

PC magazin

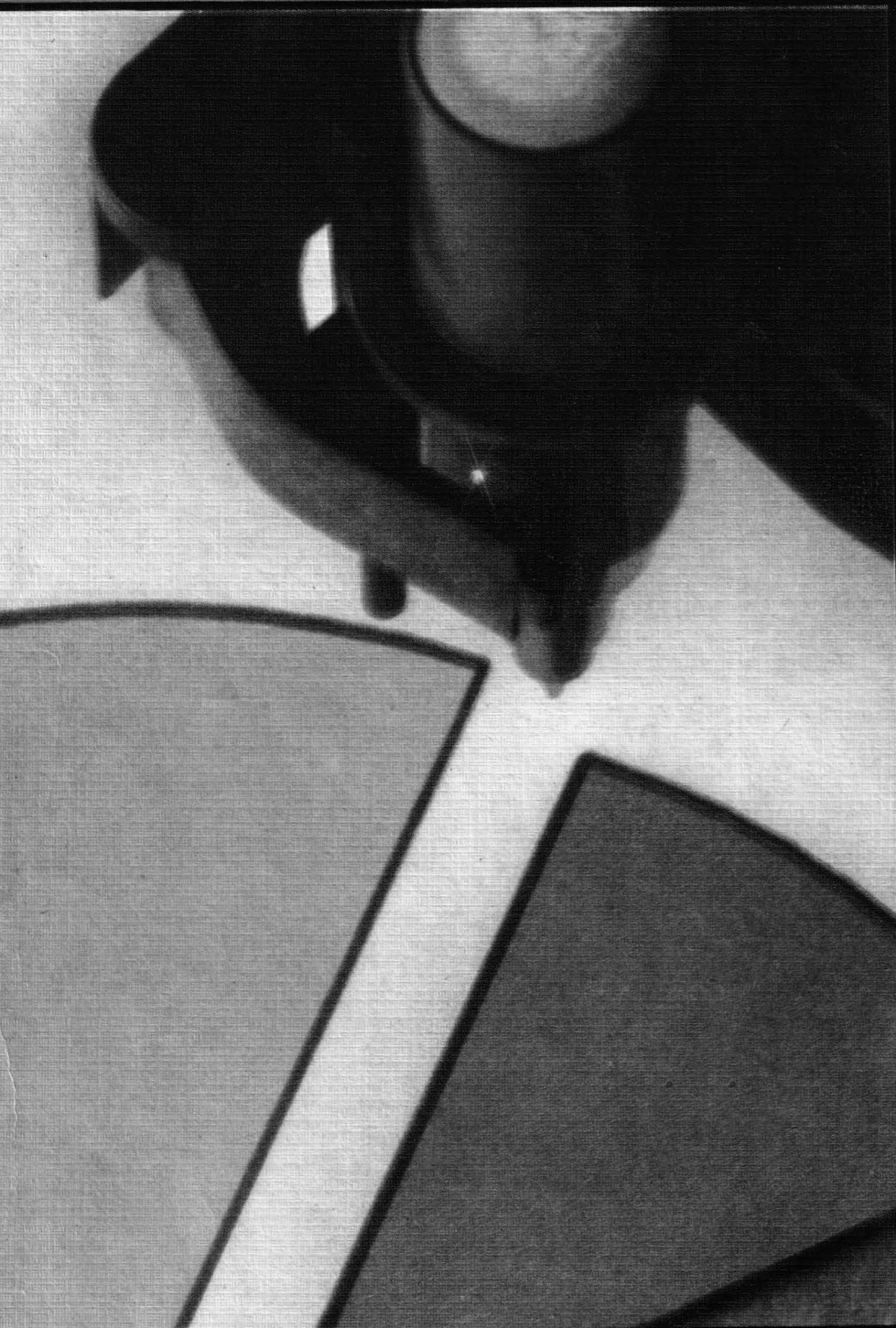


Revistă Independentă de
informatică editată de
ADISAN

ANUL II, Nr. 2/1991

- Limbaajul PROLOG
- Rețele neuronale(II)
- Incursiune în ORACLE
- Totul despre memorie
- "Spargerea" programelor
- A fost odata "UNIX"
- Prezentări SOFTWARE
- Noutăți





FEPA S.A.
72326 București
Bd. D. Pompei nr.8
Tel:88.42.65
87.67.14
Fax: 87.44.96
Telex: 10 895; 10 691

PLOTTERS

CULTURA INFORMATICĂ

Bazele logice ale inteligenței artificiale(II)	3
Limbajul PROLOG	7
Rețele de calculatoare(II)	12
Rețele neuronale(II)	15

INFORMATICA FĂRĂ PROFESOR

Limbajul C (VI)	19
Editoare și fonturi (V)	22
Inițiere în programare — Limbajul Pascal (II)	25
Incursiune în ORACLE	29

WINDOWS

Totul despre memorie	32
Probleme specifice bazelor de date documentare	33

HOME COMPUTER

Limbajul Z80 pe calculatoarele compatibile SPECTRUM (III)	35
"Spargerea" programelor	39

PERSPECTIVE

Once upon the time "THE UNIX"	42
Sosesc produsele CASE(II)	46

MARKET SOFT/HARD

Prezentări pachete de programe	48
Oferta ADISAN	49

STOP

Atenție la viruși!	53
Poșta redacției	56

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 Athanasie
conf.dr.ing. Eugenia Kallaz
ing. Sorin Deaconu
ing. Ionuț Stoica
Roxana Bicleanu
Mar'us Sturzoiu
Virgil Teculescu

ing. Cureleț-Bălan Gheorghe
Ion Paraschiv
dr. Tiberiu Spircu
Ion Mușlea
Bogdan Georgescu
Bogdan Iordănescu
Dobriță Mirel

GRAFICĂ

Octavian Penda
Lucian Mișcă
Lumința Ciupitu

ADRESA REDACȚIEI

București, str. Ion Mincu nr.11
sect.1
Tel: 17.74.30

Tiparul a fost executat la
Tipografia UNIVERSUL

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

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

ORGANIZEAZĂ PERMANENT

CURSURI

CALIFICARE, FORMARE, SPECIALIZARE,

PERFECTIONARE

PENTRU PERSOANELE CU STUDII MEDII

PE MINI-MICROCALCULATOARE (8-16 BIȚI)

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

— Formare analiști-programatori asistenți.

— Operatori inițiere în informatică, lucrări secretariat și dactilografie.

— Perfecționare pe microcalculatoare (prelucrare de texte și documentații).

— Programare în limbaj BASIC.

— Programare în limbaj dBASE.

— Utilizare microcalculatoare compatibile IBM-PC.

La cursuri pot participa și persoanele cu studii superioare.

Cursurile se organizează la sediul bazei (cu asigurarea cazării) și la BENEFICIAR cu și fără scoatere din producție atât pentru persoanele trimise de unitățile de stat, cât și pentru persoanele particulare.

Informații la telefon: 988/73534.

BAZELE LOGICE ALE INTELIGENȚEI ARTIFICIALE(II)

Prof.univ.dr. Cornel Popa

6. Derivare și respingere rezolutivă

6D1. Fie S un set de clauze și k o clauză oarecare. Spunem că din S derivă rezolutiv k și scriem $S \vdash_{\text{res}} k$, dacă și numai dacă, putem construi o șir de clauze:

$$D = (k_1, k_2, \dots, k_n)$$

care satisface condițiile:

- 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 și k_m cu $j < i$ și $m < i$ și există un l astfel încât $k_i = \text{res}_l(k_j, k_m)$.

Cu alte cuvinte, D este un șir de clauze din S sau obținute prin rezoluția din S al cărui ultim element k_n coincide cu clauza de derivat k . Relația $\vdash_{\text{res}}$ introduce o relație de ordine, punînd la început elemente din S și apoi elemente din S sau obținute prin rezoluție 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 . Și într-un caz și într-altul se procedează strict formal, prin aplicarea unor reguli de derivare. Deosebirile rezidă doar în faptul că elementele lui S nu sînt tautologii sau formule identic adevărate, ci doar formule adevărate și analog k nu este o tautologie ci tot o formulă adevărată.

6D2. Respingerea rezolutivă este o derivare rezolutivă încheiată prin obținerea clauzei vide, i.e. a clauzei fără nici un literal, notată prin $\square$ sau Nil.

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

Demonstrarea unei teoreme prin respingere rezolutivă se face astfel: 1) se obțin clauzele din premisele raționamentului; 2) se obțin clauzele din negația concluziei raționamentului; 3) se dezvoltă, prin aplicarea iterată a principiului rezoluției, o respingere rezolutivă.

Aplicarea principiului rezoluției la un set de clauze se face, de regulă, după anumite criterii și 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 raționamentului **R1**.

Demonstrarea validității raționamentului **R1** se reduce astfel la o derivare rezolutivă de forma:

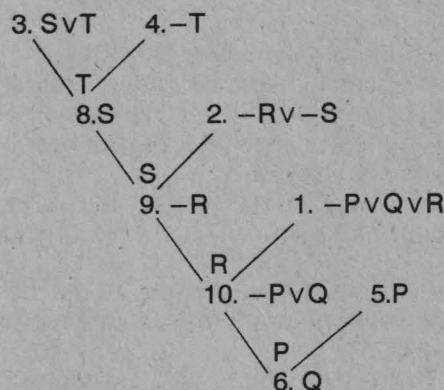
$$\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ -P \vee Q \vee R, & -R \vee -S, & S \vee T, & -T, & P, & -Q \end{array} \quad (d1)$$

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

$$\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 7 & \\ -P \vee Q \vee R, & -R \vee -S, & S \vee T, & -T, & P, & -Q & \end{array} \quad (d2)$$

De remarcat că (d2) conține în setul de clauze premise pe $-Q$, negația concluziei. Respingerea rezolutivă este un raționament prin reducere la absurd în care postulăm premisele și negația concluziei.

Derivăm mai întâi, din clauzele 1–5, clauza 6, conform lui (d1).



Clauza 3 conține literalul T , iar clauza 4 pe $-T$. Rezoluția în T din clauzele 3 și 4 duce la clauza 8. Clauzei 8 îi asociem pe 2 pentru că 2 conține pe $-S$; obținem făcînd o rezoluție în S pe 9. Și așa mai departe, pînă îl obținem pe Q , clauza de derivat.

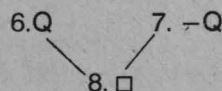
Secvența de clauze D definită în **6D1**, pentru derivarea făcută este

$$\begin{array}{cccccc} 3 & 4 & 8 & 2 & 9 & 1 & 10 & 5 & 6 \\ D1 = & S \vee T, & -T, & S, & -R \vee -S, & -R, & -P \vee Q \vee R, & -P \vee Q, & P, & Q \end{array}$$

Clauzele 3,4,2,1,5 aparțin lui $S1$, setul de premise; clauza 6 este clauza k ce trebuie derivată, iar clauzele 8, 9, 10 și 6 sînt obținute prin derivarea rezolutivă din clauzele anterioare lor în șirul $D1$.

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

Pentru a aplica o respingere rezolutivă conform relației (d2), prelungim derivarea rezolutivă anterioară obținînd drept rezolvent din: 6 și 7 pe $\square$. (Clauza 7 este negația concluziei)



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

Putem acum conchide că raționamentul **R1** este valid. Nu ne-am pronunțat încă asupra celorlalte: **R2**, **R3** și **R4**.

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

schemă validă de inferență, asemenea legii tranzitivității sau regulii detașării i.e. *modus ponens*.

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

7. Formalizarea unor raționamente în limbajul logicii predicatelor

Să ne întoarcem la raționamentele **R2**, **R3** și **R4**. În acestea apar unele particule lexicale ca "toți", "cineva", "cei care...", particulele anunță ideea de cuantificator universal sau existențial. În consecință, 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.

Experiența comună sau bunul simț, ne îndeamnă să considerăm un raționament ce are una sau două premise false, drept invalid.

Aceasta este însă o capcană în care nu trebuie să cădem cu ușurință. O implicație 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 condiții.

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

1. $\forall x (\text{piatră}(x) \supset \text{naște-pui-vii}(x))$
 2. $\forall y (\text{capră}(y) \supset \text{piatră}(y))$
 3. $\forall z (\text{capră}(z) \supset \text{naște-pui-vii}(z))$
- (R2⁰)

În **R3** intervin patru predicate și o constantă individuală. Predicatele sînt: se teme (x,y), iubește (x,y), este șef (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 obținem 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 .

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

În același timp, din predicatul iubește (x,y) trebuie să obținem predicatul "x nu este iubit de toți". Obținem mai întîi din iubește (x,y), predicatul monadic "y este iubit de toți".

$\text{iubit-de-toți}(y) = \forall x (\text{iubește}(x,y))$

De aici obținem:

$\text{non-iubit-de-toți}(y) = \neg \forall x (\text{iubește}(x,y))$

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

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

În sfîrșit, din predicatul șef(x,y) se poate obține șef al Mafiei(x) prin înlocuirea variabilei y prin numele propriu Mafia.

$\text{șef-al-Mafiei}(x) = \text{șef}(x, \text{Mafia})$

Putem acum asambla toate aceste predicate în patru

formule ce descriu premisele și concluzia raționamentului **R3**.

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

1. $\forall u (\exists t (\text{se-teme}(u,t) \supset \neg \forall v (\text{iubește}(v,u))))$
 2. $\forall y (\text{păzit}(y) \supset \exists z (\text{se-teme}(y,z)))$
 3. $\forall x (\text{șef}(x, \text{Mafia}) \supset \text{păzit}(x))$
- (R3⁰)

4. $\forall w (\text{șef}(w, \text{Mafia}) \supset \neg \forall s (\text{iubește}(s,w)))$

Trecem la formalizarea raționamentului **R4**.

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

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

$\text{este-iubit-de-cineva}(y) = \forall x (\text{iubește}(x,y))$

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

1. $\exists x (\text{iubește}(x, \text{Ion}) \supset \text{fericit}(\text{Ion}))$
 2. $\forall y (\text{iubește}(y, \text{Ana}) \supset \text{iubește}(\text{Ana}, y))$
 3. $\text{iubește}(\text{Ion}, \text{Ana})$
- (R4⁰)

4. $\text{fericit}(\text{Ion})$

(R2⁰)-(R4⁰) reprezintă formalizarea în limbajul logicii predicatelor a raționamentelor **R2-R4** propuse în 1.

Dacă dorim să decidem asupra validității raționamentelor **R2-R4** prin metoda rezoluției, atunci acestea trebuie aduse la o formă clauzală.

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

Obținerea formelor normale prenex se face prin utilizarea unor legi ale logicii predicatelor.

8. Legi ale logicii predicatelor

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

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

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

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

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

Dacă toți indivizii din D au proprietățile P și R , atunci toți au proprietatea P și toți au proprietatea R și invers.

Aceeași formulă transcrisă cu cuantificatorul existențial nu mai este adevărată. Din faptul că $\exists x P(x)$ și $\exists x R(x)$ nu decurge în mod necesar că $\exists x (P(x) \& R(x))$. Dacă există un campion al țării la săritura în înălțime și există un campion al țării la box categoria grea, nu rezultă că există un individ ce este simultan campion la săritura în înălțime și 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 și înalt, atunci este blond și este înalt.

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

Formula 8.3 descrie o lege de distributivitate a cuantificatorului universal față de conjuncție.

În mod analog, cuantificatorul existențial este distributiv față de disjuncție.

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

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 existențial nu generează o formulă adevărată.

Dacă vom considera drept domeniu D mulțimea oamenilor și vom interpreta predicatul P drept a fi bărbat iar pe R a fi femeie, atunci implicația 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 ființă umană este bărbat sau este femeie nu rezultă nicicum că toți oamenii sînt bărbați sau toți oamenii sînt femei.

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

Vor fi, de asemenea, adevărate următoarele două formule de eliminare a negației din fața cuantificatorilor:

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

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

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

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

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

Formulăm acum legea extinderii fictive a domeniului unui cuantificator.

Convenim să notăm prin Qx pe $\forall x$ sau $\exists x$ și prin * pe & sau v.

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

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

Cuantificatorul Qx din primul termen a trecut înaintea lui A în cel de al doilea, pe temeiul că A nu conține variabila x și, în consecință, Qx va lega aceleași apariții din P(x).

Dacă ținem seama că Qx ține locul lui $\forall x$ sau $\exists x$ iar * ține locul lui & sau v, atunci din (8.7) obținem lesne pentru echivalențe distincte, ca exemplificări ale lui 8.7. De exemplu:

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

se obține din (8.7) prin înlocuirea lui * prin & și a lui Qx prin $\exists x$.

Dacă într-o formulă închisă (care nu are variabile libere), aceiași variabilă individuală apare sub cuantificatori diferiți și ca argument pentru semne predicative distincte, putem înlocui aparițiile variabilei în cauză într-un

cuantificator sau într-altul și predicatele corespunzătoare printr-o altă variabilă.

$$Q_1xP(x) * Q_2xR(x) \equiv Q_1xP(x) * Q_2yR(y) \quad (8.8)$$

Păstrăm în (8.8) convențiile introduse mai sus pentru Q și *.

Schema (8.8) poate fi, între altele, instanțiată 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 întâi 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: prenex, Skolem și forme clauzale.

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

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

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

Revenim la raționamentele R2, R3 și R4 sau mai exact la formulele din logica predicatelor ce corespund acestor raționamente, R2⁰, R3⁰ și T4⁰. R2⁰ este deja la o formă normală prenexă.

R3⁰ are în formă normală prenexă numai premisa 3, celelalte trei formule, 1, 2 și 4 nefiind în forma cerută. Formula 1:

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

Trecerea de la 1 la 1.2 s-a făcut pe baza echivalenței $A \supset B \equiv -A \vee B$; trecerea de la 1.2 la 1.3 s-a făcut pe baza legilor (8.5) și (8.6) de eliminare a cuantificatorilor negați; în sfîrșit, 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 R3⁰.

2. $\forall y (\text{păzit}(y) \supset \exists z (\text{se-teme}(y, z)))$
 - 2.1. $\forall y (-(\text{păzit}(y)) \vee \exists z (\text{se-teme}(y, z)))$
 - 2.2. $\forall y \exists z (-(\text{păzit}(y)) \vee \text{se-teme}(y, z))$
- 2.2. este f.n. prenexă a lui 2 din R3⁰. S-a aplicat pentru a ajunge la această formă $A \supset B \equiv -A \vee B$ și legea extinderii fictive a domeniului unui cuantificator (8.7).

Aplicăm același tratament concluziei din R3⁰.

4. $\forall w (\text{șef}(w, \text{Mafia}) \supset -\forall s (\text{iubește}(s, w)))$
- 4.1. $\forall w (-\text{șef}(w, \text{Mafia}) \vee -\forall s (\text{iubește}(s, w)))$
- 4.2. $\forall w (-\text{șef}(w, \text{Mafia}) \vee \exists s -(\text{iubește}(s, w)))$
- 4.3. $\forall w \exists s (-\text{șef}(w, \text{Mafia}) \vee -(\text{iubește}(s, w)))$

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

Compartimentul de ediție al firmei **ADISAN** vă oferă următoarele servicii:

- machetare pe calculator pentru periodice (săptămânale, lunare, etc.) cu imprimare pe hîrtie de calitate superioară sau folie transparentă, folosind programe moderne de editare și imprimantă cu laser.
- culegere pe calculator pentru texte de orice complexitate (de la beletristică la texte științifice) folosind programe specializate.
- retroversiune și traducere de texte din orice domeniu (de către specialiști autorizați) din limbile engleză, franceză, germană, rusă, spaniolă, maghiară.
- grafică pe calculator și clasică; combinarea acestora cu text pentru realizarea de prospecte, cataloage, reclame etc.
- editarea de texte și cărți în domeniul informaticii (de la nivelul de învățare pînă la nivel universitar și profesional).
- spațiu publicitar în revista de largă circulație *PC-Magazin* — prima revistă de informatică din România.

În cadrul serviciilor menționate, specialiștii noștri pot prelua tehnoredactarea și corectarea materialelor, predînd beneficiarului textul validat pentru multiplicare.

Prețul pentru o pagină de manuscris **A4 dactilografiată la 2 rînduri**, trecută prin întreg procesul tehnologic menționat (tehnoredactare, culegere text, corectură, machetare, imprimare) pornește de la **125 lei**, fiind negociabil în funcție de complexitate și urgență.

Contractele pe **termen lung** beneficiază de condiții speciale de tarificare și prioritate.

Specialiștii noștri pot rezolva comenzile dumneavoastră în termene ferme (inclusiv urgențe) și în condiții de calitate deosebite.

Compartimentul dispune de un set complet de caractere pentru limba română și pentru majoritatea limbilor europene, precum și de caractere speciale pentru texte de specialitate.

gației unui cuantificator universal și a extinderii fictive a domeniului unui cuantificator.

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

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

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

1. $\exists x (iubește(x, Ion)) \supset fericit(Ion)$.

1.1. $\neg \exists x (iubește(x, Ion) \vee fericit(Ion))$

1.2. $\forall x \neg (iubește(x, Ion) \vee fericit(Ion))$

1.3. $\forall x (\neg (iubește(x, Ion) \vee fericit(Ion)))$

În trecerea de la 1 la 1-3 s-au aplicat în ordine legile excluderii implicației, excluderii cuantificatorului existențial și 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. $\forall y (\neg (iubește(y, Ana)) \vee iubește(Ana, y))$

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

Toate premisele din $R3^0$ și $R4^0$ au fost astfel aduse la forme normale prenex.

Limbaajul PROLOG

Ion Mușlea

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

1. Introducere

În numerele precedente revista **PC-MAGAZIN** a publicat mai multe articole privind atât programarea logică, cât și diverse posibilități de folosire ale limbajului **PROLOG**. Având în vedere cunoștințele dobândite pînă acum de cititor am decis prezentarea în continuare a unui program, de dimensiuni mai mari în care se folosesc mai multe facilități 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 anunțat în titlu, iar pe de altă parte evidențierea folosirii diverselor resurse și 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**, cât și ca o bază ce poate fi perfecționată și extinsă spre un program veritabil de joc **GO**.

Pentru cei neinițiați în **GO** va rămîne doar scopul didactic de aprofundare a cunoștințelor 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 folosește 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 caroiu de 19*19. La începutul partidei tabla e goală. Piesele se așează în intersecțiile caroiului, inclusiv pe marginile tablei — nu în interiorul "pătrățelelor" — și 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 caroiului în contururi formate din piese proprii și folosind eventual și marginile tablei drept graniță. 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 cîștiga jucătorul ce realizează un punctaj superior.

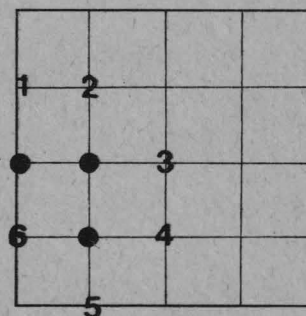
Punctele (libere) ale caroiului ce sînt adiacente unei piese se numesc **LIBERTĂȚI** ale acelei piese. Deci o piesă solitară aflată la mijlocul tablei poate avea cel mult

4 libertăți, la margine 3, iar pe colț 2. Un grup de piese adiacente pe liniile și coloanele tablei se comportă solid, însumîndu-și libertățile. Un grup de piese încercuit de adversar astfel încît nu mai are nici o libertate se va numi **CAPTURAT** și se va elimina de tablă.

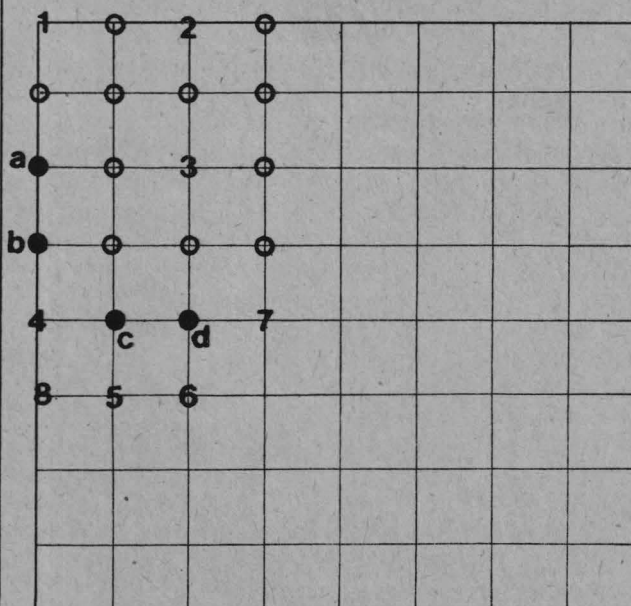
Printr-un ochi se înțelege un grup de libertăți **INTERIOARE** unui grup de piese. Pentru ca un grup de piese să fie **VIU (VIABIL)** el trebuie să aibă cel puțin doi ochi separați (neadiacenți). Acest lucru e adevărat în virtutea faptului că în **GO** e interzisă **SINUCIDAREA** (adică plasarea unei piese pe tablă de așa manieră încît să nu aibă nici o libertate), cu excepția cazului în care efectul imediat al mutării duce la capturarea unor grupuri adverse. Deci în ideea existenței a doi ochi neadiacenți, capturarea este imposibilă prin faptul că mutarea în interiorul unui ochi va duce la sinucidere fără a se obține vreo captură (căci mai există cel puțin încă o libertate interioară în cadrul celui alt ochi).

Exemple:

- grupul are 6 libertăți;



- grupul alb are 3 libertăți interioare (1,2,3);
- grupul negru format din a și b are o singură libertate (4);



- grupul negru format din c și d are 4 libertăți (4,5,6,7);
- dacă albul mută în poziția 4 capturează a și b;

— dacă negrul mută în poziția 4, va obține un singur grup a, b, c, d (cu patru libertăți: 5, 6, 7, 8).

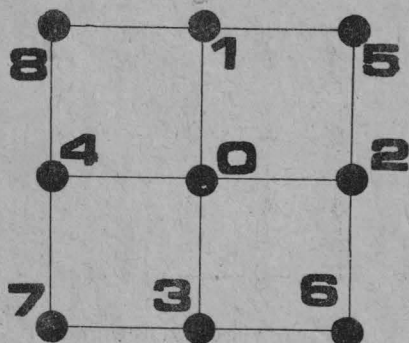
3. PROGRAMUL PROLOG

3.1. Noțiuni introductive

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

Exemplu:

Piesa 0 este în relația VECIN4 cu piesele 1-4 și în relația VECIN8 cu piesele 1-8.



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

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

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

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

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

— o bază de date de tip intern (se folosește pentru verificări rapide și pentru o eventuală salvare a unei poziții de pe tablă sau a unei partide).

3.2. Structura datelor și 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_grps,xc,yc,L_grps,
grup_piese,L_grps,integer)
exista_vecin_in_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_si_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_piesa)
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 funcție de rolul lor, subrutinele se pot împărți 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" — TELUL */
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 afișează 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(L_a8,L_n8,L_a4,L_n4,'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 și în aceea poziție să nu existe vreo altă piesă). Dacă această condiție e îndeplinită se va căuta în lista grupurilor de culoarea respectivă grupul sau grupurile ce au cel puțin o piesă care să fie în relația 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 și 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 greșită: în aceea poziție 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,C,New_a8,New_n8,
New_a4,New_n4):—
          /* Mutare greșită: coordonatele piesei
          nu aparțin 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_la8,0),
          adaugă_la_grupuri(L_a4,L_n4,X,Y,C,
          Noua_la4,Noua_la4,1),
          New_a8=Noua_la8,
          New_n8=Noua_la8,
          New_n4=Noua_la4,
          pune_piesa(X,Y,C),
          asserta(piesa(X,Y,C)).
```

NOTĂ

Într-o versiune ulterioară ar trebui testată și condiția neacceptării unei sinucideri decât în cazul în care duce la o captură.

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

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

```

Ycoord=5+Co1*10,
setfillsrle(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 relația VECIN8 */
    caută_grup_apartinător(L_a,X,Y,[],[]),
    Lista_actuala,O),
    Noua_la=Lista_actuala,
    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 relația VECIN8 */
    caută_grup_apartinător(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 1 indică căutarea după relația VECIN4 */
    caută_grup_apartinător(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 relația VECIN4 */
    caută_grup_apartinător(L_n,X,Y,[],[]),
    Lista_actuala,1),
    Noua_ln=Lista_actuala,
    Noua_la=L_a.

/* CAUTA_GRPUP_APARTINATOR verifică dacă piesa aparține unui grup existent */
cauta_grup_apartinător([],X,Y,L_beg,Noul_grup,Lista_actuala,):—
    Lista_actuala= [[coord(X,Y) | Noul_grup] | L_beg]
cauta_grup_apartinător([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),
    caută_grup_apartinător(Rest,X,Y,L_bg,G_conc,Lista_actuala,TipVecin).
cauta_grup_apartinător([Head | Tail],X,Y,L_beg,Noul_grup,L_actuala,TipVecin):—
    caută_grup_apartinător(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.
concatenează([Head | Tail],L2,Final_list):—
    concatenează(Tail,[Head | L2],Final_list).

/* EXISTA_VECIN_IN_GRPUP(Grup,X,Y) verifică dacă (X,Y) are un vecin în Grup */
exista_vecin_in_grup([],X,Y,TipVecin):—
    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.
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 cit mai eficientă continuarea a programului în scopul testării viabilității 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 așa manieră încît să se respecte situația de pe tablă (adică dacă vreo piesă a grupului e pe o margine sau într-un colț, aceeași piesă să fie în același gen de poziție și în "tabla miniaturală", iar dacă nici o piesă a grupului nu este poziționată pe vreo margine sau colț, situația să se reflecte identic și pe "noua tablă"). O dată obținut acest deziderat nu mai rămîne de realizat decît un "negativ" al imaginii obținute anterior prin încadrarea grupului și ignorarea tuturor celorlalte piese de pe tablă (în sensul că toate pozițiile caroiului neocupate de piese se vor considera drept piese de aceeași culoare iar cele ce conțineau piese se vor considera drept goale) în care să formăm grupuri de piese caracterizate de relația VECIN4. Dacă vom obține N grupuri întotdeauna unul din ele va reprezenta "restul tablei", iar celelalte N-1 vor fi ochiurile grupului.

/ VIZUALIZEAZA_GRPURI 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_GRP 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 aparțin
aceluiși 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+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(X3,Y3,X4,Y4),
    line(Xcoord,Y3,Xcoord,Y4),
    setcolor(1).
```

/ REFA_GRP elimină marcasele 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):-
    Val>Y,
    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 spațiu între piesă și 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 spațiu între piesă și 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 */
```

/ SITUAȚIA recunoaște dacă un grup e de:*

- centru(return 0);
- margine(return1-stînga/sus10-dreapta11-jos);
- colț(return2-stînga-jos,3-dreapta-jos,
 4-dreapta- sus,5-stînga-sus). */

(Continuarea articolului în numărul viitor.)

REȚELE DE CALCULATOARE(II)

Ing. Sorin Deaconu

Introducere

Continuând seria de articole referitoare la rețelele de calculatoare, prezentăm în acest număr câteva caracteristici generale ale rețelelor locale.

În modul cel mai general o rețea locală de calculatoare (LAN) permite împărțirea unor resurse comune (hard discuri, imprimante, linii de comunicație) între mai multe calculatoare interconectate pe o arie restrânsă. Aceasta permite, de exemplu, conectarea unei mașini mai slab dotate din punct de vedere hardware (fără unitați de disc) în rețea, mărirându-se astfel performanțele acesteia.

În cadrul rețelelor locale, o importanță majoră o au rețelele de PC-uri. Din punct de vedere conceptual există două tipuri de rețele de PC-uri:

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

Rețelele "peer-to-peer" sînt relativ ușor de implementat și de utilizat prezentînd în schimb două dezavantaje majore: numărul redus de utilizatori și, mai ales, performanțe scăzute ale rețelei.

b) Rețele bazate pe strategia "client-server". În acest tip de rețea unul sau mai multe calculatoare funcționează ca "file server" sau "network server", deosebindu-se de celelalte calculatoare din rețea. În rîndurile ce urmează vom face o comparație între acest tip și rețelele de tip "peer-to-peer", din care va rezulta superioritatea rețelelor "client-server".

În rețelele de tip "peer-to-peer" rețeaua 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 mașina gazdă (de exemplu mărimea volumelor de disc, structura fișierelor, lipsa posibilității de lucru multitasking, lipsa unui sistem de securitate a informațiilor vehiculate sau memorate în mediul de rețea, etc.) vor afecta performanțele rețelei.

În rețelele "client-server" există, după cum am arătat, calculatoarele 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 mașinii (ale file server-ului), folo-

sind, de exemplu, o structură proprie de formatare a discului, o structură proprie a fișierelor și a metodelor de acces la fișiere, precum și un ansamblu de facilități pentru creșterea vitezei și a fiabilității rețelei. 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*) și un sistem de securitate al rețelei organizat pe mai multe nivele.

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

Topologii de rețele locale

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

În funcție de modul de conectare a calculatoarelor în rețea se definesc tipurile de topologii de rețele 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 stație de lucru (*work station-WS*) fiind conectată la file server printr-un cablu separat (fig.1).

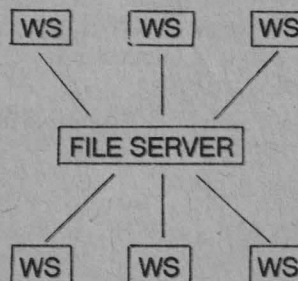


Figura 1.

Un exemplu de astfel de rețea este rețeaua S-Net a firmei Novell.

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

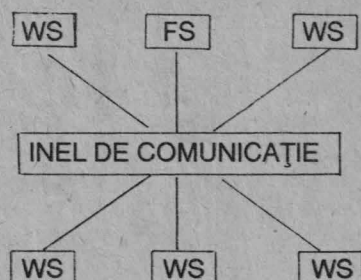


Figura 2.

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

c) Topologia *bus* utilizează un singur cablu care conectează atât file serverul cât și stațiile de lucru (fig.3).

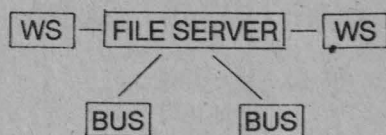


Figura 3.

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

Standarde de rețea

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

a) Standardul **ARCnet** utilizează topologia star, permițând realizarea de rețele cu performanțe medii și cost scăzut.

b) Standardul **Token Ring** utilizează topologia ring. Este dezvoltat în principal de IBM, permițând performanțe foarte bune și 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 rețea locală cel mai utilizat în lume, permițând performanțe excelente și posibilități foarte bune de interconectare.

Metode de transmisie

„Două metode se folosesc astăzi pentru vehicularea informațiilor în cadrul rețelei locale:

- metoda „*token passing*”
- metoda *CSMA/CD*

a) Metoda „*token passing*” utilizează un *token* („semn”) electronic care este trecut de la o stație la alta de-a lungul inelului. În momentul inițial este pus în circulație, de către o stație „master”, un *token* liber (*token free*). O stație 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*, stația care dorește să transmită marchează *token*-ul ca fiind ocupat (*token busy*), după care transmite pachetul de date. La stația destinație, după preluarea pachetului de date, se marchează *token*-ul *busy* ca fiind *token free*, procesul reluându-se. Prin creșterea numărului de *token*-uri *free* puse în circulație se poate mări viteza de transmisie, de la valoarea de 2 Mbps pînă la 16Mbps.

Acest proces se aseamănă cu următoarea situație: 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 acțiuni: 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 mișcare al microfonului.

Rețelele bazate pe acest tip de metoda de transmisie au o comportare bună și la încărcări mari.

b) Metoda *CSMA/CD* (*Carrier Sense Multiple Access/Collision Detection* — detecția purtătoarei, acces multiplu, detecția coliziunilor) permite accesul concurrent al tuturor stațiilor la mediul de comunicație. Acestea se află în permanentă „*ascultare*” a mediului (detecția purtătoarei). În momentul în care o stație are un mesaj de transmis, ea așteaptă pînă cînd mediul de comunicație e liber, moment în care începe transmisia. Dacă o altă stație a început în același moment transmisia propriului său mesaj, ambele stații intrate în conflict semnalează o coliziune. Datele sînt compromise, fiind ignorate. După un interval de timp aleator ambele stații 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 același mesaj depășește valoarea *n*.

Performanțele acestui tip de rețea scad cu creșterea încărcării, datorită creșterii numărului de coliziuni.

Sisteme de operare în rețea (Network Operating Systems NOS)

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

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

— *NetWare*, al firmei Novell, cu un procent de 65% din piața 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 și oferă o cale convenabilă de implementare a rețelelor generale (WAN)

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

Rețeaua NetWare a firmei Novell

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

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

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

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

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

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

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

3. Standardizarea rețelei. Novell și-a stabilit propriile standarde de rețele 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 și "gateway"-uri cu standarde ca SNA, TCP/IP, sistemul de fișiere Macintosh, etc.

NetWare cuprinde cîteva game de rețele locale:

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

b) *Advanced NetWare 286*, o rețea cu performanțe excelente și cu un număr mare de utilizatori;

c) *SFT NetWare*, asigurînd în plus fața de precedentă 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 mașini Unix.

Se studiază, pentru viitor, realizarea de soluții și pentru cuplarea calculatoarelor de tip "mainframe" într-o rețea de PC-uri.

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

I. *Network Interface Card (NIC)*, ce formează interfața dintre stația de lucru și restul rețelei.

II. *Network Cable* (cablul de rețea), la care sînt atașate toate calculatoarele rețelei

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

În prezent *NetWare* prevede soluții pentru cele trei tipuri de rețele prezentate anterior (ARCnet, Ethernet și Token Ring)

De asemenea, sînt prevăzute facilități de extensie a rețelei locale, cum ar fi *bridge*-uri (interne sau externe) și *gateway*.

Un loc aparte îl ocupă folosirea imprimantelor atașate rețelei, existînd un "manager" al acestor servicii în rețea ("printer server").

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

ANUNȚ IMPORTANT

Firmele care vor să-și promoveze prin ADISAN oferta proprie pentru reclamă, sînt rugate să trimită pe adresa firmei (str. Ion Mincu, nr.11, sect.1, București) liste conținînd oferta completă (soft + hard) pe care o pun la dispoziție.

REȚELE NEURONALE(II)

Ing. Ion Stoica

1. MEMORII CLASICE. CARACTERUL DE LOCALITATE

În calculatoarele moderne informația este păstrată în circuite electronice speciale numite memorii. În general, o dată memorată se poate citi specificând adresa locației din memorie unde se găsește. Fiecărei adrese îi corespunde o singură locație de memorie de unde se citește, respectiv se scrie data corespunzătoare locației adresate.

Exemplu: Să considerăm o memorie cu 8 locații, fiecare locație conținând 4 biți (fig. 1). Pentru a putea adresa fiecare locație sînt suficienți 3 biți de adresă ($2^3 = 8$) notați A2, A1 și A0. Pentru fiecare adresă distinctă este selectată o singură locație de memorie la ieșire obținându-se valoarea memorată în locația respectivă. Astfel, pentru adresa 100 ($A_2 = 1, A_1 = 0$ și $A_0 = 0$) la ieșire se obține valoarea 1111.

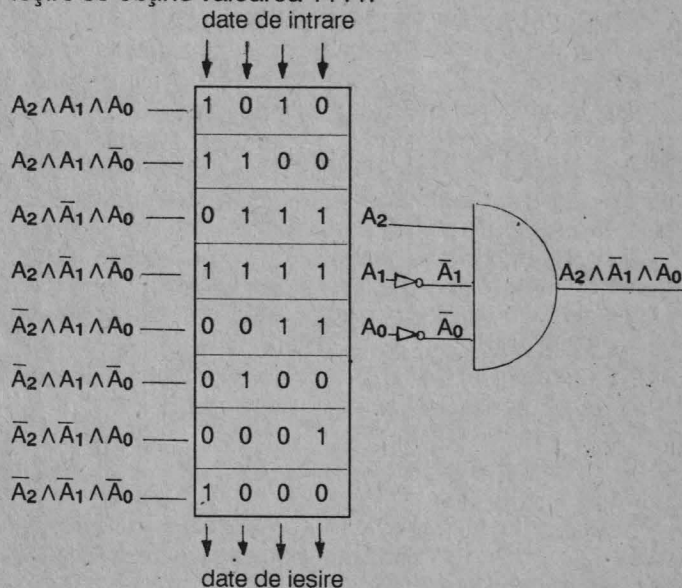


figura 1

Evident, funcțiile logice care asigură selectarea unei locații pot fi implementate cu ajutorul porților logice (SI, INVERSOR). În cazul memoriilor reale toată această logică este integrată pe același circuit și poartă denumirea de decodificator. De obicei, un decodificator are n intrări și 2^n ieșiri, fiecare ieșire fiind activată pentru o singură combinație de biți de intrare. În figura 2 este prezentat un decodificator cu 3 intrări și 8 ieșiri.

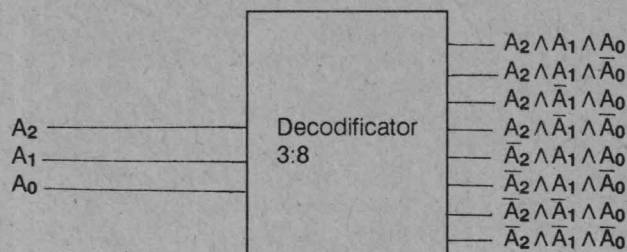


figura 2

Un alt tip de memorie întâlnită în calculatoarele clasice este memoria adresată prin conținut numită și *memorie asociativă*. La acest tip de memorie decodificatorul este înlocuit cu o memorie de construcție specială care în fiecare locație conține o "cheie" (figura 3). Fiecărei "chei" îi corespunde o locație din memoria principală în care se găsește informația utilă. La aplicarea unei combinații de biți la intrarea memoriei asociative, această combinație este comparată cu "cheile" memorate. Dacă se găsește cheia atunci la ieșire se obține informația din locația de memorie principală, corespunzătoare "cheii" de intrare.

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

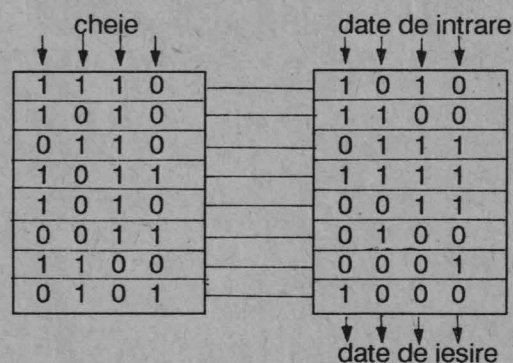


figura 3

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

Conceptul de memorie asociativă se regăsește într-un sens mai larg în cazul *memoriei biologice*. Pe de altă parte, între modelele de memorie întâlnite în calculatoarele clasice (prezentate succint mai sus) și *memoria biologică* există diferențe fundamentale. Astfel, în timp ce memoriile clasice au un caracter de localitate (fiecare informație este conținută într-o locație), în cazul rețelelor neuronale informația se regăsește dispersată în întreaga structură. Memorarea unei informații 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 funcționarea este analogică.

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

tivă care pentru fiecare șablon ("pattern") de intrare oferă la ieșire o informație completă (eventual restaurată) asupra șablonului de intrare.

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

2. Rețeaua HOPFIELD

2.1. Descriere

Acest tip de rețea a fost conceput de către J.J. Hopfield și în cursul timpului a cunoscut o serie de îmbunătățiri. În acest articol se descrie modelul inițial propus de Hopfield, care funcționează ca memorie asociativă.

Rețeaua, 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 și al unei conexiuni de ieșire. În plus, ieșirea unui nod este conectată la intrarea celorlalte noduri (obs: ieșirea unui nod nu este legată la intrarea aceluiași nod). O astfel de rețea, la care ieșirile sînt legate direct sau indirect la intrări este o rețea cu conexiune inversă (feed-back).

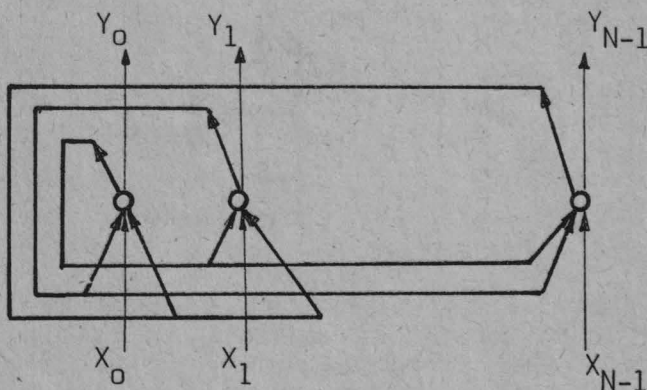


figura 4

În general acest tip de rețea este folosit cu intrări și ieșiri binare. Aceste intrări pot fi 0, 1 sau -1, 1. În cazul de față va fi preferat cel de-al doilea grup de valori.

Exemplu: Intrările unei rețele de tip Hopfield pot fi intensitățile 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 rețea poate fi utilizat, dacă este cazul, și pentru intrări nebinare. În această situație intrările se pot discretiza în prealabil.

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

Fiecare nod din rețeaua Hopfield calculează ieșirea prin intermediul unei funcții prag. Reamintim modul de

definire al acestei funcții:

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

2.2. Funcționare

Ne propunem să memorăm cu ajutorul unei rețele neuronale de tip Hopfield M șabloane cu o lungime de N biți fiecare, apoi la prezentarea unui șablon necunoscut să obținem la ieșire cel mai "apropiat" șablon față de cel prezentat. Pentru a memora M șabloane într-o astfel de rețea este suficient să se inițializeze ponderile asociate conexiunilor după cum urmează:

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

unde:

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

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

Așa cum s-a subliniat, informația memorată într-o rețea neuronală nu are un caracter de localitate ci ea se distribuie în întreaga structură. În cazul de față memorarea unui nou șablon duce la modificarea ponderilor asociate tuturor conexiunilor (ecuația 2).

Odată memorate cele M șabloane, rețeaua poate fi interogată în cazul unui șablon necunoscut anterior. Algoritmul care calculează răspunsul rețelei Hopfield comportă 2 pași:

Pasul 1. Inițializarea ieșirilor nodurilor cu valorile prezentate la intrările rețelei:

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

unde $Y_i(0)$ este ieșirea nodului i la momentul de timp inițial, iar x_i este valoarea corespunzătoare intrării nodului i pentru șablonul necunoscut.

Pasul 2. Calcularea iterativă a valorilor ieșirilor rețelei pînă la convergență:

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

unde $Y_i(t)$ reprezintă evident ieșirea nodului i la momentul de timp t . Procesul specificat în ecuația 4 se repetă pînă în momentul în care toate ieșirile rețelei rămîn neschimbate, adică $Y_i(t+1) = Y_i(t)$ cu $0 \leq i \leq N-1$. Hopfield a demonstrat că rețeaua este convergentă cînd ponderile sînt simetrice ($T_{ij} = T_{ji}$, în 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 îi 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 $0.15 * N$, unde 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 mulți 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 valoarea 1 deoarece $S1$ diferă de $S2$ prin al patrulea bit. Dacă luăm $S3 = 1100$, atunci vom avea $d(S1, S3) = 2$ și $d(S2, S3) = 3$.

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 scop rețeaua Hamming are N intrări, două nivele a câte M noduri fiecare și M ieșiri (o ieșire corespunzătoare fiecărei 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.

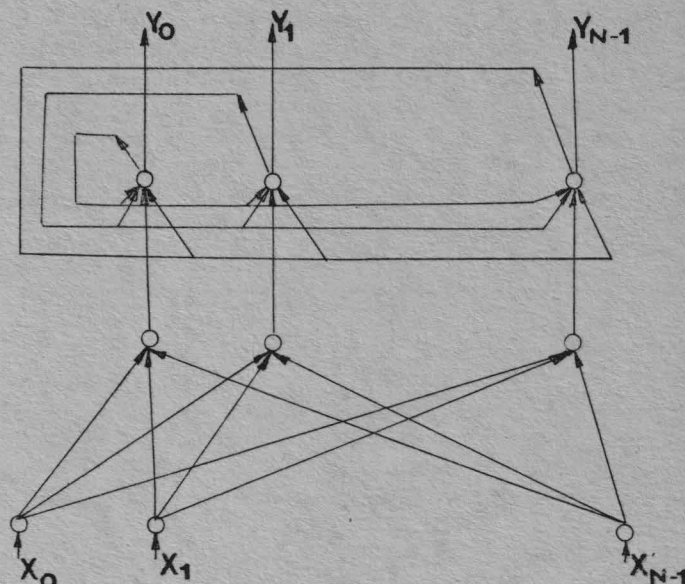


figura 5

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

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

3.2. Funcționare

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

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

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

$$0 \leq k, l \leq M-1$$

unde:

- W_{ij} reprezintă ponderea asociată conexiunii între intrarea i și 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 aparținând nivelului superior.

Exemplu: Considerăm nivelul inferior al unei rețele

Hamming cu 4 intrări și 2 noduri (figura 6). Această rețea este capabilă să clasifice șabloane cu o lungime de 4 biți în 2 clase. Fie "pattern"-urile corespunzătoare celor două clase $S1=1010$, respectiv $S2=1011$. Ca și în cazul rețelei Hopfield se aleg valorile binare de 1 și -1 pentru intrările rețelei. Astfel, pentru șablonul $S1$ se va aplica la intrare combinația 1, -1, 1, -1, iar pentru $S2$ combinația 1, -1, 1, 1. În figura 6 sînt prezentate valorile ponderilor asociate conexiunilor din rețeaua inferioară.

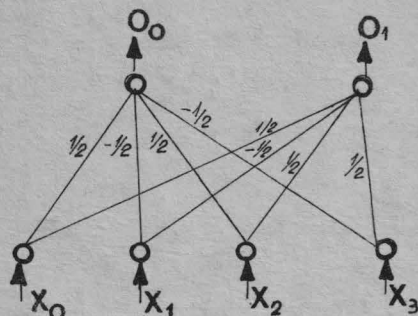


figura 6

Operația de clasificare a unui șablon se desfășoară și în acest caz în doi pași.

Pasul 1. Inițializarea ieșirilor nodurilor în funcție de valorile prezentate la intrările rețelei:

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

unde:

– $Y_j(0)$ este ieșirea nodului j din rețeaua superioară la momentul inițial.

De notat că ieșirea nodului j aparținând nivelului inferior este egală cu diferența dintre numărul de intrări (N) și distanța Hamming calculată între șablonul reprezentativ al clasei j și șablonul aplicat la intrare. Pentru aceasta este necesar ca funcția F_t să nu se satureze pe domeniul valorilor de intrare. Aceasta se întâmplă dacă coeficientul de saturație este $\alpha \geq N$.

Exemplu: Fie șablonul de intrare $S3 = 1100$. Valorile aplicate la intrare vor fi 1, 1, -1, -1. Folosind formula de însumare ponderată a intrărilor în nodurile 0 și 1, obținem:

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

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

Aplicînd funcția de tip rampă (ecuația 5) și ținînd seama de condiția de ne-saturație obținem ieșirile nodurilor studiate:

$$O_0 = F_t(I_0) = I_0, \text{ respectiv } O_1 = F_t(I_1) = I_1$$

Deci, la ieșirea nivelului inferior într-o rețea Hamming se obține o valoare maximă la nodul corespunzător clasei pentru care distanța Hamming dintre șablonul care reprezintă această clasă și șablonul de la intrare este minimă. Deoarece distanța Hamming este pozitivă și mai mică ca N , domeniul de variație a ieșirilor nodurilor de pe nivelul inferior este cuprins între 0 și N .

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

Pasul 2. Calcularea iterativă a valorilor ieșirilor rețelei pînă la convergență:

$$(8) \quad Y_i(t+1) = F_t(Y_i(t) - \epsilon \sum_{k < i} Y_k(t)), \\ 0 \leq i, k \leq M-1$$

unde $Y_i(t)$ reprezintă ieșirea nodului i la momentul de timp t . În această etapă șablonul de intrare este înlăturat iterația desfășurîndu-se în rețeaua superioară cu ieșirile inițializate la pasul 2. De fapt, ecuația 6 reprezintă calculul ieșirilor nodurilor din rețeaua superioară la momentul $t+1$ pe baza sumei ponderate a intrărilor care au valorile de la ieșirea nodurilor de la momentul t :

$$(9) \quad Y_i(t+1) = F_t(T_{ii} * Y_i(t) + \sum_{k < i} T_{ki} * Y_k(t)) = \\ = F_t(Y_i(t) - \epsilon \sum_{k < i} Y_k(t)) = F_t(Y_i(t) - \epsilon \sum_{k < i} Y_k(t))$$

Iterația se desfășoară pînă în momentul în care o singură ieșire rămîne pozitivă celelalte fiind nule. S-a demonstrat ca acest tip de rețea este întotdeauna convergent pentru $1/M$. În plus în urma experimentelor a rezultat o convergență destul de rapidă. Astfel pentru o rețea cu 1000 de intrări și 100 de noduri sînt necesare sub 10 iterații pentru a clasifica un șablon necunoscut.

Rețeaua Hamming are o serie de avantaje comparate cu rețeaua Hopfield cînd aceasta este folosită în operații de clasificare. Acest tip de rețea necesită mai puține conexiuni decît o rețea de tip Hopfield similară. Astfel pentru a clasifica șabloane cu 100 de biți în 10 clase o rețea de tip Hamming necesită 1100 de conexiuni în timp ce o rețea de tip Hopfield necesită aproximativ 10000 de conexiuni. Pe de altă parte rețeaua Hopfield obține rezultate în clasificare mai bune decît o rețea Hamming pentru "pattern"-uri ai căror biți au fost modificați aleator.

BIBLIOGRAFIE

J.J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", Proceedings 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 puțin mai ciudate decât alte limbaje de programare. Una dintre situațiile mai "dificile" de care se lovește un începător, mai ales dacă are o experiență semnificativă în alte limbaje de programare, constă din modul de interpretare (evaluare) pentru o expresie de forma:

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

Înțelegerea unei astfel de notații presupune atât înțelegerea tipurilor de date cu care lucrează limbajul **C** cât și a modului de acțiune a operatorilor care apar în instrucțiuni și a modului în care limbajul **C** tratează prioritățile operatorilor și proprietățile 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 aceeași prioritate este definită o regulă de asociativitate. Astfel, pentru expresia:

a + b * c

se evaluează întâi produsul și apoi suma deoarece operatorul de înmulțire are o prioritate mai mare decât operatorul de adunare. Pentru a schimba ordinea de evaluare se pot utiliza paranteze. Adică, pentru expresia:

(a + b) * c

se va evalua întâi suma și apoi produsul. Pentru expresia:

a + b * c + d

ordinea de evaluare este aceeași cu cea din expresia:

(a + (b * c)) + d

deoarece pentru operația de adunare se consideră regula de asociativitate stînga. Pînă aici textul anterior se aplică majorității limbajelor de programare. În limbajul **C** lucrurile sînt 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 funcțiilor *f* și *g*. Datorită acestui fapt efectele laterale pe care poate să le producă unul dintre aceste apeluri pot să influențeze datele asupra cărora operează cealaltă funcție. Ceea ce este mai grav, datorită acestei libertăți în contexte diferite aceeași expresie se poate evalua în mod diferit. Să considerăm de exemplu următoarele programe:

```
/*
    eval1.c
    program pentru studiul evaluării 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);
}
```

și

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

compilînd și executînd aceste programe utilizînd compilatorul **TURBO C++** se vor obține efecte diferite pentru cele două programe deși ar trebui să fie echivalente ca efect. Ceea ce conduce la diferențe este modul în care se execută instrucțiunea:

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

Dacă se urmărește codul generat de către compilator pentru cele două exemple se va constata că în primul caz secvența de operații corespunzătoare execuției instrucțiunii 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 secvența de operații:

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 și nimeni nu stabilește ordinea de evaluare a operanzilor săi.

Revenind la discuția inițială în tabelul 1 sînt prezentați operatorii limbajului C în ordinea descrescătoare a priorității și 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 cunoaștere a limbajului C constă acum în parcurgerea tabelului și specificarea semnificației fiecărui operator conținut în tabel. De exemplu, ce înseamnă 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 afișa textul :

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

Analizînd rezultatele obținute se verifică faptul că regula de evaluare conform proprietăților de asociativitate este pentru operatorul += de la dreapta la stînga în timp ce pentru operatorul % (restul împărțirii întregi) este de la stînga la dreapta. Deci instrucțiunea:

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

este echivalentă cu secvența de instrucțiuni:

```
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 afișa:

s = 1 t = 0

deoarece expresia:

a * b == c * d

este echivalentă cu:

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

deoarece operatorul de înmulțire este prioritar față de operatorul de comparație ==.

Să considerăm și programul următor :

```
/*
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);
}
```

Executând acest program, utilizând compilatorul **TUR-BO C++**, se obține următoarea secvență de afișări:

```
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 afișează un rezultat de tip float în urma împărțirii dintre o valoare de tip float (3.5) și o valoare de tip int (2). A doua linie afișează un rezultat greșit deoarece s-a încercat afișarea unui rezultat de tip int utilizând un format de tip float. Rezultatul de tip int se obține prin împărțirea a două valori de tip int. Prima valoare se obține 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 întâi se face conversia la tipul int pentru operatorul (int) și 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 execuției acestui program se va afișa 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 instrucțiuni separate de operatorul , se utilizează regula de asociativitate stînga, adică instrucțiunile se vor executa în ordine de la stînga la dreapta, rezultatul obținut fiind reprezentat de valoarea ultimului rezultat.

Să analizăm puțin însă ce se întâmplă în instrucțiunile de la 6 la 13. Comparînd efectul instrucțiunilor din liniile 7, 9, 11, 13 se observă o altă particularitate a limbajului C, referitoare la ordinea în care se evaluează argumentele funcțiilor. Și anume se observă că evaluarea argumentelor (și implicit introducerea lor în stivă) se face de la dreapta la stînga. Corespunzător, pentru linia 7 de exemplu, actualizarea valorii variabilei b se face după introducerea în stivă a valorilor celor două argumente în timp ce în linia 9 actualizarea variabilei b se face după ce s-a introdus în stivă valoarea ultimului argument și 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 ȘI FONTURI (V)

Ion Paraschiv și Tiberiu Spircu

Înterupem pentru moment prezentarea editorului **MatTex** pentru a insera câteva informații ce se doresc a fi adresate utilizatorilor limbajului **Turbo C**. După cum se știe, firma de soft Borland a pus la dispoziția 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 lângă rutinele grafice necesare, conține și un număr de 4 fișiere-font:

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

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

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

```
# include graphics.h
```

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

În primul rând, 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, direcția de scriere, mărimea caracterului:

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

În fine, lista funcțiilor 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 funcții au rolul de a "agăța" 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 întâlni în el, în continuare, o listă de denumiri ale fonturilor existente, denumiri ce sînt folosite doar pentru "demonstrație" (vezi **PCM-2**, funcția `Main Window`):

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

Apoi, în cadrul prototipurilor, vom întâlni funcțiile:

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

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

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

Dintre cele două funcții prototip noi, vom prezenta doar pe prima. Ea are ca efect afișarea 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.fișier.font */
    sprintf (buffer, "%s Caracterele fon-
tului: Fonts [font]);
Main Window (buffer);
    /* afișează numele fontului */
getviewsettings (&vp);
settextjustify (LEFT_TEXT, TOP_TEXT);
moreto (2,3);
buffer [1] = '\0';
wwidth = vp.right-vp. left;
    /* lățimea ferestrei */
lwidth = textwidth ("H");
    /* lățimea 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;
        outtext (buffer);
        /* trimis pe ecran */
        if (ewidth+getx () < wwidth)
            moveto (2, gety () +tex-
theight
("H") +3);
        ++ch;    /* următorul caracter */
    }
}
Pause ();
}
```

Un fișier-font este format din două părți: prima informa-
tivă, urmată de cea efectivă. (Vom exemplifica prin
fontul gothic.chr.)

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

- semnalul de recunoaștere (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 părții efective (2 bytes):
80 00
- denumirea codificată a fontului (4 bytes), de exem-
plu

- 47 4F 54 48 (GOTH)
- lungimea părții efective (2 bytes, interpretați ca
număr întreg) de exemplu:

- FO 20
- Partea efectivă a fontului este formată din patru zone.
Prima zonă conține caracteristici generale, anume:

- un byte semnal
- 2B
- un byte special, de exemplu
- 6 A
- 2 bytes nuli:
- 00 00
- 1 byte ce conține LOWCHR, în cazul nostru ace-
sta este 21:

- 15
- (amintim că HIGCHR = 127)
- 2 bytes ce conțin un pointer spre începutul zonei
a 4-a, în cazul nostru:

- 4E 01
- 1 byte nul:
- 1 byte ce conține ASCMAX, în cazul nostru:
- 15
- 1 byte nul:
- 1 byte ce conține -DESMAX, în cazul nostru acesta
este -7:

- F9
- 5 bytes nuli.

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

- 38 00
- deci descrierea semnului ! începe la adresa
0080h + 014 Eh + 0038h

A treia zonă este rezervată lățimilor efective. Anume,
fiecarui caracter $\geq$ LOWCHR îi corespunde câte 1 byte,
în care se află ACTWID-ul caracterului. Deci această
zonă este formată din HIGCHR-LOWCHR + 1 bytes.

În fine, zona a patra conține descrierile vectoriale ale
caracterelor. O astfel de descriere este formată din sec-
vențe de grupe, fiecare grupă (de câte 2 bytes) cores-
punzând unui nod (punct). De la punctul curent ne
putem deplasa spre noul punct (având coordonatele x,
y) cu "penița jos", în care caz acestui nou punct îi cores-
punde grupa:

- 80 + x 80 + y
- respectiv cu "penița sus" în care caz grupa este:
- 80 + x y
- Fiecare descriere se termină cu 2 bytes nuli
lată, de exemplu, (pe 26 noduri = 52 + 2 bytes) des-
crierea 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 partiții +800%; pentru execuția programelor +300%. Se înlocuiește driverul Smartdriv.Sys. Ocupă în modul rezident 9 ko.

Limbaje

Borland lansează, la un preț foarte mic, versiunea 6, a limbajului orientat pe obiect Turbo Pascal. Produsul se distinge printr-un mediu de dezvoltare integrat care oferă programatorului o productivi-

tate sporită datorită ferestrelor multiple, mouse-ului ca și unui editor multifişier îmbunătățit. Produsul conține și un asamblor integrat care oferă accesul complet la simbolii Pascal. Versiunea profesională completă conține: compilator Turbo Drive, Turbo Debugger, Turbo Profiler, Turbo Assembler.

Calcul

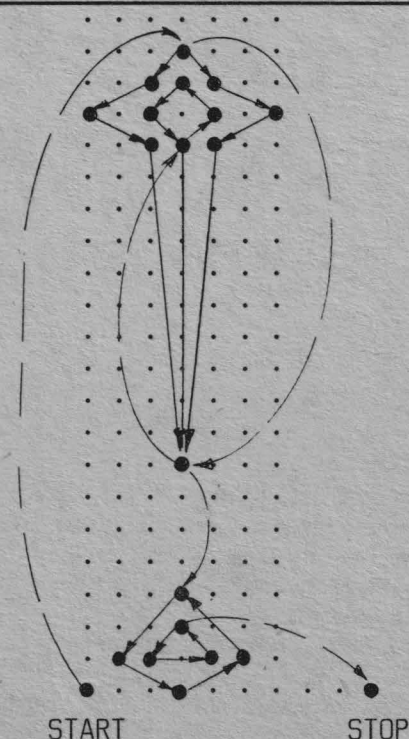
Pachetul Matlab permite efectuarea tuturor calculelor matematice sub formă literală și înlesnește punerea în pagină pentru tipărirea adecvată. Datorită unui număr de biblioteci disponibile, Matlab poate fi folosit în domenii particulare cum ar fi calculul spectral,

chimie, estimarea și identificarea funcțiilor de transfer ale unor sisteme, filtrarea numerică. Este disponibil și pentru MacIntosh.

Conversie de formate

Pachetul Software Bridge al societății Echange asigură conversia între peste 30 de procesoare de text diferite, cu recunoașterea automată a tipului documentului sursă. Este disponibil și pe MacIntosh. Mai mult, fiind compatibil cu pachetul Apple File Exchange se pot efectua transferuri de documente PC/MacIntosh utilizând programele de comunicație 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		



INIȚIERE ÎN PROGRAMARE - LIMBAJUL PASCAL (II)

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

Sperăm că ați reușit să executați cu succes toate programele prezentate în episodul trecut și chiar ați încercat să experimentați idei personale. Dacă ați făcut-o poate că v-ați pus problema — de ce să utilizăm calculatorul pentru a afișa, chiar și într-o formă elegantă, niște 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 și execuția programului. Elementele care trebuie să fie cunoscute în momentul elaborării programului sînt tipul și semnificația datelor și a rezultatelor. Cu alte cuvinte este necesară o specificare riguroasă a "resurselor" accesibile și a "obiectivului" urmărit. Numai în acest mod poate să fie elaborată rapid și eficient "strategia" de rezolvare a problemei sub forma unui algoritm.

Variabile simple

În exemplele pe care le-am prezentat pîna 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 și 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. Operațiile efectuate asupra variabilelor pot determina schimbarea valorilor asociate acestora, dar nu și 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 execuției acestui program se obține :

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

Urmărind programul `variabile_simple` se observă că a apărut o componentă nouă între titlul programului și 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` și două de tip `boolean`.

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

`variabila := expresie`

Prelucrarea descrisă de această instrucțiune se realizează în două etape:

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

Modul în care în programul `variabile_simple` au fost afișate valorile variabilelor nu este foarte elegant deoarece interpretarea valorilor afișate nu se poate face decît urmărind textul programului. Din acest motiv este de preferat soluția 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';  
{ afișare rezultate }  
    writeln('Rezultatele programului sint : ');  
    writeln;  
    write('i = ', i, ', x = ', x:6:2, ', caracter = ', caracter);  
end.
```

Pe ecran se va obține:

Rezultatele programului sint:

i = 2, x = 2.00, caracter = A

Ar fi bine să încercați să urmăriți atent modul în care au fost utilizate virgulele și blankurile în cadrul textelor care se afișează, astfel încât să se prezinte cât mai clar informațiile ce compun rezultatul.

Să mai încercăm să rezolvăm o problemă concretă și foarte simplă: efectuarea operației de schimb valutar. În acest caz datele problemei sînt reprezentate de suma schimbată și de cursul monedei respective față 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 înmulțire între suma schimbată și 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 și că suma care se schimbă nu este mai mare decît cel mai mare număr întreg reprezentabil în calculator). Suma obținută este evident de tip *real*.

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

```
citește curs valutar
citește suma schimbată
calculează suma obținută
afișează rezultatul
```

și 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 ați observat utilizarea caracterului " " ca element de legătură 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 dorința de a obține programe care să fie cât mai ușor de citit și înțeles.

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

```
readln(variabila)
```

Aceste instrucțiuni au următorul efect:

- se citește de la tastatură valoarea introdusă de către utilizator (introducerea valorii se încheie prin

apăsarea tastei RETURN);

- valoarea citită se atribuie variabilei specificate în instrucțiune;
- cursorul afișat pe ecran este poziționat la începutul următoarei linii.

Citirea se poate face și fără trecere la linia următoare, printr-o instrucțiune:

```
read(variabila)
```

Dacă ați lansat în execuție programul "schimb" poate ați sesizat faptul că sînteți așteptat să introduceți datele problemei după care se va afișa rezultatul calculat pe baza celor două valori. De exemplu dacă se dorește calculul sumei ce se obține pentru un curs valutar de 19.88 și o sumă schimbată de 25 se va obține 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 și care este partea afișată de către calculator. Pentru aceleași valori, dacă utilizatorul uită ordinea în care trebuie să facă introducerea datelor, poate să aibă loc următoarea execuție a programului **schimb**:

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

Se observă că în acest caz rezultatul nu este afectat (datorită proprietății de comutativitate a operației de înmulțire). Există însă situații în care ordinea de introducere a valorii datelor este deosebit de importantă și trebuie să fie respectată cu strictețe. Pentru a realiza acest deziderat este bine ca programul să indice utilizatorului care este informația așteptată pentru fiecare citire în parte. Modificînd programul nostru în această idee, se obține:

```
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 și o suma de schimbat de 50 se va obține următoarea imagine pe ecran:

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


Suma obtinuta este 189.71

Să considerăm și o altă variantă de program pentru aceeași 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 câteva modificări. Astfel a fost modificat tipul variabilei `suma_schimbata`, care a fost declarată de tip `integer` și s-a renunțat la utilizarea unei variabile pentru memorarea rezultatului. Deci o primă observație este că în instrucțiunea de tipărire pot să apară și expresii al căror rezultat va fi afișat (evident un lucru asemănător nu se poate realiza pentru operațiile de citire). Dacă se execută programul cu aceleași date ca și în cazul anterior efectul obținut va fi același. Să presupunem însă că executăm programul și în loc de valoarea 50 introducem valoarea 50. (deci o valoare de tip `real`). Atunci, ca efect al execuției, pe ecran vom obține:

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 poziționat pe instrucțiunea de citire a sumei schimbate. În acest mod utilizatorul este informat care dintre instrucțiunile de citire sau scriere (I/O) nu s-a executat corect. Erorile din această categorie pot să fie prevenite prin afișarea 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 și un exemplu de execuție 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 instrucțiunile utilizate în limbajul Pascal este util să aprofundăm puțin discuția referitoare la expresii. Să considerăm de exemplu expresia:

$$21 - 3 * 9 \text{ div } - 6 \text{ div } 2$$

Se observă că în această expresie intervin operatorii `*`, `div` și `-`, iar operandii sînt reprezentați de constante. Valoarea expresiei depinde atît de valorile operandilor, cît și de ordinea în care se execută diferitele operații. Dacă alegem o expresie mai simplă:

$$21 - 3 * 9$$

pare evident faptul că rezultatul este obținut efectuînd întîi înmulțirea și apoi scăderea. Adică, expresia anterioară are același rezultat cu expresia:

$$21 - (3 * 9)$$

Deci evaluarea se face ca și cum operația de înmulțire este "mai importantă" decît operația de scădere. Utilizînd termenii obișnuiți în descrierea limbajelor de programare spunem că operatorul de înmulțire este prioritar față de operatorul de scădere. Să considerăm și expresia:

$$2 * -3$$

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

operatori unari: `-`
operatori binari: `*` / `div` `mod` { operatori multiplicativi }
 `+` `-` { operatori aditivi }
(operatorii care apar pe aceeași linie au aceeași 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

același datorită proprietății de asociativitate a operației de adunare. Ce se întâmplă însă în cazul expresiei:

12 div 4 div 3

În acest caz expresiile (12 div 4) div 3 și 12 div (4 div 3) au valori diferite. Cu alte cuvinte, operația de împărțire nu este asociativă. Conform definiției limbajului **Pascal** în cazul operatorilor cu aceeași prioritate evaluarea se face de la stînga la dreapta sau, altfel spus, operatorii au asociativitate stînga. Corespunzător, expresia

21 - 3 * 9 div -6 div 2

are aceeași valoare ca și:

21 - (((3 * 9) div (-6)) div 2).

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

Pentru datele de tip *boolean* se utilizează următoarele operații specifice: *not*, *and*, *or* și *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:

- Mă duc la film dacă plouă sau rulează un film bun
- Mă duc la film dacă plouă și rulează un film bun
- Mă duc la film dacă plouă și nu rulează un film bun sau nu plouă și rulează un film bun

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

Să identificăm și situațiile în care nu se merge la film pentru fiecare caz în parte:

- nu plouă și nu rulează un film bun
- nu plouă sau nu rulează un film bun
- nu plouă și nu rulează un film bun sau plouă și rulează un film bun !

Condițiile "plouă" și "rulează un film bun" au fost legate în cazul **a**. printr-o operație de tip "și logic" (*and*), în cazul **b** printr-o operație de tip "sau logic" (*or*), iar în cazul **c** prin operația "sau exclusiv" (*xor*). Definițiile 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 și operatori de tip *boolean*:

false or not false and false

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

false or ((not false) and false)

Asupra datelor de tip elementar (standard) – integer, real, byte, boolean și char – se pot aplica și operatori de comparație (operatori relaționali):

< 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 și operator de tip relațional este o valoare de tip *boolean*. Dacă rezultatul unor expresii de forma:

2 < 3

'a' <= 'b'

0 > 3

este clar (*true* pentru primele două și *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 modalități cât mai eficiente pentru stocarea, întreținerea și vizualizarea informației. Obținerea rapidă de informații complexe a devenit indispensabilă procesului decizional. Pe această cale calculatoarele, ca instrumente de gestiune a volumelor mari de date, au pătruns în toate domeniile vieții economico-sociale și ca o consecință, a crescut interesul specialiștilor în a ușura cât mai mult rezolvarea problemelor care decurg de aici.

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

De ce s-a ales relaționalul?

Primul criteriu a fost simplitatea.

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

Simplitate decurge din faptul că toată informația cuprinsă în tabelă se poate vizualiza în orice moment. Și aceasta pentru că atât obiectele, cât și legăturile dintre ele se descriu folosind aceleași mijloace: *tabele*.

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

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

Subliniem acest aspect pentru a diferenția modelul relațional de cele rețea 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 relațional a fost constatarea că, pentru baze mici și medii (mii de rînduri în tabele), timpul de răspuns pentru prelucrarea și regăsirea informației este comparabil cu cel obținut în SGBD-urile rețea și ierarhic.

S-a constatat, de asemenea, că majoritatea aplicațiilor utilizînd baze de date nu depășeau 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 și în centrele noastre de calcul este **ORACLE**.

Oracle și ORACLE

Firma ne atrage atenția asupra felului cum ortografiem acest cuvînt.

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

Apoi trebuie făcută diferența între **SGBD ORACLE** și **sistemul ORACLE**.

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

Funcțiile nucleului sînt:

- supervizează definirea și memorarea informației
- controlează și limitează accesul concurențial la date

- permite salvarea și restaurarea informației
- interpretează SQL

La rîndul lor, *utilitarele* permit:

- transferul de informație din bază către și dinspre fișiere **ORACLE**
- încărcarea în bază a datelor din fișiere 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 instrucțiunile SQL și 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**
- Interfețe cu limbajele ADA, C, Cobol, Fortran, PL/1

SQL*Plus — este conceput ca o interfață prietenoasă cu utilizatorul, care să-i ușureze formularea cererilor, obținerea de informații despre contextul de lucru și formatarea rezultatelor; este foarte util în obținerea rapidă de informații.

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

SQL*Forms — este un utilitar pentru lucrul cu ecrane formate, el poate fi folosit atât la introducerea datelor cu verificarea condițiilor de integritate a bazei, cât și la regăsirea informației.

SQL*Graph — îmbogățește posibilitățile de reprezentare a informației prin introducerea graficelor și, pentru coloanele rapoartelor create în **SQL*Plus**, a atributelor de culoare.

SQL*Menu — permite definirea de meniuri arbore-scente pentru aplicații.

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

SQL*Report Writer — este folosit pentru obține-

rea unei game foarte largi de rapoarte de ieșire.

Interfețele cu limbajele evoluat permit accesul la informațiile stocate în bază și prelucrarea lor cu mijloacele specifice limbajelor procedurale; se poate suplini în acest fel lipsa unora din utilitățile menționate mai sus.

Funcție de configurația hard pe care se lucrează, există soft pentru lucrul cu **ORACLE** în rețea. Ce ne sugerează această impresionantă enumerare?

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

În plus, autorii insistă asupra portabilității și compatibilității SGBD-ului; principiile învățate o dată pot fi utilizate apoi în diferite contexte hard și 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 soluțiile cele mai bune.

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

Standardul ANSI datează din 1986.

Majoritatea SGBD-urilor relaționale suportă azi forme de SQL.

LIMBAJUL SQL

Să începem prin a enumera calitățile cu care limbajul a cucerit piața:

- limbajul este comun tuturor tipurilor de utilizatori, de la administratorul sistemului până la utilizatorii neinformaticieni și constă din câteva instrucțiuni de bază care pot fi asimilate foarte repede

- același limbaj este folosit pentru toate operațiile specifice lucrului cu obiectele bazei: creare, interogare, actualizare, control.

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

Limbajul **SQL** trebuie privit în strânsă legătură cu teoria bazelor de date relaționale. El va oferi soluții pentru rezolvarea operațiilor specifice acestor baze: *selecția* și *join-ul*.

Selecția este operația care permite constituirea de grupuri (seturi) de rînduri (înregistrări) care răspund aceluiași condiții.

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

Instrumentele cu care se realizează aceste operații sînt comenzile, operatorii și funcțiile.

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 utilizați sînt și ei de mai multe tipuri:

- operatori aritmetici (+, -, etc.)

- operatori șir (operatorul de concatenare)
- operatori de comparație (=, <, >, IN, BETWEEN, IS [NOT] NULL, etc.)

- operatori logici (NOT, AND, OR)

- operatori set (UNION, INTERSECT, MINUS)

- alți operatori (DISTINCT, PRIOR, etc.)

Funcțiile pot fi clasificate în:

- funcții care acționează la nivelul fiecărui rînd selectat într-un grup (**DECODE**, **LPAD**, **REPLACE**, etc.)

- funcții de grup (**MAX**, **MIN**, **SUM**, **AVG**, etc.)

- funcții de conversie (**TO-CHAR**, **TO-NUMBER**, etc.)

- funcții care acționează asupra tipului de dată DATE

Menționăm în plus posibilitatea folosirii unei game largi de formate.

Forța pe care o dobîndește limbajul prin combinarea tuturor acestor elemente este uimitoare.

Condiția care trebuie respectată pentru a avea acces la toate facilitățile oferite de limbaj este proiectarea corectă, în spirit relațional a bazei.

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

SQL — LIMBAJ NEPROCEDURAL

Ne oprim asupra acestei probleme pentru că nu mulți 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 operațiile. În cazul **SQL**-ului, aceasta cade în sarcina optimizatorului, parte a nucleului, care alege soluția 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 operațiile dorite și condițiile de constituire a grupurilor.

Ca o consecință a caracterului neprocedural, din **SQL** au dispărut instrucțiunile de realizare a ciclurilor, precum și structurile de tipul **IF THEN ELSE**.

Ne putem descurca fără ele?

În foarte multe situații răspunsul este afirmativ.

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

O situație mai complicată apare atunci cînd avem nevoie de mai multe cicluri imbricate, sau cînd dorim execuția aceleiași secvențe modificînd unul sau mai mulți parametri. Deși în **SQL** nu putem rezolva această problemă, utilitățile ne oferă soluții în multe cazuri (ne referim la **SQL * Forms** și la **SQL * Report Writer**). Și bineînțeles că întotdeauna rămîne la dispoziția programatorului calea interfeței cu limbajele procedurale, care oferă performanțe de timp foarte bune.

Dacă reușim să ne menținem doar în sfera **SQL**-ului,

efortul depus pentru obținerea rezultatelor este mult redus.

Structura IF THEN ELSE o simulăm prin folosirea operatorilor și funcțiilor.

EXPRIMAREA CERERII

Problema centrală a lucrului în **SQL** rămîne felul în care cerem numai informațiile de care avem nevoie la un moment dat; operațiile principale pe care le efectuăm asupra bazei sînt, așa cum am arătat mai devreme, selecția și join-ul.

Comanda folosită pentru regăsirea informației este **SELECT**.

Deci, **SELECT**ăm dintr-o tabelă coloane reale sau expresii. În mod obligatoriu trebuie să știm despre aceste coloane cum se numesc și cărei tabele aparțin. Scriem

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

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

Dorim mai puține rînduri? Enunțăm condițiile care restricționează numărul de rînduri în clauza **WHERE**.

Vrem ca ele să apară într-o ordine anumită?

Încheiem prin menționarea clauzei **ORDER BY** ur-

mată de cheile de sort.

Dacă dorim să dispunem de informații memorate în mai multe tabele (operația de join), menționăm toate coloanele, indiferent de tabelă, avînd grijă să calificăm numele coloanelor care nu sînt unice cu denumirea tabelului 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 tabelor. Pentru a limita numărul de rînduri incluse în set, condițiile restrictive le înscrîm, firește, în clauza **WHERE**. Scriem deci

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

Pentru a obține informații totalizatoare referitoare la o tabelă aplicăm asupra numelor coloanelor funcții de grup.

Dacă dorim ca aceste rezultate să se refere nu la ansamblul tabelului, 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 Redacției

Referitor la articolul "Despre ecrane și memorarea lor", vă semnalăm cîteva erori de redactare care s-au strecurat în textul articolului și, în același 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 același 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 mașină 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 și linia 9999), cît și la **COMENZI BASIC** (penultimul rînd din pagina 39), cuvîntul **SCREEN** este urmat de simbolul de șir \$, 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 continua comanda"; ... în loc de: **"Programul BASIC trebuie sa continuiam comanda"**

- în același 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 și 22527 și este împărțită 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 mașină, se va citi:

"BASIC (cu PLOT, DRAW și CIRCLE). După desenare sau după încărcare de pe bandă..." în loc de:

"BASIC (cu PLOT, DRAW și CIRCLE). După desenare de pe bandă..."

TOTUL DESPRE MEMORIE(I)

Marius Sturzoiu

Atunci cînd IBM PC și-a început marșul triumfal, se părea că un RAM de 640 K va fi suficient pentru toate aplicațiile. Foarte curînd bariera s-a prăbușit. La început a fost dorința de a se stoca în RAM cît mai multe date; au cerut-o 123 a lui Lotus, Oracle și alții. Mai tîrziu a beneficiat și 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 convențională*. Aceasta se întinde de la 0- 640 K. Este zona în care se execută toate aplicațiile 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. Restricția provine de la aceea că sistemul de operare DOS a fost conceput pe un PC echipat cu microprocesorul 8088. Acesta avea magistrața de memorie de 20 biți, permițînd adresarea în intervalul 0 - ($2^{20} - 1$) adică 1Mo. S-a hotărît atunci ca zona 0-FFFFH640ko) să fie destinată aplicațiilor — deci RAM, iar zona A0000H- FFFFFH (384 Ko) să fie destinată sistemului — deci ROM, plăci de memorie, etc.

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

— *memoria înaltă* (high memory). Este zona cuprinsă între AO-OOOH-FFFFH, din considerentele anterioare.

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

Din păcate la 80286 trecerea mod real-mod protejat se face ușor; în sens invers numai prin reinițializarea microprocesorului, timp în care sistemul de întreruperi este blocat. Cu începere de la 80386, trecerea se face și invers fără dificultăți.

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=...). Și nu uitați, nu folosiți memoria extinsă pe o configurație XT.

Apariția lui Windows 3.0 a modificat această situație. Utilizînd driverul HIMEM.SYS, instalat în CONFIG.SYS care are la bază protocolul XMS (eXtended Memory Specification), aplicațiile adaptate lui Window 3.0 se execută în memoria extinsă, pentru microprocesoarele 80286, 80386, 80486. Aplicațiile DOS lansate sub Windows 3.0 se vor executa în memoria convențională pentru un PC echipat cu 80286; pentru PC echipat cu 80386 se pot executa in-

clusiv în memoria extinsă.

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

— *memoria extinsă, paginată* (Expanded memory). Conceptul a apărut în 1985 cînd Lotus și Intel au dezvoltat specificația soft-hard EMS 3.2 (Expanded Memory Specification — specificația de memorie extinsă). Ulterior, la aceasta a aderat și Microsoft apărînd specificația LIM (Lotus Intel Microsoft). Pentru a funcționa, specificația necesită: plăci memorie special concepute (pe modelul Intel Above Board) și 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 inițial 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 extinsă. Dacă se dorește accesul la o altă zonă din memoria extinsă (specificația admițînd un maximum de 8 Mo) intră în acțiune două mecanisme:

— întîi are loc salvarea conținutului actual al ferestrei în memoria extinsă (page swapping).

— apoi are loc transferul noilor 4 pagini selectate.

Evident procesul are loc transparent pentru utilizator, fiind lăsat în seama driverului de memorie instalat în CONFIG.SYS și a plăcii de memorie extinsă.

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

Deci execuția programelor trebuia să se facă tot în memoria convențională.

Pașul următor l-a marcat apariția normei EEMS (*Enhanced Expanded Memory Specification*), o evoluție a normei anterioare cu care, de altfel, este compatibilă. Ea a fost definită cu începere din 1987 de Lotus, Intel, Microsoft și Ast și 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 și 1 Mo. Adresa ferestrei poate fi definită și modificată prin program. Norma suportă 32 Mo memorie extinsă. De data aceasta memoria paginată poate primi cod executabil. Windows 2.10, de exemplu, folosește această zonă pentru a permite executarea simultană a mai multor aplicații.

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

— *memoria extinsă (paginată)*, sub forma unor plăci de memorie și a unui driver în CONFIG.SYS poate fi instalată și utilizată pe mașini echipate cu 8088, 80286, 80386 etc. Ea va fi adresată sub 1 Mo. Memoria extinsă, sub forma unor plăci de memorie și a unui driver în CONFIG.SYS poate fi utilizată pe mașini echipate cu 80286, 80386 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 și folosită ca memorie extinsă.

— o placă de memorie extinsă necesită regiștri de control a memoriei și o MMU (Memory Management Unit) pentru a controla acești regiștri. O placă de memorie extinsă nu are astfel de necesități.

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

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

Probleme specifice bazelor de date documentare

Țeculescu Virgil

Să presupunem că aveți la dispoziție o vastă bibliotecă și doriți să vă îmbogățiți cunoștințele într-un anumit domeniu (ex: economia de piață).

Cum procedați?

Este incomod să tot căutați printre fișele bibliotecii diferite cărți și reviste în care parcă ar exista ceva informații de acest gen. Poate dura ore bune și efort de concentrare.

Dacă aveți însă la dispoziție un computer, problema dumneavoastră e ca și rezolvată; împreună cu utilitarul CDS/ISIS, în câteva secunde veți afla cărțile și articolele ce tratează acest domeniu.

Acest utilitar vă pune la dispoziție o metodă de interogare a bazei de date bibliotecă, după anumite cuvinte cheie. Dacă am avut grijă ca printre aceste cuvinte cheie să se afle și economia de piață, vor fi afișate toate publicațiile în care acest domeniu a fost atins.

Domeniul de activitate al informaticii documentare pornește tocmai de la corelația dintre creșterea volumului de date stocate și necesitatea reducerii timpului de acces la informație.

Baza de date documentară este o bază de date a cărei informație utilă o reprezintă tocmai datele referitoare la publicații (titluri de cărți sau articole, locul unde pot fi găsite, precum și alte informații de interes). Ca orice bază de date, cea documentară are sarcina de a permite utilizatorilor interogări pe diferite câmpuri (autor, domeniu, etc.) pentru accesarea informației dorite.

Un program orientat spre proiectarea și 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 spațiu mic de "manevră" pentru proiectant.

Din punct de vedere al proiectării, baza de date documentară implică unele diferențe față de cele nespecializate. Vom avea și aici fișiere cu formate de introducere și de afișare a datelor, tabele de definire a câmpurilor, tehnici de regăsire, etc.

Tipul informației vehiculate impune însă și existența unor tehnici particulare cum ar fi: interogarea prin dicționar și prezența fișierelor de istoric, a dicționarului vid sau de tehnici de indexare.

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

Să revenim la publicațiile noastre alegând un exemplu didactic, care să răspundă la întrebarea: "Ce interesează la o carte?"

— titlul	A	11
— lista autorilor	A	12
— cota cărții (cod pentru regăsire)	A	
— data apariției	A	
— editura	A	
— lista descriptorilor		13
— rezumatul cărții	A	

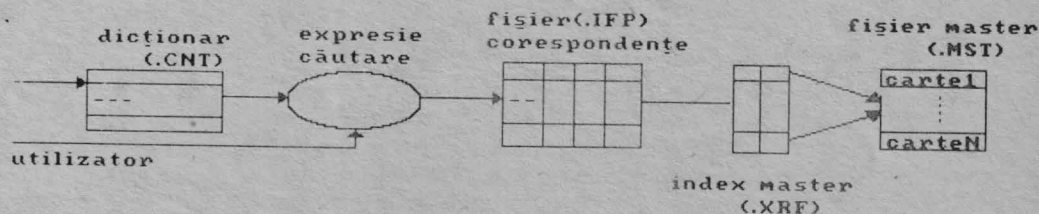
Deci am ales un fișier master cu șapte câmpuri, fiecare înregistrare conținând informații ce caracterizează o carte.

Ca scurtă explicație, lista de descriptori reprezintă un set de cuvinte-cheie (expresii) care descriu schematic conținutul cărții.

În tabelul de mai sus am notat cu 'I' câmpurile necesare la regăsirea informației (câmpuri posibil cunoscute de utilizator la interogare), iar cu 'A' pe cele ce interesează și vor fi afișate. Câmpurile notate cu 'I' vor fi tratate și introduse într-un fișier special (dicționar) ordonat alfabetic în vederea accesării rapide. De menționat că dicționarul este memorat în forma arborescentă (B-tree). Deci dicționarul va "condensa" informațiile mai multor câmpuri. Acesta va fi vizualizat la cerere de utilizator, putându-se selecta de pe ecran autorii sau domeniile de interes. De remarcat că la o singură interpretare se pot selecta mai mulți termeni ai dicționarului, fiind disponibile mai multe tehnici cum ar fi AND, OR și NOT.

Să presupunem termenii selectați de pe ecran a, b și c, cărora le vor corespunde listele de publicații A, B și C. Tehnicile vor avea ca rezultat operațiuni de intersecție, reuniune respectiv scădere a mulțimilor A, B și C.

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



Exemplu:

Dorim găsirea cărților (articolelor) în care este atins domeniul "economia de piață" și "privatizare".

Avem la dispoziție două metode:

1. interogarea prin dicționar:

Navigăm un dicționar, selectăm sintagma "economia

de piață", selectăm și cuvântul "privatizare" și lansăm cererea de interogare a bazei de date. Vom obține toate publicațiile în care ambele domenii sînt tratate.

2. interogarea explicită:

Edităm o linie astfel: <economia de piață.* privatizare>, '*' are rolul de AND logic. Lansînd cererea de interogare vom obține același rezultat ca și în cazul precedent.

Obs.: Putem căuta toate publicațiile în care apar noțiuni de economie. Interogarea se va face astfel: <economia\$>, deci rădăcina, urmată de '\$', furnizînd toate cărțile în care apar informații legate de orice tip de economie (nu doar cea de piață).

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

fișierului master.

Această corespondență este conținută într-un fișier special, fișierul tehnicilor de indexare.

Concret, la exemplul nostru aceste tehnici sînt:

- preluare text dintre delimitatori (13);
- preluare precedată de descompunerea pe subcîmpuri (12);
- preluare precedată de descompunerea cuvînt cu cuvînt a textului.

Detaliînd mecanismul descompunerii titlului cărții (11), vom constata introducerea în dicționar și a numeroase conjuncții, prepoziții și a altor cuvinte nedorite. Pentru eliminarea acestui neajuns s-a introdus mecanismul "dicționarului vid", care constă în căutarea unui cuvînt într-o listă existentă de cuvinte de legătură. Vor fi trecute în dicționarul de interogare numai cele ce nu au fost găsite în dicționarul vid.

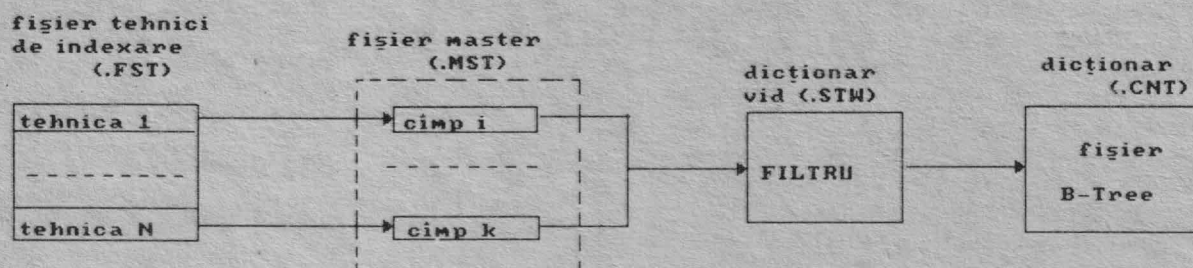


fig.2. Construcția dicționarului (CDS/ISIS)

O problemă deosebită a acestor programe destinate proiectării bazelor de date documentare o reprezintă alocarea spațiului de memorie. Se poate ușor observa din analiza exemplului fișierului de mai sus diferența între necesitățile de spațiu de la carte la carte. Unele liste de descriptori sau rezumate pot fi foarte scurte și altele foarte lungi, iar numărul autorilor poate fi și el diferit.

Apare deci necesitatea compactării zonei de memo-

rie, pentru a preveni un consum exagerat de spațiu, care se realizează prin memorarea lungimii fiecărui cîmp de dimensiune variabilă.

Recapitulînd, mergem la bibliotecă, deschidem computerul, apelăm ISIS, vizionăm dicționarul autorilor și descriptorilor, selectăm termenii de interes, interogăm baza de date și căutăm în bibliotecă publicațiile listate.

STOP

NEWS

Paradox SQL Link

SQL (*Structured Query Language* — pronunțaji sequel) este standardul în manipularea datelor pentru SGBD relaționale. Utilizînd **Paradox SQL Link** veți putea (atenție, 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 funcții C și 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 MAȘINĂ Z80 PE CALCULATOARELE COMPATIBILE SPECTRUM (III)

Bogdan Georgescu
Bogdan Iordănescu

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

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

Comunicația cu celelalte periferice se realizează prin intermediul unui port de intrare și al unui de ieșire, care împreună sînt denumite în mod convențional "portul de I/O de adresă #FE". Vom vedea în continuare de ce această denumire este mai mult o convenție de limbaj decît o denumire exactă.

Avem în codul mașină **Z80** următoarele instrucțiuni 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 regiștrii: A,B,C,D,E,H,L.

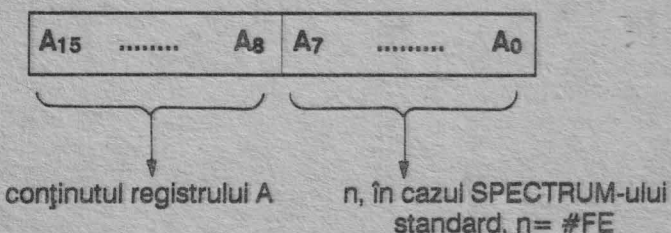
— INI sau IND; OUTI sau OUTD

— INIR sau INDR; OTIR sau OTDR

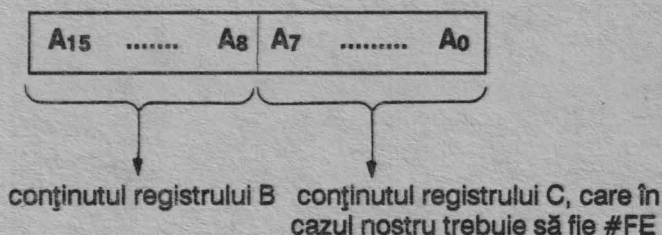
Vom explica în acest număr comportarea portului #FE ca port de input. Să vedem întâi, dintr-un punct de vedere mai apropiat de hardware, cum face microprocesorul **Z80** dacă trebuie să citească date din exterior.

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

— în cazul instrucțiunii IN A, (n):



— în cazul instrucțiunii IN r, (C):

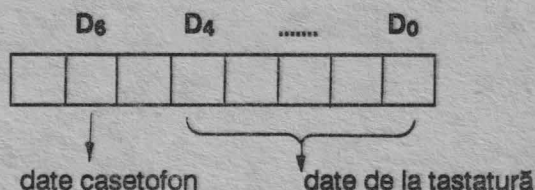


— în cazul instrucțiunilor repetitive, comportarea este identică, dar pe liniile A8-A15 se pune conținutul registrului B decrementat.

Protocolul de I/O este, în realitate, puțin mai complicat, dar ceea ce ne interesează este că microprocesorul citește, de pe magistrala de date, informațiile transmise de periferic, în registrul A la IN A, (n), în registrul r la IN r, (C) sau în locația de memorie adresată de registrul dublu HL la instrucțiunile INI, IND, INIR și INDR.

Din această scurtă caracterizare reiese faptul că adresa unui port de input-output este un număr pe 16 biți, 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 informația care vine pe magistrala de date, sub forma unui număr de 8 biți:



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

Să examinăm următorul desen:

A15	SP	SS	M	N	B
A14	CR	L	K	J	H
A13	P	O	I	U	Y
A12	0	9	8	7	6
A11	1	2	3	4	5
A10	Q	W	E	R	T
A9	A	S	D	F	G
A8	CS	Z	X	C	V
	D0	D1	D2	D3	D4

Probabil v-ați închipuit deja o rețea de fire (o matrice) în care fiecare intersecție corespunde unei taste. Observați 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 și coloana corespunzătoare acestuia se pun în contact, deci starea coloanei devine aceeași cu starea liniei. În mod normal, liniile magistralei de date sînt menținute la starea "1" logic. Dacă o linie este pe "0" logic și una dintre tastele de pe această linie este apăsată, coloana corespunzătoare ei va trece și 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 observația 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:

A15 A14 A13 A12 A11 A10 A9 A8

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:

D7 D6 D5 D4 D3 D2 D1 D0

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 și 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 și 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)    ; citește portul
AND     A,#04      ; testează bitul 2 din A
JR      Z,APASAT   ; dacă bitul este 0, salt la
                      ; rutina corespunzătoare
```

NU -----
APASAT -----

Pentru a reține mai ușor ce bit trebuie resetat pentru a selecta una din liniile de 5 taste, să le reprezentăm așa 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 corespondența dintre biții citiți de pe magistrala de date și 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 ști exact care din rînduri conține tasta apăsată. De exemplu, punînd A15 și A11 pe 0, dacă la citirea portului vom avea D2 pe 0, nu vom putea ști sigur care din tastele 3 sau M au fost apăsate, dar sigur una din ele a fost apăsată.

O altă observație ar fi cu privire la denumirea "portul #FE". Tot din schemă rezultă că, de fapt, sînt 8 porturi cu adresele obținute așa cum am arătat, iar combinațiile de rînduri citite în același timp ar reprezenta combinații între aceste porturi.

A mai rămas un singur lucru de spus, și 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 conține 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 și folosește citirea tastaturii pentru a mișca pe partea de sus a ecranului ... veți 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 mașină, evident cu anumite restricții de memorie, temporizare etc. În subrutina de tratare a întreruperii, pe lîngă citirea tastaturii prin metoda prezentată anterior, se citește tastatura și prin apelarea rutinei din ROM, de la adresa #02BF, care este indispensabilă rulării BASIC-ului.

```
ORG     #FF 00      ; subrutina de inițializare începe
INIT    LD     HL, #F000 ; la #FF00
        LD     B, 0
```


Home Computer

LIMBAJUL Z80 (III)

LD	A, #F1	; Se creează tabela de întreruperi
INIT0	LD	(HL), A ; între adresele #F000 și F100 inclu-
INC	HL	; siv, conținând 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 mașinii, 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, 180, 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 spațiu pentru DATES
POZ	DEFB	0 ; inițializează locațiile POZ, ROT și
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	; mișcarea mașinii și pentru
POP	IX	; citirea tastaturii prin ROM
POP	AF	
POP	BC	
POP	DE	
POP	HL	
EI		
RET		
BUFFER	DEFS	64 ; alocă spațiu pentru buffer
MASINA	LD	A, #FE ; IX conține 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)	; combinații nu este întâlnită
AND	#02	; desenează mașina nemișcată
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)	; ționează indicatorii pentru poziția la
AND	#0F	; dreapta (întoarcere) și desenează
JP	NZ, DESEN	
LD	(IX+1), 4	
DEC	(IX+2)	
JP	DESEN	
DRT0	BIT	2, (IX+1) ; stabilește indicatorii pentru poziția
JR	NZ, DRT1	; următoare spre dreapta
INC	(IX+1)	
JP	DESEN	
DRT1	LD	A, (ADR) ; Se incrementează adresa de afișaj,
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)	; poziționează 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	DES9	PUSH	DE
DESEN	LD	A, (POZ)		PUSH	BC
	LD	HL, DATED		LD	BC,4
	SLA	A		LDIR	
	JR	NC, DES0		POP	BC
	LD	HL, DATES		POP	DE
DES0	LD	DE, BUFFER		INC	D
	CALL	DES1		DJNZ	DES9
	PUSH	HL		RET	
	LD	A, (POZ)	CONT	DEFB	3
	AND	#03	CONTOR	LD	A, (CONT)
	JR	Z, DES3		DEC	A
	LD	D,4		AND	#03
	RR	A		LD	(CONT), A
	JR	C,DES2		JR	Z,CONT0
	LD	B,8		POP	HL
DES2	LD	DE, BUFFER+31		JP	DESEN
	LD	HL, BUFFER+30	CONT0	LD	A, (POZ)
	PUSH	BC		INC	A
	LD	BC,31		AND	#83
	LDDR			LD	(POZ), A
	POP	BC		SLA	A
	DJNZ	DES2		RET	
DES3	POP	HL	DES1	XOR	A
	LD	A, (POZ)		LD	B,8
	AND	#03	DES 10	PUSH	BC
	JR	Z, DES4		LD	BC,3
	LD	BC,24		LDIR	
	ADD	HL,BC		LD	(DE), A
	RR	A		INC	DE
	JR	C, DES4		POP	BC
	ADD	HL, BC		DJNZ	DES10
DES4	LD	DE, BUFFER+32		RET	
	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			

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

Indicatorul "POZ" specifică pe primii 2 biți poziția intermediară a mașinii (specifică partea de jos care se afișează) și este incrementat modulo 4, iar ultimul bit specifică orientarea (0 - dreapta, 1-stînga). Indicatorul "ROT" specifică pe primii 3 biți numărul de "shift-uri" spre dreapta înmulțit cu 2 (la fiecare mișcare mașina se deplasează cu 2 biți). Valorile pot fi 0, 1, 2, 3, pentru stînga și 1,2,3,4 pentru dreapta.

Locația "ADR" specifică adresa de afișare pe ecran.

Succes!

"SPARGEREA" PROGRAMELOR

Bogdan Georgescu
Bogdan Iordănescu

Cum ce-ar putea însemna un asemenea titlu? Iată o scurtă explicație: ne propunem să abordăm un alt mod de a câștiga experiență în programarea pe SPECTRUM, și anume însușirea experienței altora. Ne vom transforma deci (numai pentru un timp ...) în "piraiți" și vom încerca să vedem care sînt posibilitățile de a "sparge" protecția unui program utilitar sau a unui joc, mai ales cînd autorul programului a făcut tot ce a știut mai bine pentru a ne împiedica să o facem. Tocmai datorită varietății protecțiilor, 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 imaginație, dar necesită și o cunoaștere 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 puțin protejat, care poate conține și cod mașină sub diferite forme. Ne vom ocupa de acest bloc din toate punctele de vedere.

De obicei, BASIC-ul folosește la trecerea din interpretor la nivelul codului mașină, deci listarea urmărește 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 și a adresei de lansare.

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

O metodă mai puțin 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, reacționează ciudat sau deloc, se poate încerca încărcarea lui cu MERGE ". Această încercare poate eșua dacă programul este protejat împotriva încărcării cu această instrucțiune (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 funcțiune o rutină care să fi modificat sau mascat părți din program. Dacă acest lucru nu se observă la timp, se poate rata totul.

Dacă după cîteva jonglerii cu caracterele de control, "intuiția" vă spune că, totuși, e prea simplu ca să fie adevărat, vă recomandăm o metodă de privire a BASIC-ului prin perspectiva codului mașină și a controlului asu-

pra memoriei, care duce la rezultate sigure.

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

Acum să scriem și să asamblăm următoarea rutină în GENS:

```

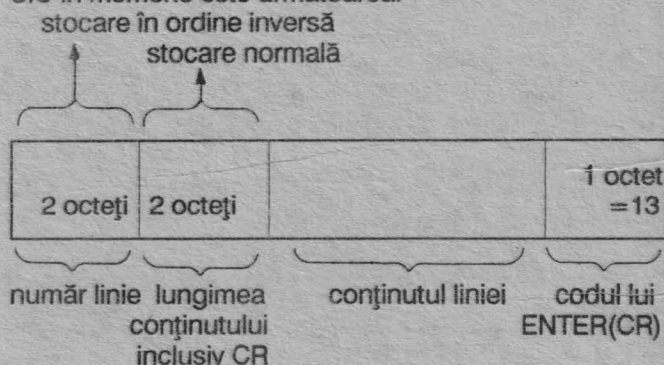
ORG      54 900    ; de exemplu!
HEADER   LD      IX, 30 000; header-ul se încarcă
          LD      DE, 17    ; la adresa 30 000 și
          SCF          ; are lungimea 17 байți
          XOR     A
          CALL    #0556
BLOC      LD      IX, 30100 ; programul propriu-zis se
          LD      DE, lung  ; încarcă la adresa 30 100
          SCF          ; și 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 fișierului (în cazul BASIC, conține 0)
- octeții 01 - 0A: numele programului
- octeții 0B-0C: lungime program + variabile
- octeții 0D-0E: linia de autolansare a programului
- octeții 0F-10: lungime BASIC fără variabile

Partea de program cu numele (label-ul) "BLOC" încarcă fișierul BASIC ca pe un fișier headerless. Dacă se cunoaște modul de stocare în memorie a BASIC-ului și variabilelor, se poate realiza un alt fel de citire a acestuia, fără a apela la interpretor, deci avînd control deplin asupra situației (în general protecțiile la BASIC folosesc particularitățile rutinei de listare din ROM).

Adresa de început a programului BASIC se ia din variabila de sistem PROG, la adresa 23635-23636 și 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 reținut că numărul de linie este stocat în ordine inversă față de celelalte constante întregi pe 16 biți (adică octetul cel mai semnificativ este primul).

Pentru a vedea cum este stocat conținutul liniei, să introducem următorul program BASIC care să listeze conținutul 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 întâlnirii 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) și a lungimii liniei (27).

Constantele numerice care apar în linie sînt stocate întîi ca șir (fiecare cifră este stocată ca un caracter), pentru a fi listate ușor, apoi urmează caracterul de control 14 și reprezentarea în virgulă mobilă, pe 5 octeți, 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 și să aplicăm cunoștințele despre BASIC în cazul jocului "SPLITTING".

Acest joc este compus din 3 blocuri:

- primul bloc de BASIC, cu lungimea de 150 octeți și autolansare la linia 10
- al doilea bloc de BASIC, cu lungimea de 735 octeți

și autolansare la linia 0

— al treilea bloc, care este un fișier 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 protecție de culoare. Cu instrucțiunea "INK 0" problema este rezolvată și se poate citi BASIC-ul, care conține doar un mesaj care ne anunță că BORDER-ul rămîne negru pe durata încărcării programului, instrucțiunea "INK 7" care a cauzat "albirea" ecranului și instrucțiunea LOAD", ceea ce înseamnă că primul bloc nu are nici o importanță, deoarece este șters la încărcarea următorului.

Faptul că BORDER-ul rămîne negru la încărcarea programului, ca și faptul că blocul de cod este headerless ne indică faptul că programul folosește o rutină proprie de încărcare. Inexistența unui alt bloc de cod arată că rutina în cod mașină este inserată în al doilea BASIC. Șansele ca această rutină să poată fi oprită cu BREAK sînt minime, deci încercăm instrucțiunea MERGE. Vedem cu această ocazie cam cum acționează o protecție la MERGE, și anume blocarea sistemului. Această protecție este realizată de altfel foarte simplu: se scrie BASIC-ul într-o singură linie și în locațiile care conțin lungimea liniei se încarcă valoarea 0. Cum rutina de MERGE din ROM folosește lungimea fiecărei linii pentru ordonarea programului, se înțelege de ce apare blocajul. Această modificare nu afectează rularea programului, în el neexistînd instrucțiuni de salt etc. Se mai poate proceda și altfel pentru a încărca BASIC-ul: se încarcă într-un copier (COPY 86/M), se dezactivează linia de autorun (cu opțiunea R, în COPY 86/M), se salvează pe casetă și apoi se încarcă cu LOAD, fără ca programul să mai intre în execuție. În cazul SPLITTING-ului, veți avea o surpriză neplăcută dacă veți 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 și 16, urmat de 1, ceea ce înseamnă PAPER 1 și INK 1. Urmează RANDOMIZE USR 24070, care este prima instrucțiune BASIC executată. Rezultă că tot ce urmează ca instrucțiuni BASIC (și care ocupă, probabil 24 070 - 23 760 = 310 octeți) sînt pentru "derută". S-ar putea ca în acest interval să fie și instrucțiuni de cod mașină, dar asta vom vedea în continuare. Listînd mai departe, observăm caracterul de control 9, care mută cursorul cu un caracter la stînga, și repetat va cauza "acoperirea" instrucțiunii "RANDOMIZE". Urmează schimbarea culorii în vizibil (17,1, apoi 16,6 și 19,1 adică PAPER1, INK 6 și BRIGHT1) și 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 "ușura" munca. Iată de ce

PARADOX 3.5

Paradox 3.5 este, așa cum arată și numele, un paradox: un SGBD (sistem de gestiune a bazelor de date), relațional (deci ceva sofisticat), ușor de învățat și utilizat. El permite: **1.** organizarea informației în tabele (relații), datele fiind alfanumerice, numerice, currency (se referă doar la sume de bani), dată; **2.** o tabelă poate conține până la 2 milioane de înregistrări, cu până la 4000 de caractere fiecare: 255 cîmpuri cu până la 255 caractere fiecare; **3.** informația stocată poate fi schimbată, regăsită în orice formă dorită, incluzînd efectuarea de calcule asupra ei, precum și reprezentări grafice. **Paradox** implementează strălucit QBE (*Query by Example*). Se dă pur și simplu un exemplu de răspuns dorit și se obțin toate răspunsurile adecvate. De notat că într-o singură "query" pot fi legate până la 24 tabele. Fiecare tabelă poate avea atașate până la 15 formate speciale de formatare a datelor și tot până la 15 forme diferite de așezare a lor în rapoarte. Bineînțeles, atît formulele cît și rapoartele pot conține informații din tabele diferite. **4. Paradox** este un produs cu precădere vizual. Totul, începînd cu proiectarea rapoartelor, formatelor, QBE-urilor și terminînd cu utilizarea pe scară largă a meniurilor, se produce pe ecran. **5. Paradox** poate fi folosit pe o rețea locală la fel de ușor ca de către un singur utilizator (fără a se utiliza comenzi speciale). Se asigură blocarea automată a fișierului 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 secvențe de taste care ulterior pot fi executate. **8.** Accesul la tabele poate fi controlat printr-un mecanism de codificare, asigurîndu-se protecția datelor. **9.** Utilizînd PAL (*Paradox Application Language*), limbaj structurat, se creează un mediu de programare permițînd dezvoltarea de aplicații.

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 și utiliza foi de calcul conținînd o mare cantitate de date.
- o grafică deosebită. Puteți alege între mai multe reprezentări grafice suprafețe, linii, XY, diagrame, inclusiv tridimensionale.

O bibliotecă de simboluri grafice vă va permite o prezentare deosebită a graficelor obținute. Fiecare obiect poate fi deplasat, colorat, redimensionat. Există și posibilitatea de a crea o înlănțuire de imagini, realizînd un "show" de diapozitive. În aceeași direcție s-a prevăzut importul (exportul) unor noi tipuri de fișiere grafice: EPS (Encapsulated PostScript), PCX, CGM (primele două formate pot fi exportate, ultimul numai importat). Veți putea alege din 9 fonte diferite. Dispuneți de asemenea de o funcție de pre-vizualizare, adevărat WYS/WIG.

— funcția de "goalseek", permițînd 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 și de o mașină pe bază de 80286 pot lansa în execuție 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. Și dacă în plus dispuneți și de accesul la o bază de date amplasată pe un mini sau pe un server, Paradox 3.5 și Paradox SQL Link permițîndu-vă accesul prin intermediul cererilor SQL, veți putea asigura o prezentare deosebită a analizelor dumneavoastră utilizînd facilitățile lui Q.P.

listarea obișnuită nu prea era recomandabilă. Mai de parte, de la adresa 30 226 pînă la 30 414 sînt octeți 0 (NOP) care ne liniștesc în privința 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 și dezasamblăm de la adresa 30 415, unde vom avea următoarele instrucțiuni.

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 octeți la adresa 41320 și se predă controlul acestui bloc.

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

Once upon the time "THE UNIX"

Ion Mușlea

UNIX - portretul unui standard

În momentul de față prețul unui WORK-STATION al firmei SUN este echivalent cu cel al unei interfețe grafice de tip WINDOWS și a citorva produse soft existente sub DOS (1-2-3, WordPerfect, Ventura). Și totuși unii beneficiari ezită încă să aleagă o mașină performantă de tipul celei amintite. Aceasta din mai multe rațiuni: pe de o parte complexitatea UNIX-ului sperie un utilizator obișnuit cu DOS-ul sau MACINTOSH-ul, iar pe de altă parte există celebra problemă a compatibilității diverselor versiuni UNIX. Altfel zis, un utilizator de WordPerfect își poate schimba liniștit PC-ul având garanția că procesorul de texte va funcționa la fel de bine și pe noua mașină, pe când cel ce folosește Word5 trebuie să aibă grijă să achiziționeze un UNIX SCO, căci Word5 nu funcționează 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, forța unui UNIX ce permite de exemplu portabilitatea unui software cum ar fi Oracle pe sute de mașini de orice tip (micro, mini sau mainframe) ar cam trebui să dea de gândit. Și apoi nu trebuie uitat faptul că în epoca rețelelor de calculatoare, raza de acțiune a UNIX-ului (mult mai mare decât în cazul DOS sau OS/2) l-ar putea impune drept soluție definitivă.

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

Având în vedere că atât OSF-ul cât și UNIX INTERNATIONAL au acceptat X/OPEN și POSIX, putem spune, fără teama de a greși, că UNIX-ul se îndreaptă spre o mult așteptată unificare. De remarcat și tendința tuturor "barosanilor" de a se alia la UNIX. Acest lucru are două explicații. În primul rând dorința guvernului american (care e cel mai important beneficiar) de a se axa în domeniul militar pe acest sistem; în al doilea rând studiile arată că perspectivele răspîndirii UNIX-ului sînt foarte mari. Astfel, în perioada 1987-1990, în America, s-a înregistrat o creștere cu 20% a pieței 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! Și apoi să nu uităm că UNIX-ul oferă incomparabil mai multe alternative de lucru superioare celor existente sub DOS, mai ales în domeniul industrial și științific.

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 întâmplare? — rulează sub UNIX. Și, în sfîrșit, cele mai sofisticate 4 baze de date existente (Sybase, Oracle, Ingres și Informix) și-au făcut — toate !!! — debutul sub UNIX. Acest fapt a permis ca numai pentru mașinile de tip IBM RISC 6 000 să avem peste 500 de "tools"-uri necesare în aplicații de gestiune.

În încheiere mai trebuie subliniat că mașinile UNIX prezintă un excepțional raport preț-performanță. Chiar și în comparație cu facilitățile grafice oferite de DOS sau OS/2. În condițiile în care oferă o viteză și niște aplicații excepționale, s-ar prea putea ca UNIX-ul să ofere și soluția 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, deși 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 și, împreună cu un alt inginer pe nume Dennis Ritchie, hotărăște să-și continue studiile. Cei doi vor descoperi un DEC vechi (cu un PDP-7) complet neutilizat, și hotărăsc să adapteze "SPACE TRAVEL" pe această mașină. În acest scop ei au dezvoltat și niște 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 și un iz de clandestinitate pentru că de cînd cu abandonarea MULTICS, subiectul a devenit tabu în cadrul laboratoarelor BELL.

Thompson și Ritchie au avut o idee care i-a determinat să persevereze în această direcție; erau convinși că ar fi formidabil să existe un sistem care să permită utilizatorilor să-și "mute" programele pe diverse mașini. Aici s-ar cuveni o paranteză; în acea vreme fabricanții de calculatoare aveau obiceiul de a oferi doar sisteme "proprietary", adică de așa natură încît, să fie incompatibile cu soft-ul altor firme; iar acest fapt îi obliga 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 mașini. Noul limbaj va fi terminat în 1970 și va purta pe "B". În anul următor UNIX este codificat în "B" și este instalat pe un DEC PDP-11. Tot-odată apar la BELL și primele aplicații practice ale noului sistem, printre care și un procesor de texte multi-user. Thompson și Ritchie profită de ocazie pentru a publica prima documentație de UNIX.

Așa a luat naștere esența filosofiei portabilității. Sistemul e prezentat ca multi-tasking, multi-user și evolutiv (adică permite crearea unor noi utilitare care să permită la rîndu-le apariția altor utilitare). Dar înainte de toate UNIX se vrea portabil.

Ritchie e de părere că limbajul "B" trebuie să fie îmbunătățit de așa manieră încât să permită cele mai diverse aplicații și să ofere atât un sistem de exploatare cât și compatibilitate. În această sarcină, Ritchie este ajutat de un programator ce se numea ... Kernigan. Cei doi vor defini 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 piața informatică. Dar, din păcate, în 1956 BELL TELEPHONE semnase un document prin care se angaja (față de guvernul american) să nu "iasă" pe piața 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 fildeș" ce-i fusese împus. "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 mașină. Pentru a răspunde acestei cereri spontane, BELL LABs distribuie gratuit câteva exemplare de UNIX V6 mai multor universități. Acestea vor primi de la BELL cite o bandă magnetică împreună cu dreptul de a face orice fel de modificări; dar BELL nu oferea nici garanție, nici asistență.

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 substanțial la maturizarea sistemului. Astfel au luat naștere un editor de texte (VI), un sistem de gestiune a rețelilor și a memoriei virtuale. Începând din 1979, la Berkley, cercetările vor continua pe mașini 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 Development*).

Anul 1977 va fi o piatră de hotar pentru UNIX: universitatea WOLLOGONG din Australia reușește în chip strălucit instalarea unui UNIX pe un calculator de tipul INTERDATA 8/32, mașina pe 32 de biți foarte diferită de PDP-11. Această reușită a contribuit enorm la răspândirea ideii că în sfârșit există un sistem de exploatare perfect portabil.

Pentru prima dată UNIX-ul a fost comercializat de firma ONYS SYSTEMS care, neputând achiziționa UNIX-ul, s-a mulțumit cu licența de utilizare a versiunii V7. Mașina ONYX a fost prezentată în 1980 la "National Computer Conference" și a avut un asemenea succes încât exemplul a fost urmat de alte firme. Pentru început ZYLOG propune o serie de "micro" de 16 biți (S 8 000) bazată pe un UNIX propriu, rebotezat ZEUS. Febra UNIX se întinde cuprinzând rând pe rând tot mai multe firme, PLEXUS, ALTOS, FORTUNE, NCR, TOWER, etc.

La începutul anilor '80, UNIX devine foarte popular în lumea cercetătorilor și inginerilor. În mai toate colțurile lumii apar asociații de "fans" ai sistemului. Unele dintre ele, cum ar fi *FreeSoft Foundation*, dezvoltă gratuit sau la prețuri modice diverse utilitare, neurmărind realizarea vreunui profit. "UNIX-istii" vor crea de asemenea o mesagerie internațională unde fiecare indică problemele pe care

nu le poate rezolva; de regulă, în scurt timp, se primesc soluții sugerate de alți utilizatori. Această bunăvoință, total lipsită de interese comerciale constituie un fapt rar în "moravurile" informaticii acelor vremuri.

În schimb, piața profesionistă cunoaște alt gen de evoluție. AT&T-ul vinde licențe de utilizare la niște prețuri destul de piperate. Doar școlile și universitățile continuă să beneficieze de tarife mai abordabile, fapt ce duce la o continuă creștere a nivelului de popularitate a UNIX-ului în rândul studenților.

În 1981 AT&T ia două decizii majore. În primul rând va publica SYSTEM III (prima versiune UNIX destinată în exclusivitate pieței comerciale) și se angajează să furnizeze asistență 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 rând AT&T a redus semnificativ prețul licențelor (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 așteptat căci o mașină 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 aparține societății SANTA CRUZ OPERATION (care însumează și azi 90% din vânzările din acest domeniu).

Cu toate acestea UNIX-ul este încă prea scump. Pentru a evita acest inconvenient unii editori vor lansa un "UNIX look-alikes", adică sisteme foarte asemănătoare cu UNIX. Cele mai cunoscute variante sînt *Unos*, *Coherent* și *UNETIX*. Aceste sisteme simulează modul de lucru al UNIX-ului, oferind același set de comenzi și putînd executa aceleași programe. Fiind create fără ajutorul AT&T, aceste UNIX look-alikes nu intră sub incidența 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 apariția primelor stații de lucru ale firmei APOLLO și SUN. În scurt timp, cea de-a doua firmă se va remarca prin asocierea versiunii UNIX BSD cu o interfață grafică de tip Macintosh, inovație ce va pulveriza recordurile de creștere stabilite de APPLE.

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

În mod firesc BELL încearcă să preia controlul UNIX-ului și a standardizării lui; în acest scop va lansa în ianuarie 1983 UNIX SYSTEM V. Această nouă versiune preia cea mai mare parte a îmbunătățirilor aduse de universitate Berkley, îmbunătățiri deja bine ancorate în mediile științifice. SYSTEM V va deveni primul punct de referință cu adevărat oficial. În plus în 1984 V va deveni primul punct de referință cu adevărat oficial. În plus în 1984 Ministerul Justiției din S.U.A. anulează decretul privind interzicerea accesului firmelor de comunicații pe piața informaticii. BELL se va împărți practic în mai multe societăți; dintre acestea AT&T va "moșteni" UNIX-ul, fapt ce-l va face să-și dezvăluie clar intenția de a controla atât norma, cât și comercializarea ei.

Anul 1985 va găsi UNIX-ul instalat pe mai mult de 100000 de mașini dintre cele mai variate (de la CRAY la IBM PC). Mai mulți constructori de calculatoare, dintre care și DEC și HEWLETT-PACKARD, confrunțați cu pericolul monopolizării unei piețe de asemenea dimensiuni, adresează IEEE-ului o propunere (ca și cum ar veni din partea unui grup de utilizatori) de adoptare a unui standard numit POSIX. În aceeași perioadă în Europa ia naștere X/OPEN GROUP (format din BULL, OLIVETTI, SIEMENS și PHILIPS) care, la rândul său adopta drept standard tot POSIX-ul. Realizând pericolul, AT&T își "apropie" universitatea Berkley în scopul obținerii unei convergențe a celor mai răspândite versiuni de UNIX.

În urma tratatelor va lua naștere SYSTEM V versiunea 3.0. În anul următor MICROSOFT și SCO își manifestă la rândul lor intenția unificării UNIX-ului; această hotărâre va duce la unificarea SYSTEM V 3.0, XENIX și BSD 4.2/4.3. În sfârșit, în 1987, AT&T anunță 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 și a difuzării informațiilor).

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

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

Pentru a contrabalansa OSF-ul, AT&T și SUN — împreună cu alți 50 de producători (printre care FUJITSU, OLIVETTI și 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 angajați și a definitivat prima versiune proprie: OSF.1 (publicată în noiembrie 1990). Sponsorii săi s-au angajat să-l finanțeze 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 și 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" aparținând universității 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 față există, prin urmare, doi pretendenți la standardizarea UNIX-ului. Ignorînd lupta pentru influență și 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 diferențe apar la nivelul interfeței grafice care, la OSF poartă numele de MOTIF (realizare DEC și HP), iar la UNIX INTERNATIONAL se numește OPEN LOOK (realizare SUN și XEROX). Dar chiar și aici există o oarecare compatibilitate, ambele avînd la bază standardul X-WINDOWS al celebrului M.I.T.

La nivel de utilizator final diferențele esențiale apar la DESKTOP (interfața generală). Cu alte cuvinte, o situație similară cu cea a GEM și WINDOWS din DOS: adică cine a folosit VENTURA într-una din aceste două "suprafețe" se va obișnui foarte repede și cu cealaltă.

O ultimă etapă constă 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 tendință constă 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 și AT&T își dau de asemenea tot interesul în obținerea 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 proveniența SCO, INTERACTIVE SYSTEMS, INTEL sau AT&T, UNIX-ul nu va mai avea nici un punct slab în comparație cu DOS-ul sau OS/2. În schimb va avea o mulțime de avantaje...

UNIX - interfețe grafice

O mașină 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 naștere la începutul anilor 1980 la M.I.T. și a fost conceput ca un "server" de ferestre destinat funcționării în rețea. Din 1984 M.I.T. scoate acest produs pe piață, permițînd astfel oricărui producător lipsit de resursele financiare necesare dezvoltării unei interfețe grafice proprii să și-l procure la comandă. Cum ideea este foarte inteligentă, s-a impus de la sine.

Cu toate acestea meritul popularizării interfețelor grafice sub UNIX aparține lui SUN. Totul a început în 1984 prin dezvoltarea produsului SUNView. Impactul acestui WORK-STATION, împletind confortul unui Macintosh cu execuția multi-tasking, a contribuit substanțial la evoluția 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 performanțe comparabile cu cele ale lui FINDER (la Mac) sau ale managerului de fișiere din WINDOWS). Cu alte cuvinte îi lipsește o definire exactă a modului de interacționare a utilizatorului cu diversele ferestre și icoane. Cu toate acestea, standardul M.I.T. oferă elemente de construcție adecvate pentru medii mai evaluate. În ianuarie 1987, DEC, APOLLO și HEWLETT PACKARD își anunță intenția folosirii versiunii X-WINDOWS în vigoare: X11. Imediat li s-au alăturat IBM, SUN și consorțiul de standardizare X/OPEN.

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

au apărut în 1988. Din soft-ul de birou ce folosește această interfață merită amintite LOTUS 1-2-3, Word Perfect și Wingz.

La rîndul său OSF-ul a fost consecvent principiului concursului de oferte, al cărui câștigător a fost de astă dată produsul MOTIF al firmelor DEC și HP. Ca o particularitate trebuie să menționăm comportarea absolut identică cu cea a lui WINDOWS și PRESENTATION MANAGER (chiar și la nivel de rol al tastelor funcționale), ceea ce asigură compatibilitatea cu cele mai răspândite interfețe grafice existente pe PC-uri.

Chiar dacă interfețele de programare ale lui MOTIF și OPEN LOOK diferă, ele pot coexista în aceeași rețea grație lui X-WINDOWS. Așadar o stație SUN aflată sub OPEN LOOK poate crea o fereastră pe o mașină IBM RISC 6 000, chiar dacă aceasta din urmă lucrează sub MOTIF. Stația SUN va acționa ca un "server" de ferestre și programul OPEN LOOK se va executa pe calculatorul Big Blue. E posibilă însă și situația inversă.

Dacă comparăm ecranul OPEN LOOK cu MOTIF, nu se remarcă decît diferențe "cosmetice". Practic deosebiri-le 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 același "desktop" (interfața generală — nivelul superior al interfeței, care va permite lansarea diverselor programe sau gestiunea fișierelor) la toate versiunile OPEN LOOK, pe cînd OSF a preferat să le lase mină liberă diverșilor producători (astfel pe stațiile RISC 6 000 se folosește AIX/WINDOWS, iar pe HP-uri vom găsi NEW WA-VE).

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

SCO însumează 70% din vânzările de UNIX pentru 80 386, în timp ce mai tinărul INTERACTIVE se mulțumește cu doar 25%. Ambele sisteme propuse par să aibă șanse mari să concureze OS/2 și WINDOWS, deși la prima vedere necesarul de resurse e de natură să îngrozească un "DOS-ist", căci au nevoie de cel puțin 6 Mo RAM și 100 Mo de hard disk. Aceasta "sete" de resurse e compensată în

schimb de unele avantaje incontestabile: interfața grafică de înaltă ținută, posibilități de conectare la rețea, deschidere către aplicații DOS, compatibilitate în mod text cu UNIX-ul clasic, iar în cazul SCO chiar și o bază de date relațională deloc "de lepădat" (INGRES). Dar "clou"-ul constă în posibilitatea de a executa SIMULTAN aplicații DOS și XENIX în ferestre diferite. Bineînțeles că cele două sisteme sînt compatibile (de fapt conțin — în mare — aceleași componente). Gama de aplicații e tot mai largă și aduce îmbunătățiri substanțiale în raport cu echivalentele de pe PC. Dintre best-seller-ele DOS regăsim WORD 5, WordPerfect și LOTUS 1-2-3.

Ultima oră

1 aprilie. Novell Inc, luînd în considerare performanțele 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 MacIntosh, DOS, OS/2, UNIX și servere Siris, Helios, etc... O nouă tinerețe pentru acestea, precum și o binevenită recunoaștere a calităților lor deosebite.

16 aprilie. Novelle Inc anunță achiziționarea unei părți a USL (Unix System Laboratories), succursală AT&T care a dezvoltat sistemul de operare UNIX. Investiția se referă la o parte din cele 20%-30% din proprietatea USL puse la dispoziție de către AT&T către 11 concerne din industria calculatoarelor.

Novell s-a aliniat și la tehnologia de rețea UNIX realizînd încă din 1983 produse de comunicație 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, permițînd utilizatorilor unei stații 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.

În ciuda tuturor celor afirmate mai sus, cea mai reușită interfață grafică nu se bazează pe această normă, ci pe Display PostScript al firmei ADOBE. E vorba de NextStep, care se găsește atît pe stațiile NEXT cit și pe IBM RISC 6 000 (opțional). În afara unei estetici ce te lasă cu gura căscată, NextStep se dovedește a fi mult mai avansată decît MOTIF și OPEN LOOK, mai ales prin faptul că dispune de o Interface Builder (un "tool" care în cîteva minute permite crearea unei interfețe cu orice aplicație, în condițiile unui lux rafinat). Importanța NextStep-ului reiese și din faptul că editorii se raliiază fără rezerve la acest mediu. Bazîndu-se pe aceste proprietăți LOTUS are un nou tablou ce permite descrierea formulelor în limbaj natural, iar Ashton-Tate realizează PowerStep (tot un tablou) ce permite adnotarea celulelor prin mesaje vocale.

În paralel QUARK e pe cale să-și adapteze PAO Xpress în sensul utilizării la maxim a capacităților pe multi-tasking real, iar ADOBE își adaptează produsul de desenat ILLUSTRATOR. În plus societatea IMAGINE pune la punct aplicații multi-media tot pe NextStep. O asemenea de poziție a marilor producători de soft riscă să aducă NextStep în situația MacIntosh-ului anilor 80: incompatibil cu PC-ul, dar capabil să se impună prin forța

atuurilor sale. Anunțul recent al realizării versiunii color a acestui mediu e susceptibil să-i aducă pînă și adeziunile celor mai reticenți. De altfel, IBM a decis să-și doteze mașinile RISV 6 000 cu versiunea color a NextStep.

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

SOSESC PRODUSELE CASE(II)

Cureleț-Bălan Gheorghe

Produsele program CASE pot fi clasificate după mai multe criterii; astfel, dacă ne referim la etapele din ciclul de viață cărora le sînt destinate, putem să le clasificăm în "front-end tools" — sau "upper CASE" (vizează etapele de început ale ciclului de viață) și "back-end tools" — sau "lower CASE" (destinate etapelor de sfîrșit ale ciclului de viață). 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 aplicațiilor de gestiune economică și în produse CASE destinate domeniului tehnico-științific. Desigur am putea să le clasificăm și după tipul de calculatoare pe care rulează: mari, medii, micro.

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

Din clasa produselor "front-end" menționăm în primul rînd pe cele suport în analiză/proiectare. Acestea implementează tehnicile de analiză/proiectare structurată fiind de fapt editoare grafice care suportă diferite metodologii de analiză/proiectare. Sistemul soft în faza de analiză este modelat printr-o mulțime de diagrame care reprezintă funcționalitățile acestuia, fluxul de date, interfețele etc. Sistemele CASE evolute permit producerea unei game largi de diagrame prin utilizarea unei multitudini de simboluri tipizate și suprapunerea facilităților de diagramare cu capacitățile de procesare text ceea ce permite adnotarea diagramelor sau introducerea unor diagrame în text. De asemenea se fac verificări privind sintaxa componentelor grafice folosite. Deși inițial produsele din aceste categorii se limitau doar la facilitățile de diagramare, în scurt timp ele au încorporat posibilități de verificare a consistenței și completitudinii. Aceasta se realizează de regulă prin dicționarele de date în care se înregistrează informații despre tipul datelor, cerințele de folosire etc. Astfel se previne folosirea inconsis-tentă a anumitor elemente de proiectare putîndu-se obține în același timp informații complete privind folosirea acestora — de exemplu un proiectant poate determina rapid toate componentele unei structuri de date, sau toate procesele ce folosesc o anumită dată. În plus, existența dicționarului permite generarea unei game variate de rapoarte privind proiectul software.

Sistemele CASE evolute din această categorie oferă facilități multiutilizator destinate proiectării în echipă pe suportul unei rețele locale de PC-uri a unui minicalculator sau calculator mare. Dintre produsele reprezentative menționăm: *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, *Excelsator* 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 și a identifica erorile de proiectare prin crearea unui model al sistemului aflat în curs de elaborare. *Prototipizarea* rupe tradiția de a considera desfășurarea secvențială a etapelor ciclului de viață adăugînd ca o necesitate iterația 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. Deși 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, întreținerea și refolosirea software-ului. Pentru a exemplifica această interdependență vom prezenta facilitățile cîtorva produse de referință. Firma Cortex Corp. oferă produsul *Corvision* (destinat mediilor VAX/VMS) care suportă fazele de proiectare, codificare și întreținere din ciclul de viață. Interfața grafică produce o specificație de proiectare care servește ca intrare în generatorul automat de cod în limbajul de generația 4, *Builder* (conceput de firmă). Se apreciază că în felul acesta se generează automat 85-95 % din codul aplicației, restul fiind completat de utilizatorul produsului.

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

Pe lîngă creșterea productivității de elaborare, generatoarele de cod oferă avantajul simplificării întreținerii software-ului, asigurîndu-i în același timp o fiabilitate crescută.

Considerăm că generatoarele de cod vor prolifera în viitor avînd în vedere atît creșterea continuă a complexității produselor program oferite elaboratorilor de software cît și tendința de apropiere a acestora de utilizatorul neinițiat. Pentru a exemplifica acestea vom prezenta facilitățile altor generatoare de cod. CASE W al firmei Caseworks, SUA este destinat elaborării aplicațiilor dezvoltate sub interfața MS Windows. Necesitatea elaborării produsului a fost justificată de volumul mare de muncă impus de elaborarea sub interfața MS Windows a unor aplicații relativ simple.

CASE W permite programatorului să-și definească elementele specifice ale interfețelor 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 aplicației vor fi plasate de programator.

De fapt CASE W generează "șablonul" (cadru) de cod C care reprezintă dependența aplicației de sistemul de gestiune al interfețelor MS Windows. Facem observația ca aceeași firmă a elaborat un produs similar destinat elaborării aplicațiilor sub interfața *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 observația ca anumite aspecte ale codificării în d'BASE au un caracter rutinier și sînt consumatoare de timp, elaboratorii produsului degrează programatorul de astfel de activități prin generarea automată a codului corespunzător. În acest fel programatorul se poate concentra asupra aspectelor creative ale dezvoltării aplicațiilor, cum ar fi proiectarea ecranelor, a structurii bazei de date etc.

Produsul constă din cinci componente: editor de ecran, dicționar de date, șabloane program, limbaj pentru șabloane, generator de cod. Activitățile rutiniere sînt înregistrate în sistem sub forma unor șabloane ("template") care apoi sînt selectate de utilizator. Programatorul, cu ajutorul editorului de ecran, își definește forma ecranului din programul său (în paralel cu aceasta se creează dicționarul de date), apoi selectează șablonul potrivit prelucrării pe care intenționează să o facă. Sistemul *UI* combină imaginea ecran definită împreună cu șablonul 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 crește productivitatea programatorilor în d'BASE.

Menționăm deasemenea că în prestigioasa revistă *Info-world* din 16 oct. '89 a fost semnalat faptul că firma Ashton-Tate a anunțat realizarea unui generator de aplicații în d'BASE numit *Crystal*.

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

Tot în categoria generatoarelor de cod pot fi incluse translatoarele interlimbaje și produsele de reinginerie. Acestea din urmă se referă la dezvoltarea sistemelor software foarte complexe a căror întreținere devine costisitoare. Redezvoltarea înseamnă desprinderea elementelor de proiectare pornind de la sursă și recodificarea într-o manieră structurată în vederea facilitării întreținerii. Produsele din această categorie se adresează în general utilizatorilor limbajului COBOL; dintre cele mai reprezentative amintim: *RETROFIT* al firmei CATALYST, *RE-CORDER* 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 și dialecte ale aceluiași limbaj (firmele Computer Associates și Sector 7 pentru limbajul COBOL, firma SOFT-TOOL pentru FORTRAN), precum și 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 assembler-limbaj înalt, cum este cazul translatorului Xtran al firmei Pennington Systems Inc., destinat translării assembler VAX-C.

Instrumentele de gestiune a modificărilor și a configurației identifică toate componentele care constituie aplicația și relațiile între ele, asigurînd sincronizarea între componente și 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.

Documentația ocupă un rol esențial în procesul software, intervenind practic în toate cele șase etape ale ciclului de viață. În cadrul sistemelor software complexe volumul ei e atît de mare încît gestiunea manuală a acesteia este practic imposibilă. Se citează astfel ca documentația de proiectare a sistemului software destinat navei spațiale americane a constat în mai mult de 200 000 de pagini, și ca în cadrul firmei Boeing se produce anual un volum de două miliarde de pagini, în cel puțin 56 tipuri diferite, de documentație pentru software. Un sistem de gestiune automată a documentației trebuie în primul rînd să o păstreze sub forma unei baze de date care să permită controlul modificărilor aduse diferitelor tipuri de documente și care în același timp să verifice consistența acestora, în sensul că modificările aduse unui document să fie reflectate și în documentele legate de acesta dar să nu altereze relațiile și documentele nemodificate. Pe lîngă faptul că un astfel de sistem ar putea sta la baza unor instrumente de asigurare și control ale calității procesului software (verificarea conformității cu anumite standarde — de analiză, proiectare, codificare, etc.) rolul său esențial intervine în etapa de întreținere cînd modificările propuse afectează de fapt tot volumul de documentație 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 degreze elaboratorul de software de unele aspecte rutinare specifice activității de elaborare. Dezvoltate inițial în scopul automatizării unei etape din ciclul de viață al produsului program produsele CASE au evoluat în direcția dezideratului de integrare și a altor etape. Produsele "front-end" evoluează spre înglobarea facilităților celor din categoria "back-end" și invers. Aceasta face ca în prezent să asistăm la o concurență acerbă în impunerea unui standard în domeniu. Despre toate acestea însă într-un număr viitor. ■

Începînd de astăzi inaugurăm o nouă rubrică care își propune să treacă în revistă majoritatea pachetelor de programe disponibile pe piață.

Astăzi:

UTILITARE PENTRU GESTIUNEA MEMORIEI

Nume: 386 Max Professional

Producător: Qualitas

Preț: 300 \$

Descriere: Optimizează performanțele 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 convențională. Transferă conținutul memoriei ROM (lentă ca timp de răspuns) în memoria RAM. Se obține o accelerare pentru funcțiile BIOS de exemplu. Necesită 80386.

Nume: Above Disc

Producător: AB Soft

Preț: 250 \$

Descriere: Produsul asigură crearea unei memorii expandate conformă cu specificația 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 specificației LIM 4.0. De notat că în această situație frecvența la care se lucrează este de 25 Mhz (corespunzător plăcii de bază) și nu de 8 Mhz coresponzător magistralei de extensie în cazul plăcilor de memorie conectate separat.

Pentru a se exemplifica eficiența produsului se recurge la Lotus 123 Versiunea 2.01; cu produsul instalat se pot încărca în memorie 4 816 linii din foaia de calcul față de 448 linii în cazul absenței 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

Preț: 200\$

Descriere: Permite încărcarea în memoria înaltă a programelor rezidente, a driverelor DOS, a driverelor de rețea. Comportă o funcție de optimizare a memoriei. Necesită plăci de memorie compatibile cu specificația EMS 4.0.

Nume: PC - Kwik prower

Producător: Multisoft

Preț: 320 \$

Descriere: Este un produs care optimizează lucrul cu hard-discul. Utilizează algoritmi de optimizare a scrierii pe disc, pe care îl gestionează rînd în "background". Implementează de asemenea un "spooler" de imprimantă, un disc

RAM, un accelerator de tastatură și unul de ecran. Ocupă doar 10 k RAM și funcționează atît în memoria convențională cît și în cea expandată. Necesită 640 K RAM și rulează pe XT/AT/PS/2.

Nume: QRAM sau QEMM 50/60

Producător: QuarterDeck

Preț: 200 \$

Descriere: În funcție de configurația de care dispuneți, produsul QRAM va găsi orice fel de memorie pe care DOS nu o recunoaște (EMS, EEMS, XMS) și va instala acolo toate programele rezidente precum și driverele existente în memorie eliberînd zona de memorie convențională. Pentru un PC echipat cu ecran VGA sau EGA, dar pe care nu se rulează aplicații 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 și 60.

Nume: HEAD ROOM

Producător: HELIX

Preț: 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 execuție. Gestionează pînă la 32 Mo memorie accesibilă DOS. Versiunea rețea permite instalarea soft-ului de rețea în memoria expandată (EMS 4.0). Produsul funcționează pe mașinile 8088-80486.

Nume: QEMM 386

producător: QUATERDECK

Preț: 195 \$

Descriere: Asigură conversia unei părți a memoriei extinse în memorie expandată. Optimizează funcționarea calculatorului, detectînd viteza de acces la memorie și 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 mașini virtuale 8086 precum și utilizarea produsului DESKVIEW 2.2 pentru implementarea lucrului în multi-tasking. 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 înființată în martie 1990 de un grup numeros de specialiști de vîrf - în domeniul tehnicii de calcul și comunicație de date, pentru a oferi tuturor specialiștilor în acest domeniu o alternativă la numeroasele oferte primite din străinătate. Societatea are în prezent 16 filiale în București și în întreaga țară și întreține legături cu un mare număr de parteneri străini, dintre care cel mai important este **IBM** (prin **SOCODIAG ASIST - IBM Franța**) prim agent național în România. Prin expertizii proprii **ADISAN** prospectează în permanență piața mondială de microcalculatoare depistînd cea mai avantajoasă ofertă intermediînd contracte între producătorii de microcalculatoare și beneficiari, negociînd achiziționarea cu plata în lei sau valută în funcție de posibilități.

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

Departamentul Comunicații vă oferă soluții de informatizare și comunicații globale legate de întreprinderea dumneavoastră.

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

Departamentul Comercial vă pune la dispoziție o ofertă completă în lei și valută de echipamente de calcul și telecomunicații. Începînd cu cele mai pretențioase echipamente din punct de vedere al dinamicii, prețului și calității.

Societatea ADISAN vă oferă, prin **Editura ADISAN**, informații de specialitate în revista "**PC-MAGAZIN**", culegere de text, machetare pe calculator și imprimantă **LASER JET**, pentru orice tip de publicații-ziare, reviste, cărți, etc.

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

- SERIOZITATE - COMPETENȚĂ - 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				

STOP

ATENȚIE LA VIRUȘI!

PROGRAM ANTIVIRUS: "CHKVIR.BAT"

Acest utilitar este o procedură de tip BATCH, ce se poate folosi de către utilizatorii de IBM-pc și compatibile pentru a testa prezența sau absența unui virus în sistemul de calcul. De fapt, programul "CHKVIR.BAT" verifică integritatea dimensiunii și a datei fișierului de comenzi DOS, COMMAND.COM, asupra căruia operează cele mai mulți din virușii cunoscuți. 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 fișierul propriu AUTOEXEC.BAT, fiind executat la începutul fiecărei sesiuni de lucru. "Cheia" acestui utilitar se află în liniile 2 și 9, în care se creează fișierul temporar "fisier1", în care, prin intermediul comenzii externe FIND, care trebuie să existe pe disc, se încarcă (prin directiva "fisier1"), mărimea și respectiv data standard a lui COMMAND.COM.

Dacă programul descoperă diferențe între mărimea fișierului COMMAND.COM existent pe disc, și mărimea standard a acelui fișier descrisă în linia 2, de ex. "25276", atunci fișierul "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" eșuează, "fisier1" fiind vid. Artificiul "NUL" a fost adăugat pentru a inhiba

mesajul de confirmare a copierii. Linia 4 verifică prezența pe disc a lui "fisier2". Dacă acesta există, atunci nici un virus nu a afectat mărimea originală a lui COMMAND.COM, și se trece controlul subrutinei numită "secventa", care după același principiu tratează situația datei lui COMMAND.COM. Dacă "fisier2" nu a fost copiat pe disc, la linia 5 se afișează un mesaj care atenționează asupra virusului ce a modificat dimensiunea fișierului de comenzi DOS. La linia 14 începe rutina "mesaj", care în cazul în care nici data și nici dimensiunea lui COMMAND.COM nu au fost modificate, ne asigură că nici un virus nu a modificat sistemul.

Linia 17, șterge fișierele temporare "fisier1" și "fisier2". Trebuie făcută mențiunea că acest program simplu nu este un panaceu universal pentru toate tipurile de viruși. El nu testează, de pildă memoria video, sau sistemul de întreruperi.

Pentru aceste teste se folosesc "ANTI-VIRUȘI" mult mai complicați, dar și mult mai scumpi. Totuși, CHKVIR.BAT poate fi interesant, fie și numai prin simplitatea și concepția lui.

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

:secventa
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

Cîtiva viruși mai cunoscuți

Calculatoarele compatibile IBM se confruntă cu o adevărată "epidemie" de viruși. Gama extrem de largă a acestora are un numitor comun: toate se instalează din programe comerciale de o proveniență incertă: rețele, copii ilegale, etc. Nu toți virușii au același grad de pericolozitate. Totuși, pînă și cei mai aparent "nevinovați" viruși, pot produce multe neplăceri. De aceea, ei trebuie identificați și pe cît posibil distruși cu programe "VIRUS KILLER", pentru a evita contaminări în lanț.

Iată acum tipologia celor mai cunoscuți viruși de tip IBM-PC:

1. BRAIN(sau PAKISTAN sau LAHORE)

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

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

Efect: supărător.

2. FU MANCHU

Simptom: fișierele COM și EXE cresc artificial cu cca. 2088 octeți. La startarea sistemului, ecranul se șterge și apare mesajul: "The world will hear from me again!"

Acțiune: reacționează la anumite cuvinte, cum ar fi: "Fu Manchu", "Thatcher", "Reagan", "Waldheim", "Botha", dar și la alt gen de cuvinte.

Efect: supărător.

3. DATACRIME

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

Acțiune: formează nivelul de jos al hard-discului între cilindrii 0 și 8.

Efect: foarte supărător.

4. CITY AVENGER

Simptom: fișierele COM sau .EXE cresc cu aproape 1800 octeți.

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 roșu pe monitoarele color: "DENZUK". De asemenea, modifică eticheta de volum în: "Y.C.E.R.P".

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

Efect: iritant.

6. CASCADE(sau BLACKJACK)

Simptom: fișierele .COM cresc cu 1704 sau 1701 octeți. Cînd se inițializează 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.

Acțiune: niciuna.

Efect: lipsit de pericol major.

7. SATURDAY 14th

Simptom: fișierele .COM și .EXE cresc cu un număr de octeți cuprins între 669 și 684.

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

Efect: supărător.

8. VIENNA

Simptom: fișierele .COM cresc cu 648 de octeți.

Acțiune: infestază programul în curs, ca și fișierul COMMAND.COM

Efect: supărător.

STOP

9. 405

Simptom: fișierele mai mici de 405 octeți cresc pînă la această valoare.

Acțiune: programele infestate nu mai funcționează.

Efect: foarte periculos.

este anormal. Pe floppy se găsesc 3 ko de sectoare afectate. Dacă se lasă în funcțiune computerul pentru 48 de ore și apoi se accesează hard-diskul, ecranul se șterge și 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".

Acțiune: modifică întregul hard-disk, făcîndu-l inaccesibil.

Efect: distrugător.

10. OGRE

Simptom: se pierde 8 ko de pe disc. Sectorul "boot"

Rubrică realizată de Dobrilă Mirel ■

NOU

Se asigură legarea hard și soft a imprimantelor Band Print (tip RCD) la orice tip de PC - IBM compatibil sau la o rețea (tip Novell) de PC - IBM compatibile.

Pentru PC - IBM 286/386 vă recomandăm interfața multiuser cu soft-ul aferent pentru conectarea de terminale obișnuite pentru introducerea de date.

Asigurăm legarea în rețele tip Novell a oricărui tip de PC - IBM compatibil, precum și legătura CORAL - PC - IBM compatibil.

Echipamentele furnizate de firma ADISAN au o calitate deosebită. De aceea, manopera în perioada post-garanție asigurată de ADISAN este GRATUITĂ.

Nedelcu Adrian, Sîngeorz-Băi, 17 ani

În primul rînd, vă mulțumim pentru frumoasele aprecieri la adresa revistei. Din păcate, în privința documentației de care aveți nevoie nu vă putem ajuta. Dacă aveți probleme de vedere și totuși ați lucrat pînă acum atît de mult cu calculatorul înseamnă că puteți urma și cursurile uneia dintre facultățile de profil, fie în cadrul unui Institut Politehnic în care există o secție sau facultate de Calculatoare, fie în cadrul unei Universități în cadrul căreia există o secție de Informatică.

ing. Dumitrescu Mihai, București.

Dacă doriți să colaborați cu articole la revista noastră, vă așteptăm cu plăcere pentru o discuție la redacție.

Mercea Mario Matei, Timișoara.

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

Radu Constantinescu, Brăila.

Ne bucură mult faptul că apreciați revista noastră și vă asigurăm că vom căuta să publicăm numai articole utile pentru cititorii interesați 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 mulțumim pentru comentariile competente asupra articolelor apărute în revistă. O să încercăm să publicăm și articole pe teme propuse de dumneavoastră.

Stoean Gilberto, București, 16 ani.

După cum demonstrează și titlul revistei, ne preocupăm în special de calculatoare din familia compatibilelor IBM-PC. Sperăm ca și în continuare să găsiți în revista noastră lucruri care să vă intereseze.

Vasilescu Grigore, Brăila.

Dacă ați început să lucrați pe un calculator de tip PC-IBM probabil că ați descoperit deja programe ca dBASE III+ (sau IV), ORACLE sau altele care permit exploatarea bazelor de date.

Răzvan Dumitru, București.

Performanțele calculatoarelor JET, CIP, MICROTIM sînt aproximativ aceleași cu ale calculatorului HC-85. CIP are 64 Ko RAM, dar numai 4Ko. ROM. COBRA prezintă în plus și compatibilitate CP/M (în configurație completă), ca și HC 88. În privința protejării programelor preferăm codul mașină și... originalitatea (v. almanah TEHNIIUM '90).

Silviu Dumitrescu, Focșani.

Limbajul C nu are legătură cu limbajul COBOL. În afară de BASIC sub interpretor și asamblare pentru cod mașină, pe SPECTRUM se poate lucra, cu ajutorul compilatoarelor, în PASCAL, C, PROLOG, FORTH, LOGO.

Alexandru Kopatz, Sibiu.

Mulțumim pentru idei, stilul nu contează. Începînd cu numărul următor al revistei, vom aborda, în legătură cu codul mașină din ROM și problema variabilelor de sistem. Ne vom gîndi la un concurs pe teme de cod mașină, cu un top al rezolvitorilor.

Horațiu Coifan, Suceava.

Din numărul viitor, vom prezenta, pe rînd, diversele rutine din ROM. Considerăm că GENS 3M21 este mai eficient decît ZEUS, care, deși posedă un editor mai puternic, are unele greșeli.

Valer Bocan, Deva.

Împreună cu caseta îți trimitem și o scrisoare care sperăm să ajungă la tine mai repede decît a ajuns scrisoarea ta la noi.

Aurel Enache (București), Matei Mario Mercea (Timișoara), Bogdan Horotan București, Alexandru Kopatz (Sibiu), Horațiu Coifan (Suceava).

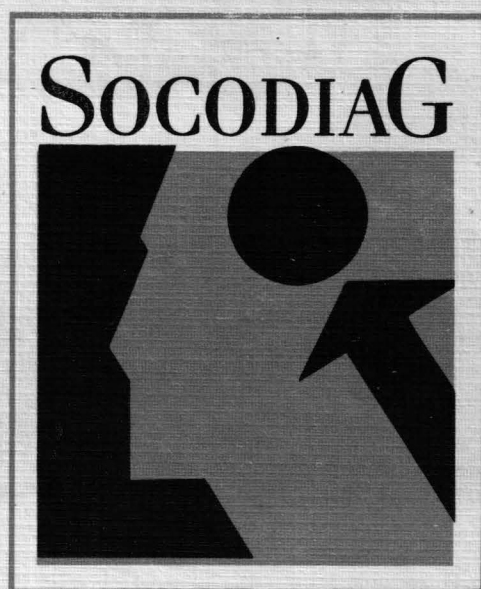
Deocamdată, din cauza problemelor legate de tipărire, nu putem onora anunțul din numărul 3 al revistei.

În legătură cu articolul din numărul 4, la pagina 65, în loc de "nu trebuie citată instrucțiunea CALL #02BF", se va citi "nu trebuie uitată instr...".

Pentru consultări în legătură cu utilitățile pe SPECTRUM, vă puteți adresa direct la BOGDAN IORDANESCU, tel.: 53.34.67 sau BOGDAN GEORGESCU, tel.: 27.78.55.

*Pentru ceilalți care ne-au scris,
răspunsurile în numerele viitoare!*

prin
SOCODIAG — ASYST
INFORMATICA
DISPONIBILA



Relații: IONEL RUSE tel. 43.11.88
ALEXANDRU BABIN tel. 15.81.65;14.54.55



**INNOVATOR NOTEBOOK
MADE IN HOLLAND**