am intrat deja în domeniul codului maşină, cu care vom continua în numărul următor. ■

Once upon the time "THE UNIX"

Ion Muşlea

UNIX - portretul unui standard

În momentul de faţă preţul unui WORK-STATION al firmei SUN este echivalent cu cel al unei interfeţe grafice de tip WINDOWS şi a cîtorva produse soft existente sub DOS (1-2-3, *WordPerfect, Ventura).* Şi totuşi unii beneficiari ezită încă să aleagă o maşină performantă de tipul celei amintite. Aceasta din mai multe raţiuni: pe de o parte complexitatea **UNIX**-ului sperie un utilizator obişnuit cu DOS-ul sau MACINTOSH-ul, iar pe de altă parte există celebra problemă a compatibilităţii diverselor versiuni **UNIX**. Altfel zis, un utilizator de *WordPerfect* îşi poate schimba liniştit PC-ul avînd garanţia că procesorul de texte va funcţiona la fel de bine şi pe noua maşină, pe cînd cel ce foloseşte Word5 trebuie să aibă grijă să achiziţioneze un **UNIX SCO**, căci *Word5* nu funcţionează pe un **UNIX SUN**.

E adevărat că **UNIX**-ul nu e standardizat la nivel binar (ca DOS sau OS/2) ci la nivel sursă (adică la nivelul limbajului ales în dezvoltarea produsului), dar acesta este unicul "punct negru" al acestui sistem de operare. În rest, forţa unui **UNIX** ce permite de exemplu portabilitatea unui software cum ar fi Oracle pe sute de maşini de orice tip (micro, mini sau mainframe) ar cam trebui să dea de gîndit. Şi apoi nu trebuie uitat faptul că în epoca reţelelor de calculatoare, raza de acţiune a **UNIX**-ului (mult mai mare decît în cazul DOS sau OS/2) l-ar putea impune drept soluţie definitivă.

Azi, lupta pentru standardizarea **UNIX**-ului se dă practic între două grupări: de o parte este OSF-ul (IBM, DEC, *Hewllet-Packard*), iar de cealaltă este *Unix International* (AT&T, SUN,FUJITU). În aparenţă s-ar zice că fiecare din ele "trage" în altă direcţie, dar în fapt se pare că forţa comitetelor de normalizare apărute sub presiunea utilizatorilor este mai mare decît cea a fabricanţilor. Căci e unanim recunoscut faptul că "legea în **UNIX**" o face X/OPEN GROUP care a ales POSIX ... şi POSIX a rămas.

Avînd în vedere că atît OSF-ul cît şi **UNIX INTERNATIONAL** au acceptat X/OPEN şi POSIX, putem spune, fără teama de a greşi, că **UNIX**-ul se îndreaptă spre o mult aşteptată unificare. De remarcat şi tendinţa tuturor "barosanilor" de a se ralia la **UNIX**. Acest lucru are două explicaţii. În primul rînd dorinţa guvernului american (care e cel mai important beneficiar) de a se axa în domeniul militar pe acest sistem; în al doilea rînd studiile arată că perspectivele răspîndirii **UNIX**-ului sînt foarte mari. Astfel, în perioada 1987-1990, în America, s-a înregistrat o creştere cu 20% a pieţei **UNIX**, iar în Europa cu 40% !!! Conform unui studiu IDC, se preconizează ca, în 1992, **UNIX**-ul să reprezinte 18% din cifra de afaceri în domeniul informaticii pe plan mondial! Şi apoi să nu uităm că **UNIX**-ul oferă incomparabil mai multe alternative de lucru superioare celor existente sub DOS, mai ales în domeniul industrial şi ştiinţific.

Iar dacă încercăm să facem un top al celor mai avansate dezvoltări de soft, pe primul loc îl vom găsi pe *NextStep* care — să fie oare o întîmplare ? - rulează sub **UNIX**. Şi, în sfîrşit, cele mai sofisticate 4 baze de date existente (*Sybase, Oracle, Ingres şi Informix*) şi-au făcut — toate !!! — debutul sub **UNIX**. Acest fapt a permis ca numai pentru maşinile de tip IBM RISC 6 000 să avem peste 500 de "tools"-uri necesare în aplicaţii de gestiune.

În încheiere mai trebuie subliniat că maşinile **UNIX** prezintă un excepţional raport preţ-performanţă. Chiar şi în comparaţie cu facilităţile grafice oferite de DOS sau OS/2. În condiţiile în care oferă o viteză şi nişte aplicaţii excepţionale, s-ar prea putea ca **UNIX**-ul să ofere şi soluţia de viitor a PC-urilor.

UNIX — Povestea unui standard

UNIX-ul "s-a născut" în laboratoarele BELL TELEPHONE la finele anilor 60. Povestea sa începe odată cu colaborarea unui oarecare Ken Thompson la un proiect de sistem intitulat MULTICS destinat unui calculator de tip GE45 al firmei GENERAL ELECTRIC. Pentru testarea acestui proiect el a conceput un software numit "SPACE TRAVEL" care, deşi lipsit de orice utilitate comercială, era de-a dreptul captivant. După un timp BELL TELEPHONE abandonează proiectul MULTICS, fapt ce-l aruncă pe Thompson în cea mai neagră disperare. Dar el nu abandonează şi, împreună cu un alt inginer pe nume Dennie Ritchie, hotărăşte să-şi continue studiile. Cei doi vor descoperi un DEC vechi (cu un PDP-7) complet neutilizat, şi hotărăsc să adapteze "SPACE TRAVEL" pe această maşină. În acest scop ei au dezvoltat şi nişte rutine de bază ale unui sistem de exploatare.Acestea au constituit prima versiune **UNIX**, scrisă în întregime în limbaj de asamblare PDP-7. Premiera a avut şi un iz de clandestinitate pentru că de cînd cu abandonarea MULTICS, subiectul a devenit tabu în cadrul laboratoarelor BELL.

Thompson şi Ritchie au avut o idee care i-a determinat să persevereze în această direcţie; erau convinşi că ar fi formidabil să existe un sistem care să permită utilizatorilor să-şi "mute" programele pe diverse maşini. Aici s-ar cuveni o paranteză; în acea vreme fabricanţii de calculatoare aveau obiceiul de a oferi doar sisteme "proprietar", adică de aşa natură încît, să fie incompatibile cu soft-ul alfor firme; iar acest fapt îi obligă pe informaticieni să se cantoneze în arhitecturi date.

Datorită limitelor ansamblorului, Thompson a decis să dezvolte un limbaj de nivel mai ridicat care să permită portabilitatea **UNIX**- ului pe diverse maşini. Noul limbaj va fi terminat în 1970 şi va purta pe "B". În anul următor **UNIX** este codificat în "B" şi este instalat pe un DEC PDP-11. Tot-otodată apar la BELL şi primele aplicaţii practice ale noului sistem, printre care şi un procesor de texte multiuser. Thompson şi Ritchie profită de ocazie pentru a publica prima documentaţie de **UNIX**.

Aşa a luat naştere esenţa filosofiei portabilităţii. Sistemul e prezentat ca multi-tasking, multi-user şi evolutiv (adică permite crearea unor noi utilitare care să permită la rîndu-le apariţia altor utilitare). Dar înainte de toate **UNIX** se vrea portabil.

ONCE UPON THE TIME "THE UNIX"

Ritchie e de părere că limbajul "B" trebuie să fie îmbunătăţit de aşa manieră încît să permită cele mai diverse aplicaţii şi să ofere atît un sistem de exploatare cît şi compatibilitate. în această sarcină, Ritchie este ajutat de un programator ce se numea ... Kernigan. Cei doi vor definitiva un limbaj de nivel înalt ce va face epoca; "C"-ul. În 1973, inginerii de la BELL vor rescrie în întregime **UNIX**-ul în "C".

Se părea că totul era pregătit pentru ca **UNIX** să invadeze piaţa informatică. Dar, din păcate, în 1956 BELL TELEPHONE semnase un document prin care se angaja (faţă de guvernul american) să nu "iasă" pe piaţa informaticii; de aceea **UNIX** a fost folosit la început doar pentru uzul intern al laboratoarelor BELL.

Norocul **UNIX**-ului a constat în faptul că pasiunea "universitarilor" l-a ajutat să scape din "turnul de fildeş" ce-i fusese impus. "Minunea" s-a produs în momentul publicării unei expuneri despre **UNIX** în *"Communications of ACM"*. Acest articol a produs un adevărat potop de scrisori din partea cercetătorilor dornici să descopere ciudatul sistem ce permitea portabilitatea programelor pe orice tip de maşină. Pentru a răspunde acestei cereri spontane, BELL LABs distribuie gratuit cîteva exemplare de **UNIX V6** mai multor universităţi. Acestea vor primi de la BELL cîte o bandă magnetică împreună cu dreptul de a face orice fel de modificări; dar BELL nu oferea nici garanţie, nici asistenţă.

În 1974 universitatea Berkley devine unul din cele mai importante centre de dezvoltare **UNIX** pe DEC PDP-11. Se vor crea multe utilitare ce vor contribui substanţial la maturizarea sistemului. Astfel au luat naştere un editor de texte (VI), un sistem de gestiune a reţelelor şi a memoriei virtuale. Începînd din 1979, la Berkley, cercetările vor continua pe maşini DEC de tipul VAX. Nu e deci de mirare faptul că primul **UNIX** comercializat de DEC în 1980 — ULTRIX — este derivat dintr-o versiune BSD (*Berkley Software Developpment*).

Anul 1977 va fi o piatră de hotar pentru UNIX: universitatea WOLLOGONG din Australia reuşeşte în chip strălucit instalarea unui **UNIX** pe un calculator de tipul INTERDATA 8/32, maşina pe 32 de biţi foarte diferită de PDP-11. Această reuşită a contribuit enorm la răspîndirea ideii că în sfîrşit există un sistem de exploatare perfect portabil.

Pentru prima dată **UNIX**-ul a fost comericalizat de firma ONYS SYSTEMS care, neputînd achiziţiona **UNIX**-ul, s-a mulţumit cu licenţa de utilizare a versiunii V7. Maşina ONYX a fost prezentată în 1980 la "National Computer Conference" şi a avut un asemenea succes încît exemplul a fost urmat de alte firme. Pentru început ZYLOG propune o serie de "micro" de 16 biţi (S 8 000) bazată pe un **UNIX** propriu, rebotezat ZEUS. Febra **UNIX** se întinde cuprinzînd rînd pe rînd tot mai multe firme, PLEXUS, ALTOS, FORTUNE, NCR, TOWER, etc.

La începutul anilor '80, **UNIX** devine foarte popular în lumea cercetătorilor şi inginerilor. În mai toate colţurile lumii apar asociaţii de "fans" ai sistemului. Unele dintre ele, cum ar fi *FreeSoft Foundation*, dezvolta gratuit sau la preţuri modice diverse utilitare, neurmărind realizarea vreunui profit. "UNIX- iştii" vor crea de asemenea o mesagerie internaţională unde fiecare indică problemele pe care nu le poate rezolva; de regulă, în scurt timp, se primesc soluţii sugerate de alţi utilizatori. Această bunăvoinţă, total lipsită de interese comerciale constituie un fapt rarisim în "moravurile" informaticii acelor vremuri.

În schimb, piaţa profesionistă cunoaşte alt gen de evoluţie. AT&T-ul vinde licenţe de utilizare la nişte preţuri destul de piperate. Doar şcolile şi universităţile continuă să beneficieze de tarife mai abordabile, fapt ce duce la o continuă creştere a nivelului de popularitate a **UNIX**-ului în rîndul studenţilor.

În 1981 AT&T ia două decizii majore. În primul rînd va publica SYSTEM III (prima versiune **UNIX** destinată în exclusivitate pieţei comerciale) şi se angajează să furnizeze asistenţa tehnică necesară. De asemenea, posesorii de SYSTEM III sînt liberi să-l redistribuie altor editori sau constructori, după ce i-au adus propriile ameliorări. În al doilea rînd AT&T a redus semnificativ preţul licenţelor (în jur de 20 000 $). Cam în această perioadă tînăra societate Microsoft va publica propria-i adaptare de SYSTEM III. Dar efectul e departe de cel aşteptat căci o maşină oarecare, pe nume . . . IBM-PC, va lansa un alt standard numit MS-DOS cu care va monopoliza aproape toată energia MICROSOFT- ului. Succesul comercializării de sisteme multi-user pentru PC- uri va aparţine societăţii SANTA CRUZ OPERATION (care însumează şi azi 90% din vînzările din acest domeniu).

Cu toate acestea **UNIX**-ul este încă prea scump. Pentru a evita acest incovenient unii editori vor lansa un "**UNIX look-alikes**", adică sisteme foarte asemănătoare cu **UNIX**. Cele mai cunoscute variante sînt *Unos, Coherent* şi UNETIX. Aceste sisteme simulează modul de lucru al **UNIX**-ului, oferind acelaşi set de comenzi şi putînd executa aceleaşi programe. Fiind create fără ajutorul AT&T, aceste **UNIX** *look-alikes* nu intră sub incidenţa dreptului de autor al acestei firme, fapt ce a dus la o rapidă răspîndire a lor. Un alt factor ce a accelerat succesul **UNIX**-ului a fost apariţia primelor staţii de lucru ale firmei APOLLO şi SUN. În scurt timp, cea de-a doua firmă se va remarca prin asocierea versiunii **UNIX BSD** cu o interfaţă grafică de tip MacIntosh, inovaţie ce va pulveriza recordurile de creştere stabilite de APPLE.

În aceste condiţii de proliferare a diverselor tipuri de **UNIX**, deşi există o oarecare compatibilitate, vor apărea diverse confuzii păgubitoare. Nevoia unui standard se face tot mai mult simţită; iniţiativa va porni de la un grup de utilizatori (USR/group), cu care se vor asocia în timp tot mai mulţi producători.

În mod firesc BELL încearcă să preia controlul **UNIX**-ului şi a standardizării lui; în acest scop va lansa în ianuarie 1983 **UNIX SYSTEM V**. Această nouă versiune preia cea mai mare parte a îmbunătăţirilor aduse de universitate Berkley, îmbunătăţiri deja bine ancorate în mediile ştiinţifice. SYSTEM V va deveni primul punct de referinţă cu adevărat oficial. În plus în 1984 V va deveni primul punct de referinţă cu adevărat oficial. În plus în 1984 Ministerul Justiţiei din S.U.A. anulează decretul privind interzicerea accesului firmelor de comunicaţii pe piaţa informaticii. BELL se va împărţi practic în mai multe societăţi; dintre acestea AT&T va "moşteni" **UNIX**-ul, fapt ce-l va face să-şi dezvăluie clar intenţia de a controla atît norma, cît şi comercializarea ei.

ONCE UPON THE TIME THE UNIX

Anul 1985 va găsi UNIX-ul instalat pe mai mult de 100000 de maşini dintre cele mai variate (de la CRAY la IBM PC). Mai mulţi constructori de calculatoare, dintre care şi DEC şi HEWLETT- PACKARD, confruntaţi cu pericolul monopolizării unei pieţe de asemenea dimensiuni, adresează IEEE-ului o propunere (ca şi cum ar veni din partea unui grup de utilizatori) de adoptare a unui standard numit POSIX. În aceeaşi perioadă în Europa ia naştere X/OPEN GROUP (format din BULL, OLIVETTI, SIEMENS şi PHILIPS) care, la rîndul său adopta drept standard tot POSIX-ul. Realizînd pericolul, AT&T îşi "apropie" universitatea Berkley în scopul obţinerii unei convergenţe a celor mai răspîndite versiuni de UNIX.

În urma tratativelor va lua naştere SYSTEM V versiunea 3.0. În anul următor MICROSOFT şi SCO îşi manifestă la rîndul lor intenţia unificării UNIX-ului; această hotărîre va duce la unificarea SYSTEM V 3.0, XENIX şi BSD 4.2/4.3. În sfîrşit, în 1987, AT&T anunţă fuzionarea lui SYSTEM V cu POSIX, măsură menită să-i întărească controlul asupra viitorului sistem.

Dar motivele de îngrijorare ale producătorilor de calculatoare nu încetează să sporească; AT&T semnează un acord cu SUN prin care acesta din urmă devine un partener privilegiat (SUN va fi avantajat prin primirea preversiunilor UNIX şi a difuzării informaţiilor).

Marea ruptură apare în 1988, cînd, dacă ne luăm după surse de IBM, principalii producători nu au obţinut nici un rezultat în urma parlamentarilor cu AT&T în domeniul ieftinirii licenţelor. Ba chiar se zice că AT&T ar fi declarat — cu riscul de a intra în contradicţie cu comenzile efectuate — că va decide după bunu-i plac evoluţia sistemului său. Mai trebuie specificat faptul că IBM-ul investise deja sume considerabile în dezvoltarea unui UNIX propriu destinat generaţiei PS/2. Acesta poartă numele de AIX şi elimină cîteva slăbiciuni ale versiunilor AT&T (în special la nivelul securităţii şi a gestiunii discurilor).

În mai 1988 cîţiva din marii producători mondiali (IBM, HEWLETT PACKARD, DEC, BULL, APOLLO) anunţă formarea OSF (OPEN SOFTWARE FOUNDATION), un organism independent şi puternic susţinut de fondatorii săi (mai ales printr-o donaţie de 120 milioane dolari). Oficial, membrii acestui grup vor căuta să scape de sub tutela AT&T. După cum era de aşteptat, OSF va alege cel mai bine pus la punct UNIX al momentului, adică IBM AIX ...

Pentru a contrabalansa OSF-ul, AT&T şi SUN — împreună cu alţi 50 de producători (printre care FUJITSU, OLIVETTI şi INTEL) — vor fonda UNIX INTERNATIONAL care va alege drept standard SYSTEM V.4. Ca o curiozitate, se poate aminti faptul că există unele companii ce au aderat la ambele grupări: CONCURENT COMPUTER, INTEL, TOSHIBA ...

Azi OSF-ul are 85 de membri, 200 de angajaţi şi a definitivat prima versiune proprie: OSF.1 (publicată în noiembrie 1990). Sponsorii săi s-au angajat să-l finanţeze timp de trei ani, după care OSF va trebui să se descurce singur. Modul de lucru al OSF e foarte original: înainte de a dezvolta un produs, se organizează un "concurs" de oferte şi se aleg cele mai avantajoase. Astfel, în OSF/1 vom regăsi multe aspecte ale lui IBM AT/X, dar nucleul provine de la MACH — o "dezvoltare" aparţinînd universide tăţii CARNEGIE-MELLON.

Spre deosebire de OSF, UNIX INTERNATIONAL se limitează la a adresa sfaturi unui compartiment al AT&T (numit USL) care continuă să facă cercetări asupra dezvoltării UNIX-ului.

În clipa de faţă există, prin urmare, doi pretendenţi la standardizarea UNIX-ului. Ignorînd lupta pentru influenţă şi luînd în considerare doar faptele, se remarcă un lucru interesant: cele două încercări de standardizare sînt foarte asemănătoare datorită faptului că ambele au la bază POSIX. Unicele diferenţe apar la nivelul interfeţei grafice care, la OSF poartă numele de MOTIF (realizare DEC şi HP), iar la UNIX INTERNATIONAL se numeşte OPEN LOOK (realizare SUN şi XEROX). Dar chiar şi aici există o oarecare compatibilitate, ambele avînd la bază standardul X-WINDOWS al celebrului M.I.T.

La nivel de utilizator final diferenţele esenţiale apar la DESKTOP (interfaţa generală). Cu alte cuvinte, o situaţie similară cu cea a GEM şi WINDOWS din DOS: adică cine a folosit VENTURA într-una din aceste două "suprafeţe" se va obişnui foarte repede şi cu cealaltă.

O ultimă etapă consta în faptul, că multă, vreme UNIX-ul a păstrat compatibilitatea doar la nivel de surse (la trecerea de pe un sistem pe altul sursele trebuiau recompilate). Noua tendinţă consta în a găsi o compatibilitate binară în genul celei existente sub DOS. În august 1990 INTEL s-a angajat să definească o serie de teste care să permită o astfel de compatibilitate a UNIX-urilor. SCO şi AT&T îşi dau de asemenea tot interesul în obţinerea unei astfel de norme. În ziua cînd un software scris pentru UNIX se va putea executa pe orice versiune PC a acestui sistem de operare, indiferent ce e de provenienţa SCO, INTERACTIVE SYSTEMS, INTEL sau AT&T, UNIX-ul nu va mai avea nici un punct slab în comparaţie cu DOS-ul sau OS/2 . În schimb va avea o mulţime de avantaje...

UNIX - interfeţe grafice

O maşină pe care există un X-WINDOWS poate genera ferestre de lucru pe mai multe calculatoare de tipuri total diferite. O astfel de proprietate ne face să ne gîndim imediat la portabilitatea UNIX-ului. Acest standard grafic a luat naştere la începutul anilor 1980 la M.I.T. şi a fost conceput ca un "server" de ferestre destinat funcţionării în reţea. Din 1984 M.I.T. scoate acest produs pe piaţă, permiţînd astfel oricărui producător lipsit de resursele financiare necesare dezvoltării unei interfeţe grafice proprii să şi-l procure la comandă. Cum ideea este foarte inteligentă, s-a impus de la sine.

Cu toate acestea meritul popularizării interfeţelor grafice sub UNIX aparţine lui SUN. Totul a început în 1984 prin dezvoltarea produsului SUNWiew. Impactul acestui WORK-STATION, împletind confortul unui Macintosh cu execuţia multi-tasking, a contribuit substanţial la evoluţia UNIX-ului spre un "look" grafic (de notat faptul că din punct de vedere al "barbariei" comenzilor UNIX-ul întrece cu mult DOS-ul).

X-WINDOWS nu este însă suficient de "tare" pentru a realiza performanţe comparabile cu cele ale lui FINDER (la Mac) sau ale managerului de fişiere din WINDOWS). Cu alte cuvinte îi lipseşte o definire exactă a modului de interacţionare a utilizatorului cu diversele ferestre şi icoane. Cu toate acestea, standardul M.I.T. oferă elemente de construcţie adecvate pentru medii mai evoluate. În ianuarie 1987, DEC, APOLLO şi HEWLETT PACKARD îşi anunţă intenţia folosirii versiunii X-WINDOWS în vigoare: X11. Imediat li s-au alăturat IBM, SUN şi consorţiul de standardizare X/OPEN.

UNIX INTERNATIONAL şi-a ales drept interfaţa grafică OPEN LOOK-ul lui SUN şi XEROX, ale cărui kit-uri

au apărut în 1988. Din soft- ul de birou ce foloseşte această interfaţă merită amintite LOTUS 1-2-3, Word Perfect şi Wingz.

La rîndul său OSF-ul a fost consecvent principiului concursului de oferte, al cărui cîştigător a fost de asta dată produsul MOTIF al firmelor DEC şi HP. Ca o particularitate trebuie să menţionăm comportarea absolut identică cu cea a lui WINDOWS şi PRESENTATION MANAGER (chiar şi la nivel de rol al tastelor funcţionale), ceea ce asigură compatibilitatea cu cele mai răspîndite interfeţe grafice existente pe PC-uri.

Chiar dacă interfeţele de programare ale lui MOTIF şi OPEN LOOK diferă, ele pot coexista în aceeaşi reţea graţie lui X-WINDOWS. Aşadar o staţie SUN aflată sub OPEN LOOK poate crea o fereastră pe o maşină IBM RISC 6 000, chiar dacă aceasta din urmă lucrează sub MOTIF. Staţia SUN va acţiona ca un "server" de ferestre şi programul OPEN LOOK se va executa pe calculatorul *Big Blue*. E posibilă însă şi situaţia inversă.

Dacă comparăm ecranul OPEN LOOK cu MOTIF, nu se remarcă decît diferenţe "cosmetice". Practic deosebirile apar doar la nivel de detalii ale utilizării curente. Pentru folosirea OPEN LOOK e necesar un mouse cu trei butoane, pe cînd la MOTIF e suficient unul de tip PC. **UNIX INTERNATIONAL** impune acelaşi "desktop" (interfaţa generală — nivelul superior al interfeţei, care va permite lansarea diverselor programe sau gestiunea fişierelor) la toate versiunile OPEN LOOK, pe cînd OSF a preferat să le lase mînă liberă diverşilor producători (astfel pe staţiile RISC 6 000 se foloseşte AIX/WINDOWS, iar pe HP-uri vom găsi NEW WAVE).

În ceea ce priveşte PC-urile există două desktop-uri: X.DESKTOP (al IXI) şi LOOKING GLASS (al VISIX SOFTWARE). Ele au reuşit să convingă cele două firme aflate în competiţie: SCO şi INTERACTIVE. Ambele oferă medii **UNIX** complete pe PC-uri dotate cu 80 386; versiunile se numesc OPEN DESKTOP şi ARCHITECT.

SCO însumează 70% din vînzările de **UNIX** pentru 80 386, în timp ce mai tînărul INTERACTIVE se mulţumeşte cu doar 25%. Ambele sisteme propuse par să aibă şanse mari să concureze OS/2 şi WINDOWS, deşi la prima vedere necesarul de resurse e de natură să îngrozească un "DOS-ist", căci au nevoie de cel puţin 6 Mo RAM şi 100 Mo de hard disk. Aceasta "sete" de resurse e compensată în schimb de unele avantaje incontestabile: interfaţa grafică de înaltă ţinută, posibilităţi de conectare la reţea, deschidere către aplicaţii DOS, compatibilitate în mod text cu **UNIX**-ul clasic, iar în cazul SCO chiar şi o bază de date relaţională deloc "de lepădat" (INGRES). Dar "clou"-ul constă în posibilitatea de a executa SIMULTAN aplicaţii DOS şi XENIX în ferestre diferite. Bineînţeles că cele două sisteme sînt compatibile (de fapt conţin — în mare — aceleaşi componente). Gama de aplicaţii e tot mai largă şi aduce îmbunătăţiri substanţiale în raport cu echivalentele de pe PC. Dintre best-seller-ele DOS regăsim WORD 5, *WordPerfect* şi LOTUS 1-2-3.

În ciuda tuturor celor afirmate mai sus, cea mai reuşită interfaţă grafică nu se bazează pe această normă, ci pe *Display PostScript* al firmei ADOBE. E vorba de *NextStep*, care se găseşte atît pe staţiile NEXT cît şi pe IBM RISC 6 000 (opţional). În afara unei estetici ce te lasă cu gura căscată, *NextStep* se dovedeşte a fi mult mai avansată decît MOTIF şi OPEN LOOK, mai ales prin faptul că dispune de o *Interface Builder* (un "tool" care în cîteva minute permite crearea unei interfeţe cu orice aplicaţie, în condiţiile unui lux rafinat). Importanţa *NextStep*-ului reiese şi din faptul că editorii se raliază fără rezerve la acest mediu. Bazîndu-se pe aceste proprietăţi LOTUS are un nou tablor ce permite descrierea formulelor în limbaj natural, iar Ashton-Tate realizează *PowerStep* (tot un tablor) ce permite adnotarea celulelor prin mesaje vocale.

În paralel QUARK e pe cale să-şi adapteze PAO Xpress în sensul utilizării la maxim a capacităţilor pe multi-tasking real, iar ADOBE îşi adaptează produsul de desenat ILLUSTRATOR. În plus societatea IMAGINE pune la punct aplicaţii multi-0media tot pe *NextStep*. O asemenea de poziţie a marilor producători de soft riscă să aducă *NextStep* în situaţia MacIntosh-ului anilor 80: incompatibil cu PC-ul, dar capabil să se impună prin forţa atuurilor sale. Anunţul recent al realizării versiunii color a acestui mediu e susceptibil să-i aducă pînă şi adeziunile celor mai retincenţi. De altfel, IBM a decis să-şi doteze maşinile RISV 6 000 cu versiunea color a *NextStep*.

În concluzie, putem afirma că, atîta vreme cît **UNIX**-ul va fi mai puţin răspîndit decît DOS-ul, duelul dintre ele se va da indirect prin intermediul aplicaţiilor disponibile. Nu vom putea neglija însă NextStep-ul, care are mari şanse să devină noua referinţă în domeniul informaticii de vîrf. ■

Ultima oră

1 aprilie. *Novell Inc, luînd în considerare performanţele extrem de ridicate ale calculatorului Felix (C256, 512,1024) a hotărît lansarea urgentă a unei cooperări cu întreprinderea de profil, care va culmina cu includerea nucleului de servicii Net Ware (Portable Net Warw) în sistemele de operare Siris, Helios, etc. În felul acesta vom avea pe lîngă servere MacIntoch, DOS, OS/2, UNIX şi servere Siris, Helios, etc... O nouă tinereţe pentru acestea, precum şi o binevenită recunoaştere a calităţilor lor deosebite.*

16 aprilie. *Novelle Inc anunţă achiziţionarea unei părţi a USL (Unix System Laboretories), sucursală AT&T care a dezvoltat sistemul de operare UNIX. Investiţia se referă la o parte din cele 20%-30% din proprietatea USL puse la dispoziţie de către AT&T către 11 concerne din industria calculatoarelor.*

Novell s-a aliniat şi la tehnologia de reţea UNIX realizînd încă din 1983 produse de comunicaţie pe protocolul TCP/IP, pentru calculatoare rulînd sub UNIX. Produsele furnizate de Novell asigură servicii Net Ware pe calculatoare Host cu UNIX fiind disponibile la Altos, Data General, HP, MIPS, NCR, Prime, Unisys, Wang. Cu începere din martie 1991, Novell distribuie Net Ware NFS (Network File System, protocol de transport) ca un serviciu, permiţînd utilizatorilor unei staţii UNIX să partajeze resursele amplasate pe un server Net Ware cu utilizatori DOS, Windows, OS/2, MacIntosh. În curînd mergînd pe principiul utilizării Portable NetWare calculatoarele UNIX vor putea să ofere servicii Net Ware.

SOSESC PRODUSELE CASE(II)

Cureleţ-Bălan Gheorghe

Produsele program CASE pot fi clasificate după mai multe criterii; astfel, dacă ne referim la etapele din ciclul de viaţa cărora le sînt destinate, putem să le clasificăm în "front-end tools" — sau "upper CASE" (vizează etapele de începutale ciclului de viaţă) şi "back-end tools" —sau "lower CASE" (destinate etapelor de sfîrşit ale ciclului de viaţă). O altă clasificare este relativă la domeniul de folosire al sistemelor soft dezvoltate cu ajutorul lor. Putem să le clasificăm astfel în produse CASE destinate aplicaţiilor de gestiune economică şi în produse CASE destinate domeniului tehnico-ştiinţific. Desigur am putea să le clasificăm şi după tipul de calculatoare pe care rulează: mari, medii, micro.

Vom face o prezentare a principalelor facilităţi oferite de produsele CASE urmărind prima clasificare.

Din clasa produselor "front-end" menţionăm în primul rînd pe cele suport în analiză/proiectare. Acestea implementează tehnicile de analiză/proiectare structurată fiind de fapt editoare grafice care suportădiferite metodologii de analiză/proiectare. Sistemul soft în faza de analiză este modelat printr-o mulţime de diagrame care reprezintă funcţionalităţile acestuia, fluxul de date, interfeţele etc. Sistemele CASE evoluate permit producerea unei game largi de diagrame prin utilizarea unei multitudini de simboluri tipizate şi suprapunerea facilităţilor de diagramare cu capacităţile de procesare text ceea ce permite adnotarea diagramelor sau introducerea unor diagrame în text. De asemenea se fac verificări privind sintaxa componentelor grafice folosite. Deşi iniţial produsele din aceste categorii se limitau doar la facilităţile de diagramare, în scurt timp ele au încorporat posibilităţi de verificare a consistenţei şi completitudinii. Aceasta se realizează de regulă prin dicţionarele de date în care se înregistrează informaţii despre tipul datelor, cerinţele de folosire etc. Astfel se previne folosirea inconsis-tentă a anumitor elemente de proiectare putîndu-se obţine în acelaşi timp informaţii complete privind folosirea aces-tora — de exemplu un proiectant poate determina rapid toate componentele unei structuri de date,sau toate procesele ce folosesc o anumită dată. În plus, existenţa dicţionarului permite generarea unei game variate de rapoarte privind proiectul software.

Sistemele CASE evoluate din această categorie oferă facilităţi multiutilizator destinate proiectării în echipă pe suportul unei reţele locale de PC-uri a unui minicalculator sau calculator mare. Dintre produsele reprezentative menţionăm: *Software through Pictures* al firmei Interactive Development Environments; *Design/2.0, Design/IDEF* ale firmei Meta Software Corporation; *Planning/Analysis/ Design Workstation* al firmei KnowledgeWare, *Excelerator* al firmei Index Technologies, *Teamwork* al firmei Cadre Technologies etc.

O altă categorie de produse CASE strîns legată de prima o reprezintă instrumentele CASE de *prototipizare*. Conceptul nu e nou, el a făcut întotdeauna parte din procesul de dezvoltare în domeniul ingineriei. *Prototipizarea* este folosită în primul rînd pentru a valida un concept şi a identifica erorile de proiectare prin crearea unui model al sistemului aflat în curs de elaborare. *Prototipizarea* rupe tradiţia de a considera desfăşurarea secvenţială a etapelor ciclului de viaţă adăugînd ca o necesitate iteraţia primelor trei faze (analiză, proiectare, codificare). Prin confruntarea utilizatorului cu prototipul sistemului soft, se realizează feedback-ul necesar în etapele de analiză/proiectare în urma căruia ies la iveală aspecte care sînt dificil de dedus pe hîrtie. Avantajele *prototipizării*, relativ la productivitate, cost, calitate sînt evidente. Deşi aparută destul de recent, *prototipizarea* beneficiază deja de mai multe metodologii care îngreunează însă eforturile de standardizare; dintre acestea amintim: RIPP —*Rapid Iterative Production Prototyping*, SDLC/R —*Rapid System Development Life Cycle;* metodologia DEC, etc.

Generatoarele automate de cod apar de cele mai multe ori integrate fie în produsele "front-end" fie în cealaltă categorie, ele fiind legate de prototipizarea, întreţinerea şi refolosirea software-ului. Pentru a exemplifica această interdependenţă vom prezenta facilităţile cîtorva produse de referinţă. Firma Cortex Corp. oferă produsul *Corvision* (destinat mediilor VAX/VMS) care suportă fazele de proiectare, codificare şi întreţinere din ciclul de viaţă. Interfaţa grafică produce o specificaţie de proiectare care serveşte ca intrare în generatorul automat de cod în limbajul de generaţia 4, *Builder* (conceput de firmă). Se apreciază că în felul acesta se generează automat 85-95 % din codul aplicaţiei, restul fiind completat de utilizatorul produsului.

Firma Pansophic Systems Inc. acoperă jumătate din piaţa produselor "lower CASE", cu generatorul de cod Telon destinat calculatoarelor mari. Un alt exemplu este produsul *Ne-tron/Cap Development Center* al firmei Netron Inc. Produsul ajută în proiectarea, prototipizarea, dezvoltarea şi întreţinerea aplicaţiilor COBOL, încorporînd tehnologia Basett Frame, care constă în folosirea de componente COBOL adaptabile şi refolosibile numite *"frames"*. În felul acesta 80-90 % dintr-o aplicaţie poate fi generată din frame-uri; întreţinerea software- ului fiind simplificată considerabil.

Pe lîngă creşterea productivităţii de elaborare, generatoarele de cod oferă avantajul simplificării întreţinerii software-ului, asigurîndu-i în acelaşi timp o fiabilitate crescută.

Considerăm că generatoarele de cod vor prolifera în viitor avînd în vedere atît creşterea continuă a complexităţii produselor program oferite elaboratorilor de software cît şi tendinţa de apropiere a acestora de utilizatorul neiniţiat. Pentru a exemplifica acestea vom prezenta facilităţile altor generatoare de cod. CASE W al firmei Caseworks, SUA este destinat elaborării aplicaţiilor dezvoltate sub interfaţa MS Windows. Necesitatea elaborării produsului a fost justificată de volumul mare de muncă impus de elaborarea sub interfaţa MS Windows a unor aplicaţii relativ simple.

CASE W permite programatorului să-şi definească elementele specifice ale interfeţelor pentru programul său, după care generează codul C care implementează în MS

Windows aceste elemente. În codul C generat, comentarii speciale indică locul în care detaliile specifice aplicaţiei vor fi plasate de programator.

De fapt **CASE W** generează "şablonul" (cadrul) de cod C care reprezintă dependenţa aplicaţiei de sistemul de gestiune al interfeţelor MS Windows. Facem observaţia ca aceeaşi firmş a elaborat un produs similar destinat elaborării aplicaţiilor sub interfaţa *Presentation Manager* (PM) a sistemului de operare OS/2.

UI Programmer, produs al firmei Wallsoft Systems, SUA, este destinat programatorilor în d'BASE. Pornind de la observaţia ca anumite aspecte ale codificării în d'BASE au un caracter rutinier şi sînt consumatoare de timp, elaboratorii produsului degrevează programatorul de astfel de activităţi prin generarea automată a codului cores-punzător. În acest fel programatorul se poate concentra asupra aspectelor creative ale dezvoltării aplicaţiilor, cum ar fi proiectarea ecranelor, a structurii bazei de date etc.

Produsul constă din cinci componente: editor de ecran, dicţionar de date, şabloane program, limbaj pentru şabloane, generator de cod. Activităţile rutiniere sînt înregistrate în sistem sub forma unor şabloane (*"template"*) care apoi sînt selectate de utilizator. Programatorul, cu ajutorul editorului de ecran, îşi defineşte forma ecranului din programul său (în paralel cu aceasta se creează dicţionarul de date), apoi selectează şablonul potrivit prelucrării pe care intenţionează s- o facă. Sistemul *UI* combină imaginea ecran definită împreună cu şablonul selectat generînd codul d'BASE corespunzător.

Spre deosebire de alte generatoare de cod d'BASE (cum ar fi Genifer, care-i destinat neprogramatorilor) *UI* este destinat pentru a creşte productivitatea programatorilor în d'-BASE.

Menţionăm deasemenea că în prestigioasa revistă *Infoworld* din 16 oct. '89 a fost semnalat faptul că firma Ashton-Tate a anunţat realizarea unui generator de aplicaţii în d'BASE numit *Crystal.*

O altă direcţie o reprezintă generatoarele de interfeţe. Reprezentativ în acest sens este sistemul *"C-scape"* şi modulul său de proiectare interactivă a ecranelor de aplicaţii "Look & Feel", ale firmei Oakland Group Inc., SUA. Ele dispun de biblioteci puternice de lucru cu ferestre, meniuri, validări de date, help-uri, funcţii de editare avînd o proiectare orientată către obiect. Lucrul cu aceste sisteme se finalizează prin generarea automată a unui cod C flexibil care implementează intefaţa proiectată.

Tot în categoria generatoarelor de cod pot fi incluse translatoarele interlimbaje şi produsele de reinginerie. Acestea din urmă se referă la redezvoltarea sistemelor software foarte complexe a căror întreţinere devine costisitoare. Redezvoltarea înseamnă desprinderea elementelor de proiectare pornind de la sursă şi recodificarea într-o manieră structurată în vederea facilitării întreţinerii. Produsele din această categorie se adresează în general utilizatorilor limbajului COBOL; dintre cele mai reprezentative amintim: *RETROFIT* al firmei CATALYST, *RECORDER* al firmei Language Technology, *Data Analyst* al firmei Bachman Information Systems Inc.

Translatoarele interlimbaje reprezintă instrumente software de mare productivitate în migrarea sistemelor software complexe. Există realizări de translatoare între diferite versiuni şi dialecte ale aceluiaşi limbaj (firmele Computer Associates şi Sector 7 pentru limbajul COBOL, firma SOFTOOL pentru FORTRAN), precum şi translatoare între limbaje diferite (FORTRAN-C, COBOL- C, PASCAL-C, FORTRAN-ADA (firma Rapitech Systems Inc.), BASIC-C (firma Sector 7, Anglia), etc.). Au apărut chiar translatoare asembler-limbaj înalt, cum este cazul translatorului Xtran al firmei Pennington Systems Inc., destinat translatării asembler VAX-C.

Instrumentele de gestiune a modificărilor şi a configuraţiei identifică toate componentele care consituie aplicaţia şi relaţiile între ele, asigurînd sincronizarea între componente şi controlul modificărilor efectuate asupra acestora. Produsele din această categorie sînt extensii logice ale celor de gestiune a versiunilor, disponibile mai devreme în mediile de programare ale calculatoarelor mari. În prezent produsele din această categorie acoperă toate tipurile de calculatoare, produsele reprezentative fiind: CCC (*Change and Configuration Control*) al firmei Softool, *Endevor* al firmei Business Software Technology, CMS (*Code Management System*), MMS (*Module Management System*) al firmei DEC, etc.

Documentaţia ocupă un rol esenţial în procesul soft- ware, intervenind practic în toate cele şase etape ale ciclului de viaţă. În cadrul sistemelor software complexe volumul ei e atît de mare încît gestiunea manuală a acesteia este practic imposibilă. Se citează astfel ca documentaţia de proiectare a sistemului software destinat navetei spaţiale americane a constat în mai mult de 200 000 de pagini, şi ca în cadrul firmei Boeing se produce anual un volum de două miliarde de pagini, în cel puţin 56 tipuri diferite, de documentaţie pentru software. Un sistem de gestiune automată a documentaţiei trebuie în primul rînd să o păstreze sub forma unei baze de date care să permită controlul modificărilor aduse diferitelor tipuri de documente şi care în acelaşi timp să verifice consistenţa acestora, în sensul că modificările aduse unui document să fie reflectate şi în documentele legate de acesta dar să nu altereze relaţiile şi documentele nemodificate. Pe lîngă faptul că un astfel de sistem ar putea sta la baza unor instrumente de asigurare şi control ale calităţii procesului software (verificarea conformităţii cu anumite standarde — de analiză, proiectare, codificare , etc.) rolul său esenţial intervine în etapa de întreţinere cînd modificările propuse afectează de fapt tot volumul de documentaţie elaborat în etapa de realizare. Realizări în acest domeniu există deja; putem cita astfel produsul *DOCUMENTOR* al firmei Context Corp. — folosit de firma Boeing în cadrul mediului software integrat SWIE (*Software Integrated Environment).*

După cum s-a văzut produsele CASE încearcă să degreveze elaboratorul de software de unele aspecte rutinare specifice activităţii de elaborare. Dezvoltate iniţial în scopul automatizării unei etape din ciclul de viaţă al produsului program produsele **CASE** au evoluat în direcţia dezideratului de integrare şi a altor etape. Produsele "front-end" evoluează spre înglobarea facilităţilor celor din categoria "back-end" şi invers. Aceasta face ca în prezent să asistăm la o concurenţă acerbă în impunerea unui standard în domeniu. Despre toate acestea însă într-un număr viitor. ■

Începînd de astăzi inaugurăm o nouă rubrică care îşi propune să treacă în revistă majoritatea pachetelor de programe disponibile pe piaţă.
Astăzi:

UTILITARE PENTRU GESTIUNEA MEMORIEI

Nume: **386 Max Professional**
Producător: **Qualitas**
Preţ: **300 $**
Descriere: Optimizează performanţele unui 80386.
Se instalează în CONFIG.SYS, ocupînd doar 4 ko.
Transformă o parte din memoria extinsă în memorie expandată (LIM.4.0). Transferă programele rezidente în memoria înaltă, eliberînd memoria convenţională. Transferă conţinutul memoriei ROM (lentă ca timp de răspuns) în memoria RAM. Se obţine o accelerare pentru funcţiile BIOS de exemplu. Necesită 80386.

Nume: **Above Disc**
Producător: **AB Soft**
Preţ: **250 $**
Descriere: Produsul asigură crearea unei memorii expandate conformă cu specificaţia LIM 4.0 pornind de la:
- în cazul calculatoarelor fără extensie de memorie, memoria se creează utilizînd discul floppy sau discul hard.
- în cazul calculatoarelor fără placă de extensie de memorie, avînd plantată pe placă de bază memorie suplimentară, aceasta este convertită conform specificaţiei LIM 4.0.

De notat că în această situaţie frecvenţa la care se lucrează este de 25 Mhz (corespunzător plăcii de bază) şi nu de 8 Mhz corespunzător magistralei de extensie în cazul plăcilor de memorie conectate separat.
Pentru a se exemplifica eficienţa produsului se recurge la Lotus 123 Versiunea 2.01; cu produsul instalat se pot încărca în memorie 4 816 linii din foaia de calcul faţă de 448 linii în cazul absenţei produsului (memorie de 640 k).
De notat utilitatea în cazul PC-urilor portabile care nu dispun de sloturi de extensie.

Nume: **Move 'Em**
Producător: **Qualitas**
Preţ: **200$**
Descriere: Permite încărcarea în memoria înaltă a programelor rezidente, a driverelor DOS, a driverelor de reţea. Comportă o funcţie de optimizare a memoriei. Necesită plăci de memorie compatibile cu specificaţia EMS 4.0.

Nume: **PC - Kwik prower**
Producător: **Multisoft**
Preţ: **320 $**
Descriere: Este un produs care optimizează lucrul cu hard-discul.. Utilizează algoritmi de optimizare a scrierii pe disc, pe care îl gestionează rulînd în "background". Implementează de asemenea un "spooler" de imprimantă, un disc RAM, un accelerator de tastatură şi unul de ecran. Ocupă doar 10 k RAM şi funcţionează atît în memoria convenţională cît şi în cea expandată. Necesită 640 K RAM şi rulează pe XT/AT/PS/2.

Nume: **QRAM sau QEMM 50/60**
Producător: **QuarterDeck**
Preţ: **200 $**
Descriere: În funcţie de configuraţia de care dispuneţi, produsul QRAM va găsi orice fel de memorie pe care DOS nu o recunoaşte (EMS, EEMS, XMS) şi va instala acolo toate programele rezidente precum şi driverele existente în memorie eliberînd zona de memorie convenţională. Pentru un PC echipat cu ecran VGA sau EGA, dar pe care nu se rulează aplicaţii necesitînd o grafică deosebită, se eliberează 96 Ko memorie. Integrează produsul MANIFEST, dînd o imagine exactă a utilizării memoriei. QEMM 50/60 este versiunea adoptată pentru PS/250 şi 60.

Nume: **HEAD ROOM**
Producător: **HELIX**
Preţ: Versiunea single-user: **240 $**
versiunea 8 utilizatori: **460 $**
versiunea n utilizatori: **950 $**
Descriere: Gestionează programele rezidente, putînd să le stocheze pe disc, în memoria extinsă sau expandată. De asemenea, încarcă mai multe programe sub forma unei liste care poate fi baleiată circular prin apăsarea unei singure taste, programul selectat intrînd în execuţie. Gestionează pînă la 32 Mo memorie accesibilă DOS. Versiunea reţea permite instalarea soft-ului de reţea în memoria expandată (EMS 4.0). Produsul funcţionează pe maşinile 8088-80486.

Nume: **QEMM 386**
producător: **QUATERDECK**
Preţ: **195 $**
Descriere: Asigură conversia unei părţi a memoriei extinse în memorie expandată. Optimizează funcţionarea calculatorului, detectînd viteza de acces la memorie şi recurgînd ori de cîte ori este posibil la memoria rapidă (să nu uităm că accesul la RAM pe placa de bază se face la 25 Mhz iar pentru memoria extinsă la 8 Mhz. Produsul execută programele rezidente, eventualele "spooler" în memoria înaltă. Produsul permite utilizarea unui 80386 sub forma a mai multe maşini virtuale 8086 precum şi utilizarea produsului DESKWIEW 2.2 pentru implementarea lucrului în multitasking. Integrează produsul MANIFEST oferind o imagine exactă a memoriei. Necesită procesor 80 386.

Marius Sturzoiu

ADISAN
11 ,Arh.Ion Mincu str.
Phone: 14.54.55
15.81.65
COMERCIAL DEP.

Adisan SRL este o societate înfiinţată în martie 1990 de un grup numeros de specialişti de vîrf - în domeniul tehnicii de calcul şi comunicaţie de date, pentru a oferi tuturor specialiştilor în acest domeniu o alternativă la numeroasele oferte primite din străinătate. Societatea are în prezent 16 filiale în Bucureşti şi în întreaga ţară şi întreţine legături cu un mare număr de parteneri străini ,dintre care cel mai important este **IBM** (prin **SOCODIAG ASIST - IBM** Franţa) prim agent naţional în Romania. Prin experţii proprii **ADISAN** prospectează în permanenţă piaţa mondială de microcalculatoare depistînd cea mai avantajoasă ofertă intermediind contracte între producătorii de microcalculatoare şi beneficiari,negociind achiziţionarea cu plata în lei sau valută în funcţie de posibilităţi.

Departamentul SERVICE vă oferă service cu cei mai buni specialişti ai tehnicii de calculatoare, la toate gamele de echipamente produse în ţară: **Felix C256/512/104, Felix 5000, I-100/102F/106, Coral, Felix PC, Cub Z, Junior XT,** etc, precum şi la toate echipamentele **PC-IBM** compatibile provenite din import pe care le achiziţionaţi prin noi. Centre mari de calcul din Bucureşti şi din ţară sînt în service la noi (ex: CINOR, Electromagnetica, C.Teritorial de calcul Cluj, Sf.Gheorghe, Focşani, I.U.G.-Craiova, I.U.C.Ploieşti,etc).

Departamentul Comunicaţii vă oferă soluţii de informatizare şi comunicaţii globale legate de intreprinderea dumneavoastră.

Departamentul soft şi reţele de calculatoare vă asigură instalarea şi configurarea de reţele de **PC-IBM** compatibile precum şi soluţii informatice soft sau hard pentru întreprinderea dumneavoastră, incluzînd şi instruirea personalului. Se asigură 1 an garanţie hard pentru lucrările executate, precum şi maintenance soft şi hard pe bază de contract.

Departamentul Comercial vă pune la dispoziţie o ofertă completă în lei şi valută de echipamente de calcul şi telecomunicaţii. Începînd cu cele mai pretenţioase echipamente din punct de vedere al dinamicii, preţului şi calităţii.

Societatea ADISAN vă oferă, prin **Editura ADISAN** , informaţii de specialitate în revista **"PC-MAGAZIN"**, culegere de text, machetare pe calculator şi imprimantă **LASER JET**, pentru orice tip de publicaţii-ziare, reviste, cărţi, etc.

SOCIETATEA ADISAN *ESTE* **NR.1** *ÎN SERVICE PRIVAT PENTRU TEHNICA DE CALCUL DIN ROMÂNIA!*

- SERIOZITATE - COMPETENŢĂ - PROMPTITUDINE -

Cu deosebită stimă
Director general
Ing. A. Babin

ADISAN

ARTICOL	SPECIFICATII	HERCULES	VGA MONOCROM	VGA COLOR	SVGA COLOR	SVGA MSYNC
1028612040xx0	DESKTOP 286/12, 1MB RAM, 1.2MB sau 1.44MB FDD, 40MB HDD	259403	281344	356951	369997	375631
10286120403x0	DESKTOP 286/12, 1MB RAM, 1.2MB si 1.44MB FDD, 40MB HDD	282826	304767	380375	393124	398758
1028616040xx0	DESKTOP 286/16, 1MB RAM, 1.2MB sau 1.44MB FDD, 40MB HDD	267705	289646	365253	378299	383933
10286160403x0	DESKTOP 286/16, 1MB RAM, 1.2MB si 1.44MB FDD, 40MB HDD	291128	312773	388380	401426	407060
1030016040xx0	DESKTOP 386SX/16, 1MB RAM, 1.2MB sau 1.44MB FDD, 40MB HDD	350132	372073	447680	460430	466063
10300160403x0	DESKTOP 386SX/16, 1MB RAM, 1.2MB si 1.44MB FDD, 40MB HDD	373259	395200	470807	483853	489487
1038620040xx0	DESKTOP 386/20, 1MB RAM, 1.2MB sau 1.44MB FDD, 40MB HDD	412990	434931	510538	523584	529218
10386200403x0	DESKTOP 386/20, 1MB RAM, 1.2MB si 1.44MB FDD, 40MB HDD	436413	458354	533962	546711	552345
11286120000x0	MINI DESKTOP 286/12, 1MB RAM, DISKLESS	179348	201289	276896	289942	295576
1128612000xx0	MINI DESKTOP 286/12, 1MB RAM, 1.2MB sau 1.44MB FDD	202771	224712	300320	313366	318999
11286120003x0	MINI DESKTOP 286/12, 1MB RAM, 1.2MB si 1.44MB FDD	224119	246060	321668	334417	340051
1128612040xx0	MINI DESKTOP 286/12, 1MB RAM, 1.2MB sau 1.44MB FDD, 40MB HDD	274821	296762	372369	385415	391049
11286120403x0	MINI DESKTOP 286/12, 1MB RAM, 1.2MB si 1.44MB FDD, 40MB HDD	296169	318110	393717	406763	412397
1128612100xx0	MINI DESKTOP 286/12, 1MB RAM, 1.2MB sau 1.44MB FDD, 100MB HDD	368515	390456	466063	478813	484446
11286121003x0	MINI DESKTOP 286/12, 1MB RAM, 1.2MB si 1.44MB FDD, 100MB HDD	389863	411507	487115	500161	505794
1128612180xx0	MINI DESKTOP 286/12, 1MB RAM, 1.2MB sau 1.44MB FDD, 180MB HDD	495120	517061	592669	605715	611348
11286121803x0	MINI DESKTOP 286/12, 1MB RAM, 1.2MB si 1.44MB FDD, 180MB HDD	516468	538409	614017	627063	632696
11286160000x0	MINI DESKTOP 286/16, 1MB RAM, DISKLESS	187650	209591	285198	298244	303878
1128616000xx0	MINI DESKTOP 286/16, 1MB RAM, 1.2MB sau 1.44MB FDD	211073	233014	308622	321371	327005
11286160003x0	MINI DESKTOP 286/16, 1MB RAM, 1.2MB si 1.44MB FDD	232421	254362	329970	342719	348353
1128616040xx0	MINI DESKTOP 286/16, 1MB RAM, 1.2MB sau 1.44MB FDD, 40MB HDD	283123	305064	380671	393717	399351
11286160403x0	MINI DESKTOP 286/16, 1MB RAM, 1.2MB si 1.44MB FDD, 40MB HDD	304471	326412	402019	415065	420699
1128616100xx0	MINI DESKTOP 286/16, 1MB RAM, 1.2MB sau 1.44MB FDD, 100MB HDD	376520	398461	474069	487115	492748
11286161003x0	MINI DESKTOP 286/16, 1MB RAM, 1.2MB si 1.44MB FDD, 100MB HDD	397868	419809	495417	508463	514096
1128616180xx0	MINI DESKTOP 286/16, 1MB RAM, 1.2MB sau 1.44MB FDD, 180MB HDD	503422	525363	600971	614017	619650
11286161803x0	MINI DESKTOP 286/16, 1MB RAM, 1.2MB si 1.44MB FDD, 180MB HDD	524770	546711	622319	635068	640702
11300160000x0	MINI DESKTOP 386SX/16, 1MB RAM, DISKLESS	270077	292018	367625	380671	386305

ARTICOL	SPECIFICATII	HERCULES	VGA MONOCROM	VGA COLOR	SVGA COLOR	SVGA MSYNC
1028612040xx0	DESKTOP 286/12, 1MB RAM, 1.2MB sau 1.44MB FDD, 40MB HDD	259403	281344	356951	369997	375631
1130016000xx0	MINI DESKTOP 386SX/16, 1MB RAM, 1.2MB sau 1.44MB FDD	293204	315145	390752	403798	409432
11300160003x0	MINI DESKTOP 386SX/16, 1MB RAM, 1.2MB si 1.44MB FDD	314552	336493	412100	425146	430780
1130016040xx0	MINI DESKTOP 386SX/16, 1MB RAM, 1.2MB sau 1.44MB FDD, 40MB HDD	365550	387491	463098	475848	481481
11300160403x0	MINI DESKTOP 386SX/16, 1MB RAM, 1.2MB si 1.44MB FDD, 40MB HDD	386898	408839	484446	497196	502829
1130016100xx0	MINI DESKTOP 386SX/16, 1MB RAM, 1.2MB sau 1.44MB FDD, 100MB HDD	458947	480888	556496	569542	575175
11300161003x0	MINI DESKTOP 386SX/16, 1MB RAM, 1.2MB si 1.44MB FDD, 100MB HDD	480295	502236	577844	590890	596523
1130016180xx0	MINI DESKTOP 386SX/16, 1MB RAM, 1.2MB sau 1.44MB FDD, 180MB HDD	585849	607790	683398	696147	701781
11300161803x0	MINI DESKTOP 386SX/16, 1MB RAM, 1.2MB si 1.44MB FDD, 180MB HDD	606901	628842	704449	717495	723129
11386200000x0	MULTI DESKTOP 386/20, 1MB RAM, DISKLESS	332935	354876	430483	443529	449163
1138620000xx0	MULTI DESKTOP 386/20, 1MB RAM, 1.2MB sau 1.44MB FDD	356358	378299	453907	466953	472586
11386200003x0	MULTI DESKTOP 386/20, 1MB RAM, 1.2MB si 1.44MB FDD	377706	399647	475255	488004	493638
1138620040xx0	MULTI DESKTOP 386/20, 1MB RAM, 1.2MB sau 1.44MB FDD, 40MB HDD	428408	450349	525956	539002	544636
11386200403x0	MULTI DESKTOP 386/20, 1MB RAM, 1.2MB si 1,44MB FDD, 40MB HDD	449756	471697	547304	560350	565984
1138620100xx0	MULTI DESKTOP 386/20, 1MB RAM, 1.2MB sau 1.44MB FDD, 100MB HDD	522102	544043	619650	632400	638033
11386201003x0	MULTI DESKTOP 386/20, 1MB RAM, 1.2MB si 1.44MB FDD, 100MB HDD	543153	565094	640702	653748	659381
1138620180xx0	MULTI DESKTOP 386/20, 1MB RAM, 1.2MB sau 1.44MB FDD, 180MB HDD	648707	670648	746256	759302	764935
11386201803x0	MULTI DESKTOP 386/20, 1MB RAM, 1.2MB si 1.44MB FDD, 180MB HDD	670055	691996	767604	780650	786283
1138620300xx0	MULTI DESKTOP 386/20, 1MB RAM, 1.2MB sau 1.44MB FDD, 320MB HDD	960625	982566	1058174	1071220	1076853
11386203003x0	MULTI DESKTOP 386/20, 1MB RAM, 1.2MB si 1.44MB FDD, 320MB HDD	983752	1005693	1081301	1094347	1099980
11386330000x2	MULTI TOWER 386/33, 64KB CACHE, 1MB RAM, DISKLESS	466063	488004	563612	576361	581995
1138633000xx2	MULTI TOWER 386/33, 64KB CACHE, 1MB RAM, 1.2MB sau 1.44MB FDD	489487	511428	587035	600081	605715
11386330003x2	MULTI TOWER 386/33, 64KB CACHE, 1MB RAM, 1.2MB si 1.44MB FDD	512910	534851	610459	623208	628842
1138633040xx2	MULTI TOWER 386/33, 64KB CACHE, 1MB RAM, 1.2MB sau 1.44MB FDD, 40MB HDD	561536	583477	659085	672131	677764
11386330403x2	MULTI TOWER 386/33, 64KB CACHE, 1MB RAM, 1.2MB si 1.44MB FDD, 40MB HDD	584960	606901	682508	695554	701188
1138633100xx2	MULTI TOWER 386/33, 64KB CACHE, 1MB RAM, 1.2MB sau 1.44MB FDD, 100MB HDD	655230	677171	752779	765825	771458
11386331003x2	MULTI TOWER 386/33, 64KB CACHE, 1MB RAM, 1.2MB si 1.44MB FDD, 100MB HDD	678357	700298	775906	788952	794585
1138633180xx2	MULTI TOWER 386/33, 64KB CACHE, 1MB RAM, 1.2MB sau 1.44MB FDD, 180MB HDD	781836	803777	879384	892430	898064
11386331803x2	MULTI TOWER 386/33, 64KB CACHE, 1MB RAM, 1.2MB si 1.44MB FDD, 180MB HDD	805259	827200	902808	915854	921487

ARTICOL	SPECIFICATII	HERCULES	VGA MONOCROM	VGA COLOR	SVGA COLOR	SVGA MSYNC
1028612040xx0	DESKTOP 286/12, 1MB RAM, 1.2MB sau 1.44MB FDD, 40MB HDD	259403	281344	356951	369997	375631
1138633300xx2	MULTI TOWER 386/33, 64KB CACHE, 1MB RAM, 1.2MB sau 1.44MB FDD, 320MB HDD	1093457	1115102	1190709	1203755	1209389
11386333003x0	MULTI TOWER 386/33, 64KB CACHE, 1MB RAM, 1.2MB si 1.44MB FDD, 320MB HDD	1116584	1138525	1214133	1227179	1232812
1138633650xx2	MULTI TOWER 386/33, 64KB CACHE, 1MB RAM, 1.2MB sau 1.44MB FDD, 650MB HDD	1267799	1289740	1365348	1378394	1384027
11386336503x2	MULTI TOWER 386/33, 64KB CACHE, 1MB RAM, 1.2MB si 1.44MB FDD, 650MB HDD	1290926	1312867	1388475	1401521	1407154
11486250000x3	MULTI TOWER 486/25, 128KB CACHE, 1MB RAM, DISKLESS	693182	715123	790731	803777	809410
1148625000xx3	MULTI TOWER 486/25, 128KB CACHE, 1MB RAM, 1.2MB sau 1.44MB FDD	716606	738547	814154	827200	832834
11486250003x3	MULTI TOWER 486/25, 128KB CACHE, 1MB RAM, 1.2MB si 1.44MB FDD	740029	761970	837578	850624	856257
1148625040xx3	MULTI TOWER 486/25, 128KB CACHE, 1MB RAM, 1.2MB sau 1.44MB FDD, 40MB HDD	788952	810893	886500	899250	904883
11486250403x3	MULTI TOWER 486/25, 128KB CACHE, 1MB RAM, 1.2MB si 1.44MB FDD, 40MB HDD	812079	834020	909627	922673	928307
1148625100xx3	MULTI TOWER 486/25, 128KB CACHE, 1MB RAM, 1.2MB sau 1.44MB FDD, 100MB HDD	882349	904290	979898	992944	998577
11486251003x3	MULTI TOWER 486/25, 128KB CACHE, 1MB RAM, 1.2MB si 1.44MB FDD, 100MB HDD	905773	927714	1003321	1016367	1022001
1148625180xx3	MULTI TOWER 486/25, 128KB CACHE, 1MB RAM, 1.2MB sau 1.44MB FDD, 180MB HDD	1009251	1031192	1106800	1119549	1125183
11486251803x3	MULTI TOWER 486/25, 128KB CACHE, 1MB RAM, 1.2MB si 1.44MB FDD, 180MB HDD	1032378	1054319	1129927	1142973	1148606
1148625300xx3	MULTI TOWER 486/25, 128KB CACHE, 1MB RAM, 1.2MB sau 1.44MB FDD, 320MB HDD	1320873	1342814	1418421	1431171	1436804
11486253003x3	MULTI TOWER 486/25, 128KB CACHE, 1MB RAM, 1.2MB si 1.44MB FDD, 320MB HDD	1344000	1365941	1441548	1454594	1460228
1148625650xx3	MULTI TOWER 486/25, 128KB CACHE, 1MB RAM, 1.2MB sau 1.44MB FDD, 650MB HDD	1495215	1517156	1592763	1605809	1611443
11486256503x3	MULTI TOWER 486/25, 128KB CACHE, 1MB RAM, 1.2MB si 1.44MB FDD, 650MB HDD	1518638	1540579	1616187	1628936	1634570
100102	NOTEBOOK 286/16, 1MB RAM, 1.44MB FDD, 40MB HDD	307436				
100103	NOTEBOOK 286/16, 1MB RAM, 1.44MB FDD, 80MB HDD	353393				
100202	NOTEBOOK 286/16 LAN-STATION, 1MB RAM, 1.44MB FDD	254362				

ATENŢIE LA VIRUŞI!

PROGRAM ANTIVIRUS: "CHKVIR.BAT"

Acest utilitar este o procedură de tip BATCH, ce se poate folosi de către utilizatorii de IBM-pc şi compatibile pentru a testa prezenţa sau absenţa unui virus în sistemul de calcul. De fapt, programul "CHKVIR.BAT" verifică integritatea dimensiunii şi a datei fişierului de comenzi DOS, COMMAND.COM, asupra căruia operează cele mai mulţi din viruşii cunoscuţi. Programul se introduce cu un editor de texte oarecare lansat din MS-DOS(de exemplu EDLIN), apoi se salvează pe floppy sau hard-disk, putînd fi utilizat ca atare prin simpla apelare a numelui său, ori se poate include în fişierul propriu AUTOEXEC.BAT, fiind executat la începutul fiecărei sesiuni de lucru. "Cheia" acestui utilitar se află în liniile 2 şi 9, în care se creează fişierul temporar "fisier1", în care, prin intermediul comenzii externe FIND, care trebuie să existe pe disc, se încarcă(prin directiva "fisier1"), mărimea şi respectiv data standard a lui COMMAND.COM.

Dacă programul descoperă diferenţe între mărimea fişierului COMMAND.COM existent pe disc, şi mărimea standard a aceluaşi fişier descrisă în linia 2, de ex."25276", atunci fişierul "fisier1" nu se mai creează, el avînd dimensiunea zero. În acest caz, în linia 3 "COPY fisier1 fisier2 NUL", încercarea de copiere a lui "fisier1" în "fisier2" eşuează, "fisier1" fiind vid. Artificiul " NUL" a fost adăugat pentru a inhiba mesajul de confirmare a copierii. Linia 4 verifică prezenţa pe disc a lui "fisier2". Dacă acesta există, atunci nici un virus nu a afectat mărimea originală a lui COMMAND.COM, şi se trece controlul subrutinei numită "secventa", care după acelaşi principiu tratează situaţia datei lui COMMAND.COM. Dacă "fisier2" nu a fost copiat pe disc, la linia 5 se afişează un mesaj care atenţionează asupra virusului ce a modificat dimensiunea fişierului de comenzi DOS. La linia 14 începe rutina "mesaj", care în cazul în care nici data şi nici dimensiunea lui COMMAND.COM nu au fost modificate, ne asigură că nici un virus nu a modificat sistemul.

Linia 17, şterge fişierele temporare "fisier1" şi "fisier2". Trebuie făcută menţiunea că acest program simplu nu este un panaceu universal pentru toate tipurile de viruşi. El nu testează, de pildă memoria video, sau sistemul de întreruperi.

Pentru aceste teste se folosesc "ANTI-VIRUŞI" mult mai complicaţi, dar şi mult mai scumpi. Totuşi, CHKVIR.BAT poate fi interesant, fie şi numai prin simplitatea şi concepţia lui.

NOTĂ: Este posibil ca data şi/sau dimensiunea fişierului de comenzi utilizat să difere de la o versiune MS-DOS la alta. De aceea, folosind versiunes originală(de firmă), sigur neafectată de vreun virus se dă comanda "DIR COMMAND.COM" şi se află cei doi parametri ai acestui fişier: mărimea şi data. Apoi aceşti doi parametri se introduc în liniile 2, respectiv 9 ale programului CHKVIR.BAT în locul valorilor aflate între ghilimele. Iată şi programul:

```
        @ECHO OFF
        DIR A:COMMAND.COM  |  A:FIND "25276"  fisier1
        COPY fisier1 fisier2  NUL
        IF EXIST fisier2 GOTO secventa
        ECHO Marimea fisierului COMMAND.COM este modificata. VIRUS ACTIV!
        GOTO sfirsit
:secvcenta
        DEL fisier2
        DIR A:COMMAND.COM |  A:FIND "24-07-87"  fisier1
        COPY fisier1 fisier2  NUL
        IF EXIST fisier2 GOTO mesaj
        ECHO Data fisierului COMMAND.COM este modificata. VIRUS ACTIV!
        GOTO sfirsit
:mesaj
        ECHO Marime si data fisier COMMAND.COM nealterate. VIRUS NEDETECTAT!
:sfirsit
        DEL fisier2
```

STOP

Cîţiva viruşi mai cunoscuţi

Calculatoarele compatibile IBM se confrntă cu o adevărată "epidemie" de viruşi. Gama extrem de largă a acestora are un numitor comun: toate se instalează din programe comerciale de o proveniență incertă: reţele, copii ilegale, etc. Nu toţi viruşii au acelaşi grad de periculozitate. Totuşi, pînă şi cei mai aparent "nevinovaţi" viruşi, pot produce multe neplăceri. De aceea, ei trebuie identificaţi şi pe cît posibil distruşi cu programe "VIRUS KILLER", pentru a evita contaminări în lanţ.

Iată acum tipologia celor mai cunoscuţi viruşi de tip IBM-PC:

1. BRAIN(sau PAKISTAN sau LAHORE)

Simptom: afişare mesaj "(c) Brain" ca etichetă de volum, simultan cu distrugerea a 3 ko. din sectoarele discului.

Acţiune: virusul se autoscrie în "boot sector"-ul fiecărui disc.

Efect: supărător.

2. FU MANCHU

Simptom: fişierele COM şi EXE cresc artificial cu cca. 2088 octeţi. La startarea sistemului, ecranul se şterge şi apare mesajul: "The world will hear from me again!"

Acţiune: reacţionează la anumite cuvinte, cum ar fi: "Fu Manchu", "Thatcher", "Reagan", "Waldheim", "Botha", dar şi la alt gen de cuvinte.

Efect: supărător.

3. DATACRIME

Simptom: pe data de 13 octombrie a fiecărui an, şi în fiecare zi de după 13 octombrie, pînă la sfîrşitul anului, apare mesajul: "DATACRIME VIRUS RELEASED 1 MARCH 1989". Fişierele .COM cresc cu 1168 sau 1280 octeţi.

Acţiune: formarează nivelul de jos al hard-discului între cilindrii 0 şi 8.

Efect: foarte supărător.

4. CITY AVENGER

Simptom: fişierele COM sau .EXE cresc cu aproape 1800 octeţi.

Forma: scrie un sector care începe cu: "Eddie lives somewhere in time", afectînd sectoare aleatoare de pe hard-disk, la intervale de timp neregulate.

Efect: neplăcut.

5. DENZUK

Simptom: după un start cald al sistemului de calcul, apare un mesaj roşu pe monitoarele color: "DENZUK". De asemenea, modofică eticheta de volum în: "Y.C.E.R.P".

Acţiune: este creat mai ales pentru a se infiltra pe discurile de 360 ko, dar se pare că afectează şi alte formate de discuri.

Efect: iritant.

6. CASCADE(sau BLACKJACK)

Simptom: fişierele .COM cresc cu 1704 sau 1701 octeţi. Cînd se iniţializează caracterele de pe ecran, încep să "cadă" în partea de jos a ecranului, într-o grămadă sau într-o linie. Difuzorul emite mici "clic"-uri la căderea fiecărei litere.

Acţiune: niciuna.

Efect: lipsit de pericol major.

7. SATURDAY 14th

Simptom: fişierele .COM şi .EXE cresc cu un număr de octeţi cuprins între 669 şi 684.

Acţiune: în fiecare zi de sîmbătă, 14, se vor scrie 100 de sectoare cu caractere nesemnificative, începînd cu primele sectoare ale drive-urilor: D, C, B şi A, în această ordine.

Efect: supărător.

8. VIENNA

Simptom: fişierele .COM cresc cu 648 de octeţi.

Acţiune: infestează programul în curs, ca şi fişierul COMMAND.COM

Efect: supărător.

9. 405

Simptom: fişierele mai mici de 405 octeţi cresc pînă la această valoare.
Acţiune: programele infestate nu mai funcţionează.
Efect: foarte periculos.

10. OGRE

Simptom: se pierd 8 ko de pe disc. Sectorul "boot" este anormal. Pe floppy se găsesc 3 ko de sectoare afectate. Dacă se lasă în funcţiune computerul pentru 48 de ore şi apoi se accesează hard- diskul, ecranul se şterge şi apare mesajul: "Disk Killer - version 1.00 by Computer Ogre 04/01/1989. Warning do not turn off the power or remove the diskette while Disk Killer is processing".
Acţiune: modifică întregul hard-disk, făcîndu-l inaccesibil.
Efect: distrugător. ■

Rubrică realizată de *Dobrilă Mirel*

Poşta redacţiei

Nedelcu Adrian, Sîngeorz-Băi, 17 ani

În primul rînd, vă mulţumim pentru frumoasele aprecieri la adresa revistei. Din păcate, în privinţa documentaţiei de care aveţi nevoie nu vă putem ajuta. Dacă aveţi probleme de vedere şi totuşi aţi lucrat pînă acum atît de mult cu calculatorul înseamnă că puteţi urma şi cursurile uneia dintre facultăţile de profil, fie în cadrul unui Institut Politehnic în care există o secţie sau facultate de Calculatoare, fie în cadrul unei Universităţi în cadrul căreia există o secţie de Informatică.

ing.Dumitrescu Mihai, Bucureşti.

Dacă doriţi să colaboraţi cu articole la revista noastră, vă aşteptăm cu plăcere pentru o discuţie la redacţie.

Mercea Mario Matei, Timişoara.

În curînd apare în librării: "ABC de calculatoare personale", autori Adrian Petrescu şi colectiv. Această carte publicată în Editura Tehnică conţine numeroase informaţii referitoare la calculatoarele compatibile cu Spectrum.

Radu Constantinescu, Brăila.

Ne bucură mult faptul că apreciaţi revista noastră şi vă asigurăm că vom căuta să publicăm numai articole utile pentru cititorii interesaţi de informatică. În forma în care sînt prezentate programele care apar în revistă pot să fie rulate pe calculatoare din familia compatibilelor IBM-PC.

Bölöni Ladislau, Tîrnăveni.

Vă mulţumim pentru comentariile competente asupra articolelor apărute în revistă. O să încercăm să publicăm şi articole pe temele propuse de dumneavoastră.

Stoean Gilberto, Bucureşti, 16 ani.

După cum demonstrează şi titlul revistei, ne preocupăm în special de calculatoare din familia compatibilelor IBM-PC. Sperăm ca şi în continuare să găsiţi în revista noastră lucruri care să vă intereseze.

Vasilescu Grigore, Brăila.

Dacă aţi început să lucraţi pe un calculator de tip PC-IBM probabil că a'i descoperit deja programe ca dBASE III+ (sau IV), ORACLE sau altele care permit exploatara bazelor de date.

Răzvan Dumitru, Bucureşti.

Performanţele calculatoarelor JET, CIP, MICROTIM sînt aproximativ aceleaşi cu ale calculatorului HC-85. CIP are 64 Ko RAM, dar numai 4Ko. ROM. COBRA prezintă în plus şi compatibilitate CP/M (în configuraţie completă), ca şi HC 88. În privinţa protejării programelor preferăm codul maşină şi... originalitatea (v. almanah TEHNIUM '90).

Silviu Dumitrescu,Focşani.

Limbajul C nu are legătură cu limbajul COBOL. În afară de BASIC sub interpretor şi asambloare pentru cod maşină, pe SPECTRUM se poate lucra, cu ajutorul compilatoarelor, în PASCAL, C, PROLOG, FORTH, LOGO.

Alexandru Kopatz,Sibiu.

Mulţumim pentru idei, stilul nu contează. Începînd cu numărul următor al revistei, vom aborda, în legătură cu codul maşină din ROM şi problema variabilelor de sistem. Ne vom gîndi la un concurs pe teme de cod maşină, cu un top al rezolvitorilor.

Horaţiu Coifan,Suceava.

Din numărul viitor, vom prezenta, pe rînd, diversele rutine din ROM. Considerăm că GENS 3M21 este mai eficient decît ZEUS, care, deşi posedă un editor mai puternic, are unele greşeli.

Valer Bocan, Deva.

Împreună cu caseta îţi trimitem şi o scrisoare care sperăm să ajungă la tine mai repede decît a ajuns scrisoarea ta la noi.

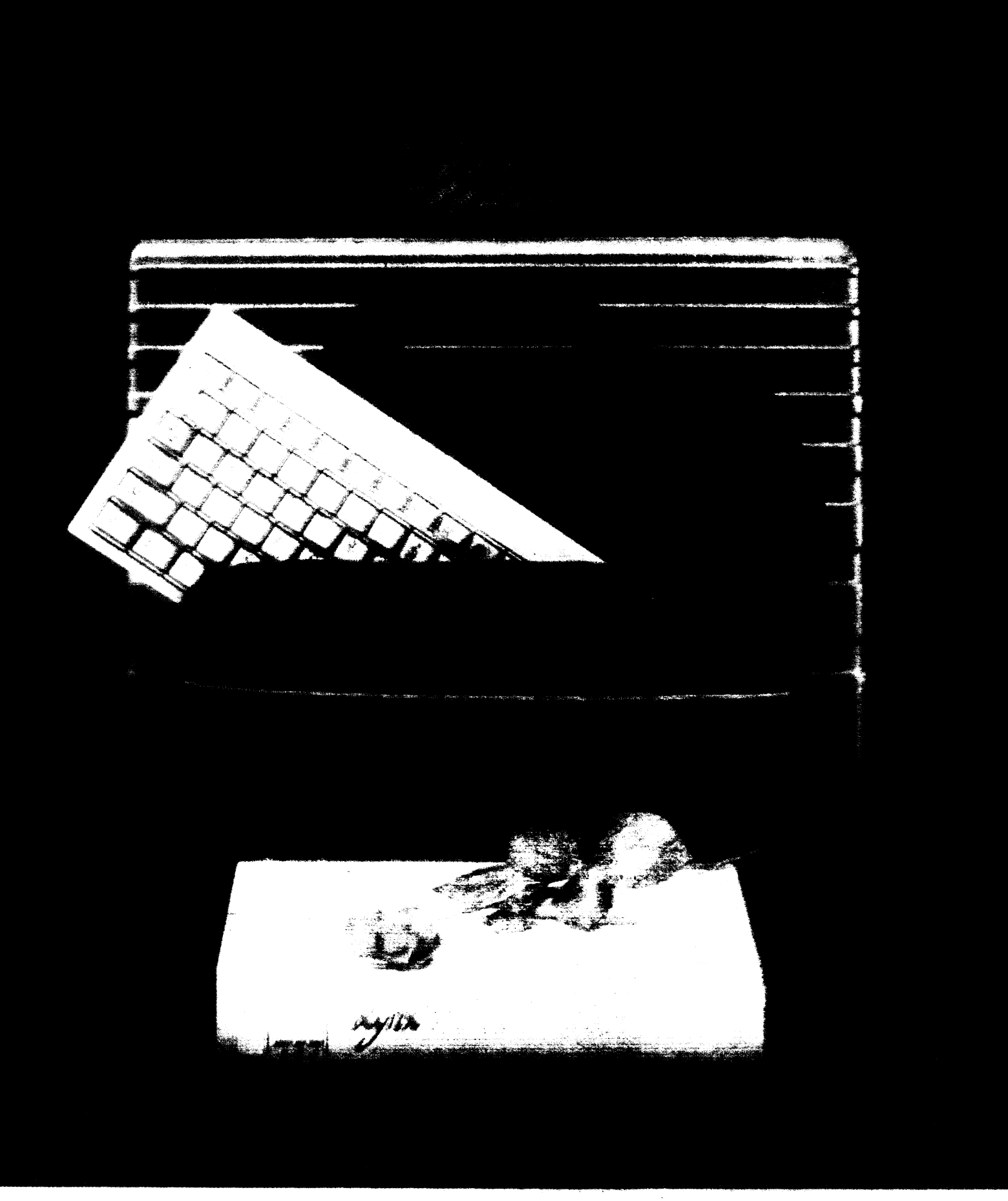
Aurel Enache (Bucureşti), Matei Mario Mercea (Timişoara), Bogdan Horotan Bucureşti), Alexandru Kopatz (Sibiu), Horaţiu Coifan (Suceava).

Deocamdată, din cauza problemelor legate de tipărire, nu putem onora anunţul din numărul 3 al revistei.

În legătură cu articolul din numărul 4, la pagina 65, în loc de "nu trebuie citată instrucţiunea CALL #02BF", seva citi "nu trebuie uitată instr...".

Pentru consultări în legătură cu utilitarele pe SPECTRUM, vă puteţi adresa direct la BOGDAN IORDĂNESCU,tel.: 53.34.67 sau BOGDAN GEORGESCU, tel.: 27.78.55.

Pentru ceilalţi care ne-au scris, răspunsurile în numerele viitoare!



INNOVATOR NOTEBOOK
MADE IN HOLLAND