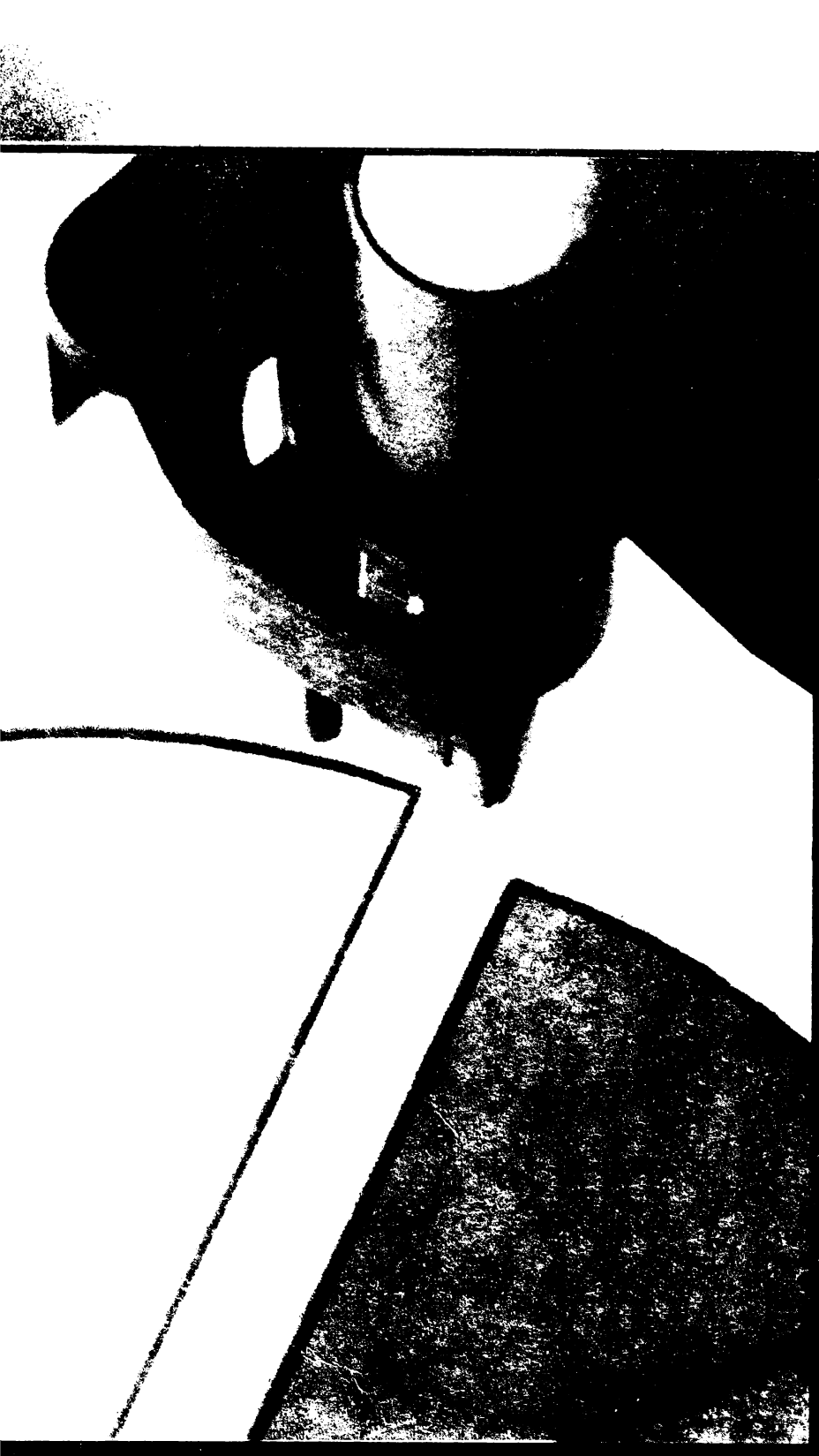
PC magazin

ADISAN

REVISTĂ INDEPENDENTĂ
DE INFORMATICĂ
EDITATĂ DE FIRMA ADISAN

- CULTURA INFORMATICĂ
  - Reţele de calculatoare (I)
  - Reţele neuronale
- INFORMATICA FĂRĂ PROFESOR
  - Limbajul C (V)
  - Iniţiere în programare - Limbajul PASCAL
- WINDOWS
  - Interfaţarea Turbo PASCAL - MASM
  - PC-uri şi protecţie
- HOME COMPUTER
  - Limbajul Z80 (II)
  - Despre ecrane şi memorarea lor
- PERSPECTIVE



Cuprins

PC-MAGAZIN
ANUL II NR.1/1991

FONDATORI
Adrian Negru
Alexandru Babin

DIRECTOR
Alexandru Babin

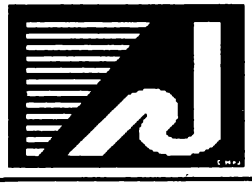
REDACTOR COORDONATOR
Mircea Ciobotaru

COLABORATORI
Prof.univ.dr. Cornel Popa
conf.dr.ing. Irina Athanasiu
conf.dr.ing. Eugenia Kalisz
Ing. Adrian Mircea
Ing. Ionuţ Stoica
Ing. Florin Dobrian
Ing. Florin Sultan
Ing. Cureleţ-Bălan Gheorghe
Ing. Călin Şandru
Ion Paraschiv
dr. Tiberiu Spircu
Ion Muşlea
Bogdan Georgescu
Bogdan Iordănescu
Cornel Negru
Floricel Popescu
Anca Dumitru
Şerban Silvinschi

GRAFICĂ
Octavian Penda
Lucian Mişcă
Luminiţa Ciupitu

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

Tiparul a fost executat la
Tipografia UNIVERSUL



ADISAN
11,Arh.Ioh Mincu str.
Bucharest-ROMANIA
Phone: 17.74.30
COMERCIAL DEP.

Adisan SRL este o societate înfiinţată în martie 1990 de un grup numeros de specialişti de vîrf în domeniul tehnicii de calcul şi comunicaţie de date, pentru a oferi tuturor specialiştilor în acest domeniu o alternativă la numeroasele oferte primite din străinătate. Societatea are în prezent 16 filiale în Bucureşti şi în întreaga ţară şi întreţine legături cu un mare număr de parteneri străini dintre care cel mai important este **IBM** (prin **SOCODIAG ASYST - IBM**, Franţa, prim agent naţional în România).

Prin experţii proprii **ADISAN** prospectează în permanenţă piaţa mondială de microcalculatoare depistînd cea mai avantajoasă ofertă intermediind contracte între producătorii de microcalculatoare şi beneficiari, negociind achiziţionarea cu plata în lei sau valută în funcţie de posibilităţi.

Departamentul SERVICE vă oferă service cu cei mai buni specialişti ai tehnicii de calculatoare, la toate gamele de echipamente produse în ţară: **Felix C256/512/1024, Felix 5000, I-100/102F/106, Coral, Felix PC, Cub Z, Junior XT** etc., precum şi la toate echipamentele **PC-IBM** compatibile provenite din import pe care le achiziţionaţi prin noi. Centre mari de calcul din Bucureşti şi din ţară sînt în service la noi (ex: CINOR, Electromagnetica, C.T.C. Cluj, C.T.C. Sf.Gheorghe, C.T.C. Focşani, I.U.G.- Craiova, I.U.C.-Ploieşti etc.

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

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

Departamentul Comercial vă pune la dispoziţie o ofertă completă în lei şi valută de echipamente de calcul şi telecomunicaţii (vezi oferta de la sfîrşitul acestui număr al revistei), începînd cu cele mai pretenţioase echipamente din punct de vedere al dinamicii, preţului şi calităţii. Se asigură garanţie 1 an şi service postgaranţie prin contract.

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

SERIOZITATE—COMPETENŢĂ—PROMPTITUDINE

REŢELE DE CALCULATOARE (I)

Ing. Ion Stoica
Ing. Florin Dobrian
Ing. Călin Şandru

Unul dintre cele mai ample fenomene observate în ultimul deceniu în domeniul tehnicii de calcul îl constituie dezvoltarea rapidă a reţelelor de calculatoare. Această spectaculoasă evoluţie se explică prin importanţa pe care o are schimbul infomaţional şi avantajele pe care reţelele le oferă în această direcţie.

Informaţia reprezintă, după cum probabil se cunoaşte, o înlăturare a nedeterminării, vehicularea ei rapidă şi sigură apărînd din ce în ce mai mult ca o necesitate a acestui sfîrşit de secol. Calculatoarele pot realiza transferuri informaţionale cu performanţe ce asigură prelucrări eficiente. Ele pot fi interconectate rezultînd reţele de calculatoare. Aceste reţele au o serie întreagă de caracteristici, putînd fi însă distinse trei mari categorii şi anume:

- **reţele locale (LAN)** - sînt funcţionale pe o distanţă de cîteva sute de metri şi sînt amplasate în instituţii sau întreprinderi pentru a implementa suportul informatic al sistemului informaţional al respectivei unităţi economice;
- **reţele metropolitane (MAN)** - sînt amplasate la scara unui oraş şi interconectează principalele obiective economice ale oraşului respectiv;
- **reţele pe arii întinse (WAN)** - interconectează centrele informatice la nivel continental sau planetar.

Evoluţia cea mai rapidă a avut loc în cadrul reţelelor locale. Ele tind să înlocuiască modelul calculatorului central. Ideea centrului de calcul cu un calculator puternic (mainframe) la care operatorii aduc programele pentru a fi rulate a devenit învechită. Acest model are o fiabilitate scăzută, fapt datorat existenţei unui singur calculator care poate introduce timpi morţi în procesul de rulare în cazul apariţiei unei defecţiuni. Modelul este înlocuit treptat cu un altul în care mai multe calculatoare cu performanţe mai scăzute sînt interconectate prin cabluri electrice, fibre optice sau unde radio. Acestea pot fi amplasate în birourile utilizatorilor sau în casele acestora. Calculatoarele nu se află într-o relaţie de tip master-slave ci sînt echivalente şi autonome în funcţionare. Din punct de vedere al fiabilităţii se poate observa că aceasta creşte în cazul modelului reţea deoarece la defectarea unuia sau mai multor calculatoare rămîn totuşi o parte în stare funcţională iar reţeaua în ansamblul ei poate continua rularea diverselor aplicaţii.

Şi din punct de vedere al raportului cost/performanţă o reţea locală este avantajoasă faţă de un calculator de tip mainframe. Acesta din urmă dispune de o putere de calcul de circa 10-20 de ori mai mare decît aceea a unui calculator personal utilizat de obicei într-o reţea. În acelaşi timp preţul este de 1000 de ori mai mare. Rezultă deci clar un raport cost/performanţă favorabil reţelelor de calculatoare. Desigur, nu trebuie contestat rolul unui mainframe într-o aplicaţie specifică în care caracteristicile acestuia sînt puse în valoare.

În cazul reţelelor metropolitane sau a celor pe arii extinse există avantajul de a putea accesa baze de date situate la mare distanţă. Spre exemplu, o companie aeriană foloseşte o astfel de reţea pentru operaţia de rezervare a locurilor. Baza de date este în general distribuită în mai multe aeroporturi. Astfel, pentru un avion care zboară pe ruta Bucureşti, Paris, New-York, rezervarea de locuri la Paris se poate face pe baza locurilor rămase disponibile la plecarea din Bucureşti accesînd din Paris baza de date referitoare la ocuparea locurilor din Bucureşti.

După această prezentare generală a reţelelor de calculatoare vom trece acum la introducerea unor concepte concrete care le definesc şi anume conceptele de structură şi arhitectură.

Pentru a fixa ideile, să considerăm un exemplu care, prin analogie, să facă înţelese o mare parte din conceptele utilizate în descrierea structurii şi arhitecturii reţelelor. Să presupunem că într-o ţară de dimensiuni mici, s-o numim Liliput, se pune problema urmăririi spaţiului aerian. Pentru aceasta sînt necesare trei staţii radar în punctele A, B, C ca în *figura 1.* Fiecare staţie are un punct (centru) de comandă unde există o hartă a întregii ţări pe care se notează poziţia avioanelor observate. Apare astfel necesitatea unui sistem de comunicaţie între cele trei centre de comandă. Poziţia avioanelor din sectorul propriu de observare va fi comunicată de staţia locală în timp ce a avioanelor din afara spaţiului local va fi

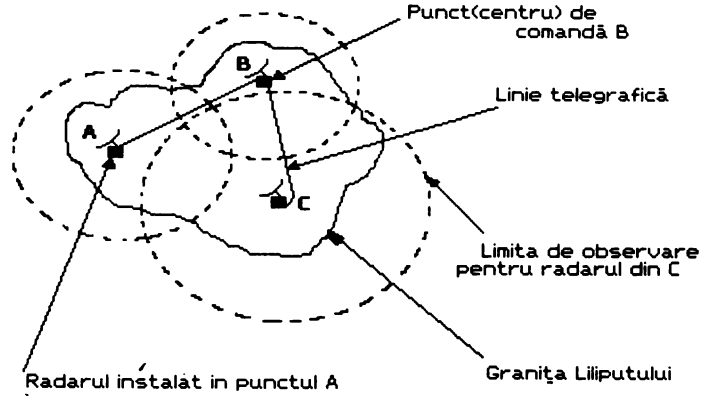


Figura 1.

obţinută de la celelalte puncte de comandă. Presupunem că suportul fizic pentru comunicaţie este o linie telegrafică care conectează punctele de comandă A, B şi C. Informaţiile care vor fi transmise constau în poziţia şi eventual traiectoria şi viteza avioanelor care sînt observate pe radarul local. Pentru a asigura circulaţia corectă a fluxului informaţional, în fiecare punct de comandă există o structură organizată formată din mai mulţi operatori, fiecare cu atribuţii specifice. Astfel, un operator urmăreşte ecranul radarului, un operator notează poziţia

pe hartă a avionului, un operator interpretează poziția de pe hartă și formează mesajul ce trebuie transmis și în sfîrșit un operator (telegrafistul) transmite efectiv mesajul.

Acest exemplu, evident, prezintă simplificări majore față de sistemele reale de supraveghere a spațiului aerian. Ceea ce ne interesează pe noi este folosirea acestuia în dorința de a fi cît mai clari în explicarea conceptelor de bază ale rețelelor de calculatoare.

STRUCTURA REȚELELOR

Atunci cînd vorbim de structura unei rețele de calculatoare ne referim la părțile componente ale acesteia și la organizarea și interacțiunile dintre acestea. Astfel, într-o rețea există o colecție de mașini pe care se rulează aplicațiile utilizatorilor. Aceste mașini poartă numele de calculatoare gazdă (*host*). Ele sînt interconectate printr-o subrețea de comunicație, a cărei sarcină este de a transporta mesajele de la un calculator la altul.

În general subrețeaua este alcătuită din două componente majore: elementele de comutare și liniile de transmisie. Primele sînt calculatoare specializate și poartă numele de procesoare de mesaje (*IMP - Interface Message Processors*). Liniile de comunicație sînt denumite și canale. În *figura 2* este prezentată structura generală a unei rețele de calculatoare. În cazul exemplului nostru linia de transmisie este reprezentată de linia telegrafică, iar un IMP poate fi asociat cu telegrafistul (chiar dacă acum destinația unui IMP nu este suficient de clară ea va fi lămurită ulterior).

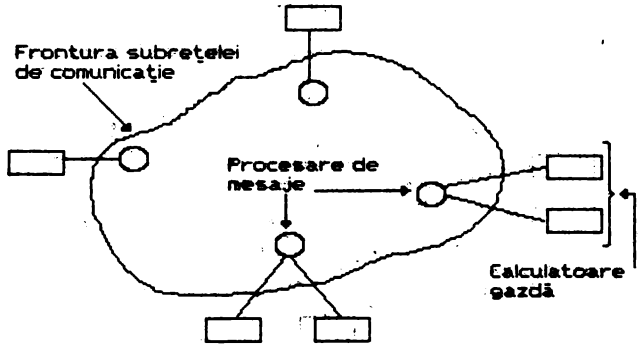


Figura 2.

Există două modele generale pentru realizarea subrețelei de comunicație: canale punct la punct și canale broadcast. În primul caz rețeaua este formată dintr-o serie de linii ce conectează cîte o pereche de IMP-uri. Dacă două IMP-uri nu sînt conectate direct ele pot comunica indirect prin intermediul altor IMP-uri. Astfel, să modificăm exemplul nostru și să presupunem că există două linii telegrafice distincte care leagă centrele de comandă A, B, respectiv B, C (între A și C nu există o legatură fizică). Pentru a transmite un mesaj de la A la C operatorul (telegrafistul) din B trebuie să preia mesajul de pe linia telegrafică AB și să-l transmită pe linia BC.

Un aspect important al modelului punct la punct îl constituie topologia rețelei. *Figura 3* descrie o serie de topologii posibile.

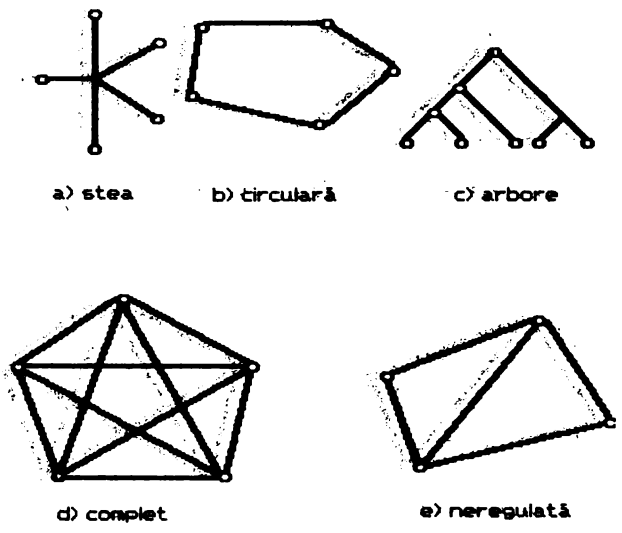


Figura 3.

Modelul broadcast presupune existența unui singur canal partajat de toate IMP-urile. Un mesaj transmis de un IMP este astfel recepționat de toate celelalte IMP-uri, fiecare dintre acestea urmînd a decide dacă mesajul îi este sau nu adresat.

Deci, revenind la exemplu, dacă A, B, C sînt legate cu o singură linie telegrafică (așa cum s-a presupus inițial), atunci la transmisia unui mesaj din A fiecare telegrafist "ascultă" mesajul transmis și îl preia numai dacă i se adresează. Cîteva exemple de configurații de tip broadcast sînt prezentate în *figura 4*.

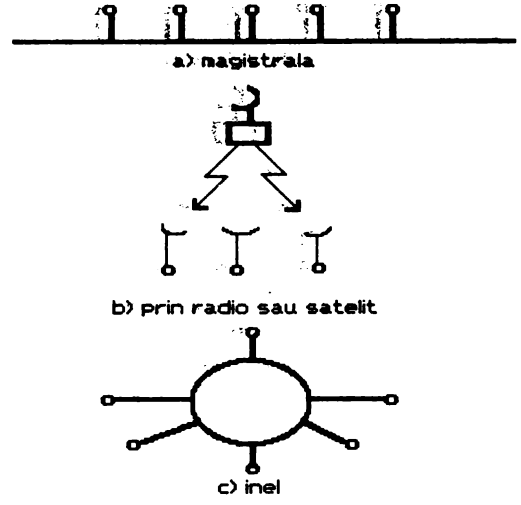


Figura 4.

ARHITECTURA REŢELELOR

Al doilea concept pe care trebuie să îl luăm în discuţie este cel de arhitectură. Pentru definirea lui vom porni de la organizarea proiectării unei reţele.

Reţelele moderne sînt proiectate într-o manieră puternic structurată în vederea reducerii complexităţii. Această structurare are ca scop împărţirea serviciilor asigurate de reţea în mai multe categorii începînd cu funcţiile de bază privind transportul fizic al unităţilor de informaţie, împachetarea acestora pentru formarea unor pachete, rutarea acestor pachete şi terminînd cu aplicaţiile cele mai generale. Astfel, funcţiile cele mai simple stau la baza altor funcţii care le apelează şi care sînt la rîndul lor apelate de funcţii mai complexe. Organizarea se face deci ierarhizat pe o serie de nivele, fiecare dintre acestea fiind construit deasupra predecesorului. Scopul unui asemenea nivel este de a oferi servicii sub forma unor funcţii apelabile nivelelor de deasupra sa, scutindule pe acestea din urmă de amănunte privind implementarea acestor servicii.

Deşi nu există o analogie perfectă cu exemplul nostru, totuşi, se poate considera că fiecare nivel este reprezentat de un operator care oferă anumite servicii celorlalţi operatori. Pentru clarificare să considerăm serviciile oferite de fiecare operator la transmisia unui mesaj de la A la B.

La transmisie:

— *operatorul* 4 - urmăreşte ecranul radarului şi comunică coordonatele avionului (într-un sistem de referinţă local);

— *operatorul* 3 - face transformarea coordonatelor locale în coordonate globale (la nivelul întregii ţări) şi notează poziţia avionului pe hartă;

— *operatorul* 2 - citeşte poziţia de pe hartă şi, eventual pe baza poziţiilor anterioare calculează viteza şi traiectoria. Apoi transmite într-un mesaj codificat toate datele necesare la operatorul 1;

— *operatorul* 1 - transmite efectiv mesajul pe linia telegrafică.

La recepţie:

— *operatorul* 1 - preia mesajul recepţionat şi îl comunică operatorului 2;

— *operatorul* 2 - interpretează mesajul primit (îl decodifică) şi comunică poziţia avionului;

— *operatorul* 3 - notează poziţia avionului pe hartă şi dacă avionul urmează să intre în spaţiul aerian local transmite poziţia în coordonate locale operatorului 4:

— *operatorul* 4 - preia urmărirea avionului (eventual modifică parametrii staţiei pentru a putea observa avionul în condiţii optime).

Un nivel de pe o anumită maşină poate conversa cu acelaşi nivel de pe o altă masină. Regulile utilizate în conversaţie sînt cunoscute sub numele de protocolul nivelului respectiv. Acest protocol foloşeste serviciile furnizate de nivelele inferioare. Protocolul unui anumit nivel are deci la bază protocolul nivelului imediat inferior. Astfel, de exemplu operatorii de pe nivelele 2 ale fiecărui centru de comandă comunică între ei folosind acelaşi protocol (adică aceleaşi scheme de codificare şi de decodificare a mesajelor). Dar, ei nu comunică direct,ci se folosesc de protocolul de comunicaţie de pe nivelul 1.

În *figura 5* este prezentată o reţea cu şapte nivele.

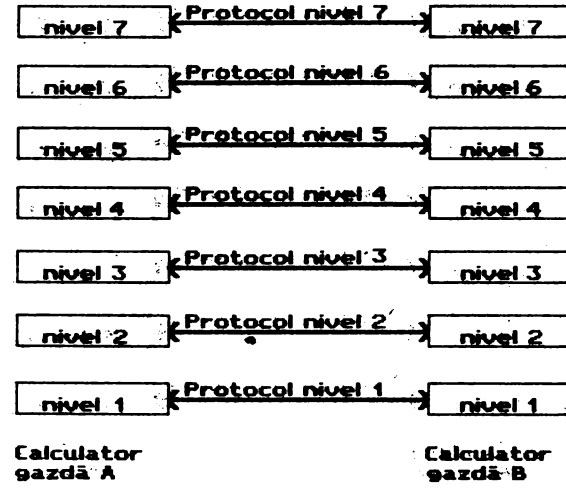


Figura 5

Trebuie înţeles faptul că un astfel de nivel este constituit dintr-un set de funcţii implementate, fie în hardware,fie în software pe un anumit calculator din reţea. De exemplu, nivelul 1 poate executa funcţii de transportare a biţilor prin mediul de comunicaţie, nivelul 2 poate împacheta aceşti biţi în octeţi şi asigura un anumit grad de siguranţă a transmisiei iar nivelele cele mai de sus pot implementa funcţii de transfer de fişiere, apeluri de funcţii situate pe un alt calculator din reţea etc. Un nivel superior apelează funcţiile puse la dispoziţie de un nivel inferior. Structurarea este impusă de complexitatea funcţiilor utilizate. Comunicarea între două nivele adiacente se face printr-o interfaţă bine stabilită,astfel că este posibilă înlocuirea totală a unui anumit nivel cu condiţia respectării riguroase a interfeţelor sale cu nivelele vecine. Astfel, pentru operatorii de pe nivelul 2 nu contează dacă pe nivelul 1 mesajele se transmit folosind telegraful, telefonul sau un dispozitiv radio atîta timp cît mesajele primite şi transmise au exact aceeaşi formă.

Datele nu se transferă direct între nivelele echivalente de pe două calculatoare gazdă, cu excepţia celui mai de jos nivel. La staţia care transmite, nivelele superioare comunică datele către cele inferioare, apelînd la serviciile acestora, pînă se ajunge la ultimul nivel care transmite efectiv datele pe mediul de comunicaţie; la recepţie acelaşi nivel de jos preia datele de pe mediu şi le furnizează către nivelele superioare. Astfel, între nivelele de jos are loc o comunicaţie fizică în timp ce între oricare alte nivele are loc o comunicaţie virtuală. Comunicaţia între două nivele adiacente aparţinînd aceleiaşi maşini este realizată cu ajutorul unei interfeţe ce defineşte operaţiile primitive şi serviciile pe care nivelul inferior le oferă celui superior.

Mulţimea nivelelor şi protocoalele asociate acestora formează arhitectura reţelei.

În *figura 6* se poate observa fluxul informaţiei între două calculatoare gazdă în cadrul reţelei structurate.

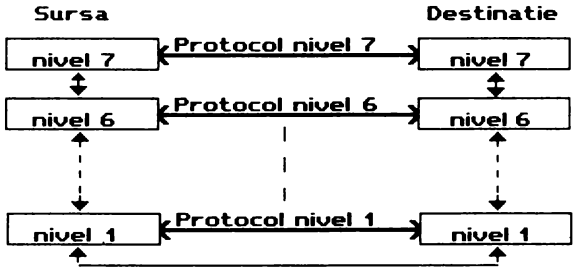


Figura 6

Protocolul unui nivel are la bază o serie de algoritmi care asigură o anumită calitate a comunicaţiei. Pentru ca aceasta să fie ridicată se folosesc diferite metode cum ar fi:împărţirea mesajelor în pachete, numerotarea acestora în cadrul unei secvenţe pentru păstrarea ordinii datelor, transmiterea mesajelor pachet cu pachet, confirmarea recepţiei fiecărui pachet de date, retransmisia pachetelor în cazul pierderii lor pe mediul de comunicaţie sau în cazul nerecepţionării acestora de către staţia destinaţie. Revenind la exemplul nostru un mesaj este format din una sau mai multe fraze, iar o frază din mai multe cuvinte. Cuvintele sînt transmise literă cu literă de către telegrafist folosind alfabetul Morse. Telegrafistul care recepţionează mesajul poate confirma recepţia lui dacă l-a primit în bune condiţiuni sau poate cere retransmisia lui dacă mesajul este neînţeles. De notat că, recuperarea erorilor de transmisie se poate face la mai multe nivele. Astfel, telegrafistul(nivelul 1) poate cere retransmisia mesajului dacă primeşte litere sau cuvinte incorecte, sau operatorul 2 poate cere retransmisia mesajului dacă conţinutul mesajului nu este inteligibil (lipsesc cuvinte, simboli etc.).

INTERCONECTAREA REŢELELOR

O dată cu creşterea numărului de reţele locale (**LAN**) a apărut firesc necesitatea de a conecta mai multe **LAN**-uri între ele. Problema interconectarii nu are o rezolvare generală datorită marii diversităţi de tipuri de reţele existente. Astfel, fiecare mare producător de calculatoare are propria lui reţea. Acestea diferă între ele atît ca protocoale utilizate, cît şi ca viteză de transmisie şi suport fizic utilizat (ex: linii de cupru, fibre optice etc.). De aici rezultă mai multe moduri de abordare optimă a problemei interconectării.

În prezent se folosesc 3 tipuri de dispozitive de interconectare:

1. Bridge

Bridge-ul este cel mai simplu dispozitiv de conectare a două sau mai multe reţele locale cu aceleaşi caracteristici. Pentru a reprezenta acest tip de interconectare, în exemplul nostru vom face unele modificări. Presupunem că avem o ţară cu o întindere mai mare împărţită în regiuni. Spaţiul aerian al fiecărei regiuni este supravegheat de mai multe radaruri conectate ca în Liliput. Comunicarea între două centre de comandă aparţinînd la două regiuni diferite se face printr-un punct de comunicaţie care în principiu va avea caracteristicile unui bridge. Acesta este format dintr-un singur telegrafist care are acces la reţelele de comunicaţie ale celor două regiuni. Acesta recepţionează toate mesajele transmise. Dacă un mesaj este transmis de la un punct de comandă al unei regiuni la un punct de comandă al celei de-a doua regiuni telegrafistul îl preia şi îl retransmite pe cealaltă reţea pentru a ajunge la centrul de comandă destinaţie.

2. Router

Router-ul se foloseşte la interconectarea a două reţele care respectă acelaşi standard,dar cu implementări care pot fi diferite. De exemplu, pot diferi prin lungimea şi formatul mesajelor trimise. **Router**-ul trebuie să asigure transformarea mesajelor între reţelele pe care le interconectează. Deci, mesajele primite de la o reţea sînt modificate pentru a putea fi recunoscute şi manipulate corect în cealaltă reţea. Revenind la exemplul nostru, să presupunem că, Liliput împreună cu o ţară vecină, s-o numim Blefuscu, organizează supravegherea spaţiului aerian comun. În Blefuscu observaţia este organizată similar ca în Liliput. În acest caz, se foloseşte un centru de comunicaţie asemănător cu cel descris la punctul 1. Spre deosebire de acesta, în acest caz este nevoie pe lîngă telegrafist de un translator care să traducă mesajele dintr-o limbă în alta.

3. Gateway

Pînă acum am arătat cum se pot conecta reţele cu protocoale care se conformează aceluiaşi standard. Aceasta nu se poate dacă există o reţea deja instalată cu o arhitectură proprie. Datorită investiţiilor deja făcute în softul de comunicaţie nu se poate pune problema schimbării acestuia. Această problemă este rezolvată printr-un dispozitiv numit **Gateway** care permite coexistenţa reţelelor bazate pe standarde de comunicaţie diferite. Pentru a clarifica ideile putem considera exemplul anterior, dar cu anumite amendamente. Să presupunem că Blefuscu are un alt mod de supraveghere a spaţiului aerian (sisteme de coordonate diferite, modalităţi de transmisie diferite etc). Nu vom detalia aici acest sistem de organizare deoarece este lipsit de importanţă. În acest caz centrul de comunicaţie va avea o organizare mai complexă: el va cuprinde toată organizarea ierarhică regăsită într-un centru de comandă din Liliput şi, în paralel, într-un centru de comandă din Blefuscu. O informaţie transmisă din Liliput către Blefuscu va fi recepţionată de telegrafist(operatorul 1), care o va comunica operatorului 2 şi aşa mai departe pînă la operatorul 4. La acest nivel,informaţia va fi transmisă către nivelul superior al structurii corespunzătoare organizării din Blefuscu. Informaţia preluată de acesta este prelucrată şi apoi transmisă în modul specific protocoalelor şi modalităţilor de comunicaţie din Blefuscu către nivelele inferioare.

ÎN LOC DE ÎNCHEIERE

Acest articol se adresează tuturor acelora care doresc o iniţiere în lumea reţelelor de calculatoare. Intrarea în această lume depinde în ultima instanţă de gradul de cunoaştere şi de pregătire. Prin seria de articole pe care ne-o propunem dorim să contribuim la apropierea momentului în care vom vedea pe biroul fiecăruia un calculator personal conectat la reţeaua publică, avînd acces la diverse baze de date, putînd transfera programe şi avînd posibilitatea de a lucra acasă.

BAZELE LOGICE ALE INTELIGENŢEI ARTIFICIALE

Prof.univ.dr. Cornel Popa

Dacă silogistica aristotelică a fost un organon sau un instrument pentru analiza unor raţionamente din viaţa cotidiană, logica matematică modernă este un organon pentru analiza limbajelor ştiinţifice şi a bazelor de cunoştinţe din programele expert şi de inteligenţa artificială.

Dorim să argumentăm această teză în mai multe eseuri de logică.

1. Formalizare, capacitate inferenţială şi inteligenţă

Orice fiinţă umană îşi preţuieşte puterea de judecată. Fiecare dintre noi acceptăm mult mai uşor că, într-o împrejurare sau alta, nu a deţinut informaţia necesară, decît că a deţinut-o, dar nu am avut capacitatea logic-raţională de a degaja din ea concluziile necesare. Sîntem fiinţe orgolioase ce nu acceptă nici o ofensă adusă competenţei intelectului propriu, capacităţii lui de a prelucra adecvat informaţia dată.

Aptitudinea de a deduce şi de a calcula, capacitatea de a degaja consecinţe cît mai îndepărtate şi neaşteptate este indubitabil un atribut al inteligenţei.

În afară de capacitatea de a infera sau deduce, inteligenţa presupune chiar înainte de aptitudinea deductiv-calculatorie, abilitatea de a reprezenta intern în imagini subiective sau configuraţii lexicale, relaţii şi structuri din mediul extern. Înainte de a calcula şi de a infera, trebuie să ştim să codificăm realitatea în sisteme de ecuaţii sau structuri semiotice.

Să pornim de la concret. Să admitem că ni se dau raţionamentele:

R1. 1. Dacă ninge şi plouă, mă duc la munte.
2. Dacă mă duc la munte nu citesc
3. Citesc sau joc şah.
4. Dar eu nu joc şah.
5. Ninge.

---

6. Deci, plouă.

R2. 1. Toate pietrele nasc pui vii.
2. Toate caprele sînt pietre.

---

3. Toate caprele nasc pui vii.

R3. 1. Cei ce se tem de cineva nu sînt iubiţi de toţi.
2. Cei ce sînt păziţi se tem de cineva.
3. Şefii Mafiei sînt păziţi.

---

4. Şefii Mafiei nu sînt iubiţi de toţi.

R4. 1. Ion e fericit, dacă îl iubeşte cineva.
2. Ana iubeşte pe oricine o iubeşte.
3. Ion o iubeşte pe Ana.

---

4. Prin urmare, Ion e fericit.

Ni se cere:

a) Să le formalizăm în limbajul logicii predicatelor;

b) Să transpunem sub formă clauzală (eventual sub formă de clauze Horn, formulele obţinute la punctul a);

c) Să decidem prin metoda rezoluţiei asupra raţionamentelor date.

Admitem, deci, că pentru a putea întreprinde un calcul logic sau aritmetic, trebuie mai întîi să reprezentăm şi să simbolizăm o problemă într-un anumit limbaj formal, trebuie să transpunem obiectele şi relaţiile sesizate într-un domeniu empiric, în simboluri, relaţii şi operaţii proprii unei discipline formale. Rezolvitorii de probleme ştiu că alegerea cadrului de reprezentare, simbolizarea şi determinarea ecuaţiilor sînt momentele iniţiale, mai dificile, în drum spre găsirea soluţiei. Pentru a putea face o traducere dintr-o limbă în alta, trebuie să cunoşti ambele limbi. Dar mai ales limba în care traduci.

În ceea ce ne priveşte, în rîndurile care urmează, limbajul în care vom face traducerea unor fapte, enunţuri este cel al logicii predicatelor de ordinul întîi. Este, deci, firesc să-l definim înainte de a-l utiliza.

2. Limbajul logicii predicatelor

Un limbaj formal se defineşte prin alfabet şi reguli de formare. Mai ortodox, acestea din urmă, pot fi redate şi ca reguli de rescriere sau producţiuni.

Alfabetul logicii predicatelor, pe care îl vom nota prin A, presupune mai multe subalfabete:

$A_{CI} = \{a,b,c,\ldots\}$	constante individuale
$A_{VI} = \{x,y,z,x_1,\ldots\}$	variabile individuale
$A_F = \{f,g,h,f_1,\ldots\}$	semne pentru funcţii sau operaţii sau semne funcţionale
$A_P = \{P,Q,R,P_1,\ldots\}$	semne predicative.
$A_{CL} = \{-,\&,\vee,\supset,\equiv,\ldots\}$	conective logice
$A_Q = \{\forall,\exists\}$	cuantificatori
$A_G = \{(,),[,],\{,\}\}$	semne de grupare

Este util, probabil, să amintim că alfabetul constantelor individuale A_C, va servi pentru redarea numelor proprii sau a descripţiilor singularizatoare. Variabilele individuale vor fi folosite pentru a desemna obiecte individuale oarecare dintr-un domeniu determinat, iar semnele funcţionale din alfabetul A_F vor servi pentru a codifica operaţii şi funcţii matematice.

Prin semnele din alfabetul A_P vom codifica proprietăţi şi relaţii instituite între diferite mulţimi de obiecte individuale.

Demn de menţionat este faptul că simbolurile din A_F şi A_P denotînd operaţii sau relaţii au întotdeauna un număr $0 \leq n < \infty$ de argumente ce descriu aritatea funcţiei sau relaţiei în cauză. Dacă $f \in A_F$ şi $P \in A_P$, atun-

ci $\delta(f)=n$ sau $\delta(P)=m$ vor exprima numărul de argumente al lui f şi respectiv al lui P.

Conectivele logice nu caracterizează obiecte din lumea empirică, ci sînt instrumente de asamblare şi analiză a formulelor ce descriu propoziţii declarative, predicate sau formule deschise.

Cuantificatorii $\forall x$ "ori care ar fi x..." şi $\exists x$ "există un x..." sînt instrumente de prelucrare a predicatelor şi de formare a propoziţiilor din predicate sau formule deschise. Totodată, ei intervin esenţial în formularea unor reguli de inferenţă din logica predicatelor.

Definirea limbajului logicii predicatelor presupune introducerea succesivă a conceptelor de termen, formulă atomară şi formulă bine formată în logica predicatelor.

Definim, inductiv, conceptul de termen, cu ajutorul unor reguli de formare.

R1. Constantele individuale sînt termeni.
Dacă $\alpha \in A_{CI}$, atunci $\alpha \in T$, mulţimea termenilor.

R2. Variabilele individuale sînt termeni.
Dacă $\alpha \in A_{VI}$, atunci $\alpha \in T$.

R3. Dacă $f \in A_F$ şi $\delta(f)=n$, i.e. f are n argumente şi dacă $t_1,\ldots,t_n$ sînt termeni, atunci $f(t_1,\ldots,t_n)$ este un termen compus.

DT. Numim *termeni* obiectele matematice construite cu ajutorul regulilor **R1, R2** şi *R3* din alfabeltele A_{CI}, A_{VI} şi A_F.

Exemple. a,x şi y vor fi termeni în virtutea regulilor **R1** şi **R2.** Dacă $f,g,h \in A_F$ şi $\delta(f)=1$, $\delta(g)=2$, $\delta(h)=2$ şi a, x şi y sînt termeni, atunci vor fi termeni, pe baza regulii **R3**, şi expresiile $f(a)$, $f(x)$, $g(a,y)$, $h(f(x), g(a,y))$, $g(f(a), f(x))$.

Pentru a ajunge la $h(f(x), g(a,y))$ am procedat astfel:

1. a	**R1**
2. x	**R2**
3. y	**R2**
4. $f(x)$	**R3,2**
5. $g(a,y)$	**R3,1,3**
6. $h(f(x), g(a,y))$	**R3,4,5**

Putem defini acum conceptul de atom sau formulă atomară în logica predicatelor.

Dăm mai întîi două reguli.

R4. Dacă t_1 şi t_2 sînt termeni, atunci $t_1=t_2$ este o formulă atomară.

Semnul '=' este un element privilegiat din A_P şi desemnează relaţia de identitate referenţială sau egalitate.

R5. Dacă $P \in A_P$ şi $\delta(P)=n$ şi $t_1,\ldots,t_n$ sînt termeni, atunci $P(t_1,..,t_n)$ este un atom sau o formulă atomară în logica predicatelor.

DA. Numim *atomi* sau formule predicative atomare orice formulă obţinută prin regulile **R4** sau **R5.**

Atomi se obţin, aşadar, din aplicarea unor semne predicative de aritate n la şiruri de n termeni.

Ecuaţiile matematice sînt predicate binare formate în concordanţă cu regula **R4.**

În manualele clasice de logică semnele predicative sînt redate prin litere majuscule $P(x)$, $R(x,y)$, $S(x,f(y),a)$. În limbajul **PROLOG** simbolurile predicative sînt notate prin cuvinte scrise cu litere mici, iar argumentele sînt redate prin cuvinte scrise cu iniţială majusculă; variabilele sînt redate prin litere majuscule.

Expresia "x vinde lui y, z" i.e. vinde (X,Y,Z) este un predicat ternar. Tot predicate ternare vor fi: plus (X,Y,Z), scad (X,Y,Z) sau $g(X,Y)=Z$ sau $h(X,Y)=Z$.

Dacă un atom predicativ conţine numai constante, atunci vom spune că a fost *instanţializat* sau interpretat într-un domeniu oarecare. Un atom instanţializat poate fi adevărat sau fals.

Atomii sînt cele mai simple formule ce pot fi adevărate sau false.

Pornind de la atomi putem forma, cu ajutorul conectivelor logice (vezi A_{CL}), formule predicative complexe. Adăugăm la regulile **R1- R5** încă trei reguli:

R6. Atomii sînt formule predicative.

R6 constituie iniţializarea definiţiei inductive a noţiunii de formulă în logica predicatelor. Regulile următoare extind mulţimea formulelor predicative.

R7. Dacă α este o formulă predicativă, atunci $-\alpha$ este şi ea o formulă predicativă bine formată.

R8. Dacă α şi β sînt formule predicative şi $*$ este un conectiv logic binar, i.e. $* \in A_{CL}$, atunci $\alpha * \beta$ va fi o formulă bine formată în logica predicatelor.

DL. Atomii, cu şi fără negaţie, se numesc *literali.*

DC. Clauzele sînt mulţimi de literali legaţi prin disjuncţie.

Exemple. Fie atomii: $P(x)$, $R(x,f(x))$, $S(y,z,t)$. Atunci: $-P(x)$, $-S(y,z,t)$, $R(x,f(x)) \vee -P(x)$ sînt formule bine formate în logica predicatelor.

Ultima regulă ne va permite formarea unor formule cu cuantificatori.

R9. Dacă α este o formulă bine formată în logica predicatelor şi Q este un cuantificator (i.e. Q= $\forall$ sau $\exists$), atunci $Qx\ \alpha\ (x)$ este o formulă bine formată.

Fie $S(x,y,z)\&R(x,u)$ o formulă a logicii predicatelor. Atunci $\exists$ y $\exists$ $x(S(x,y,z)\&R(x,y))$ va fi o formulă bine formată cu cuantificatori, care descrie un predicat $P(z)$.

DL. Pred. Numim *limbaj al logicii predicatelor* formulele obţinute din alfabetul $A=A_{CI} \cup A_{VI} \cup A_F \cup A_P \cup A_{CL} \cup A_Q \cup A_G$ prin intermediul regulilor **R1-R9.**

Obs.1. Regulile **R1-R3** conduc la formarea termenilor sau a expresiilor denotative.

Obs.2. Regulile **R4** şi **R5** permit construirea atomilor sau a formulelor predicative elementare.

Obs.3. **R6, R7** şi **R8** operează asupra atomilor pentru alcătuirea formulelor "moleculare" sau compuse, iar **R9** conduce la formarea formulelor cuantificate.

Obs.4. Cînd un semn funcţional este de aritate 0, i.e $\delta(f)=0$ sau f nu are nici un argument (se scrie f^0), atunci f devine o constată. La fel, cînd un semn predicativ P este de aritate zero, i.e. $\delta(P)=0$, atunci P devine o *variabilă propoziţională* ce poate lua valorile də 1 sau 0 (adevărat sau fals).

Obs.5. Limbajul logicii predicatelor şi cu atît mai mult sublimbajele termenilor şi al logicii propoziţiilor pot fi captate în cadrul unei gramatici generative independente de context sau în cadrul unei gramatici analitice de acelaşi tip.

Predicatele sînt formule deschise în care cel puţin o variabilă individuală apre nelegată de vreun cuantificator. Predicatele mai sînt numite şi funcţii propoziţionale predicative sau scheme cuantificaţionale. Ele pot fi privite ca structuri sau cadre generatoare de propoziţii prin cuantificare mai departe sau prin substituirea variabilelor individuale prin constante.

Limbajul logicii predicatelor are o mare capacitate descriptivă. În cadrul lui pot fi formalizate propoziţii complexe din limbile naturale şi din teoriile ştiinţifice.

3. Formalizarea unor raţionamente în limbajul logicii propoziţiilor

Să ne întoarcem la raţionamentele **R1-R4** date la început. Primul raţionament este formalizabil în limbajul logicii propoziţiilor, care este un sublimbaj al logicii predicatelor. Pentru redarea formală a lui **R1** vom folosi în exclusivitate alfabetele A_P,A_{CL},A_G. Mai mult A_P va fi serios restrîns prin postularea absenţei oricărui argument pentru variabilele P,Q,R,S i.e. $\delta(P) = \ldots = \delta(S) = 0$. Astfel P,Q... decad din statutul de variabile predicative la statutul de variabile propoziţionale; vor ţine locul unor propoziţii oarecare ce pot fi adevărate sau false; A_{CL} vor desemna conectivele logice $-$, &, $\vee$, $\supset$, $\equiv$, iar A_G vor fi semne de grupare.

Regulile de formare utilizate vor fi **R6, R7** şi **R8**, cu "degenerarea" provocată căderea variabilelor predicative la starea de variabile propoziţionale.

Prepoziţiile "şi" şi "sau" din limba română vor fi redate prin semnele conjuncţiei "&" şi corespunzător al disjuncţiei "$\vee$"; sintagma "dacă..., atunci" va fi redată prin implicaţie.

Pe această cale **R1** devine:

1. $(P\&\text{-}Q) \supset R$
2. $R \supset \text{-}S$
3. $S \vee T$
4. $-T$

---

5. P
6. Q

Este uşor de văzut că propoziţiile "ninge", "plouă", "mă duc la munte", "citesc", "joc şah" au fost redate, în ordine, prin variabilele P,Q,R,S,T.

Cerinţa a) a fost satisfăcută pentru **R1**. Întrucît clauzele sînt disjuncţii de literali, va trebui să înlocuim formulele ce conţin implicaţii cu formule echivalente cu ele ce nu conţin implicaţii. Vom înlocui, de asemenea, formulele ce conţin semnul negaţiei în faţa unor conjuncţii sau disjuncţii, cu altele echivalente cu ele ce conţin semnul negaţiei pe variabilele. Pentru aceasta vom folosi echivalenţele:

$$A \supset B \equiv -A \vee B$$
$$-(A\&B) \equiv -A \vee -B$$
$$-(A \vee B) \equiv -A \,\&\, -B \quad \text{legile lui De Morgan}$$
$$--A \equiv A \quad \text{legea dublei negaţii}$$

Obţinem astfel clauzele:

R1a.	**R1.b.**
1. $-P \vee Q \vee R$	1. {-P,Q,R}
2. $-R \vee -S$	2. {-R,-S}
3. $S \vee T$	3. {S,T}
4. $-T$	4. {−T}
5. P	5. {P}
6. Q	6. {Q}

R1 şi **R1b** sînt transpuneri clauzele a raţionamentului R1. Am satisfăcut astfel cerinţa b).

Pentru a satisface cerinţa c), adică a decide asupra unei formule prin metoda rezoluţiei, trebuie să ştim mai întîi ce este principiul rezoluţiei şi în ce constă metoda rezoluţiei.

4. Principiul rezoluţiei

Fie k_1 şi k_2 două clauze d isjuncte astfel că $l \in k_1$ şi $-l \in k_2$ (l este un literal). Dacă k_1 şi k_2 sînt adevărate, atunci va fi adevărată şi clauza rezolvent în l din k_1 şi k_2, $res_l(k_1,k_2)$ unde

$$res_l(k_1,k_2)=k_1 \setminus \{l\} \cup k_2 \setminus \{-l\} \qquad (4.1)$$

Rezolventul în l din k_1 şi k_2 este clauza ce conţine literalii din k_1 şi literalii din k_2, după ce s-au omis l din k_1 şi $-l$ din k_2.

Exemplu.

$k_1=\{P,Q\}$
$k_2=\{-P,R\}$
$res_P(k_1,k_2)=\{P,Q\}\setminus\{P\}\cup\{-P,R\}\setminus\{-P\}=\{Q\}\cup\{R\}$
$=\{Q,R\}$

Principiul rezoluţiei este, ca *modus ponens, modus tollens* sau legea tranzitivităţii o schemă sau o regulă validă de inferenţă. El garantează conservarea adevărului de la premise (sau clauze) la concluzie (sau rezolvent).

Ori de cîte ori premisele sau clauzele părinți sînt adevărate este adevărată și clauza rezolvent.

Ultimul enunț reprezintă teorema rezolventului.

5. Teorema rezolventului

Fie două clauze k_1 și k_2 astfel că $l \in k_1$ și $-l \in k_2$. Dacă k_1 și k_2 sînt adevărate, atunci și rezolventul lor, $res_l(k_1,k_2)$, va fi adevărat.

Demonstrație

1.	$k_1 \& k_2$	ip.
2.	$l \in k_1$	ip.
3.	$-l \in k_2$	ip.
4.	$l = 0$	ip.supl.
5.	k_1	(E&,1)
6.	k_2	(E&,1)
7.	$k_1 \setminus \{ l \}$	(5,4,∨)
8.	$k_1 \setminus \{ l \} \cup k_2 \setminus \{ -l \}$	(v,7)
9.	$res_l(k_1,k_2)$	(**RE**,8,4,l)
10.	$l=1$	ip.supl.
11.	$-l=0$	(def.neg.)
12.	$k_2 \setminus \{ -l \}$	(6,11,∨)
13.	$k_1 \setminus \{ l \} \cup k_2 \setminus \{ -l \}$	(v,12)
14.	$res_l(k_1,k_2)$	(**RE**, 13, 4.1)

Observații. Demonstrația este din ipoteze. Acceptînd adevărul clauzelor, rezultă adevărul rezolventului. 1-3 sînt ipoteze. Din ipoteza suplimentară 4 că $l=0$ și ipotezele 1-3 rezultă 9, respectiv rezolventul. Din ipoteza suplimentară 10, că $l=1$ și celelalte ipoteze rezultă, de asemenea, rezolventul. Dar l nu poate fi decît 0 sau 1.

În consecință, rezolventul este necesar adevărat, dacă, clauzele sale "părinți" sînt adevărate.

5 și 6 rezultă din 1, prin definiția conjuncției. Dacă o conjuncție este adevărată, atunci sînt adevărate ambele propoziții conjugate. Trecerile de la 4 și 5 la 7 și de la 6 și 11 la 12 se întemeiază pe definiția disjuncției: dacă o disjuncție este adevărată și unul dintre literalii ei este fals, atunci în mod necesar va fi adevărată și disjuncția formată din cea anterioară prin omiterea literalului fals:

$$\frac{A \vee 0}{A}$$

Trecerile de la 7 la 8 și de la 12 la 13 se întemeiază, de asemenea, pe proprietățile disjuncției:

$$\frac{A}{A \vee B} \qquad \frac{A}{\{A,B\}} \qquad A \supset A \vee B,\ A \supset B \vee A$$

Teorema rezolventului ne arată că principiul rezoluției este o schemă validă de inferență.

Principiul rezoluției a fost formulat de J.A.Robinson în 1965 în articolul "A Machine Oriented Logic and the Principle of Resolution". El este o generalizare a unor scheme de inferență utilizate încă de logicienii antici din școlile megarică și stoică.

El conține drept cazuri particulare schemele clasice de inferență modus ponens, modus tollens sau legea tranzitivității. Acestea pot fi reduse la principiul rezoluției după cum se vede:

modus ponens

$$\frac{A \supset B \quad A}{B} \qquad \frac{-A \vee B = k_1 \quad A = k_2}{B = res_A(k_1,k_2)}$$

modus tollens

$$\frac{A \supset B \quad -B}{-A} \qquad \frac{-A \vee B = k_1 \quad -B = k_2}{-A = res_A(k_1,k_2)}$$

tranzitivitatea

$$\frac{A \supset B \quad B \supset C}{A \supset C} \qquad \frac{-A \vee B = k_1 \quad -B \vee C = k_2}{-A \vee C = res_A(k_1,k_2)}$$

Exerciții de logica propozițiilor

I. Se dau formulele:

1. $(p \supset (q \supset r)) \supset (p \supset q) \supset (p \supset r))$
2. $(p \supset r) \supset ((q \supset r) \supset ((p \vee q) \supset r)))$
3. $((q \supset s) \& (p \supset q) \& (p \vee r) \& (r \supset t) \& (t \supset u)) \supset (s \vee u)$

a) Să se determine clauzele provenite din premise și din concluzii.

b) Să se verifice validitatea formulelor 1, 2 și 3 prin derivare rezolutivă.

c) Să se verifice validitatea aceloraşi formule prin respingere rezolutivă.

d) Să se construiască un raționament în limba naturală avînd structura formulelor 1, 2 și 3.

II. Să se verifice prin derivare și respingere rezolutivă validitatea raționamentelor.

A) *Dilemă 1*

1. Dacă alerg, transpir.
2. Dacă joc șah, devin sedentar.
3. Dacă transpir, răcesc.
4. Alerg sau joc șah.
5. Dacă devin sedentar, mi se atrofiază mușchii.

---

6. Prin urmare, răcesc sau mi se atrofiază mușchii.

B) *Dilemă 2*

1. Dacă fac sport, cheltuiesc energie.
2. Dacă cheltuiesc energie, nu devin obez.
3. Fac sport sau privesc opere de artă.
4. Dacă nu devin obez, mă plac fetele.
5. Dacă privesc opere de artă, mă cultiv artistic.
6. Dacă mă plac fetele, sînt fericit.

---

7. Prin urmare, sînt fericit sau mă cultiv artistic.

REŢELE NEURONALE

Ing. Florin Dobrian
Ing. Ion Stoica
Ing.Florin Sultan

Deşi studiate încă de la jumătatea acestui secol calculatoarele neuronale au produs un adevărat impact în lumea tehnicii de calcul abia în ultimul deceniu, fapt datorat în mare măsură atît progreselor tehnologice cît şi spectaculoasei dezvoltări a software-ului înregistrate în această perioadă. Puternica renaştere a acestei direcţii de vîrf a cercetării a stîrnit vii dezbateri şi controverse printre specialişti. Ce sînt aceste neurocalculatoare? Sînt ele capabile să concureze creierul uman? Vor înlocui oare calculatoarele tradiţionale? Iată cîteva întrebări la care încercăm să răspundem printr-o serie de articole, deschisă în acest număr al revistei.

SURSA DE INSPIRAŢIE ESTE MODELUL BIOLOGIC

Întrebat la ora actuală despre creierul uman, nici un psiholog sau neurochirurg nu ar putea da o imagine globală despre funcţionarea acestuia. Problema este atît de complexă încît rezolvarea ei completă mai trebuie să aştepte încă foarte mult timp. Anumite lucruri se cunosc totuşi. Se ştie în general că sistemul nervos este alcătuit din celule numite neuroni între care există o reţea de conexiuni. Neuronii conţin un anumit număr de dendrite şi cîte un axon, prin intermediul cărora se fac interconexiunile. O legătură între doi neuroni poartă numele de sinapsă. Dendritele constituie intrările în celula neuronală în timp ce axonul constituie ieşirea. Aşadar, un neuron are mai multe intrări şi o singură ieşire. Cu toate acestea, axonul se ramifică, astfel că neuronul poate conecta ieşirea sa la mai multe intrări ale altor neuroni. O sinapsă se realizează între axonul unui neuron şi una din dendritele altui neuron.

Un neuron poate fi activat prin impulsuri electrice la nivelul dendritelor. Totalitatea impulsurilor prezentate la intrare pot excita neuronul astfel încît acesta să genereze la rîndul său un impuls electric la ieşire, către ceilalţi neuroni cu care este conectat. Condiţia de excitare a neuronului este ca impulsurile de la intrare să depăşească un anumit prag de sensibilitate specific celulei nervoase.

Determinant pentru modul de funcţionare a creierului este gradul foarte ridicat de paralelism în procesarea informaţiilor. Astfel, creierul uman conţine în jur de zece miliarde de neuroni, fiecare cu peste o mie de sinapse. Această organizare îi permite să prelucreze informaţii cu o viteză foarte ridicată, deşi un neuron individual este destul de lent (viteza de activare a unui neuron este de un milion de ori mai mică decît viteza de comunicare a circuitelor electronice actuale).

REŢELE NEURONALE ARTIFICIALE

Avînd ca model reţeaua de neuroni naturală, specialiştii au studiat posibilitatea construirii de sisteme artificiale care să îndeplinească funcţii similare. Aceste sisteme pot fi întîlnite sub denumirea de reţele neuronale sau modele conexioniste iar implementările fizice ale unor calculatoare bazate pe tehnicile neuronale se numesc calculatoare neuronale sau pe scurt neurocalculatoare. Primele reţele neuronale au fost dezvoltate sub forma unor simulatoare software rulabile pe calculatoare secvenţiale clasice. Ulterior au apărut şi implementări hardware ale acestor reţele sub forma unor circuite electronice specializate.

O reţea neuronală este alcătuită dintr-o mulţime de noduri în care se află elementele de procesare respectiv neuronii. Ele reprezintă elemente de calcul neliniare care operează în paralel şi sînt organizate în moduri similare celor din reţelele biologice.

Prin analogie cu neuronul biologic, un neuron artificial are un anumit număr N de intrări (corespondente dendritelor) şi o singură ieşire (corespondentă axonului), dar care se poate conecta la intrările mai multor neuroni. Fiecare intrare are asociată o pondere care reprezintă importanţa pe care un impuls prezent la intrarea respectivă o are în activarea neuronului. Conexiunile se realizează prin legarea ieşirilor unor neuroni la intrările altora, ponderile asimilîndu-se cu sinapsele. Impulsurile de la intrările unui neuron pot avea un caracter excitant (impulsuri pozitive) sau inhibitor (impulsuri negative), totalitatea lor determinînd activarea sau nu a respectivului neuron. Pragul de sensibilitate amintit la modelul biologic este prezent şi în neuronul artificial. El este notat θ. O altă caracteristică a neuronului este comportarea acestuia, definită printr-o funcţie de activare neliniară *f*.

Ponderile asociate intrărilor unui neuron se notează W_i. Ele multiplică valorile impulsurilor X_i corespunzătoare respectivelor intrări. Ponderile W_i iau în general valori reale. Neuronul calculează suma ponderată a tuturor impulsurilor de intrare din care scade valoarea de prag θ pentru a determina dacă combinaţia de la intrare a depăşit această valoare. În funcţie de rezultatul calculului precum şi de tipul funcţiei de activare caracteristice neuronului, acesta generează un semnal *Y* la ieşire. În *figura 1* este ilustrat un neuron, cu notaţiile corespunzătoare.

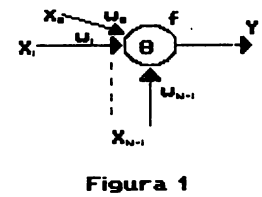


Figura 1

Ieşirea se generează conform formulei:

$$Y = f(\sum_{I=1}^{N-1} Wi * Xi - \theta)$$

Există mai multe tipuri de neliniarităţi specifice comportării unui neuron. Dintre cele mai cunoscute putem aminti următoarele trei:

- funcţia treaptă;
- funcţia rampă;
- funcţia sigmoidală;

Variaţia acestor trei modele de funcţii neliniare de activare este prezentată în graficele din *figura 2.*

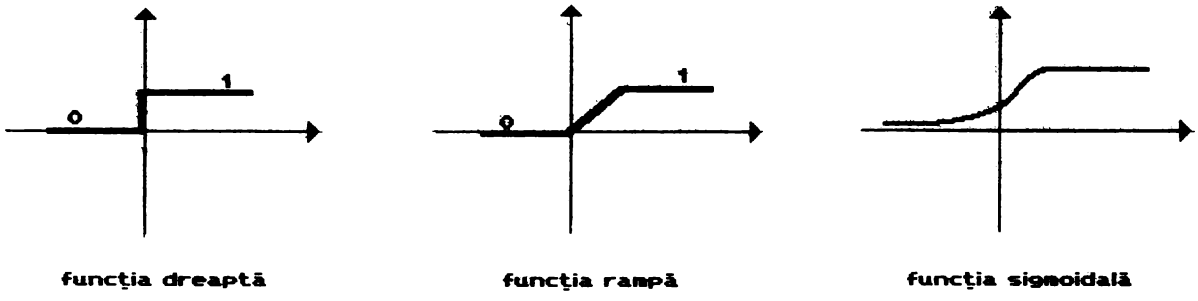


Figura 2

În cadrul unor reţele neuronale complexe se pot întîlni variaţii temporale ale parametrilor precum şi operaţii matematice mai complicate în locul simplei sumări.

Fiecare element de procesare dintr-o reţea neuronală are aşadar un număr de intrări şi o ieşire care este funcţie de intrări şi care este utilizată ca intrare de către alte elemente. Aceste elemente de procesare sînt organizate în mai multe straturi (nivele). Introducerea informaţiilor în reţea se face prin intermediul unui strat de neuroni de ieşire. Între aceste două straturi care asigură comunicarea cu mediul înconjurător este posibilă existenţa unor straturi intermediare numite şi straturi ascunse. Legăturile între neuronii din diversele straturi sînt specificate printr-o matrice de conexiuni ponderate. În funcţie de modurile de interconectare se pot deosebi o serie întreagă de topologii neuronale. Structura generală a unei reţele neuronale este ilustrată în *figura 3.*

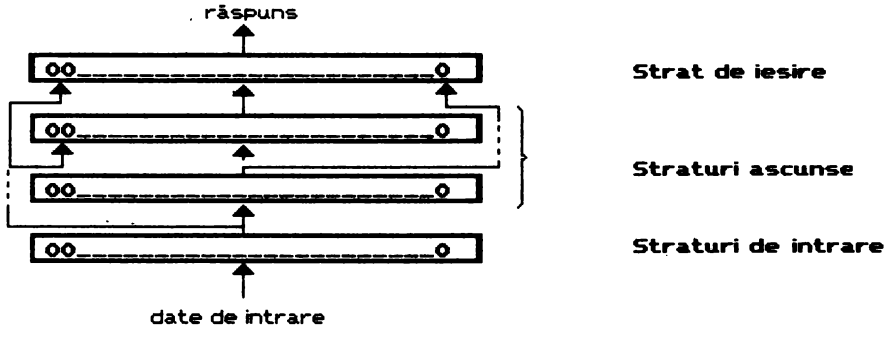


Figura 3.

Primele modele de reţele neuronale studiate conţineau numai unul sau două straturi de neuroni. În primul caz, singurul strat existent ţinea loc atît de strat de intrare cît şi de strat de ieşire. Aceste prime implementări limitau rolul respectivelor reţele. Ulterior s-au construit modele cu mai multe straturi. Adăugarea de straturi ascunse permite sistemelor concepute să-şi formeze o reprezentare internă a problemelor analizate. Aceasta înseamnă că în urma prezentării unor date la intrare, o reţea neuronală îşi organizează intern informaţiile într-un anumit mod.

Reţelele cu un singur strat rezumau această reprezentare internă numai la forma sub care erau prezentate informaţiile la intrare. Reţelele multistrat sînt mult mai puternice prin posibilitatea de a crea în interiorul straturilor ascunse propriile reprezentări ale problemelor pe care le rezolvă. *Figura 4* descrie trei exemple de reţele, cu cîte unul, două şi trei straturi.

Asigurarea de performanţe ridicate se face prin gradul ridicat de interconectare care determină un paralelism foarte accentuat. Datorită acestuia tehnicile de prelucrare a informaţiilor specifice calculatoarelor neu-

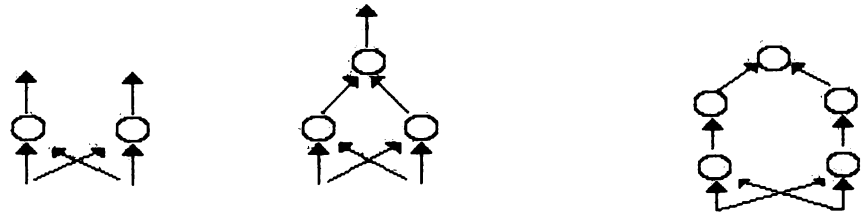


Figura 4.

ronale le fac să devină preferate calculatoarelor clasice (calculatoare secvențiale de tip von Neumann) într-o serie de aplicații. Principalele avantaje prezentate de către un neurocalculator sînt:

- toleranța la defecte (continuarea normală a activității rețelei chiar și în cazul în care un număr de neuroni individuali au încetat să mai funcționeze corect);
- recuperarea asociativă (regăsirea informațiilor stocate pe bază de conținut și formularea unor răspunsuri adecvate chiar dacă nu se găsește un corespondent intern la datele de intrare);
- degradare lentă (recuperarea situațiilor de încetare a funcționării unor neuroni).

Diferența față de un calculator secvențial constă tocmai în existența unui număr foarte mare de elemente de procesare care operează în paralel. Defectarea unor astfel de elemente sau a unor conexiuni între acestea nu conduce la o scădere semnificativă a performanțelor.

Majoritatea algoritmilor utilizați în diversele modele de rețele neuronale adaptează ponderile astfel încît să îmbunătățească performanțele. Acest proces poartă numele de învățare și constituie conceptul de bază în domeniul neurocalculatoarelor.

Cu toate aceste avantaje pe care le prezintă, rețelele neuronale nu sînt potrivite pentru aplicații care reclamă calcule complexe și de mare precizie cu numere. Datorită modului lor de funcționare asemănător creierului se poate spune că puterea de calcul nu reprezintă punctul lor forte. În consecință, pentru marea majoritate a aplicațiilor dezvoltate pînă la ora actuală, calculatoarele convenționale se dovedesc de departe a fi mai eficiente. În momentul în care se pune însă problema activităților care utilizează seturi de date incomplete sau informații neclare și contradictorii, rețelele neuronale depășesc calculatoarele convenționale, chiar și procesoarele paralele. Aplicații specifice acestor activități sînt cele ca recunoașterea vorbirii sau a imaginilor, unde mai multe ipoteze sînt urmărite în paralel. În această direcție, sistemele cele mai bine puse la punct bazate pe tehnicile clasice sînt departe de a egala performanțele unui creier uman. Abilitatea în adaptare pe care o au rețelele neuronale este esențială în domenii ca cele amintite mai sus, în care seturile de date pentru învățare sînt limitate iar în cursul funcționării pot să apară practic o infinitate de noi situații.

PROCESUL DE ÎNVĂȚARE

Învățarea unei rețele neuronale este o problemă de ajustare a ponderilor. Acest lucru poate avea loc atît manual cît și automat. Inițial, se atribuie anumite valori ponderilor conexiunilor dintre neuroni. Se prezintă apoi datele de intrare și se rulează rețeaua în mai multe iterații. În timpul acestui proces ponderile își schimbă valorile. Operația de iterare se termină în momentul în care rețeaua a ajuns să satisfacă anumite cerințe în legătură cu corectitudinea răspunsurilor furnizate, adică să răspundă corect la situațiile obișnuite cu care a fost învățată și să poată rezolva și o serie de situații noi care prezintă anumite variații față de cele cunoscute. Aceasta este activitatea inițială de învățare, în care rețeaua este deprinsă cu situațiile cele mai des întîlnite dintr-o aplicație specifică. În continuare pot să apară situații noi. Datorită modului specific de funcționare rețeaua generalizează pe baza informațiilor stocate și furnizează răspunsurile corecte.

Există trei elemente care caracterizează o rețea neuronală. Primele două sînt tipul elementelor de procesare și topologia rețelei. Cel de-al treilea element este schema de învățare. Au fost dezvoltați o serie întreagă de algoritmi care implementează aceste scheme. Din punct de vedere al asistării a procesului de învățare se disting următoarele modalități:

- învățarea supervizată;
- învățarea nesupervizată;
- învățarea autosupervizată.

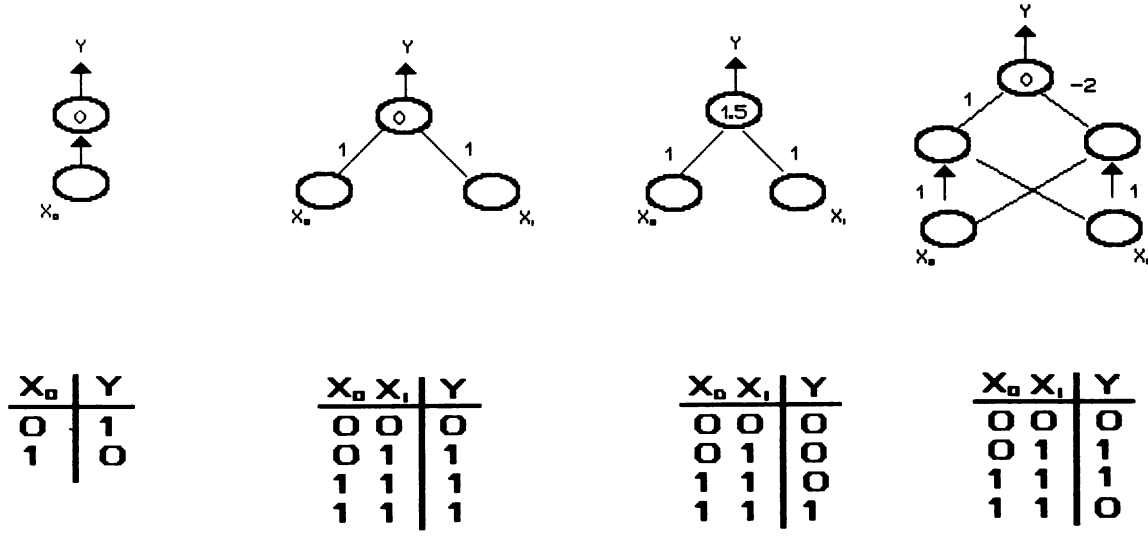
Învățarea supervizată are loc în cazul în care un agent extern introduce informațiile de probă și ieșirea pe care dorește să o obțină. Diferența între ieșirea obținută pe baza informațiilor de intrare și ieșirea dorită este folosită la recalcularea ponderilor rețelei. În cazul învățării nesupervizate adaptarea ponderilor se face numai pe baza intrărilor, fără intervenția agentului extern. Acest proces conduce la gruparea internă a datelor în vederea furnizării răspunsului corect. Învățarea autosupervizată apare în cazul în care rețeaua își monitorizează singură activitatea și corectează erorile în interpretarea datelor prin reacție inversă de reglare.

În procesul de învățare, în urma stabilirii unor ponderi inițiale și a prezentării datelor de intrare rețeaua trece

printr-o serie de stări pînă cînd ajunge într-o stare stabilă. Aceasta este atinsă în momentul în care ponderile asociate elementelor de procesare nu se mai schimbă. În perioada următoare, în care rețeaua este pusă în situația de a soluționa o anumită problemă, ponderile sînt fixe. Introducerea problemei se face prin furnizarea corespunzătoare a datelor către neuronii din stratul de intrare. Ca urmare, se declanșează un proces de activitate distribuită, adică se începe prelucrarea informațiilor de către toatele elementele de procesare, pe baza valorilor ponderilor stabilite în perioada de învățare. Răspunsul la problemă se prezintă prin intermediul neuronilor din stratul de ieșire.

CÎTEVA EXEMPLE SIMPLE

Pentru clarificarea ideilor să considerăm implementarea celor trei funcții logice elementare (NOT, OR, AND) precum și a funcției locale XOR cu ajutorul rețelelor neuronale. *Figura 5* ilustrează aceste exemple. Astfel, rețeaua *a* implementează funcția NOT, *b* funcția OR, *c* funcția AND iar *d* funcția XOR. În figură sînt indicate ponderile conexiunilor dintre neuroni precum și valorile interne de prag ale neuronilor. Funcția de activare utilizată de neuronii acestor rețele este funcția treaptă al cărei grafic a fost prezentat anterior. Funcționarea rețelelor *a, b* și *c* se



X_0	Y
0	1
1	0

X_0	X_1	Y
0	0	0
0	1	1
1	1	1
1	1	1

X_0	X_1	Y
0	0	0
0	1	0
1	1	0
1	1	1

X_0	X_1	Y
0	0	0
0	1	1
1	1	1
1	1	0

Figura 5.

poate deduce ușor din figură aplicînd formula de calcul a ieșirii pe baza intrărilor, a ponderilor, a valorii de prag și funcției de activare. De exemplu, în cazul rețelei *b*, pentru intrările $X_0=1$ și $X_1=0$, ieșirea are valoarea:

$$Y=f(W_0*X_0+W_1*X_1-\theta)=f(1*1+1*0-0)=f(1)=1$$

Se observă că în timp ce pentru implementarea primelor trei rețele sînt suficiente două straturi de neuroni, unul de intrare și unul de ieșire, în cazul rețelei care realizează funcția XOR mai este necesar un al treilea strat ascuns. Acesta este folosit pentru o reprezentare intermediară a informațiilor de intrare care să permită obținerea la ieșire a funcției dorite. În legătură cu acest aspect vom reveni într-un articol ulterior.

GLOSAR DE TERMENI

Activare distribuită — Proces de activare simultană a tuturor neuronilor dintr-o rețea neuronală.

Axon — Componentă a celulei nervoase prin care impulsurile părăsesc corpul celulei.

Conexiune — Cale între elementele de procesare pe care le unește în cadrul unei rețele.

Dendrită — Componentă a celulei nervoase prin care impulsurile sosesc în corpul celulei.

Funcţie de activare — Funcţie matematică după care se calculează valoarea ieşirii unui element de procesare.

Învăţare — Etapa în care unei reţele neuronale i se introduc date primare în vederea adaptării ponderilor.

Neuron — Unitate structurală şi funcţională a sistemului nervos.

Pondere — Valoare reală care exprimă importanţa unei intrări a neuronului.

Prag — Nivel energetic de excitare minim.

Sinapsă — Punctul de legătură între doi neuroni prin care se propagă impulsurile.

Strat ascuns — Strat de neuroni situat între stratul de intrare şi cel de ieşire.

LIMBAJUL C (V)

Conf.dr.ing. Irina Athanasiu

Cuvînt înainte

Spre surprinderea generală al treilea grup de autori preia rubrica destinată limbajului **C**. Ca orice echipă nouă vom începe prin a schimba modul de abordare,astfel încît,cititorii să înregistreze schimbarea.

Abordarea noastră pleacă de la două idei. În primul rînd, în paginile unei reviste nu poate să încapă un manual exhaustiv pentru un limbaj de programare, deci cititorul care doreşte într-adevăr să înveţe limbajul **C** va trebui în mod obligatoriu să apeleze la o carte. În al doilea rînd considerăm că rubrica prezentă este destinată unor cititori avansaţi care au cunoştinţe de programare (eventual în alte limbaje de programare) şi pentru care este interesant să discutăm aspecte specifice programării în limbajul **C** (este cunoscut faptul că o programare de tip **FORTRAN** este posibilă în orice limbaj de programare de nivel superior, oricît de sofisticat ar fi acesta - sper că nu am jignit pe nimeni prea tare). Cititorii începători sînt sfătuiţi să înveţe să lucreze întîi în limbajul Pascal pentru a căpătă o educaţie corectă în domeniul programării.

Din aceste motive vom încerca în această rubrică să discutăm anumite aspecte care ridică de obicei probleme celor care învaţă limbajul **C** venind din alte limbaje de programare. De asemenea vom încerca să discutăm modul în care se pot rezolva aplicaţii tipice utilizînd limbajul **C**.

Utilizarea pointer-*ilor* în limbajul C

Înţelegerea şi utilizarea corectă a *pointer*-ilor este una dintre cheile realizării unor programe **C** corecte şi eficiente. Utilizarea *pointer*-ilor este necesară pentru a realiza transferul parametrilor funcţiilor prin referinţă şi realizarea unor structuri de date dinamice complexe. De asemenea,utilizarea *pointer*-ilor în locul accesului indexat la vectori conduce la realizarea unor programe mai eficiente. Pe de altă parte existenţa mecanismului de adresare prin *pointer*-i poate să conducă la apariţia unor erori deosebit de dificil de localizat. Să considerăm de exemplu următorul program :

```
            void main( )
            {
/* 1 */      float x = 3.14, y ;
/* 2 */      int *p;
/* 3 */      p = &x;
/* 4 */      y = *p
/* 5 */      printf("%f",y);
            }
```

Pentru a uşura explicarea programului am numerotat în comentarii liniile din corpul acestuia. Să urmărim acest program. În linia 1 sînt declarate două variabile reale *x* şi *y*, şi se realizează iniţializarea variabilei *x*. În linia 2 este declarată variabila *p* reprezentînd un pointer către o valoare de tip *int*. În linia 3, acestei variabile i se atribuie ca valoare adresa variabilei *x*. Aparent, în linia 4 variabila *y* primeşte valoarea 3.14 (valoarea variabilei *x*). În linia 5 se afişează valoarea variabilei *y*. Dacă veţi încerca să executaţi acest program (după ce veţi ignora avertismentul referitor la linia 3, în care se atribuie un pointer către un *int* adresa unei valori de tip *float*), veţi constata că valoarea afişată nu este corectă (se va afişa valoarea -2621.000000). Explicaţia este simplă: în linia 4 datorită modului diferit în care sînt reprezentate valorile de tip *int* faţă de cele de tip *float* conţinutul adresei memorate în variabila *p* este interpretat în mod incorect. Desigur, avem de a face cu un caz fericit, eroarea fiind semnalată în avertismentul referitor la linia 3. Din păcate erori de tipul *pointer* neiniţializat nu pot să fie diagnosticate la fel de simplu la nivelul compilatorului.

Să consideram şi un exemplu tipic de utilizare a *pointer*-ilor: un program pentru afişarea în baza 16 a conţinutului unei zone de memorie :

```
/*
dump.c
Program pentru afişarea unei zone de memorie
începînd de la o adresă specificată de către utilizator. Se afişează 256 de octeţi.
*/
            void dump(unsigned long int);
            void main( )
            {
/* 1 */      unsigned long int start;
/* 2 */      printf("\nAdresa de început : ");
/* 3 */      scanf("%lu", &start);
/* 4 */      dump(start);
            }
            void dump(start)
            unsigned long int start;
            {
/* 6 */      unsigned char *p;
/* 7 */      int t;
/* 8 */      p = (unsigned char *) start;
/* 9 */      for (t = 0; t  255 ; t++, p++)
               {
/*10 */        if(!(t % 16)) printf("\n");
/* 11 */       printf("%2X ",*p);
               }
            }
```

În linia 1 este declarată variabila *start* de tip *unsigned long int*. În linia 3 această variabilă primeşte ca valoare adresa începînd de la care se doreşte preluarea octeţilor pentru afişare. Fiind o valoare de tip *unsigned long int* valorile posibile sînt în cadrul segmentului curent de date. În linia 6 este definit un *pointer* către o valoare de tip *char*. Se observă că acest *pointer* este neiniţializat. În linia 8 variabila *p* primeşte ca valoare, prin conversia

valorii variabilei *start,* adresa primului octet din zona de memorie care se parcurge. În ciclul format din liniile 9 — 11 se realizează afişarea pe rînd în baza 16 a 256 de octeţi începînd de la adresa indicată de utilizator. Pe fiecare linie se afişează 16 valori.

Utilizarea pointer-ilor pentru transferul parametrilor

În limbajul **C** mecanismul de transfer al argumentelor pentru funcţii este aşa numitul transfer prin valoare. Adică, ceea ce se transmite unei funcţii pentru fiecare argument actual este valoarea acestuia pe care funcţia îl copiază în contextul său. Din acest motiv orice modificare suferită de un astfel de argument în cadrul funcţiei se referă numai de la copia sa şi nu are nici un efect în afara funcţiei. Avînd în vedere că de multe ori este necesară realizarea unei actualizări pentru un argument în cadrul unei funcţii este necesar un mecanism prin care funcţia să aibă acces la argumentul propriu-zis şi nu numai la valoarea sa. În alte limbaje de programare (**PASCAL, FORTRAN**) acest mecanism se realizează prin transferul argumentelor prin referinţă. În acest caz ceea ce se transmite unei funcţii este adresa argumentului, pe baza căruia funcţia are atît posibilitatea să determine valoarea curentă a argumentului cît şi să modifice această valoare. În limbajul **C** se poate realiza un astfel de mecanism de transfer de argumente cu ajutorul *pointer*-ilor. Să considerăm de exemplu programul **test.c.**

```
/*
test.c
Program de demonstraţie pentru transferul argumentelor funcţiilor
*/
/* 1 */    void test(int *ip);
           void main( )
           {
/* 2 */     int i = 50, *ip = &i;
/* 3 */     printf("\ni = %i", i);
/* 4 */     test(ip);
/* 5 */     printf("\nDupă execuţie test \ni = %i",i);
           }
/* 6 */    void test(int *int_p)
           {
/* 7 */     *int_p = 100;
           }
```

Dacă veţi executa programul **test.c** veţi obţine :

```
i = 50
După execuţie test
i = 100
```

Să urmărim puţin programul. În linia 6 au fost declarate: tipul funcţiei **test** şi tipul argumentului. Se observă că argumentul este un *pointer* către o valoare de tip *int.* În linia 7 valoarea argumentului (a cărui adresă este conţinută în argumentul *int_p* transmis funcţiei) este modificată, aceasta primind valoarea 100. În cadrul programului principal în linia 2 se face iniţializarea şi declararea variabilelor *i* de tip *int* cu valoarea 50 şi *ip* de tip *pointer* către *int* cu adresa variabilei *i.* Linia 3 va tipări valoarea curentă a variabilei *i.* În urma apelului în linia 4 a funcţiei test, valoarea variabilei *i* se modifică conform efectului funcţiei **test.** Să modificăm programul **test.c** renunţînd la argumentul de tip *pointer.* Se obţine programul **test1.c**

```
/*
test1.c
Program de demonstraţie pentru transferul argumentelor funcţiilor
*/
/* 1 */    void test1(int i);
           void main( )
           {
/* 2 */     int i = 50, *ip = &i;
/* 3 */     printf("\ni = %i", i);
/* 4 */     test1(ip);
/* 5 */     printf("\nDupă execuţie test \ni = %i",i);
           }
/* 6 */    void test1(int i)
           {
/* 7 */     i = 100;
           }
```

Pe ecran se va afişa :

```
i = 50
După execuţie test
i = 50
```

deoarece în acest caz modificarea valorii variabilei i în funcţia test1 nu are nici un efect în afară funcţiei.

O altă situaţie în care utilizarea *pointer*-ilor pentru transferul argumentelor este obligatorie, este cea a argumentelor care nu sînt constante sau variabile simple. În acest caz utilizarea *pointer*-ilor este absolut necesară. Să considerăm de exemplu programul **copystr.c.**

```
/*
copystr.c
Program de demonstraţie pentru transferul argumentelor vectori
*/
/* 1 */    void copystr(char *destinaţie, char *sursă);
           void main( )
           {
/* 2 */     char *s1 = "primul şir";
/* 3 */     static char s2[ ] = {"al doilea şir"};
/* 4 */     char s3[80];
/* 5 */     copystr(s3, s1);
/* 6 */     printf("\n%s\n", s3);
/* 7 */     copystr(s3, s2);
/* 8 */     printf("%s\n", s3);
/* 9 */     copystr(s3, "al treilea şir");
/* 10 */    printf("%s\n", s3);
           }
/* 11 */   void copystr (char *destinaţie, char *sursă)
           {
/* 12 */    while(( *destinaţie++ =*sursă++) != '\0')
/* 13 */    ;
           }
```

Ca efect al execuţiei acestui program se afişează :

primul şir
al doilea şir
al treilea şir

Funcţia **copystr** este definită ca avînd două argumente de tip *pointer* către valori de tip *char*. În ciclul conţinut în linile 11 şi 12 se face copierea caracterului curent a cărui adresă este valoarea curentă a variabilei sursă la adresa care este valoarea curentă a variabilei destinaţie. După referirea (obţinerea valorii) fiecăreia dintre aceste variabile se face incrementarea valorilor variabilelor. Fiind vorba de valori de tip *pointer,* incrementarea se face conform spaţiului de memorie ocupat de tipul de date către care indică variabila *pointer* respectivă. Ciclul se repetă pînă cînd se ajunge la sfîrşitul şirului indicat de terminatorul de şir de caractere (un caracter cu valoarea \0). În cadrul programului principal sînt realizate trei apeluri ale funcţiei **copystr.** Toate cele trei apeluri utilizează ca prim argument variabila *s3.* Această variabilă a fost declarată în linia 4 ca vector de tip *char,* cu alte cuvinte numele *s3* fără paranteze desemnează un *pointer* către o dată de tip *char.* În primul apel este utilizată variabila *s1* care a fost declarată ca fiind un *pointer* către o valoare de tip *char,* deci corespunde tipului argumentului funcţiei. Al doilea apel utilizează variabila *s2* care ca şi *s3* este de tip *pointer* către o valoare de tip *char.* În al treilea apel ca valoare actuală pentru al doilea argument se transmite expresia "al treilea şir". Valoarea acestei expresii este un *pointer* către şirul constant "al treilea şir".

Pointer-i către structuri de date

Un alt exemplu tipic de utilizare a *pointer*-ilor constă din utilizarea acestora pentru parcurgerea mai eficientă a vectorilor. Să considerăm de exemplu programul **vect.c.**

```
/*
vect.c
Program pentru ilustrarea parcurgerii vectorilor cu ajutorul pointerilor
*/
        void main( )
        {
/* 1 */   static int data[5] = {1, 2, 3, 4, 5};
/* 2 */   int *int_ptr, i;
/* 3 */   printf("\nParcurgere fără pointeri \n");
/* 4 */   for (i = 0; i <= 4; i++)
/* 5 */       printf("%d ", data[i]);
/* 6 */   printf("\nParcurgere utilizînd pointeri \n");
/* 7 */   for(int_ptr = data;int_ptr <= &data[4];
            ++int_ptr)
/* 8 */       printf("%d ", *int_ptr);
/* 9 */   printf("\n");
        }
```

Efectul programului **vect.c** constă din afişarea textului

```
Parcurgere fără pointeri
1 2 3 4 5
Parcurgere utilizînd pointeri
1 2 3 4 5
```

Aparent, cele două parcurgeri sînt echivalente. Dar să urmărim puţin care sînt operaţiile care se execută de fapt pentru ciclurile descrise în liniile 4, 5 şi respectiv 7, 8. Pentru reprezentarea întregilor limbajul **C** utilizează 2 octeţi. Pentru a realiza accesul la valoarea unui element din vectorul *data* se realizează următoarele operaţii :

1. se obţine valoarea variabilei *i*;
2. se înmulţeşte această valoare cu 2 (dimensiunea unui element al vectorului);
3. se adună rezultatul cu adresa de început a vectorului *data*;
4. se obţine valoarea elementului.

Deci, pentru fiecare element de vector se realizează două citiri din memorie, o înmulţire (eventual deplasare) şi o adunare.

Pentru a realiza accesul la valoarea unui element din vectorul *data* utilizînd *pointer*-ul *int_ptr* se realizează două citiri din memorie: o citire pentru obţinerea valorii curente a *pointer*-ului *int_ptr* şi o citire utilizînd această valoare pentru a obţine valoarea elementului curent din vectorul *data*. De remarcat în acest program modul în care variabila *int_ptr* este utilizată ca variabilă de control pentru ciclul din liniile 7, 8. Valoarea de iniţializare este adresa primului element din vector. În loc de instrucţiunea:

```
int_ptr = data;
```

se putea utliliza instrucţiunea:

```
int_ptr = &data[0];
```

Valoarea de comparaţie finală este adresa ultimului element. Deoarece şi numele vectorului este considerat ca fiind de tip *pointer* o notaţie echivalentă pentru valoarea de comparaţie este

```
int_ptr  data + 4
```

şi nu

```
data + 4 * sizeof(int)
```

Actualizarea variabilei *int_ptr* se face conform modului de reprezentare a tipului *int.*

Operaţii asupra pointerilor

Aşa cum s-a văzut deja în exemplele anterioare asupra unui *pointer* se pot executa operaţii de adunare şi scădere cu valori întregi. Rezultatul unei astfel de operaţii este adresa unui obiect de acelaşi tip cu cel cu care a fost definit *pointer*-ul. Adică, dacă într-un program a fost definită o structură de tipul *tip,* şi există un *pointer p* definit ca *pointer* către o structură de tipul *tip,* atunci o

operaţie de forma *p* + 7 înseamnă de fapt că la valoarea conţinută în *p* se adaugă valoarea 7 * *sizeof(tip)*. Să considerăm însă programul **lungime.c.**

```
/*
lungime.c
Program pentru ilustrarea operaţiilor cu pointeri
*/
/* 1 */     int lungime (char *sir);
/* 2 */     void main( )
            {
/* 3 */      printf ("Lungimea şirului : 'Asta este ' este
             %i", lungime("Asta este"));
            }
/* 4 */     int lungime(char *sir)
            {
/* 5 */      char *ptr = sir;
/* 6 */      while (*ptr)
/* 7 */      ++ptr;
/* 8 */      return(ptr - sir);
            }
```

În urma execuţiei programului **lungime.c** se afişează textul :

Lungimea şirului : 'Asta este ' este 9.

Funcţia **lungime** primeşte ca argument un *pointer* către un şir de elemente de tip *char*. În cadrul procedurii utilizînd *pointer*-ul *ptr* (care de asemenea este un *pointer* către *char*) se parcurge şirul de caractere primit ca argument căutîndu-se sfîrşitul acestuia. În linia 8 se calculează diferenţa a doi *pointer*-i. Pentru ca rezultatul să fie corect *pointer*-ii trebuie să indice către acelaşi tip de date. Valoarea diferenţei este egală cu numărul de elemente cuprinse între valorile celor doi *pointer*-i. Deci dacă în exemplul considerat ar fi fost vorba de *pointer*-i către tipul *tip* de date atunci rezultatul scăderii valorilor celor doi *pointer*-i ar fi fost împărţit automat la valoarea *sizeof(tip)* obţinîndu-se rezultatul corect.

Pointer-i către funcţii

În cazul unui *pointer* către o funcţie compilatorul trebuie să ştie nu numai că *pointer*-ul adresează o funcţie dar şi tipul acesteia. De exemplu, declaraţia

```
int (*fp)(void);
```

declară un *pointer* către o funcţie care întoarce un rezultat de tip *int* şi nu are argumente. Parantezele în care este inclus *pointer*-ul (*fp) sînt necesare pentru a diferenţia această declaraţie de declararea unei funcţii fără argumente care întoarce ca valoare un *pointer* către un *int* :

```
int *fp (void);
```

Pentru a ilustra modul în care se poate utiliza un astfel de *pointer* să considerăm programul **comparaţie.c.**

```
/*
comparaţie.c
Program pentru ilustrarea utilizării pointerilor către
funcţii
*/
/* 1 */     void compara(char *a,char *b,int(*cmp)( ));
/* 2 */     int strcmp( );
            void main( )
            {
/* 3 */      char s1[80], s2[80];
/* 4 */      int (*p) (char *s1, char *s2);
/* 5 */      p = strcmp;
/* 6 */      printf("\nIntroduceţi două şiruri :\n");
/* 7 */      gets(s1);
/* 8 */      gets(s2);
/* 9 */      compara(s1, s2, p);
            }
/* 10 */void compara(char *a, char *b, int(*cmp)( ))
            {
/* 11 */     printf("\nCele două şiruri sînt ");
/* 12 */     if (!(*cmp)(a,b))
                 printf("egale");
/* 13 */     else printf("diferite");
            }
```

Declararea în linia 2 a funcţiei **strcmp** anunţă compilatorul că numele **strcmp** este un nume de funcţie care întoarce o valoare de tip *int*. Actualizarea valorii *pointer*-ului p se face prin instrucţiunea:

```
p = strcmp
```

(atenţie: numele funcţiei apare fără paranteze, altfel apariţia în membrul drept al operaţiei de atribuire a numelui funcţiei urmat de paranteze este interpretată ca un apel de funcţie). Se observă că nu se utilizează notaţia:

```
p = &strcmp;
```

Funcţia **compara** are trei argumente ultimul, fiind de tip *pointer* către o funcţie cu valoare de tip *int*. În linia 12 se realizează un apel al funcţiei transmise ca parametru funcţiei **compara**. Apelul funcţiei **compara** se putea face şi sub forma

```
compara(s1, s2, strcmp).
```

Din exemplul considerat nu rezultă poate foarte clar care sînt avantajele utilizării *pointer*-ilor către funcţii. Să ne imaginăm însă scenariul (cuvînt la modă) în care în acelaşi program doream să realizam şi compararea unor vectori avînd elemente de alte tipuri. În acest caz funcţia **compara** poate să rămînă neschimbată şi se definesc funcţii de comparaţie corespunzatoare tipurilor elementelor vectorilor.

EDITOARE ŞI FONTURI (IV)

Ion Paraschiv şi Tiberiu Spircu

Vom începe acum prezentarea sistemului de editare **Mattex**, realizat în Institutul de Matematică al Academiei Române, sistem creat pe ideea **WYSIWYG** (*"what you see is what you get"*) adică o corespondenţă biunivocă pixel ecran — pixel hîrtie.

Sistemul de editare **Mattex** a fost conceput în urma studierii experienţei dobîndite de către operatoarele maşinilor de editat **"Xerox 860"** în realizarea concretă de texte matematice complexe. (Un text complex necesită multe semne grafice distincte — de regulă cîteva sute — deci mai multe fonturi, precum şi posibilităţi de a realiza exponenţi, indici şi suprapuneri de semne; toate acestea pe lîngă cerinţele "clasice": centrări de rînduri, sublinieri sau bolduiri de semne, tabulări.)

Odată cu schimbarea maşinilor "Xerox 860" cu aparatură IBM PC compatibilă, prin acest sistem de editare s-a încercat, pe cît posibil, păstrarea habitudinilor precum şi abordarea în acelaşi stil "clasic" a activităţilor specifice aparaturii existente.

Avînd la dispoziţie o configuraţie **Compact AT** + monitor **VGA** + imprimantă laser **Xerox 4045**, sistemul de editare a fost creat special pentru această configuraţie, deci în prezent nu este portabil. Poate fi adaptat pentru monitoare **EGA** sau **CGA** (dar în acest ultim caz fineţea reprezentării suferă), respectiv pentru imprimante laser **HP Laser Jet.**

Sistemul lucrează de regulă cu ecranul în mod grafic (direct pe *videobuffer*) ceea ce asigură materializarea rapidă (practic insesizabilă de către ochiul uman) a tuturor comenzilor.

Sistemul **Mattex** este format dintr-un program executabil (versiunea actuală de mărime 250 KB) căruia i se ataşează familia de fonturi pentru ecran şi pentru imprimantă (o familie de fonturi conţine un font-imprimantă, precum şi fonturile-ecran reduse 1/2, 1/3, 1/4 corespunzătoare, şi ocupă în medie 40 KB). Crearea de noi fonturi este o operaţie relativ uşoară, iar recunoaşterea lor de către sistem este uşor de realizat.

Descrierea funcţionării.

Se presupune că pe disc există creat un *directory* special, denumit **DACT**. În acesta, fiecare utilizator îşi are propriul său *subdirectory*, în care se păstrează fişiere intermediare de lucru (acestea au o structură specială). Să descriem funcţionarea sistemului, pas cu pas, pe un exemplu.

Lansînd în execuţie programul **Mattex**, vom obţine în primul rînd un ecran asemănător celui prezentat în figura 1. În acesta ni se permite selectarea între discul rigid C: şi discul *floppy* A: . Să admitem că am selectat C: ; vom observa sus şi jos cîte o linie de mesaje. Linia de sus ne arată că ne aflăm în *directory* C:\DACT, în regim de SELECTARE. Linia de jos ne indică posibilităţile de folosire a tastaturii: tastele şi pentru selectare, tasta **Enter** pentru acceptarea subregistrului selectat în cazul nostru RODICA), tasta **ESC** pentru revenire la starea anterioară. După ce am selectat (ca subdirectory) RODICA, ecranul se modifică. Se observă apariţia conţinutului acestui subdirectory (lista fişierelor intermediare existente) precum şi alternativa de selecţie **Fişier existent — Fişier nou.** Pentru moment, dacă selectăm "Fişier existent", vom trece instantaneu la un ecran în care linia de jos de mesaje are un conţinut mai bogat, iar deasupra ei apare o nouă linie de mesaje, care ne indică tastele cu care putem efectua selecţia fişierului dorit. Acest fişier selectat poate fi: editat (tasta **F1**), tipărit (ceea ce înseamnă prelucrarea lui, crearea fişierului final corespunzător şi trimiterea spre imprimantă — tasta **F2**), redenumit (tasta **F3**),duplicat (tasta **F5**), distrus (tasta **F8**). Vom prefera însă, pentru moment, să apăsăm tasta **ESC** şi să revenim la etapa anterioară.

Selectînd "Fişier nou" ni se va cere tastarea unei denumiri. În *figura 1* este surprins momentul în care am reuşit deja să tastăm numele **"nelu"** pentru acest nou fişier. (Să observăm că tasta **Back** ne permite să corectăm eventualele greşeli, ceea ce va constitui o regulă generală.) Acceptînd acest nume,vom trece la o nouă etapă, cea de selectare a stilului, prezentată în *figura 2*.

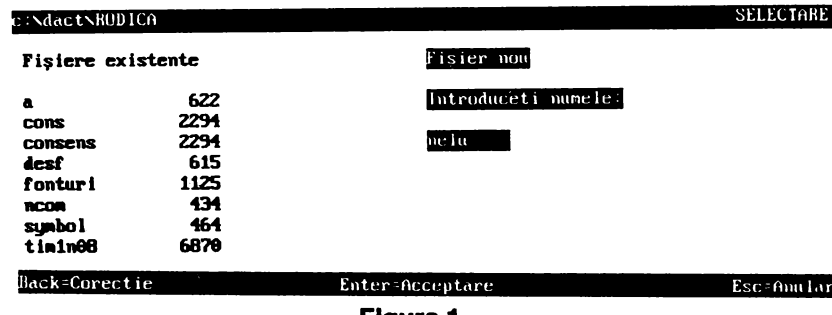


Figura 1

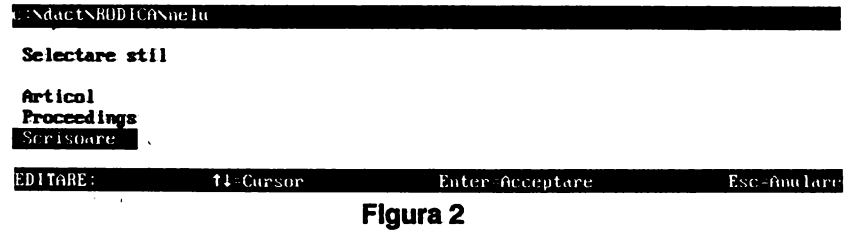


Figura 2

Stilul de editare constă din precizarea fonturilor folosite, a marginilor textului scris şi a tabulatoarelor introduse. În urma experienţei anterioare de lucru au fost identificate trei stiluri principale, intitulate "articol", "proceedings", respectiv "scrisoare".

Odată cu selectarea stilului ieşim din regimul de SELECTARE şi intrăm în regimul de EDITARE. În acest moment linia de mesaje de sus va conţine:

a) denumirea completă a fişierului intermediar în lucru

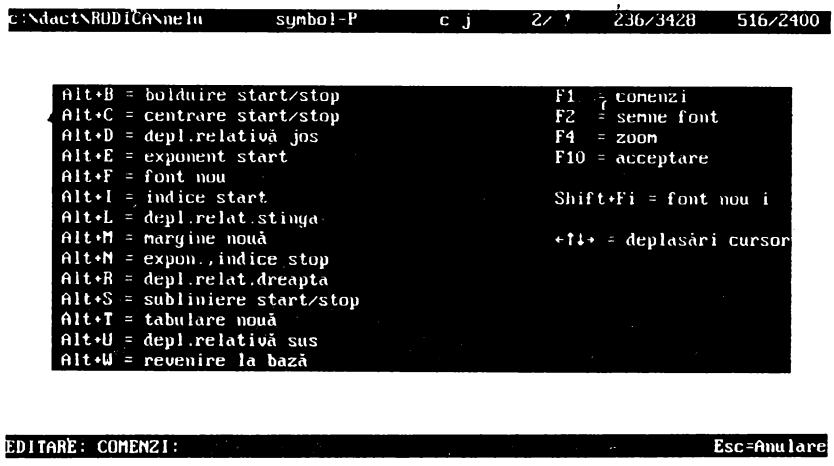


Figura 3

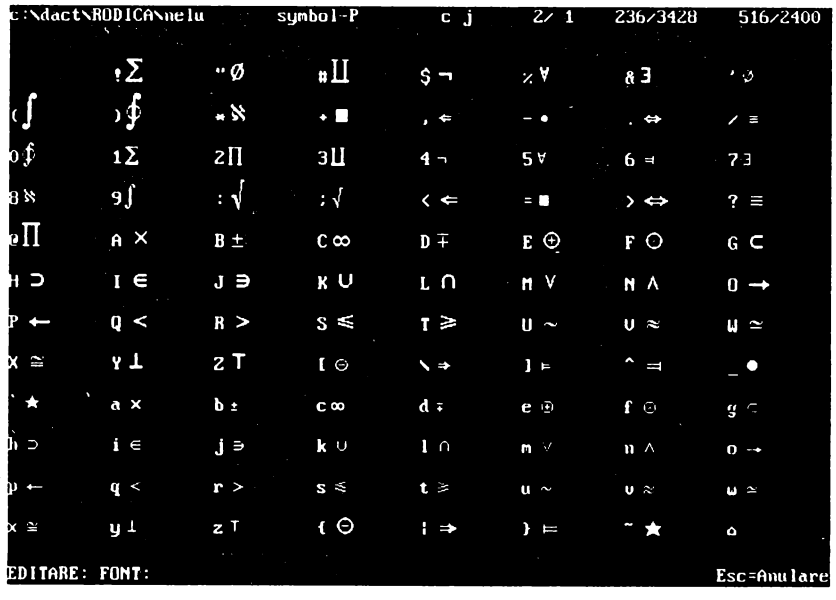


Figura 4

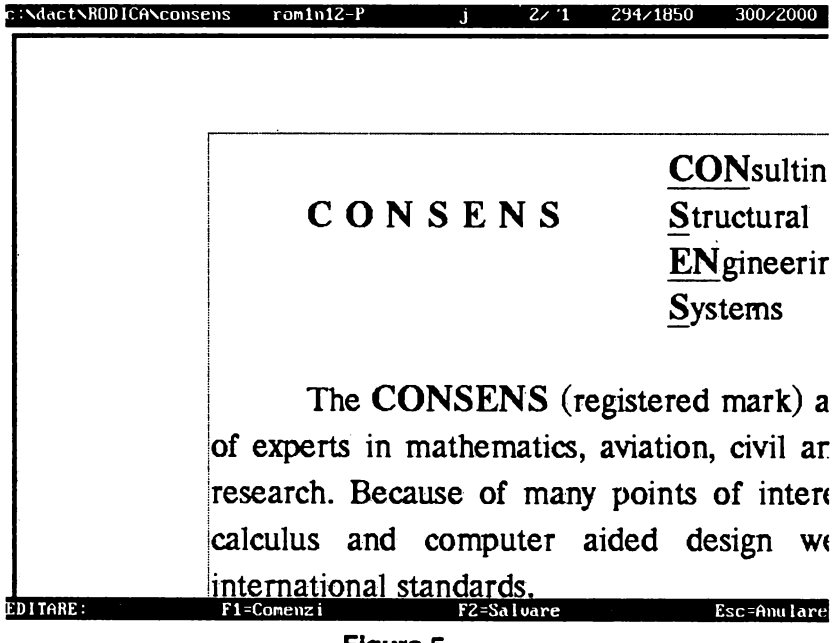


Figura 5

(*drive* + *path* + *name*, în cazul nostru C:\DACT\RODICA\nelu);

b) fontul curent, primul font utilizat fiind rom1n12-P (cu semne româneşti, normale, de 12 puncte tipografice înălţime);

c) caracteristicile setate: bolduire (b), centrare de rînd (c), justify (j), subliniere (s);

d) rîndul curent şi pagina curentă;

e) ordonata curentă şi lungimea paginii (în pixeli);

f) abscisa curentă şi lăţimea paginii (în pixeli).

Iar linia de mesaje de jos va conţine permanent informaţia privind starea curentă a regimului de lucru (în cazul nostru EDITARE) şi modul de folosire a tastelor speciale.

Apăsînd de exemplu tasta **F1**, după ce în prealabil am schimbat fontul (selectînd fontul symbol-P), vom obţine ecranul surprins în *figura 3.* Aici ni se prezintă lista comenzilor ce pot fi obţinute prin combinaţii de taste. Majoritatea se obţin tastînd **Alt** împreună cu o tastă literală sugestivă.

Astfel, **Alt** + B înseamnă schimbarea caracteristicii de bolduire a textului; **Alt** + C însemană schimbarea caracteristicii de centrare; **Alt** + E înseamnă trecerea la exponent; **Alt** + M înseamnă schimbarea marginilor textului; **Alt** + R înseamnă deplasarea laterală spre dreapta (cu un număr de pixeli ce va trebui indicat). De asemenea, ţinînd seama că au fost selectate iniţial, o dată cu stabilirea stilului, un număr de 9 fonturi, trecerea rapidă la fontul al 3-lea (de exemplu) se efectuează tastînd **shift** + **F3** etc.

Tastînd **ESC** revenim în regimul normal de EDITARE. Apăsînd tasta **F2** vom obţine semnele grafice ale fontului curent în cazul nostru symbol-P), aşa ca în *figura 4.*

Utilizînd comenzile prezentate în *figura 3,* şi bineînţeles tastele obişnuite, se realizează de fapt "dactilografierea" brută a textului. Avantajul principal al acestui sistem constituie faptul că, în fiecare moment, operatorul "vede" pe ecran situaţia exactă a paginii "dactilografiate", aşa cum ar ieşi ea din imprimantă. Subliniem faptul că pe ecran nu apar secvenţe liniare de semne (combinaţii de texte şi comenzi) aşa cum procedează sistemele de editare cunoscute (**WORDSTAR, VENTURA**); dimpotrivă, comenzile nu apar pe ecran (ca semne), apare vizibil doar efectul lor. Despre acest lucru vom reveni ulterior.

Apăsînd tasta **F10** ieşim din regimul normal de "dactilografiere" şi intrăm într-un regim de "prelucrare" a textului introdus. Acesta se materializează în primul rînd în apariţia unui cursor de alt tip şi în modificarea liniei de jos de mesaje.

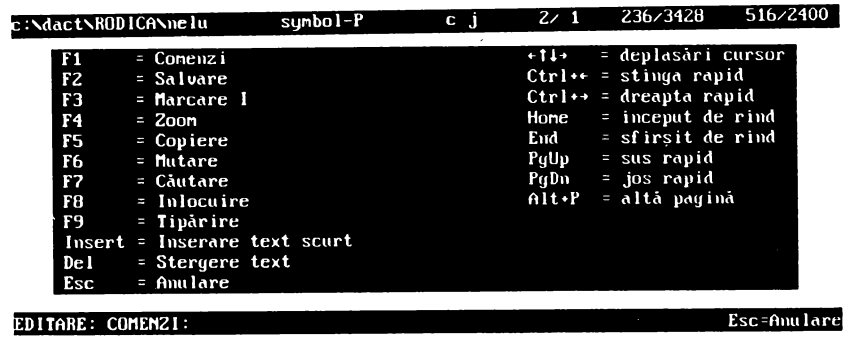


Figura 6

Să apăsăm din nou tasta **F10.** Vom constata apariţia unui mesaj "Se salvează? Da/Nu". Alegînd "Nu" se pierde, bineînţeles, tot ceea ce s-a lucrat. Alegînd "Da" se va cere un nou fişier intermediar, avînd numele dorit (în cazul nostru "nelu"), nume ce va fi adăugat listei fişierelor existente (vezi *figura 1*).

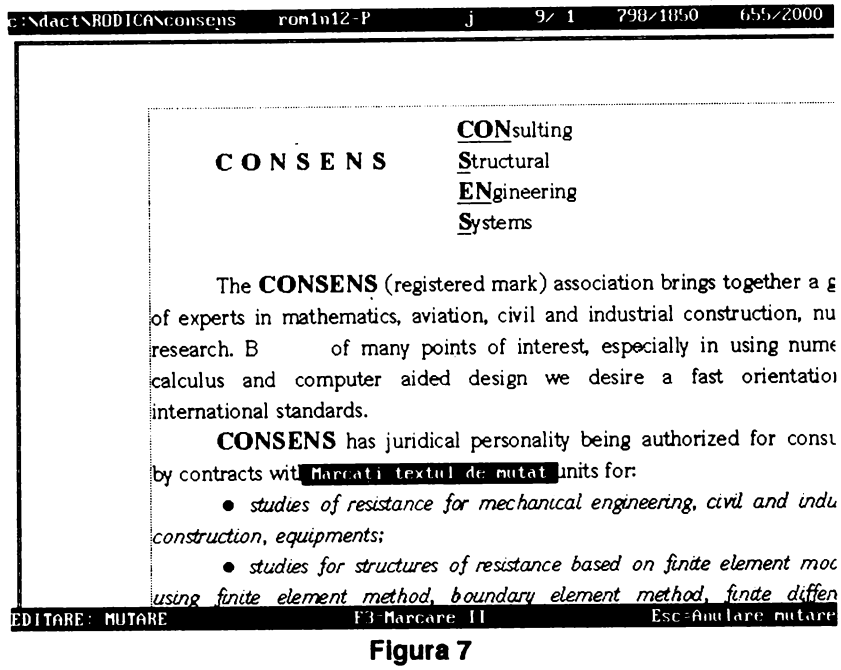


Figura 7

Să revenim, apăsînd de cîteva ori tasta **ESC**, la situaţia anterioară celei prezentate în *figura 1* şi să selectăm dintre fişierele existente unul, fie acesta fişierul "consens". Dintre comenzile disponibile în acest moment, cea mai interesantă este cea de editare (tasta **F1**), avînd acum sensul de "prelucrare" a conţinutului.

Pe ecran va apare imaginea primei pagini, începînd din stînga-sus (vezi *figura 5*). Lista comenzilor de "prelucrare" poate fi consultată în orice moment apăsînd tasta **F1** (vezi *figura 6* pentru un exemplu). Prezentăm acum efectul comenzii "Zoom" obţinută prin apăsarea tastei **F4**.

Aşadar, apăsînd tasta **F4** pe linia de jos vor apare mesaje specifice. Se va indica faptul că sîntem în regimul de

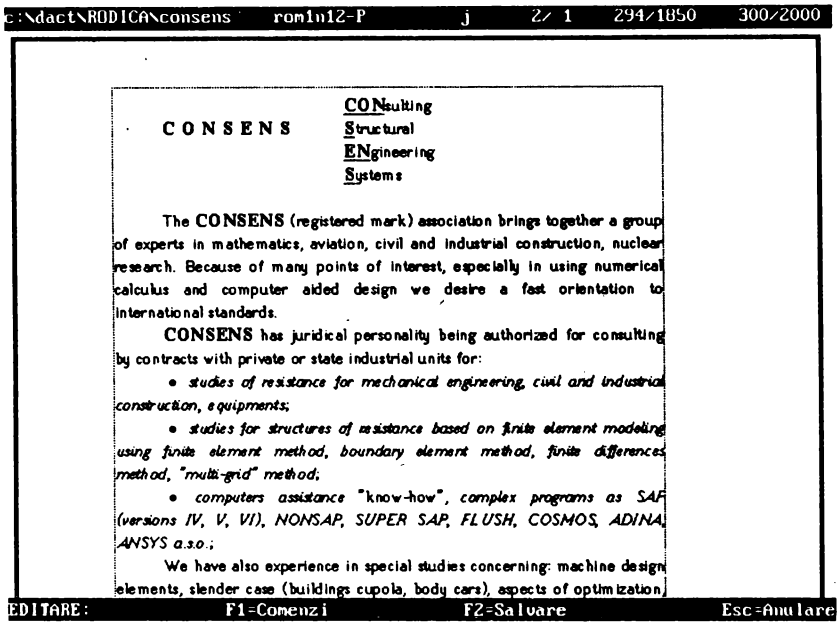


Figura 8

lucru EDITARE : ZOOM şi că avem de ales între trei posibilităţi de prezentare a paginii pe ecran: mare (vezi *figura 5*), mijlocie (vezi *figura 7*) şi mică (vezi *figura 8*).

Încheiem aici prima parte a prezentării sistemului de editare **Mattex,** cu menţiunea că *figura 7* prezintă o situaţie ce va fi explicată ulterior.

INIŢIERE ÎN PROGRAMARE - LIMBAJUL PASCAL (I)

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

Introducere

Fiind vorba de un curs adresat celor ce vor să se iniţieze în programare se poate pune întrebarea de ce a fost ales ca limbaj de iniţiere tocmai limbajul Pascal, cînd există limbaje mai "simple" cum este limbajul BASIC ori limbaje mai puternice şi mai sofisticate cum este limbajul PROLOG. Răspunsul este destul de simplu : este singurul limbaj de programare larg răspîndit şi care a fost proiectat special pentru învăţarea programării. Autorul acestui limbaj (Niklaus Wirth - profesor la Universitatea Tehnică din Zürich) s-a străduit să realizeze un limbaj de programare cît mai pur, care să "sileasca" programatorul "să se poarte frumos", nepermiţîndu-i acestuia să scrie programe dezordonate. Evident însă fantezia şi puterea omului poate să învingă orice bariere şi se pot scrie programe la fel de proaste în Pascal ca şi în orice alt limbaj de programare.

Limbajul Pascal este deja un limbaj vechi, fiind un produs al anilor '70. Toate implementările semnificative ale acestui limbaj pornesc de la ceea ce se numeşte "limbajul Pascal standard", a cărui descriere apare în aşa-numitul "Raport Revizuit" publicat de către Niklaus Wirth în 1974). Implementările existente pentru diferite calculatoare au fost realizate prin extinderea limbajului cu facilităţi legate de sistemul de operare şi hardware-ul specific calculatoarelor respective.

Implementarea cea mai spectaculoasă a acestui limbaj aparţine firmei BORLAND, care prin compilatorul TURBO Pascal a revoluţionat pur şi simplu domeniul, atît prin performanţele intrinseci ale compilatorului (timp de compilare, memorie ocupată de către compilator) cît şi prin uşurinţa de utilizare a acestuia.

Limbajul TURBO Pascal este disponibil în versiunea 3.0 sub sistemul de operare CP/M pe microcalculatoare bazate pe microprocesorul Z80 şi în versiunile de la 3.0 pînă la 5.5 sub sistemul de operare MS-DOS pe microcalculatoare compatibile IBM. Din acest motiv pentru prezentarea iniţială a limbajului vom considera pentru exemplificări limbajul TURBO Pascal versiunea 3.0.

Intenţia cursului pe care îl începem acum nu este de a produce un text exhaustiv referitor la limbajul Pascal ci de a iniţia cititorul în arta programării utilizînd limbajul de programare Pascal.

Elementele de bază ale limbajului Pascal

Cum arată un program?

Să vedem întîi cum arată un program banal scris în limbajul de programare Pascal :

```
program unu;
begin
 write('Felicitari, ati reusit !')
end.
```

În primul rînd ar fi bine să încercaţi să executaţi acest program pentru ca să vă convingeţi ca stăpîniţi operaţiile de "bucătărie" legate de compilatorul TURBO Pascal, aceasta însemnînd că ştiţi să introduceţi textul programului, să îl compilaţi şi să îl lansaţi în execuţie.

Să urmărim puţin programul prezentat. Se observă că programul apare sub forma unui text care se termină cu punct. Programul are cîteva elemente pe care le vom regăsi la orice program Pascal :

- *titlul programului* — în exemplul considerat titlul este "unu". Numele programului este semnificativ numai pentru documentarea programului;
- cuvintele "*begin*" şi "*end*" fac parte din categoria cuvintelor cu semnificaţie predefinită, denumite cuvinte rezervate (cheie) ale limbajului şi sînt utilizate în acest context pentru a delimita corpul executabil al programului;
- *corpul programului* este conţinut între cuvintele cheie begin şi end şi este, pentru exemplul considerat, format dintr-o singură instrucţiune.

Un program real va conţine şi alte componente pe care le vom prezenta mai tîrziu.

După cum sperăm că aţi ghicit, efectul execuţiei acestui program constă din afişarea pe ecran a textului :

```
Felicitari, ati reusit !
```

Să ne propunem acum să modificăm acest program pentru a afişa textul :

```
Felicitari, ati reusit
sa executati programul
```

Programul corespunzător este :

```
program doi;
begin
 writeln('Felicitari, ati reusit');
 write('sa executati programul')
end.
```

Se observă că în acest caz corpul programului este format din două instrucţiuni. Amîndouă instrucţiunile realizează afişarea textului conţinut între caracterele apostrof. Prima instrucţiune realizează în plus şi trecerea

la începutul liniei urmatoare, astfel încît execuţia următoarei instrucţiuni să determine afişarea pe o linie nouă.

În exemplul considerat apare şi un nou element de punctuaţie şi anume caracterul ";" , care este utilizat pentru separarea instrucţiunilor. În primul exemplu considerat acest caracter nu a fost necesar deoarece în corpul programului nu există decît o singură instruciune.

Programul doi poate să fie scris şi sub forma :

```
program doi;begin writeln('Felicitari, ati reusit');
write('să executaţi programul') end.
```

Pentru compilatorul limbajului Pascal cele două forme sînt echivalente dar, evident, prima formă de scriere este mai uşor de urmărit, deci este de preferat.

Dacă profesorul Niklaus Wirth ar vedea exemplele noastre, nu ar fi foarte mulţumit pentru că nu am utilizat un element important din punct de vedere al documentării programelor, şi anume nu am utilizat nici un comentariu. Să corectăm această greşeală :

```
{ Program de demonstraţie pentru
afişarea unui text simplu }

program doi;
begin
  { afisare pe prima linie }
  writeln('Felicitari, ati reusit');
  { afisare pe urmatoarea linie }
  write("sa executati programul')
end.
```

Se observă că un comentariu este un text cuprins între acolade. Evident, în exemplul considerat utilizarea comentariilor este exagerată, mai ales că nu oferă informaţii suplimentare faţă de ceea ce rezultă imediat din textul programului. Ca regulă de comportare, ori de cîte ori avem impresia că am fost "deştepţi" probabil că este bine să introducem un comentariu care să explice notaţia, construcţia sau algoritmul utilizat.

Tipuri simple de date

În general se spune că un program realizează o prelucrare de informaţie. Termenul de prelucrare trebuie să fie considerat într-un sens foarte general. De exemplu, chiar în programele prezentate anterior, prelucrarea este suferită de texte (care reprezintă în acest caz informaţia) şi constă din afişarea acestora.

Informaţiile pe care le prelucrează un program sînt în general numere, texte, valori logice. Prelucrările cele mai simple suferite de către aceste informaţii sînt reprezentate de operaţii definite asupra acestora. Evident, în funcţie de tipul informaţiei, regulile de execuţie a operaţiilor pot să fie foarte diferite. De exemplu, dacă pentru toata lumea este clar ce înseamnă operaţia de adunare definită asupra numerelor, operaţia de adunare definită asupra textelor poate să însemne de exemplu concatenarea acestora, dar pot să fie imaginate şi alte definiţii. De asemenea operaţia de adunare asupra valorilor logice adevărat şi fals poate să corespundă unei operaţii de tip "*sau logic*", dar poate să corespundă şi operaţiei "*şi logic*". Din acest motiv în majoritatea limbajelor de programare se utilizează noţiunea de tip de date. Specificarea unui tip de date presupune descrierea valorilor pe care le pot lua datele de tipul respectiv şi specificarea operaţiilor şi semnificatiilor acestora.

Limbajul Pascal a fost prevăzut cu o serie de tipuri de date standard (predefinite) dar şi cu un mecanism prin care programatorul poate să construiască tipuri noi pe baza tipurilor predefinite.

Tipurile simple de date prevăzute de către limbajul Pascal standard sînt : **integer, real, char, boolean.** În afară de aceste tipuri în limbajul TURBO Pascal ca tip simplu de date este prevăzut şi tipul **byte**.

Tipurile **integer, real** şi **byte** sînt tipuri de date care descriu numere, deci le putem denumi tipuri de date numerice. Diferenţele dintre aceste tipuri sînt date de modul de reprezentare în calculator. Acest mod de reprezentare dictează ordinul de mărime al valorilor posibile pentru datele de tipul respectiv, iar în cazul tipului **real** şi precizia reprezentării acestor valori. Modul de reprezentare şi particularităţile ce decurg din acesta sînt funcţie de compilatorul utilizat. Astfel, pentru limbajul TURBO Pascal datele de tip **integer** sînt numere intregi cu valori în intervalul -32768 pînă la 32767; datele de tip **byte** sînt numere naturale cu valori în intervalul 0 pînă la 255. Valorile de tip **real** se încadrează în intervalul -10^{38}, 10^{38}, iar cel mai mic număr real pozitiv are ordinul de mărime 10^{-38}. Indiferent de ordinul de mărime al numărului el poate să fie reprezentat cu maximum 11 cifre semnificative exacte. Aceasta înseamnă că numerele 0,0012345678901 şi $123{,}45678901 \times 10^{20}$ pot să fie reprezentate exact utilizind limbajul TURBO Pascal (nu explicăm acum motivul, rugînd cititorul să ne creadă pe cuvînt - ...nu cerceta aceste legi ...). Să executăm de exemplu şi următorul program :

```
program trei;
begin
 writeln('Ce părere aveţi despre rezultatele afişate ?');
 writeln(3);
 writeln(-77);
 writeln(32767);
 writeln(3.14);
 writeln(0.0012345678901);
 writeln(0.00123456789012);
end.
```

Ca efect al execuţiei acestui program pe ecran se afişează :

```
Ce părere aveţi despre rezultatele afişate ?
3
-77
32767
  3.1400000000E+00
  1.2345678901E-03
  1.2345678901E-03
```

LIMBAJUL PASCAL (I)

Se observă că modul de afişare depinde de tipul datelor ce se afişează. Astfel, datele întregi apar în notaţia matematică uzuală, cu condiţia să se încadreze în domeniul tipului respectiv. Dacă se încearcă utilizarea într-un program a unei instrucţiuni

```
write(60000)
```

compilatorul va semnala eroarea corespunzătoare utilizării unei constante întregi prea mari. Datele de tip real se afişează în aşa numita formă normalizată (cu mantisă şi exponent), cu numărul maxim de cifre semnificative (11). Evident, o astfel de notaţie nu este potrivită în orice situaţie. Limbajul TURBO Pascal oferă posibilitatea de a specifica cu puţin efort şi alte forme de afişare pentru numere. Să executăm de exemplu următorul program :

```
program patru;
begin
  writeln('Numere întregi');
  writeln(3 : 7);
  writeln(-77 : 7);
  writeln(-77 : 1);
  writeln('Numere reale');
  writeln(-3.141 : 25);
  writeln(3.141 : 8);
  writeln(3.141 : 9);
  writeln(3.141 : 2);
  writeln(-3.141 : 2 : 0);
  writeln(-3.141 : 1 : 0);
  writeln(3.141 : 6 : 4);
  writeln(3.141 : 4 : 2);
  writeln(3.141 : 3 : 3);
  writeln(3.141 : 3 : 5);
end.
```

Ca efect al execuţiei pe ecran se afişează :

```
Numere intregi
      3
    -77
-77
Numere reale
          -3.1410000000E+00
3.14E+00
3.141E+00
3.1E+00
-3
-3
3.1410
3.14
3.141
3.14100
```

Aţi remarcat efectul notaţiilor de forma " : intreg" şi respectiv " : intreg1 : intreg2" ?

Notaţia " : intreg" precizează lungimea zonei în care se afişează numărul, aliniat la dreapta, cu condiţia ca această zonă să fie suficientă pentru afişarea întregală a numărului, dacă acesta este de tip întreg, sau a reprezentării cu exponent, cu minimum 2 cifre semnificative, dacă numărul este de tip real. Se observă că în caz contrar zona este extinsă automat la dimensiunea necesară afişarii.

Notaţia " : intreg1 : intreg2" se utilizează pentru afişarea fără exponent a numerelor de tip real. Această notaţie precizează atît lungimea zonei în care se afişează numărul ("intreg1"), cît şi lungimea zonei destinate afişarii părţii subunitare ("intreg2"). În cazul în care lungimea zonei de afişare este insuficientă, ea este extinsă automat, astfel încît să poată fi afişată integral partea întreagă a numărului, urmată de numărul de zecimale cerut.

Dacă zona destinată afişarii unui număr are o lungime mai mare decît cea necesară afişarii conform ordinului său de mărime, se va face o completare la stînga cu blancuri.

Pentru a uşura identificarea numerelor afişate, în exemplele anterioare s-a făcut afişarea acestora pe linii separate. Aceasta nu este însă unica posibilitate de afişare. Vă rugăm să încercaţi să executaţi şi următorul program:

```
program cinci;
begin
  write('Afisare urită :');
  write(1);
  write(2);
  write(-3);
  write(-4.12);
  writeln; trecere la linia următoare
  writeln;
  write('Afisare normala : ');
  write(1:3);
  write(2:3);
  write(-3:3);
  write(-4.12:6:2);
end.
```

Ar trebui să obţineţi următorul rezultat :

```
Afisare urita :12-3 -4.1200000000E+00

Afisare normala :  1  2 -3 -4.12
```

Se observă din nou diferenţa dintre writeln (afişare urmată de trecere la începutul liniei următoare) şi write (afişare cu păstrarea poziţiei). De asemenea se remarcă posibilitatea realizării unei spaţieri între numerele afişate pe acelaşi rînd, prin specificarea unei lungimi mai mari decît cea necesară.

Dacă veţi executa şi programul şase veţi constata că are acelaşi efect ca şi programul cinci, deşi este scris mai compact.

```
program şase;
begin
  writeln('Afisare urita :', 1, 2,- 3,- 4.12);
  writeln;
  write('Afisare normala : ', 1 : 3, 2 : 3,- 3 : 3,
                              -4.12 : 6 : 2);
end.
```

Rezultă deci că într-o instrucţiune write sau writeln se poate specifica o listă de informaţii (numere sau texte, separate prin virgule) care se vor afişa pe acelaşi rînd.

Asupra datelor numerice se pot executa operaţii aritmetice. În limbajul Pascal operaţiile aritmetice sînt specificate cu ajutorul următoarelor notaţii :

+	adunare
–	scădere
*	înmulţire
/	împărţire cu rezultat de tip real
div	împărţire cu rezultat de tip întreg (operaţie definită în cazul limbajului TURBO Pascal pentru tipurile **integer** şi **byte**
mod	restul împărţirii întregi (operaţie definită în cazul limbajului TURBO Pascal pentru tipurile **integer** şi **byte**)

Efectele execuţiei acestor operaţii sînt ilustrate de următorul program :

```
program şapte;
begin
  {operatii aritmetice "fara probleme"}
  writeln('2 + 3 = ', 2 + 3);
  writeln('2 – 3 = ', 2 – 3);
  writeln('2 * 3 = ', 2 * 3);
  writeln('2 / 3 = ', 2 / 3);
  writeln('2 / 3 = ', 2 / 3 : 5 : 2);
  writeln('2 div 3 = ', 2 div 3);
  writeln('2 mod 3 = ', 2 mod 3);
  writeln('6000. * 6000 / 6000 = ', 6000. * 6000 / 6000);
  writeln;
  {operatii aritmetice "cu probleme"}
  writeln('32767 + 2 = ', 32767 + 2);
  writeln('-20 - 32767  = ', -20 - 32767);
  writeln('6000 * 6000 / 6000 = ', 6000 * 6000 / 6000);
  writeln('1E-20 * 1E-20 = ',1E-20 * 1E-20);
end.
```

În urma execuţiei acestui program se va afişa următorul text :

```
2 + 3 = 5
2 – 3 = -1
2 * 3 = 6
2 / 3 =  6.6666666667E-01
2 / 3 = 0.67
2 div 3 = 0
2 mod 3 = 2
6000. * 6000 / 6000 =  6.0000000000E+03

32767 + 2 = -32767
-20 - 32767  = 32749
6000 * 6000 / 6000 =  3.4560000000E+00
1E-20 * 1E-20 =  0.0000000000E+00
```

Presupunem că partea de operaţii fără probleme nu necesită comentarea rezultatelor obţinute (nici măcar pentru operaţia 2 / 3). Să urmărim însă puţin cazul operaţiilor aritmetice "cu probleme". Primele trei cazuri ilustrează faptul că depăşirea domeniului de valori pentru numerele întregi, ca rezultat al unor operaţii aritmetice, conduce la obţinerea unor rezultate incorecte.

Este interesant să se compare valorile expresiilor :

6000. * 6000 / 6000 şi respectiv 6000 * 6000 / 6000

În primul caz rezultatul este cel corect deoarece rezultatul înmulţirii este un număr real a cărui valoare se încadrează în domeniul tipului de date real. În al doilea caz rezultatul înmulţirii depăşeşte domeniul tipului întreg şi din acest motiv rezultatul obţinut este incorect.

În afară de operaţiile aritmetice asupra tipurilor de date numerice, în limbajul Pascal pot să fie efectuate şi alte tipuri de operaţii. De asemenea asupra celorlalte tipuri simple de date (char, boolean) sînt definite operaţii specifice. Preferăm să amînăm prezentarea acestora pentru momentul în care vor fi fost descrise toate elementele necesare scrierii unor programe ilustrative edificatoare.

Să precizăm însă cum arată valorile posibile pentru tipurile de date char şi boolean. O dată de tip **char** are ca valoare codul unui caracter. Din nou, modul de reprezentare a unei astfel de valori în calculator este specific compilatorului utilizat (de exemplu pentru TURBO Pascal se utilizează codul ASCII) dar nesemnificativ în general din punctul de vedere al programatorului. Ceea ce este totuşi important de semnalat este faptul că valorile corespunzătoare caracterelor respectă ordinea lexicografică, astfel că 'A' < 'B', '0' < '1'. De asemenea pentru TURBO Pascal codurile cifrelor sînt mai mici decît codurile literelor iar literele mari au coduri cu valori mai mici decît literele mici (adică '9' < 'A' < 'a'). O dată de tip boolean poate avea una dintre cele două valori logice : adevărat, respectiv fals. Pentru specificarea acestor valori în limbajul Pascal se utilizează notaţiile : **true,** respectiv **false.** Modul de reprezentare în calculator depinde de compilator, dar se respectă regula **false < < true.**

PC-uri ŞI PROTECŢIE

Ion Muşlea

Conceperea PC-urilor ca sisteme de calcul "deschise" (programatorului), ce nu îngrădesc libertatea de acţiune a utilizatorului, a dus la o reacţie promptă şi oarecum previzibilă: producătorii de soft au fost obligaţi să-şi ia măsuri de precauţie pentru a nu fi "furaţi" (mai bine zis copiaţi pe gratis) sistematic. Altfel spus, au început să-şi protejeze programele.

Una din metodele curente de control a răspîndirii produselor soft este cea de creare a unor programe ce să nu poată fi copiate decît de un număr finit de ori. Acest lucru se realizează foarte simplu: undeva în INSTALL (programul ce se ocupă cu "instalarea" produsului: crearea directoarelor corespunzătoare, setarea unor configuraţii, copierea fişierelor corespunzătoare,...) se va iniţia o acţiune de scriere pe dischete sursă (fie în scopul de a incrementa un contor, fie pentru a memora o anumită configuraţie de biţi). În momentul în care s-a depăşit numărul permis de copii se va da un mesaj de avertizare, iar instalarea va fi oprită. Chiar dacă piratul "soft" a fost precaut şi şi-a protejat discheta sursă la scriere, "furtul" nu va reuşi pentru că producătorul are grijă să sisteze operaţia de instalare în caz de nereuşită a scrierii pe disc.

Deşi în aparenţă octetul "buclucaş" nu pare dificil de găsit, în practică acest lucru e greu realizabil. În primul rînd dimensiunea INSTALL-ului poate fi de sute de Ko, apoi acest program poate fi compus din mai multe module (scrise eventual în limbaje diferite) a căror interapelare continuă duce la exasperarea şi debusolarea autorului unei eventuale trasări. De asemenea putem avea mai multe condiţii de testat, octeţii de test putînd avea la rîndu-le poziţii variabile în fişier (după nişte algoritmi deosebit de alambicaţi); ca trucuri clasice putem adăuga şi programele care se automodifică sau se autodistrug. Dar acestea sînt doar cîteva idei, dintre care unele destul de "previzibile"; cînd însă colective întregi de cercetători de la IBM sau Microsoft îşi pun mintea la lucru, treburile pot lua o turnură nebănuită.

În continuare este prezentat un exemplu de program ce permite doar un număr finit de rulări. Dacă îl veţi completa cu partea de instalare şi-i veţi adăuga cîteva trucuri ingenioase, veţi obţine poate un INSTALL de "primă mînă".

```
DATA    SEGMENT  PARA          PUBLIC        'DATA'
nf      db  "a:/filename. ext", 0
nre     db  50
DATA    ENDS

CODE    SEGMENT  PARA          PUBLIC        'CODE'
START   PROC     FAR
        assume   cs: code,     ds: data
                 push          ds
                 xor           ax, ax
                 push          ax            ; INIT FOR RETURN
                 mov           ax, data
                 mov           ds, ax        ; INIT DS
                 mov           ah, 3dh
                 mov           al, 2
                 lea           dx, nf        ; OPEN FILE
                 int           21 h
                 push          ax            ; SAVE HANDLE
                 mov           ah, 42 h
                 mov           al, 00
                 pop           bx
                 push          bx            ; SET THE FILE POINTER
                 mov           cx, 0
                 mov           dx, 6
                 int           21 h
                 mov           ah, 3 fh
                 pop           bx
                 push          bx
                 mov           cx, 1
                 lea           dx, nre       ; GET NUMBER OF COPIES
                 int           21 h
                 mov           ah, 42 h
```

```
                    mov     al, 00
                    pop     bx
                    push    bx          ; SET THE FILE POINTER ON THE PREVIOUS
                    mov     cx, 0       ;  POSITION
                    mov     dx, 6
                    int     21 h
                    cmp     ds: nre, 54 ; TEST FLAG INDICATING NUMBER OF COPIES
                    jg      sf          ; TO MANY COPIES
                    inc     ds: nre     ; IT'S OK; INCREMENTATE NUMBER OF COPIES
retry:              mov     ah, 40h
                    pop     bx
                    push    bx
                    mov     cx, 1       ; STORE NUMBER OF COPIES
                    lea     dx, nre
                    int     21 h
                    jnc     retry       ; RETRY IF THERE IS AN ERROR
sf:                 pop     bx
                    mov     ah, 3 eh    ; CLOSE FILE
                    int     21 h
                    RET                 ; FAR return to DOS
START       ENDP
code        ends
END         START
```

O problemă aparte la acest capitol o constituie programele "documentaţie" de tipul BIOS-HELP sau NORTON-GUIDE. Un astfel de produs are de regulă două părţi distincte: fişierul de date şi programul ce îl face accesibil. Particularitatea lor? Pentru a fi ferite de priviri indiscrete, fişierele de date au fost concepute de aşa manieră încît la citirea cu WORDSTAR sau NEDIT vor fi vizibile doar cîteva rînduri ce conţin numele produsului şi al firmei producătoare. Astfel se ajunge la o situaţie inedită: DOS-ul indică un fişier cu o lungime de cîteva sute de Ko, iar editorul de texte îl tratează ca şi cum ar avea cîteva zeci de octeţi.

Dacă se încearcă vizualizarea conţinutului aceluiaşi fişier cu TYPE sau funcţia VIEW din NORTON COMMANDER (apelată cu **F3**) se va remarca un fapt interesant: cartuşul vizibil este urmat de un caracter "→", după care mai există alte caractere. Acest "→" are codul ASCII zecimal 26, este echivalent cu EOF (pentru editoarele de text mai sus citate) şi nu trebuie confundat cu "caracterul" CURSOR-RIGHT (cod ASCII zecimal 77).

Deci e clar: "clou"-ul afacerii este un simplu caracter. Din acest moment există două puncte de vedere: al celui ce protejează produsul şi al celui ce încearcă să-l "fure". În esenţă amîndoi au de făcut acelaşi lucru: să înlocuiască un caracter cu un altul (piratul un "→" cu un blanc, iar "producătorul" un cracter oarecare cu un "→"). Acest lucru se poate realiza foarte uşor folosind spre exemplu funcţiile DOS (se poate lucra atît din MASM ci şi din limbaje evoluate de tip TURBO C sau TURBO PASCAL). În esenţă structura de program înainte prezentată rămîne valabilă aşa că în continuare vom trata alte modalităţi de a realiza acelaşi lucru.

Cum din editoarele de text nu putem şterge acel caracter "→" pentru că ele îl tratează ca pe un EOF, am putea încerca acelaşi lucru cu funcţia EDIT din NORTON COMMANDER. Vom remarca însă că din cauza lungimii fişierelor, ele nu pot fi decît vizualizate, nu şi modificate. Deci din punct de vedere al piratului se pare că soluţia este una singură. Pentru producător ar mai fi însă încă o posibilitate: să scrie "cartuşul" cu un editor oarecare, după care să continue editarea cu funcţia EDIT (se apelează cu tasta **F4**) din NORTON COMMANDER. La acest apel se va remarca faptul că putem "continua" fişierul şi după acel caracter "→". După cum se vede această a doua soluţie este oarecum restrictivă prin faptul că nu se permite crearea unor fişiere la fel de lungi ca şi "sub" WORDSTAR sau NEDIT, şi în plus "spargerea" se poate face mult mai comod de către un eventual "hobbyst" care ştie trucul. Practic această modalitate de ascundere a informaţiei ţine mai mult de domeniul lui "Ştiaţi că...", dar oricum este o "ciudăţenie" ce merită a fi reţinută. Tot ca fapt divers ar mai fi de amintit faptul că fişierele create cu NEDIT se încheie cu un singur "→", pe cînd cele WORDSTAR cu o suită de "→".

Interfaţarea Turbo Pascal - MASM

Ion Muşlea

Noţiunea de UNIT

Pentru a putea realiza interfaţarea Turbo Pascal-ului cu limbajul de asamblare pentru 8086 e necesară în primul rînd prezentarea conceptului de UNIT.

Un UNIT se defineşte ca fiind o colecţie de constante, tipuri de date, variabile, proceduri şi funcţii. Altfel spus un UNIT este o bibliotecă ce poate fi folosită de orice program.

Un UNIT are două secţiuni: "INTERFACE SECTION" şi "IMPLEMENTATION SECTION". Prima va conţine toate structurile accesibile utilizatorului (de notat faptul că aici vor figura doar declaraţiile de proceduri şi funcţii, nu şi corpul acestora), iar cea de a doua va conţine structurile invizibile utilizatorului UNIT-ului împreună cu corpurile procedurilor şi funcţiilor descrise în prima secţiune.

De notat faptul că dacă UNIT-ul va folosi proceduri sau funcţii externe (cum ar fi cele scrise în asamblor, de exemplu), vor fi necesare directive de tipul

```
{$L filename}
```

care vor indica compilatorului numele fişierului .OBJ în care acestea pot fi găsite.

O altă observaţie ar fi legată de faptul că în "INTERFACE SECTION" nu se vor utiliza declaraţii de tip FORWARD.

În acest moment cel mai indicat lucru este prezentarea unui exemplu care să ilustreze cele afirmate.

Exemplu de folosire a UNIT-ului

Cu ajutorul editorului de texte al Turbo Pascal-ului se va crea fişierul MYUNIT. PAS care va avea următorul conţinut:

```
unit MyUnit;
interface
procedure increment (var variabila: integer);
implementation
procedure view (oVariabila: integer);
{această funcţie nu poate fi apelată din programul principal}
begin
 writeln( 'variabila vizualizată are valoarea ', oVariabila);
end;
procedure increment; { nu mai este necesară lista de parametri }
begin
 variabila:=variabila+1;
 view(variabila);
end;
end.
```

Apoi în meniul "COMPILE" vom seta opţiunea "Compile to disk" şi vom compila fişierul astfel obţinut tastînd Alt-F9. Rezultatul acestei acţiuni va fi obţinerea în directorul curent a unui fişier cu numele de MYUNIT. TPU.

În final vom mai crea încă un fişier, cu numele UNIT_USR.PAS, care va conţine următorul program principal:

```
program UnitUser;
uses MyUnit;
var var1:integer;
begin
 var1:=1;
 writeln(' inainte de incrementare variabila are valoarea ' , var1);
 increment(var1);
end.
```

Pe acesta îl vom compila şi executa normal, obţinînd în cele din urmă programul executabil UNIT USR.EXE care va folosi procedura INCREMENT definită în UNIT-ul MYUNIT.PAS.

Remarcăm faptul că în programul ce utilizează UNIT-ul se va face o declaraţie de tipul

```
uses MyUnit;
```

care este absolut necesară pentru ca Turbo Pascal-ul să poată identifica structurile declarate în UNIT.

Proceduri şi funcţii externe scrise în asamblor

Toate procedurile şi funcţiile scrise în limbaj de asamblare vor fi declarate în cadrul UNIT-ului ca fiind externe.

Exemplu:

```
procedure numara; external;
function cauta(var NumarCautat: integer): real; external;
```

Aceste proceduri şi funcţii vor fi scrise în limbaj de asamblare şi compilate cu MASM, obţinîndu-se astfel nişte fişiere .OBJ ce-i vor fi indicate compilatorului Pascal prin directive de tipul $L.

Observaţie:

Pentru fiecare fişier .OBJ vom avea nevoie de cîte o directivă $L.

De remarcat faptul că rutinele scrise în asamblor trebuie să lase nemodificaţi registrii BP, SP, SS, DS (deci în cazul în care modificarea lor este necesară în interiorul rutinei, ei vor fi salvaţi la început şi restauraţi în final).

Toate rutinele declarate externe în cadrul UNIT-ului trebuie să apară în segmentul de cod ("CODE SEGMENT") al unui fişier .ASM, unde vor fi declarate PUBLIC. Trebuie verificat ca rutinele scrise în asamblor să respecte modul în care au fost declarate în Pascal (NEAR/FAR după cum sînt declarate în "IMPLEMENTATION SECTION" sau în "INTERFACE SECTION"), să aibă acelaşi număr de parametri şi în plus tipurile parametrilor şi ale rezultatelor trebuie să coincidă.

Variabilele folosite în asamblor vor fi declarate în segmentul de date ("DATA SEGMENT") şi nu vor fi apelabile din Pascal sau din UNIT. În schimb orice variabilă, procedură sau funcţie declarată în UNIT poate fi accesată din asamblor dacă va fi declarată în fişierul .ASM ca fiind externă (deci cu EXTRN).

Pentru ca Turbo Pascal-ul să poată reuşi formarea UNIT-ului ce conţine directive $L e necesară respectarea cîtorva reguli:

a) - orice procedură sau funcţie externă va apărea într-un CODE SEGMENT;
   - orice variabilă privată din asamblor va fi declarată în DATA SEGMENT;
   - orice alt segment ce va apare în asamblor va fi ignorat;
   - segmentele de definiţie pot fi aliniate BYTE sau WORD şi nu trebuie să specifice vreo clasă de memorie.

b) - toate variabilele declarate în DATA SEGMENT vor fi neiniţializate (se va folosi "?"), căci Turbo Pascal-ul va ignora oricum iniţializările făcute la acest nivel.

Exemplu:

```
BUFFER DB 128 DUP (?)
COUNT DW?
```

c) - procedurile şi funcţiile declarate cu EXTRN nu pot fi apelate folosindu-ne de un OFFSET. Deci, dacă avem o declaraţie de tipul EXTRN PROC1:NEAR , un apel în stilul CALL PROC1+8 este incorect;
   -această restricţie nu e valabilă şi pentru variabile.

d) - la variabilele externe (definite cu EXTRN) nu se admit referiri la nivel de byte (deci nu pot fi folosiţi operatorii HIGH şi LOW).

Reprezentarea internă a datelor în Turbo Pascal

- INTEGER: -128..127 1 octet cu semn
  0..255 1 octet fără semn
  -32768..32767 1 cuvînt (2 octeţi) cu semn
  pentru LONGINT se vor folosi 2 cuvinte
- CHAR: 1 octet fără semn
- BOOL : 1 octet conţinînd valoarea 0 sau 1
- ENUMERATED: 1 octet fără semn (pînă la 25 enumerări inclusiv)
  1 cuvînt fără semn (mai mult de 25 enumerări)
- FLOATING POINT:
  real 6 octeţi
  single 4 octeţi
  double 8 octeţi
  extended 10 octeţi
  comp 8 octeţi
- POINTER: 2 cuvinte (OFFSET în LOW, SEGMENT în HIGH)
- STRING : pentru N caractere vom avea N+1 octeţi, dintre care primul va conţine lungimea STRING-ului (deci N)
- SET: cel mult 256 octeţi
- ARRAY: locaţii succesive de elemente componente; la cele multidimensionale cea mai din dreapta dimensiune creşte prima.
- RECORD : locaţii succesive de elemente componente
- FILE : se reprezintă ca RECORD-uri.

Convenţii de apel

Transmitera parametrilor procedurilor şi funcţiilor se va face prin stivă. Înainte de apelul rutinei parametrii vor fi puşi în stivă în ordinea declarării.

Există două tipuri de parametri: transmişi prin referinţă (în stivă va fi pus un *pointer* la acea variabilă sau transmişi prin valoare (în stivă va fi pusă valoarea lor în acel moment).

La rîndul lor parametri sînt de două tipuri:

a) "VARIABLE PARAMETERS": se transmit întotdeauna prin referinţă;

b) "VALUE PARAMETERS": în funcţie de tip şi de lungime se vor transmite fie prin valoare, fie prin referinţă. Regula generală spune că în cazul în care au o lungime de 1,2 sau 4 octeţi se vor transmite prin valoare, altminteri prin referinţă. Deci, să vedem cum vor fi transmise diversele tipuri de date:

- INTEGER: 1,2,4octeţi (în ultimul caz prima dată se va pune pe stivă cuvîntul mai semnificativ);
- CHAR: 1 octet fără semn;
- BOOL: 1 octet cu valoare 0 sau 1;
- ENUMERATED: 1 octet fără semn (max. 256 enumerări);
  1 cuvînt fără semn (peste 256 enumerări);
- REAL e o EXCEPŢIE; 6 octeţi puşi pe stivă (cel mai semnificativ primul);
- POINTER: 2 cuvinte (SEGMENT-ul e primul);
- STRING: pointer la o valoare;
- SET: pointer la 32 de octeţi;
- ARRAY/RECORD: dacă au 1,2,4 octeţi se pun pe stivă, altfel prin pointer la valoare.

Rezultatele funcţiilor vor fi returnate după cum urmează:

- dacă sînt de tip ordinal (INTEGER, BOOL, ENUMERATED) vor fi returnate în regiştri (1 octet în AL, 1 cuvînt în AX, un dublu cuvînt în DX(HIGH-ORDER):AX(LOW-ORDER);
- dacă sînt de tip REAL în regiştri DX(HIGH-ORDER), BX(MIDDLE-ORDER) şi AX(LOW-ORDER);
- POINTER-ii vor fi returnaţi în DX (SEGMENT) şi AX (OFFSET);
- STRING-urile: apelantul va transmite la apel un *pointer* la o locaţie de memorie temporară, la care adresa va regăsi apoi valoarea de string "returnată"; de remarcat că funcţia nu trebuie să schimbe *pointer*-ul.

Tot acum mai trebuie discutată problema tipului de apel: NEAR sau FAR (controlul e predat unei rutine din acelaşi segment sau din altul). Un CALL de tip NEAR va pune pe stivă 2 octeţi (OFFSET), pe cînd unul FAR va încărca stiva cu 4 octeţi (SEGMENT şi OFFSET). Vom reaminti faptul că rutinele declarate în INTERFACE vor fi de tip FAR, iar cele din IMPLEMENTATION vor fi NEAR.

De subliniat şi faptul că orice procedură sau funcţie "încuibărită" ("NESTED" - adică declarată în interiorul alteia) va fi întotdeauna tratată drept NEAR. La apelul unei astfel de rutine compilatorul va genera înaintea lui CALL un PUSH BP, transmiţînd astfel un parametru adiţional. Astfel rutina "încuibărită" va putea folosi variabilele locale ale apelantului.

Observaţie

Rutinele "încuibărite" nu pot fi declarate externe!

Exemplu:

Se începe prin a se crea cu ajutorul unui editor de text fişierul **proc_asm.asm** care va avea următorul conţinut:

```
data    segment     word    public
extrn VARREALA:word         ;e o variabilă reală externă
data    ends
code    segment     byte    public
        assume      cs: code
extrn TIPREAL:far,NUMĂR:near   ;declarare de rutine externe
        public      subr1   ;declararea unei rutine folosite de alt modul
subr1   proc        far
        push        ds      ;salvarea registrului DS
        mov         ax,data ;  ;pregătire DS pentru accesarea segm. de date
        mov         ds,ax   ;a nu se uita ca DS nu suportă încasări imediate
        mov         bx,sp

;urmează accesarea parametrului rutinei

        mov         ax,ss:[bx+6]  ;în mod normal ar fi trebuit să avem bx+4,
        mov         dx,ss:[bx+8]  ;bx+6, bx+8 dar prin punerea pe stivă a lui DS
        mov         cx,ss:[bx+10] ;e necesară o corecţie cu 2 octeţi;
;se va folosi procedura TIPREAL (scrisa în Pascal) pentru a vizualiza valoarea parametrului transmis
        push        cx
        push        dx                                          →
```

```
                push        ax
                call        TIPREAL
        ;se apelează funcţia NUMAR (scrisă tot în Pascal care va returna un număr real (se folosesc
        ;regiştrii DX, BX, AX)
                call        NUMĂR
        ;variabilei VARREALA i se atribuie valoarea returnată de funcţia NUMAR
                mov         ds:VARREALA,ax
                mov         ds:VARREALA+2,bx
                mov         ds:VARREALA+4,dx
        ;se vizualizează valorea returnată de funcţia NUMAR pentru a o putea compara cu cea pe care o
    ;se ;are variabila VARREALA
                push        dx
                push        bx
                push        ax
                call        TIPREAL
        ;se restaurează registrul BS
                pop         ds
                ret
        subr1   endp
        code    ends
                end
```

O dată creat, acest fişier se va compila cu MASM (testaţi MASM şi apoi RETURN; la prima întrebare testaţi PROC.ASM, iar la următoarele doar RETURN).

Apoi, se va crea fişierul MYUNT2.PAS care va conţine:

```
unit myunit2;
interface
var VARREALA:real;
procedure subr1(x:real);
procedure TIPREAL(x:real);
implementation
procedure subr1; external;
{$1 proc_asm}
procedure TIPREAL;
begin
 writeln('procedure TIPREAL tipareste valoarea ',x);
end;
function NUMAR:real;
begin
 NUMAR:=876.54321;
end;
end.
```

Acesta va fi compilat cu Turbo Pascal, obţinîndu-se astfel fişierul MYUNIT2.TPU.

În sfîrşit se va crea fişierul TEST1.PAS cu conţinutul:

```
program test1;
uses myunit2;
begin
 varReala:=1.23;
 writeln('initial variabila varReala = ',varReala);
 subr1(12.3456);
 writeln('in final varReala = ',varReala);
end.
```

Acest ultim fişier se va rula cu Ctrl-F9 şi rezultatele sale vor demonstra corectitudinea acestui scurt program demonstrativ.

Toate exemplele din acest articol au fost rulate folosind Turbo Pascal versiunea 5.0 şi MASM versiunea 5.10; autorul articolului nu e răspunzător de eventualele probleme ce pot apărea datorită folosirii altor variante de compilatoare.

I N F O R M A T I C Ă

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

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

ORGANIZEAZĂ PERMANENT

CURSURI

CALIFICARE, FORMARE, SPECIALIZARE,

PERFECŢIONARE

PENTRU PERSOANELE CU STUDII MEDII

PE MINI-MICROCALCULATOARE (8-16 BIŢI)

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

— Formare analişti-programatori asistenţi.

— Operatori iniţiere în informatică, lucrări secretariat şi dactilografie.

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

— Programare în limbaj BASIC.

— Programare în limbaj dBASE.

— Utilizare microcalculatoare compatibile IBM-PC.

La cursuri pot participa şi persoanele cu studii superioare.

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

Informaţii la telefon: 988/73534.

LIMBAJUL MAŞINĂ Z80 PE CALCULATOARELE COMPATIBILE SPECTRUM (II)

Bogdan Georgescu
Bogdan Iordănescu

Aşa cum am promis în numărul trecut, vom prezenta un program care foloseşte întreruperile Z80 şi care are un efect apropiat de cel al unui virus informatic, lipsindu-i evident protecţia şi capacitatea de autoreproducere. Pe SPECTRUM, de altfel, nu poate exista un virus în adevăratul sens al cuvîntului, datorită lipsei discurilor şi a conectării în reţea, deci a căilor de transmitere (acest lucru ar fi posibil pe variantele extinse, 48K cu Microdrive şi Interfaţa 1 sau +2, respectiv +3). Am asemănat programul pe care îl vom prezenta cu un virus deoarece, odată iniţializate întreruperile, se îngreunează lucrul cu calculatorul. Astfel, la fiecare 32 de întreruperi apărute în sistem şi detectate de program (32 de baleieri ale ecranului efectuate pe sincrogenerator), se efectuează cîte o translatare (scroll) a întregului ecran spre dreapta sau spre stînga cu un caracter.

Se vor executa patru scroll-uri spre dreapta, urmate de patru spre stînga ş.a.m.d. Datorită faptului că se deplasează numai punctele pe ecran, fără informaţia de culoare (atribute), efectul este foarte derutant, scrierea unei linii în **BASIC** devenind aproape imposibilă, deşi între două întreruperi succesive, deci în majoritatea timpului, interpretorul lucrează, aşteptînd introducerea instrucţiunilor. Menţionăm că este afectată numai imaginea, respectiv memoria ecran, în rest instrucţiunile vor fi memorate corect, odată introduse; astfel, un "RANDOMIZE" va fi memorat corespunzător, chiar dacă pe ecran apare ceva de genul "OMIZE"...

Prezentăm în continuare programul, urmînd ca după aceea să explicăm modul în care subrutinele de scroll (singurul lucru nou faţă de prezentarea din numărul trecut) rulează pentru a da rezultatul care se vede pe ecran.

Aşadar...

```
                ORG     #FF10           ; 65296 zecimal.
                LD      HL,#FE00        ; se creează tabela conţinînd octeţi cu valoarea
                LD      A,#FC           ; #FC, de lungime 257 baiţi începînd la adresa
                LD      B,0             ; #FE00 (65024), deci subrutina va fi la adresa
LABEL1          LD      (HL),A          ; #FCFC.
                INC     HL
                DJNZ    LABEL1
                LD      (HL),A
                XOR     A               ; valoarea 0 în acumulator.
                LD      (#FFFE),A       ; contor număr întreruperi şi număr scroll-uri.
                LD      (#FFFF),A
                LD      A,#FE           ; în I octetul cel mai semnificativ al adresei tabelei.
                LD      I,A             ; trece în IM 2 în timp ce întreruperile sînt dezacti-
                DI                      ; vate, pentru a evita "probleme".
                IM      2
                EI
                RET

                ORG     #FCFC           ; 64764 în zecimal.
                DI                      ; dezactivează întreruperile şi salvează regiştrii pe
                PUSH    HL              ; durata subrutinei-propriu-zise.
                PUSH    DE
                PUSH    BC
                PUSH    AF
                LD      A,(#FFFF)       ; incrementează contorul de întreruperi.
                INC     A
                LD      (#FFFF),A
                AND     #1F             ; multiplu de 32 (#20)?
                CALL    Z,SUB           ; dacă da, apelează SUB.
                CALL    #02BF           ; oricum, citeşte tastatura.
                POP     AF              ; restabileşte situaţia iniţială şi revine acolo unde
                POP     BC              ; a fost întrerupt programul principal.
                POP     DE
```

→

```
        POP     HL
        EI
        RET

SUB     LD      A,(#FFFE)       ; incrementează contor scroll-uri.
        INC     A
        AND     7               ; modulo 7.
        LD      (#FFFE),A
        CP      4               ; dacă este între 4 şi 7 inclusiv, scroll left, altfel
        JP      NC,LEFT         ; scroll right.
RIGHT   LD      HL,#57FE        ; ultimele două adrese display.
        LD      DE,#57FF
        LD      A,192           ; număr rînduri pixeli.
        LD      B,0             ; în BC valoarea 31.
LOOP1   LD      C,31
        LDDR                    ; deplasează un rînd.
        INC     HL              ; HL la începutul rîndului.
        LD      (HL),C          ; pune 0 acolo.
        DEC     HL              ; HL şi DE la sfîrşitul rîndului următor.
        DEC     HL
        DEC     DE
        DEC     A               ; face asta pentru 192 de rînduri.
        JP      NZ, LOOP1
        RET                     ; gata.

LEFT    LD      HL,#4001        ; primele două adrese display.
        LD      DE,#4000
        LD      A,192           ; în continuare aproape similar cu RIGHT dar în
        LD      B,0             ; sens invers.
LOOP2   LD      C,31
        LDIR
        DEC     HL
        LD      (HL),C
        INC     HL
        INC     HL
        INC     DE
        DEC     A
        JP      NZ,LOOP2
        RET                     ; gata.
```

Programul a fost asamblat cu GENS 3M21. Un exemplu de configurare a memoriei astfel încît să nu existe pericolul blocării claculatorului prin scriere peste anumite date din memorie este: se dă CLEAR 39999, apoi se încarcă GENS-ul la adresa 40000 şi se apelează cu RAND USR, se scrie programul, se asamblează şi apoi se rulează subrutina de iniţializări din BASIC, cu RAND USR 65296. Stiva a fost deplasată atît de jos din două motive: primul, pentru a nu fi afectată de execuţia programului în cod maşină, care rulează la adrese "înalte", acolo unde în mod normal este stiva, şi al doilea, pentru a putea readuce calculatorul la o comportare mai prietenoasă, prin NEW, fără a pierde din memorie nici programul, nici asamblorul, deci pentru a vă putea "juca", scriind propriile voastre variante de program. Revenirea la normal poate fi realizată şi prin reintrarea în GENS şi scrierea următoarei mici rutine:

```
        ORG     61000
        DI
        IM      1
        EI
        RET
```

Este esenţial ca instrucţiunea CLEAR să fie executată înainte de prima apelare a asamblorului, deoarece o greşeală în scrierea GENS-ului face ca acesta să funcţioneze incorect dacă între două apelări succesive se modifică poziţia stivei.

Şi acum, sîntem datori cu cîteva explicaţii cu privire la modul cum rulează subrutinele de scroll. Vom încerca să explicăm, cît mai puţin pretenţios posibil, cum se stochează în memorie punctele şi culorile de pe ecran.

După zona de memorie ROM dintre adresele 0 şi 16383 (3FFFH), urmează două zone fixe de memorie RAM: fişierul display între adresele 16384 şi 22527 inclusiv (4000H — 57FFH) şi atributele între 22528 şi 23295 (5800H — 5AFFH).

În fişierul display, fiecărui punct de pe ecran i se asociază un bit din memorie, punctul avînd culoarea *paper* dacă bitul este 0 şi *ink* dacă bitul este 1. Pe un octet încap deci 8 puncte alăturate. Pentru a arăta cum se

poate determina adresa octetului din care face parte un punct de pe ecran, vom face cîteva convenţii.

Să pornim de la lucruri cunoscute din BASIC, şi anume linia şi coloana asociate unui caracter de 8×8 pixeli. Vom folosi aceste două denumiri cu acelaşi sens, cu deosebirea că vor fi 24 de linii în loc de 22 (codul maşină are acces şi la ultimele două linii, rezervate altfel sistemului BASIC).

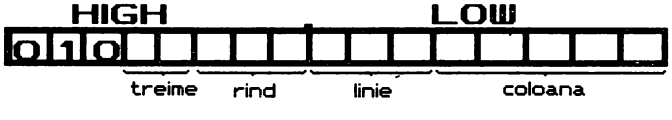
Cele 24 de linii, sînt grupate în 3 zone a cîte 8 linii, zone numerotate de sus în jos de la 0 la 2 şi numite, cum credeţi? — treimi. În aceste zone comportarea memoriei ecran este identică, deci să considerăm una din ele. Într-o treime se află 8 linii numerotate 0-7 de sus în jos şi 32 de coloane numerotate 0-31 care determină caracterele. Mergînd mai departe cu împărţirile, să împărţim şi fiecare linie în 8 rînduri de pixeli numerotate 0-7 tot de sus în jos şi cu asta puteţi răsufla uşuraţi, am terminat. Examinînd această convenţie, puteţi constata că specificînd treimea, linia, rîndul şi coloana în care se află un punct, avem unic determinat grupul de 8 pixeli în care se află acesta, deci şi octetul în care se află bitul asociat punctului.

Foarte bine, veţi spune, dar de ce să ne complicăm cu asemenea sistem de denumiri pe care deja le-am uitat, cînd putem scăpa foarte simplu: 32 coloane şi 192 (8×24) linii, şi avem acelaşi rezultat. Pentru a vedea motivele care ne-au împins la o asemenea activitate sinucigaşă, vă propunem să rulaţi următoarea rutină:

```
        LD      HL, 16384
        LD      BC, nr. octeţi
LOOP1   LD      (HL),#FF
        LD      DE,700          \
LOOP2   DEC     DE              | ciclu de
        LD      A,D             | tempo-
        OR      E               | rizare
        JP      NZ,LOOP2        /
        INC     HL
        DEC     BC
        LD      A,B
        OR      C
        JP      NZ,LOOP1
        RET
```

Rutina nu face altceva decît să încarce în memorie, la adrese consecutive, octetul #FF, echivalent cu înnegrirea tuturor celor 8 puncte. Ar fi de aşteptat ca punînd în loc de "nr.octeţi", valori ca 32, 256, 512, 1024, 2048, 4096, 6144 (nu mai mult), să vedem cum rîndurile de pixeli sînt înnegrite unul după altul, şi totuşi nu se întîmplă aşa: se ocupă întîi primul rînd din fiecare din cele 8 linii din treimea 0, apoi al doilea ş.a.m.d., iar cînd se ocupă toată treimea se trece la următoarea şi se face acelaşi lucru.

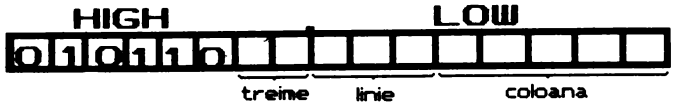
Pornind de la aceste observaţii, se poate scrie forma generală, binară, a adreselor din memoria display, şi anume:



unde fiecare căsuţă reprezintă un bit.

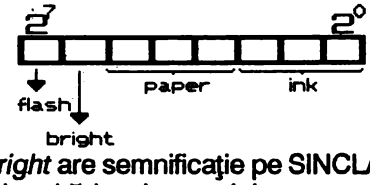
Această formă respectă regulile de incrementare stabilite mai sus şi este utilă în programare. Dar ce se întîmplă cînd în cei doi biţi rezervaţi treimii se scrie 11 binar, adică 3 zecimal? Adresele care rezultă nu mai fac parte din fişierul display, ci din zona atributelor, care stochează informaţia de culoare pentru fiecare caracter.

Relaţia dintre memorie şi ecran în cazul atributelor nu este aşa complicată, pentru că nu mai intervin rîndurile de pixeli, atributele referindu-se numai la caractere. Forma binară a adreselor de atribute este:



Notaţiile sînt cele stabilite anterior şi puteţi observa că în loc de treime şi linie s-ar fi putut folosi o singură notaţie, "linie extinsă" cu valori 0-23, deci aici parcurgerea ecranului se face continuu, fără salturi, ca la memoria display. Aceste salturi sînt, de fapt consecinţe ale structurii hardware şi, folosite bine, pot uşura, uneori, programarea.

Mai rămîne de clarificat un singur lucru: cum se pot stoca într-un singur octet toate informaţiile referitoare la un caracter, şi anume: *paper, ink, bright, flash*? Avînd în vedere că *paper* şi *ink* sînt numere între 0 şi 7, deci încap exact pe 3 biţi, şi că *flash* şi *bright* sînt 0 sau 1, deci li se poate asocia cîte un bit, nu avem nevoie de o prea mare putere de calcul pentru a constata că se ocupă exact cei 8 biţi ai unui octet. Ordinea în care aceşti biţi sînt aranjaţi în interiorul octetului este următoarea:



Bitul de *bright* are semnificaţie pe SINCLAIR, dar nu şi pe HC 85, datorită hardware-ului care nu are prevăzut un circuit pentru controlul acestui bit. Acesta este motivul pentru care instrucţiunea BRIGHT din BASIC nu funcţionează pe HC 85.

Mai trebuie să facem o observaţie şi cu privire la întreruperi: pe HC85, în ciclul de acceptare a întreruperii, octetul primit de microprocesor este întotdeauna #FF (codul instrucţiunii RST #38), deci ar putea fi evitată tabela de 256 de valori. Nu dispunem însă de scheme detailate ale SPECTRUM-ului, pentru a şti sigur dacă acest lucru este valabil şi la acest calculator. Am preferat să urmăm exemplul majorităţii producătorilor de programe şi să adoptăm soluţia tabelei, pentru mai multă siguranţă.

În numărul următor vom prezenta sistemul de input/output al SPECTRUM-ului, împreună cu un program aplicativ cu efecte mai puţin distructive şi puţină animaţie.

CALCULATOARE COMPATIBILE SPECTRUM — DESPRE ECRANE ŞI MEMORAREA LOR

Sorin Ciupa

Imaginea afişată pe monitor (sau televizor) corespunde cu informaţiile conţinute într-o zonă din memoria calculatorului, începînd cu adresa 16384 şi care are o lungime de 6912 bytes. Afişarea se face prin citirea continuă a acestor 6912 bytes, cu o frecvenţă de 50 de imagini/secundă şi transmiterea spre monitor (sau după modulare în înaltă frecvenţă spre televizor) împreună cu indicaţiile de sincronism specifice imaginii TV.

Orice număr cuprins între 0 şi 255 (1 byte) introdus în această zonă de memorie a calculatorului (16384...23295) va produce o modificare a imaginii, într-un anumit punct (încercaţi POKE adresă, byte).

Pentru a înţelege semnificaţia adreselor imaginii, să ne amintim că în cîmpul afişajului există 32 de coloane (0...31) şi 24 de linii (0...23), fiecare linie fiind constituită din 8 rînduri (0...7) de pixeli, puncte.

Definirea completă a cîmpului unui caracter se face prin 9 locaţii de memorie, 8 corespunzînd celor 8 rînduri de puncte pentru desen şi al nouălea pentru atribute. Transformarea în număr binar al byte-lui conţinut într-o adresă ce defineşte un număr binar al byte-lui conţinut într-o adresă ce defineşte un rînd, are ca rezultat grafica rîndului, respectiv pentru 0 corespunde fondul (PAPER, hîrtia) iar pentru 1 un pixel (un punct scris, INK, cerneala). Însumarea efectului celor 8 rînduri de grafică dă forma caracterului în cîmpul celor 8 × 8 puncte. Ultima adresă de memorie, a 9-a, ce se referă la cîmpul unui caracter, defineşte atributele, precizînd culoarea fondului, culoarea scrisului, dacă caracterul este mai luminos (BRIGHT), sau dacă pîlpîie (FLASH). De aici apare evident că tot cîmpul unui caracter, adică toate cele 8 rînduri de puncte, au aceeaşi culoare de fond, de scriere, sunt la fel de luminoase şi pîlpîie sau nu.

Cele 9 locaţii de memorie care definesc cîmpul unui caracter nu sînt cuprinse în adrese succesive, ci se găsesc dispersate în memorie, dispuse după anumite reguli. În tabelul anexat sînt indicate adresele ecranului pe linii şi rînduri şi pe coloane, pentru grafică şi atribute.

Partea grafică este cuprinsă în 3 sectoare de cîte 8 linii fiecare (0...7,8...15,16...23). Fiecare sector are 2048 bytes (=8 linii × 8 rînduri × 32 coloane), şi întreg ecranul are 6144 bytes (=2048 × 3 sectoare).

Stocarea se face pe sectoare, păstrîndu-se adresele în ordinea succesivă a coloanelor, începînd cu primul rînd de puncte al primei linii, apoi primul rînd de puncte al liniei a doua şi mergînd pînă la primul rînd de puncte al liniei a opta. Abia după aceasta se trece la al doilea rînd de puncte al primei linii, al doilea rînd de puncte al liniei a doua pînă la al doilea rînd al liniei a opta, etc. Cu stocarea ultimului rînd de puncte (al optulea) din linia a opta se termină stocarea primului sector şi va începe în acelaşi mod stocarea sectorului următor.

Sistemul acesta de memorare întreţesut se poate urmări la încărcarea în calculator a unei "coperţi" de joc.

Partea de atribute este memorată între adresele 22528 şi 23295, avînd un număr total de 768 bytes (32 coloane × 24 linii). Aici adresele urmăresc succesiunea coloanelor şi apoi a liniilor.

Un byte ce reprezintă un atribut, are următoarea semnificaţie pentru cei 8 biţi ai săi:

BIT 8	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1
FLASH	BRIGHT	CULOARE PAPER			CULOARE INK		

Biţii FLASH şi BRIGHT pot fi 1 pentru activat sau 0 pentru dezactivat.

Cei trei biţi pentru PAPER (6,5,4) şi pentru INK (3,2,1) pot aveaurmătoarele valori:

0 0 0	pentru culoarea neagră
0 0 1	pentru culoarea albastră
0 1 0	pentru culoarea roşie
0 1 1	pentru culoarea magenta (purpuriu)
1 0 0	pentru culoarea verde
1 0 1	pentru culoarea azuriu (cyan)
1 1 0	pentru culoarea galbenă
1 1 1	pentru culoarea albă

De exemplu, pentru a scrie cu cerneală galbenă pe hîrtie roşie, cu pîlpîire, vom determina valoarea byte-lui astfel:

FLASH	BRIGHT	PAPER	INK
128	64	32 16 8	4 2 1
1	0	0 1 0	1 1 0

128 + 16 + 4 + 2 = 150, valoare care introdusă în adresa atributului ne dă efectul dorit. (Operaţia se face uşor prin funcţia BIN, POKE adresa, BIN 10010110).

Pentru memorarea unui desen, a unui ecran, să zicem începînd de la adresa 50000, s-ar putea utiliza un mic program în BASIC, cam aşa:

```
   1  Adrese utilizate pentru realizarea desenului
1000
1010  REM memorează desenul de pe ecran
1020  FOR i=0 TO 6911
1030  POKE 50000 + i, PEEK (16348 + i)
1040  CLS : STOP
1100  REM readuce pe ecran desenul din memorie
1110  FOR i=0 TO 6911
1120  POKE 16384+i, PEEK (50000+i)
1130  NEXT i
1140  PAUSE 0
```

Este interesant de rulat acest program şi de cronometrat timpul său de execuţie.

ECRANE ŞI MEMORAREA LOR

În cod maşină, pentru mutări de zone din memorie, există o rutină compusă dintr-o suită de comenzi, care se execută cu o viteză mult mai mare. Aceste comenzi sînt:

NR.ADR.	COD MAŞINĂ	MNEMONICA	SEMNIFICAŢIE
....0	33 . .	LD HL, NN	NN=adresa de început, de unde se ia
....3	17 . .	LD DE, NN	NN=adresa de început, unde se pune
....6	1 . .	LD BC, NN	NN=lungimea zonei de memorie care se mută
....9	237 176	LDIR	Transferă conţinutul adresei HL în adresa DE, creşte HL şi DE cu o unitate, scade BC cu o unitate şi repetă pînă cînd BC=0
....11	201	RET	reîntoarcere din subrutină

Avînd ca bază această rutină în cod maşină, se propune un program cu care se pot memora pînă la 5 ecrane desenate cu programe speciale de desen (M draw, THE ARTIST, leonardo, SKETCH, artstudio etc.) sau prin BASIC (cu PLOT, DRAW şi CIRCLE). După desenare de pe bandă, se memorează în RAM şi pot fi chemate în afişaj în cadrul unui program.

PROGRAM : SCREEN STORAGE

```
10 CLS : PRINT "MEMORAREA ECRANELOR"'"Acest program permite stocarea în memoria calculatorului a unor desene, realizate prin programe de desen şi memorate pe bandă ca ecrane, pentru a fi chemate în timpul rulării unui program."
20 PRINT #1; "Cite ecrane doriti sa memorati ?(1...5)": PAUSE 0: LET n=CODE INKEY$-48: INPUT;:IF n<1 OR n>5 THEN GO TO 20
30 PRINT AT 8, 3; "Se va memora un ecran": IF n<>1 THEN PRINT AT 8,3;"Se vor memora ";n;" ecrane."
40 POKE 64000,n: LET sb=65367-n*6924: PRINT AT 11,3;"Programul BASIC trebuie să continuăm comanda"; BRIGHT 1;"CLEAR ";sb
50 PRINT '"Programul BASIC + variabilele pot avea maximum";sb-23755;"bytes.";#1;"Apasati oricare clapa.": PAUSE 0: CLEAR sb: LET n=PEEK 64000
60 FOR j=1 TO n: LET ie=65368-j*6924: PRINT "Ecranul ";j;" se poate chema prin;"'BRIGHT 1;"RANDOMIZE USR";ie:LET h=255-j*27:LET l=100-j*12
70 RESTORE: FOR i=0 TO 11: READ x: POKE ie+i,x: NEXT i
80 PRINT'"Porniti banda pentru a încarca ecranul ";j: LOAD ""CODE 16384,6912: RANDOMIZE USR ie: POKE ie+1,PEEK (ie+4): POKE ie+2, PEEK (ie+5): POKE ie+4,0: POKE ie+5, 64: CLS : NEXT j
90 PRINT "S-a terminat memorarea ecranelor si acest program poate fi sters. "'"Oricind puteti rechema ecranele din memorie cu comanda indicata. "'" Pentru salvarea pe bandă a zonei de memorie care contine ecranele va trebui sa dati comanda :"' BRIGHT 1;"SAVE""nume""CODE";65368-n*6924;",";n*6924
100 PRINT '"Acum apasati NEW pentru a sterge acest program.":STOP
110 DATA 33,0,64,17,/,h,1,0,27,237,176,201
9999 CLEAR : SAVE "SCREEN STORAGE" LINE 10: REM Program SSW '89
```

OBSERVAŢII

Se pot memora maximum 5 ecrane; dacă se încarcă al şaselea, nu rămîne decît foarte puţin pentru programul BASIC.

Se păstrează n într-o adresă ce nu va fi afectată de comanda CLEAR ce urmează.
Se păstrează neafectată zona de memorie 65368...65535 care conţine graficele definite de utilizator.
Se rezervă memorie pentru ecrane.
Se recuperează variabila n.
Se determină adresa de început.
Adresa de început poate fi scrisă 256 × h + l
Se memorează rutina de mutare de pe ecran în memorie.
Mută ecranul în memorie.
Transformă rutina de mutare pentru a muta din memorie pe ecran.

NOTĂ. Pentru a realiza desenul prin BASIC, se înlocuieşte LOAD ""CODE 16384,6912 DIN LINIA 80 cu o comandă GO SUB desen.

Informaţii despre ecran sau despre elementele afişate pe ecran se pot obţine prin cîteva comenzi BASIC sau din variabilele de sistem.

COMENZI BASIC :
— SCREEN (linie, coloană) unde 0≤linie≤23 şi 0≤ ≤coloană≤31, are ca rezultat caracterul aflat la linia şi

coloana respectivă.

— ATTR (linie, coloană) unde 0 ≤ linie ≤ 23 şi 0=coloană=31, are ca rezultat atributul de la linia şi coloana respectivă, exprimat conform celor prezentate anterior.

— POINT (x,y) unde 0 ≤ *x* ≤ 255 şi 0 ≤ *y* ≤ 175, analizează punctul de pe ecran şi este 1 dacă punctul are culoarea cernelii (INK) sau 0 dacă punctul are culoarea hîrtiei.

VARIABILE DE SISTEM, se obţin prin PEEK adresa.

— BORDCR din adresa 23624 specifică atributele comune ale părţii de jos a ecranului şi a marginii (border), cu codificarea ştiută.

— DFSZ din adresa 23659 conţine numărul de linii din partea de jos a ecranului (obişnuit 2), parte utilizată pentru INPUT-uri şi rapoarte.

— COORDS din adresele 23677 şi 23678 conţin coordonatele x şi respectiv y ale ultimului punct plotat cu comenzi grafice.

— SPOSN din adresele 23688 şi 23689 conţin coordonatele poziţiei PRINT în partea de sus a ecranului, respectiv la PRINT AT linie, coloană, adresa 23688 conţine cifra 33-coloană iar adresa 23689, 24-linie.

— SPOSNL din adresele 23690 şi 23691, acelaşi lucru pentru liniile din partea de jos a ecranului.

— ATTR P din adresa 23693 conţine atributele globale, codificate obişnuit, conform celor prezentate anterior.

— ATTR T din adresa 23695 conţine atributele temporare, indicate printr-o comandă PRINT.

— K CUR din adresele 23643 şi 23644 indică adresa cursorului.

— DFCC din adresele 23684 şi 23685 conţine adresa în display a poziţiei PRINT în partea de sus a ecranului.

— DFCCL din adresele 23684 şi 23685 conţine acelaşi lucru pentru partea de jos a ecranului.

PARTEA			GRAFICĂ				ATRIBUTE			
COLOANA			0	1	...	31	0	1	...	31
SECTOR	LINIE	RÎND								
		0	16384	16385	.....	16515				
		1	16640	16641	.....	16671				
		2	16896	16897	.....	16927				
	0	3	17152	17153	.....	17183				
		4	17408	17409	.....	17439	22528	22529	.....	22559
		5	17664	17665	.....	17695				
		6	17920	17921	.....	17951				
		7	18176	18177	.....	18207				
		0	16416	16417	.....	16447				
		1	16672	16673	.....	16703				
		2	16928	16929	.....	16959				
1	1	3	17184	18185	.....	17215				
		4	17440	17441	.....	17471	22560	22561		22591
		5	17696	17697	.....	17727				
		6	17932	17933	.....	17963				
		7	18208	18209	.....	18239				
	..	.	.....				.....			
		0	16608	16609	.....	16639				
		1	16884	16885	.....	16915				
		2	17120	17121	.....	17151				
	7	3	17376	17377	.....	17407				
		4	17632	17633	.....	17663	22752	22753	—	22783
		5	17888	17889	.....	17919				
		6	18124	18125	.....	18155				
		7	18400	18401	.....	18431				
		0	18432	18433	.....	18463				
		1	18688	18689	.....	18719				
		2	18944	18945	.....	18975				
	8	3	19200	19201	.....	19231				
		4	19456	19457	.....	19487	22784	22785	.....	22815
		5	19712	19713	.....	19743				
		6	19968	19969	.....	19999				
		7	20224	20225	.....	20255				
2	..	.	.....			.....	....			
		0	18656	18657	.....	18687				
		1	18912	18913	.....	18943				
		2	19168	19169	.....	19199				
	15	3	19424	19425	.....	19455				
		4	19680	19681	.....	19711	23008	23009		23039
		5	19936	19937	.....	19967				
		6	20192	20193	.....	20223				
		7	20448	20449	.....	20479				
		0	20480	20481	.....	20511				
		1	20736	20737	.....	20767				
		2	20992	20993	.....	21023				
	16	3	21248	21249	.....	21279				
		4	21504	21505	.....	21535	23040	23041		23071
		5	21760	21761	.....	21791				
		6	22016	22017	.....	22047				
		7	22272	22273	.....	22302				
3	..	.	.....			.....	....			
		0	20704	20705	.....	20735				
		1	20960	20961	.....	20991				
		2	21216	21217	.....	21247				
	23	3	21472	21473	.....	21503				
		4	21728	21729	.....	21759	23264	23265		23295
		5	21984	21985	.....	22015				
		6	22240	22241	.....	22271				
		7	22496	22497	.....	22527				

PROGRAMARE EFICIENTĂ *

Anca Dumitru

La prima vedere, **C** pare să se bucure de o imensă popularitate. Multe din cele mai importante aplicaţii pentru PC au fost sau mai sînt trecute în **C** din limbajul în care au fost iniţial implementate, iar reclamele pentru echipamentele orientate către **C** ca şi bibliotecile lor în revistele de programare depăşesc ca număr pe cele în toate celelalte limbaje la un loc. Dar, este de aşteptat ca în relativ puţini ani să discutăm despre clasicul **C** cu nostalgie, aşa cum deja mulţi programatori privesc acum limbaje precum, COBOL, FORTRAN, BASIC. Această predicţie se bazează pe două motive:

Primul, odată cu standardizarea unui limbaj, evoluţia sa este blocată de cerinţele stricte ale consensului şi compatibilităţii cu practica existentă. Limbajul nu mai poate fi adaptat rapid la schimbări ale metodologiei de programare sau la cerinţe ale noilor echipamente de operare. Într-un anumit moment, cînd cerinţele realităţii s-au depărtat prea mult de facilităţile oferite de limbaj, devine mai uşor pentru programatori să treacă la alt limbaj sau să inventeze unul nou.

Al doilea, şi cel mai important este faptul că are loc o radicală schimbare în optica programării. Lumea limbajelor tradiţionale este gata să fie revoluţionată, iar cîştig de cauză vor avea limbajele de programare orientate pe obiect - OOPLs (Object Oriented Programming Languages). O asemenea revoluţie nu este fără precedent - vezi înlocuirea limbajului FORTRAN cu limbaje moderne, structurate în blocuri, ca PASCAL, C şi MODULA II în ultimul deceniu. Dar trecerea la OOPLs va cunoaşte un impact mult mai dramatic asupra programatorilor decît trecerea de la FORTRAN la **C**. De ce? Pentru că OOPLs cere programatorului să privească problema pe care încearcă să o rezolve într-un mod radical diferit. În limbajele tradiţionale, vizualizezi datele ca nişte elemente fixe, în timp ce diverse rutine intervin pentru a executa operaţii cu acestea. În cadrul unui echipament şi a softului asociat orientate pe obiect, rutinele sînt vizualizate ca statice — ascunse în interiorul unor mici cutii negre numite obiecte — şi datele sînt cele care acţionează: trec de la un obiect la altul sub formă de mesaje. Acest fapt are o corespondenţă interesantă cu software-uri grafice, ca Microsoft, Windows ş.a.

În ciuda agitaţiei de dată recentă în articole şi cărţi de specialitate, OOPLs nu sînt tocmai o nouă invenţie. Prototipul SIMULA a fost pentru prima oară descris în 1967 -chiar a precedat limbajul C- fiind inventat de Ole-Johan Dahl şi Krysten Nygaard de la Universitatea din Oslo şi Centrul de Calcul Norgevian. Mai cunoscutul SMALLTALK a fost inventat în mijlocul anilor '70. Au urmat altele. Evenimentul cheie în acest sens a fost apariţia în urmă cu circa 8 ani a limbajului **C++** (autor Bjarne Stroustrup, cercetător la AT&T Bell), ca un superset orientat pe obiect al limbajului **C**. Acesta a găsit imediat susţinătorii printre academîcieni şi unii programatori pe sisteme **C**. Caracteristic faţă de OOPLs care l-au precedat este compatibilitatea cu codul **C** existent permiţînd programatorilor să înveţe şi să adopte tehnici orientate pe obiect într-un mod incremental. O alternativă la **C++**, numit **Objective C** a fost dezvoltată de Brad Cox cam în acelaşi timp cu originalul lui Stroupstrup. Dar **Objective C** era mai apropiat de SMALLTALK şi mai puţin intuitiv pentru programatorii **C**. Unele caracteristici ale limbajului SMALLTALK, ca şi originalului **C++**, arată de ce OOPLs au fost privite ca nişte curiozităţi de către marea majoritate a programatorilor atîţia ani. Erau necesare o maşină puternică cu un adaptor grafic cu o înaltă rezoluţie, megabaiţi de RAM (Random Acces Memory) şi un disc fix mare şi rapid. Numai foarte recent au reuşit asemenea aparaturi să fie la îndemîna posibilităţilor financiare ale programatorului obişnuit. De asemenea, anumite caracteristici ale SMALLTALK s-au dovedit a fi slăbiciuni cînd limbajul este folosit pentru proiecte mari ce implică mulţi programatori.

În anii '80, **C++** a avut de înfruntat reputaţia unor compilatoare lente şi greoaie ce generau codul maşină ineficient. Majoritatea compilatoarelor **C++**, pur şi simplu transformau codul sursă **C++** într-un foarte întortochiat cod sursă **C** care era apoi supus compilării **C** obişnuite. Programatorii erau sceptici în legătură cu trecerea la **C++** pentru proiecte mari, pentru că limbajul evolua rapid, cîteva dialecte erau în proiect şi în plus confuzia era sporită de existenţa limbajului **Objective C**.

Iniţial, deşi creştea numărul susţinătorilor, programarea orientată pe obiect a servit numai seminariilor academice şi tezelor studenţilor pentru absolvire, pînă în mai 1989. Atunci atît BORLAND cît şi MICROSOFT au trecut de partea OOPLs introducînd echipamente şi utilităţi cu extensii orientate pe obiect. Coincidenţa în timp şi asemănarea între MICROSOFT QUICKPASCAL şi BORLAND TURBO PASCAL 5.5 a fost remarcabilă. Ambele erau compatibile cu TURBO PASCAL 5.0 şi includeau documentaţie excelentă, materiale de referinţă pentru extensii, editoare multifişier în cadrul ferestrelor şi "debugger" integrat (sistem de facilităţi destinate depanării programelor). Şi ambele au fost oferite programatorilor la preţuri sub 100 $. Aceasta a pus efectiv OOPLs în mîna milioanelor de programatori serioşi şi capabili care se considerau gata să facă încercarea. În mai 1990 BORLAND a scos **TURBO C++**, o evoluţie naturală a produsului său deja popular, **TURBO C**. Apariţia acestui nou produs cu toate facilităţile aferente ne întăreşte convingerea că am intrat în era limbajelor orientate pe obiect.

Caracteristici ale OOPLs

Reticenţa cu care se trece la scrierea programelor în OOPLs este foarte asemănătoare cu adaptarea dificilă pe care programatorii în MS-DOS o încearcă cînd încep să codifice pentru echipamente tip window ca MICRO-

* După articolul lui Ray Duncan din *PC Magazine* - SUA - nov. 1990.

SOFT WINDOW ş.a. Ei sînt obişnuiţi cu ideea că nimic nu se întîmplă decît dacă programul lor face să se întîmple — de exemplu nici un caracter nu vine de la tastatură decît dacă este cerut de program. Dar într-un mediu tip WINDOW, un program trebuie să fie construit în cu totul alt mod; pentru că este constant şi imprevizibil bombardat cu evenimente nesolicitate variind de la conectori, pînă la cerinţe ale sistemului de operare de a reface părţi ale ecranului. Nu contează ce altă activitate poate face programul, este întotdeauna important să manevreze în timp util evenimentele. La fel, un obiect într-un OOPL nu acţionează niciodată, el întotdeauna reacţionează.

Ce-l remarcă pe un limbaj de programare ca într-adevăr orientat pe obiect? Sînt trei trăsături care sînt citate în acest context: ENCAPSULATION, POLYMORPHISM şi INHERITANCE (traducerea e nepotrivită: încapsulare, polimorfism şi moştenire). Dacă ţineţi minte aceste trei cuvinte "la modă", veţi impresiona cînd veţi dori să cumpăraţi echipamente informatice la petreceri sau cînd vreţi să derutaţi interlocutorul cu privire la ceea ce faceţi.

ENCAPSULATION — un termen "interesant" pentru a desemna ascunderea informaţiilor. După cum am menţionat, cînd scrii un program într-un OOPL, procedurile, datele lor particulare şi constantele sînt incluse în cantităţi numite *obiecte.* Fiecare *obiect* "ştie" ce mesaje poate procesa, pe care să le respingă şi pe care să le treacă altui obiect, dar nici un *obiect* nu poate cunoaşte cu adevărat capabilităţile altuia. Scopul principal al acestei trăsături este izolarea activităţilor interne ale unei clase particulare de obiecte astfel încît să poată fi modificate şi îmbunătăţite fără a cauza efecte secundare dăunătoare sistemului în general.

POLYMORPHISM — este similar cu supraîncărcarea operatorului din **C** sau PASCAL, unde poţi folosi acelaşi operator pentru manipularea mai multor tipuri de date: compilatorul **C** sau PASCAL este suficient de inteligent pentru a examina tipul datelor şi a genera codul potrivit. Într-un OOPL, această trăsătură se referă la capacitatea unui obiect de a selecta procedura internă corectă (numită metodă) pe baza tipului datelor primite într-un mesaj (necunoscute în momentul scrierii programului).

INHERITANCE — este principala armă a OOPL împotriva complexităţii şi trăsătura care le apropie cel mai bine de aplicaţiile în medii cu WINDOW. O nouă clasă de obiecte derivă dintr-una existentă, prin schimbarea uneia sau mai multor metode aferente clasei vechi şi adăugîndu-i una sau mai multe. Acest proces se mai numeşte uneori şi *subclasare.* Noua clasă trebuie să conţină numai codul metodelor adăugate sau schimbate; codul pentru metodele "moştenite" neschimbate rămîne în clasa de referinţă. Este uşor de imaginat cît simplifică aceasta codificarea unui program care trebuie să controleze mai multe diferite tipuri de ferestre care sînt mici variaţiuni ale aceleiaşi.

În timp ce tranziţia către OOPLs poate fi dificilă şi confuză pentru programatorii obişnuiţi, triumful treptat al OOPLs într-o formă sau alta apare ca inevitabil acum, cînd este cerut de dificultatea de a programa interfeţe grafice în limbajele procedurale tradiţionale. Într-adevăr, la Seminarul Sistemelor Microsoft din ultimul an, vorbitorii au considerat implicit triumful OOPLs şi au vorbit. în schimb, despre planurile pentru sisteme de fişiere orientate pe obiect şi interfeţe utilizator orientate pe obiect. În plus, MICROSOFT a anunţat intenţia sa de a comercializa instrumente de programare care să permită un tip similar de programare vizuală pentru aplicaţii tip WINDOWS *and* PRESENTATION MANAGER. Interfeţele utilizator vor fi construite de către programator, folosind un instrument grafic interactiv care va genera automat programul sursă necesar pentru aplicaţia respectivă. Aceasta va elimina necesitatea de a scrie sute de linii de cod **C** obscur, doar pentru selectarea fontelor, vizionarea ferestrelor (WINDOWS), controlul meniurilor şi a zonelor de dialog, şi va permite programatorilor să se concentreze asupra scopului aplicaţiei şi mai puţin asupra mecanismului interfeţei utilizator. Echipamentul NeXT este deja prevăzut cu asemenea instrumente de programare. De asemenea, MICROSOFT a prezentat intenţiile sale de a dezvolta şi un fel de superset al limbajului QUICKBASIC, aceasta în ideea de a implementa un macrolimbaj care să permită utilizatorilor să execute programe pe aplicaţiui "din afară" şi să transfere date între ele. Utilizatorul va fi capabil să creeze scenarii în acest macrolimbaj şi apoi doar să indice paşii doriţi în ordine. Utilizatorul va avea astfel posibilitatea de a perfecţiona şi dezvolta scenariile cu un editor de texte normal. O facilitate ca aceasta este în mod deosebit necesară în cadrul echipamentelor şi aplicaţiilor grafice, unde limbajele tradiţionale sînt în principiu nefolositoare.

SOSESC PRODUSELE CASE

Cureleţ-Bălan Gheorghe

În ultimii ani piaţa produselor software a fost invadată de o categorie specială de produse, care conform anumitor păreri ar reprezenta produsele viitorului, numite produse **CASE.**

CASE reprezintă prescurtarea de la *Computer Aided Software Engineering* care în traducere înseamnă inginerie software asistată de calculator. Apariţia domeniului se justifică prin aceea că elaborarea de *software fiabil* (robust) a ajuns în prezent într-un moment de criză.

Apariţia calculatoarelor personale şi revoluţia pe care a declanşat-o au dus la o largă răspîndire a calculatoarelor în cele mai diverse domenii, şi în cele mai diverse categorii socio-profesionale. Astfel s-a estimat că la sfîrşitul anului 1988 numărul calculatoarelor personale răspîndite pe întregul mapamond era de aproximativ 20 milioane. Evident că în prezent acest număr este mult mai mare.

În aceste condiţii solicitările de software au crescut imens. Imperativul prezent în materie de software îl reprezintă elaborarea sa cît mai rapidă dar, aceasta să nu se facă în detrimentul calităţii sale. Software-ul elaborat trebuie să fie cît mai fiabil, adică să aibă cît mai puţine erori.

Soluţia acestei crize o reprezintă folosirea însăşi a calculatorului în activitatea de elaborare software. Mulţi ar putea să ne reproşeze că el a fost folosit încă de la apariţia primelor sisteme de operare, asambloare şi compilatoare şi că actualele produse de succes destinate calculatoarelor personale şi minicalculatoarelor (editoare, spreadsheet-uri, sisteme de gestiune baze de date, etc.) sînt de un real ajutor elaboratorilor de software. Acestora le putem răspunde că produsele **CASE** se ridică prin destinaţie şi facilităţi deasupra categoriei de produse sus menţionate.

Înainte însă de a vorbi despre destinaţia produselor **CASE** trebuie să definim mai bine ce înseamnă de fapt elaborarea software-ului şi ce este ingineria programării.

La început, prin elaborarea software-ului se înţelegea activitatea de codificare într-un limbaj de programare a cerinţelor beneficiarului. Această activitate avea un caracter artizanal, nedisciplinat, fără a respecta anumite norme, motiv pentru care au început să apară o sumă întreagă de inconveniente, cum ar fi : software-ul elaborat era nefiabil, necesitînd o întreţinere continuă a sa, codul (sursa) era ilizibil şi greu accesibil pentru alţi programatori, termenele de livrare erau depăşite iar costurile subestimate, etc. Toate aceste inconveniente corelate cu alte elemente cum ar fi continua ieftinire a hardware-ului, creşterea ponderii software-ului, etc. au dus la necesitatea disciplinării activităţii de elaborare a software-ului. S-a ajuns astfel la concluzia că elaborarea software-ului nu înseamnă doar codificare ci ea presupune parcurgerea unui şir de etape oarecum identice cu cele parcurse de un produs elaborat pe cale industrială. Cu alte cuvinte elaborarea software-ului trebuie să se facă după o serie de principii ştiinţifice, inginereşti, care să constituie limitele între care să se desfăşoare actul programării fără însă a impieta asupra aspectului creativ, prin excelenţă, al său. S-a stabilit astfel că activitatea de elaborare a software-ului constă din parcurgerea următoarelor etape : specificare cerinţe de realizare, analiză/proiectare, codificare, testare, implementare şi întreţinere. Această înşiruire de etape formează ciclul de viaţă a software-ului. Aspectul ciclic derivă din aceea că elaborarea unei noi versiuni a software-ului presupune reparcurgerea etapelor sus amintite.

Cercetările în această direcţie au dus la conturarea la sfîrşitul deceniului al 7-lea a unui domeniu distinct al informaticii numit *ingineria programării.* Evident că au fost elaborate o sumă întreagă de norme care trebuie respectate în parcurgerea fiecarei etape. Acestea reprezintă metodologiile. Respectarea metodologiilor presupune folosirea anumitor metode şi tehnici. Putem defini deci ingineria programării ca domeniul care se ocupă de totalitatea metodologiilor, metodelor, tehnicilor care să conducă în condiţii de rentabilitate şi eficienţă la elaborarea unui software care să satisfacă toate cerinţele beneficiarului.

Folosirea calculatorului în automatizarea activităţilor din etapele ciclului de viaţă a software-ului a reprezentat dintotdeauna un deziderat pentru autorii de software. Deşi, încercări timide de a realiza acest deziderat, au existat încă din perioada pionieratului tehnicii de calcul şi informaticii, realizările notabile apar abia în perioada anilor '80. Acestea au dus la conturarea domeniului ingineriei software asistată de calculator- domeniu cunoscut pe plan internaţional sub denumirile de : **CASE** în S.U.A., **IPSE** (*Integrated Project Support Environment*) în Europa, "*Atélier de Génie Logiciel*" (**AGL**) în Franţa.

Fără a avea pretenţia unei definiţii exhaustive putem defini **CASE** ca folosirea tehnicilor de inginerie software precum şi a tehnicii de calcul în scopul automatizării dezvoltării sistemelor software. Produsele de tip **CASE** implementează metodele formale de dezvoltare software, reprezentînd instrumente de lucru pentru elaboratori pe tot cuprinsul ciclului de viaţă a software-ului. În plus, ele oferă echipei de elaborare suport metodologic precum şi facilităţi de asigurare a calităţii (verificare şi asigurare a consistenţei, completitudinii şi conformităţii cu anumite standarde de elaborare). Putem considera deci, că **CASE** aduce dezvoltării de soft aceleaşi beneficii ca şi *CAD/CAM* producţiei.

Deşi pe plan internaţional există unele tendinţe de contestare a statutului de sine stătător al domeniului, considerăm că **CASE** şi-a conturat frontierele afîndu-se în prezent în curs de maturizare. O dovadă a viabilităţii **CASE** o reprezintă adoptarea sa de principalii "actori" de pe scena tehnicii de calcul şi informaticii mondiale (*IBM, DEC, Hewlett-Packard, Bull, Tektronix, Sun, Texas Instruments, Oracle, Relational Technology, Software AG* etc.). Menţionăm de asemenea că piaţa produselor **CASE** cunoaşte în prezent una din cele mai mari rate de creştere

din industria de software. Conform unui studiu publicat în 1988, de **CASE** Consulting Group, la sfîrşitul anului 1987 existau peste 150 de firme competitoare pe piaţa produselor **CASE**, estimîndu-se pentru 1992 o piaţă în jur de 500 milioane dolari.

Un alt element revelator îl reprezintă eforturile de cercetare-dezvoltare în domeniul **CASE** lansate în Europa, S.U.A. şi Japonia. Menţionăm astfel programele de cercetare : **Esprit** (al comunităţii europene) , **Alvey** (Marea Britanie), **STARS** (*Software Technology for Adaptable Reliable Systems*) - finanţat de DOD (300 milioane $), **Leonardo** (al corporaţiei *Microelectronics and Computer Corporation (MCC)* - SUA) şi **Sigma** - *Software Industrialized Generation and Maintenance Aids* (proiect de 100 milioane $, lansat în 1985 şi finanţat de MITI, care îşi propune să automatizeze cu 80% concepţia de soft şi să reducă cu 75% timpul de programare).

În prezent produsele **CASE**, datorită expansiunii domeniului sînt de o mare diversitate, astfel încît selectarea unuia potrivit nevoilor specifice unui utilizator a devenit o problemă - mentionam că primul produs (**CASE** Analyser) destinat rezolvării acestei probleme a fost dèja prezentat de firma *Parallax/P-Cube Corp.* cu ocazia simpozionului **CASE** de la începutul anului 1989 la Dallas şi Boston.

În numărul viitor, pornind de la anumite criterii de clasificare, vom prezenta cîteva dintre facilităţile oferite de produsele **CASE.**

NOUTĂŢI • NOUTĂŢI • NOUTĂŢI

Noul laser al Mannesmann Tally (M.T.)

M.T. a pus la punct un nou model de imprimantă cu laser MT 906. Noua imprimantă, prin caracteristicile sale oferă uşurinţă în întrebuinţare(interfaţa utilizator dotată cu menu simplu), este silenţioasă. MT 906 este superioară modelelor precedente MT 905 şi 910 prin rezoutia de 300 dpi, viteza de imprimare de 6 pagini pe minut, 512 KByte memorie RAM ce se poate extinde pînă la 4MB; se furnizează cu 3 emulatoare şi 2 interfeţe standard (RS232 şi paralela CTX). Imprimanta MT 906 se distinge şi prin modul de gestionare a hîrtiei: spre deosebire de alimentarea tradiţională poate să dispună opţional de 2 casete cu o capacitate de 200 coli fiecare.

Page Maker - Windows 3.0

Utilizatorii de Page Maker vor putea să beneficieze de performanţele aduse de versiunea 3.01 faţă de Windows 3.0. Aceasta asigură o creştere a vitezei de 40-50% faţă de versiunea precedentă, un răspuns instantaneu la anumite funcţii şi posibilitatea de a beneficia de buna funcţionare a reţelelor Windows 3.0. Page Maker 3.01 a fost îmbogăţită cu noi formate de test, ca Test Format, Word Perfect 5.0 şi 5.1.

New Wave 3.0

Hewlett-Packard îşi întăreşte propria strategie prin New Wawe. Această software permite utilizatorilor să trieze cele mai bune produse disponibile pe piaţă, fie ale Hewlett-Packard ca şi ale altor producători, şi să le integreze într-o reţea deschisă şi cooperativă, cu scopul de a simplifica diferitele operaţiuni de gestionare a informaţiilor. Noua versiune 3.0 foloseşte M.S.Windows 3.0, exploatînd într-un mod special interfaţa utilizator şi asigurînd o mai eficientă gestionare a memoriei, reducînd de la 3 la 2 MB capacitatea de memorie necesară ca suport al numeroaselor aplicaţii. Cu această nouă versiune e posibilă realizarea de operaţiuni complexe şi repetitive cu o singură comandă, şi împărţirea resurselor reţelei printr-o simplă manipulare a "icoanelor" vizualizate pe ecran.

Un 68040 pentru NeXT

Începînd de la sfîrşitul anului 1990 este disponibilă o nouă versiune de NeXT bazată pe microprocesorul Motorola MC 68040. Cipul MOTOROLA care constituie a patra generaţie din serie, e compatibil la nivel obiect cu predecesorii săi, garantînd deci o uşoară actualizare a software-ului deja existent. Acest microprocesor încorporează ce are mai bun tehnologia ASC şi RISC.

O alternativă pentru mouse

Se numeşte PC Trackball şi e produsul lui Mouse System. La prima vedere este la fel, dar mişcarea cursorului pe ecran nu se mai obţine prin deplasarea mouse-ului pe masa de lucru, ci pur şi simplu învîrtind cu un deget o bilă. PC Trackball poate fi integrat la tastatură de propriul personal printr-un efort minim, iar la instalare necesită portul serial RS 232 şi o versiune de MS-DOS 2.1 sau superioară. Rezoluţia variază între 200 şi 6400 CPI şi asigură o deplină compatibilitate cu standardul Microsoft Mouse. PC Trackball ar putea să fie o reală alternativă pentru mouse, mai ales în cazul unui spaţiu limitat, şi mai ales în cazul calculatoarelor portabile, folosite în condiţii de spaţiu precare.

Recunoaştere vocală

"Deltatre" a început comercializarea de "Micro-Intro Voice", un sistem de input/output vocal capabil să recunoască 1000 cuvinte. Sistemul transferă semnalele vocale în caractere ASCII la poarta serială şi trimite testul la sintetizatorul incorporat pentru audierea şi controlul imediat. Micro-Intro Voice e furnizat cu Host-software şi poate funcţiona în mediu DOS, OS/2, UNIX şi DEC.Sistemul e disponibil în două versiuni: prima versiune completă, cu recunoaştere şi sinteză vocală, a doua doar cu sinteza vocală.

OFERTĂ

ADISAN STUDENT

COD ARTICOL	DENUMIRE ARTICOL	HERCULES	VGA MONO	VGA 256	VGA 512
10286120401X0	DESKTOP 296/12 1MbRAM:1,2MbFD;40MbHD;28ms	283.250	311.500	354.500	375.000
10286120402X0	DESKTOP 286/12 1MbRAM;1,44MbFD;40MbHD;28ms	280.750	308.500	351.250	371.500
10286120403X0	DESKTOP 286/12 1MbRAM;1,2/1,44MbHD;40MbHD;28ms	308.750	336.500	386.000	406.500
10300160401X0	DESKTOP 386SX/16 1MbRAM;1,2MbFD;44MbHD;40MbHD;28ms	384.750	412.500	469.000	485.000
10300160402X0	DESKTOP 386SX/16 1MbRAM;1,44MbHD;40MbHD;28ms	382.000	410.000	466.500	482.750
10300160403X0	DESKOTOP 369S/X 1MbRAM;1,2/1,44MbFD;40MbHD;28ms	409.750	438.000	494.500	510.250
10386200401X0	DESKTOP 386/20 1MbRAM;1,2MbFD;40MbHD;28ms	469.00 0	497.750	554.250	569.500
10386200402X0	DESKTOP 386/20 1MbRAM;1,44MbFD;40MbHD;28ms	466.750	494.500	551.000	567.000
10386200403X0	DESKTOP 386/20 1MbRAM;1,2/1,44MbFD;40MbHD;28ms	494.750	522.500	579.000	594.750

ADISAN PRO386SX

COD ARTICOL	DENUMIRE ARTICOL	HERCULES	VGA 256	VGA 512
11300160401X0	MINI DESKTOP 386SX/16 1MbRAM;1,2MbFD;40MbHD;28ms	483.000	511.000	530.500
11300160402X0	MINI DESKTOP 386SX/16 1MbRAM;1,44MbFD;40MbHD;28ms	479.500	507.500	527.000
11300160403X0	MINI DESKTOP 386SX/16 1MbRAM;1,2/1,44MbFD;40MbHD;28ms	515.250	541.500	562.750
11300160801X0	MINI DESKTOP 386SX/16 1MbRAM;1,2MbFD;80MbHD;19ms	640.250	668.250	687.750
11300160802X0	MINI DESKTOP 386SX/16 1MbRAM;1,44MbFD;80MbHD;19ms	636.750	664.750	684.250
11300160803X0	MINI DESKTOP 386SX/16 1MbRAM;1,2/1,44MbFD;180MbHD;19ms	696.250	723.500	743.750
11300161801X0	MINI DESKTOP 386SX/16 1MbRAM;1,2MbFD;180MbHD;25ms	823.000	850.000	870.500
11300161802X0	MINI DESKTOP 386SX/16 1MbRAM;1,44MbFD;180MbHD;25ms	819.500	846.750	867.000
11300161803X0	MINI DESKTOP 386SX/16 1MbRAM;1,2/1,44MbFD;180MbHD;25ms	855.250	882.500	902.750

OFERTĂ

ADISAN PRO286

COD ARTICOL	DENUMIRE ARTICOL	HERCULES	VGA 256	VGA 512
11286120401X0	MINI DESKTOP 286/12 1MbRAM;1,2MbFD;40MbHD;28ms	352.750	381.000	401.250
11286120402X0	MINI DESKTOP 286/12 1MbRAM;1,44MbFD;40MbHD;28ms	350.250	377.500	398.000
11286120403X0	MINI DESKTOP 286/12 1MbRAM;1,2/1,44MbFD;40MbHD;28ms	385.250	413.250	433.500
11286120801X0	MINI DESKTOP 286/12 1MbRAM;1,2MbFD;80MbHD;19ms	511.000	538.250	558.500
11286120802X0	MINI DESKTOP 286/12 1MbRAM;1,44MbFD;80MbHD;19ms	507.500	534.750	555.250
11286120803X0	MINI DESKTOP 286/12 1MbRAM;1,2/1,44MbFD;80MbHD;19ms	542.500	570.500	590.750
11286121001X0	MINI DESKTOP 286/12 1MbRAM;1,2MbFD;100MbHD;19ms	534.000	562.000	582.250
11286121003X0	MINI DESKTOP 286/12 1MbRAM;1,2/1,44MbFD;100MbHD;19ms	566.250	594.250	614.750
11286121801X0	MINI DESKTOP 286/12 1MbRAM;1,2MbFD;180MbHD;25ms	692.750	721.000	741.250
11286121802X0	MINI DESKTOP 286/12 1MbRAM;1,44MbFD;180MbHD;25ms	689.500	717.500	738.000
11286121803X0	MINI DESKTOP 286/12 1MbRAM;1,2/1,44MbFD;180MbHD;25ms	725.250	753.250	773.500
11286160401X0	MINI DESKTOP 286/16 1MbRAM;1,2MbFD;40MbHD;28ms	409.750	437.750	458.250
11286160402X0	MINI DESKTOP 286/16 1MbRAM;1,44MbFD;40MbHD;28ms	406.500	434.500	454.750
11286160403X0	MINI DESKTOP 286/16 1MbRAM;1,2/1,44MbFD;40MbHD;28ms	442.000	470.250	490.500
11286160801X0	MINI DESKTOP 286/16 1MbRAM;1,2MbHD;19ms	567.000	595.000	615.500
11286160802X0	MINI DESCTOP 286/16 1MbRAM;1,44MbHD;19ms	563.750	591.750	612.000
11286160803X0	MINI DESKTOP 286/16 1MbRAM;1,2/1,44MbFD;80MbHD;19ms	599.250	627.500	647.750
11286161001X0	MINI DESKTOP 286/16 1MbRAM;1,2MbFD;100MbHD;19ms	590.750	619.000	639.250
11286161002X0	MINI DESKTOP 286/16 1MbRAM;1,44MbFD;100MbHD;19ms	587.500	615.500	636.000
11286161003X0	MINI DESKTOP 286/16 1MbRAM;1,2/1,44MbFD;100MbHD;19ms	623.250	651.250	671.500
11286161801X0	MINI DESKTOP 286/16 1MbRAM;1,2MbFD;180MbHD;25ms	749.750	777.750	798.250
11286161802X0	MINI DESKTOP 286/16 1MbRAM;1,44MbFD;180MbHD;25ms	746.500	774.500	794.750
112861618030	MINI DESKTOP 286/16 1MbRAM;1,2/1,44MbFD;180MbHD;25ms	782.000	810.250	830.500

ADISAN PRO386

COD ARTICOL	DENUMIRE ARTICOL	HERCULES	VGA 256	VGA 512
11386200401X0	MULTI DESKTOP 386/20 1MbRAM;1,2MbFD;40MbHD;28ms	590.750	618.000	638.500
11386200402X0	MULTI DESKTOP 396/20 1MbRAM;1,44MbFD;40MbHD;28ms	587.500	614.750	635.000
11386200403X0	MULTI DESKTOP 386/20 1MbRAM;1,2/1,44MbFD;40MbHD;28ms	623.250	650.250	670.750
11386200801X0	MULTI DESKTOP 386/20 1MbRAM;1,2MbFD;100MbHD;19ms	748.000	775.250	795.750
11386200802x0	MULTI DESKTOP 386/20 1MbRAM;1,44MbFD;80MbHD;19ms	744.750	772.000	792.250
11386200803X0	MULTI DESKTOP 386/20 1MbRAM;1,2/1,44MbFD;80MbHD;19ms	780.500	807.500	828.000
11386201001X0	MULTI DESKTOP 386/20 1MbRAM;1,2MbFD;100MbHD;19ms	772.000	799.000	819.500
11386201002X0	MULTI DESKTOP 386/20 1MbRAM;1,44MbFD;100MbHD;19ms	768.500	795.750	816.000
11386201003X0	MULTI DESKTOP 386/20 1MbRAM;1,2/1,44MbFD;100MbHD;19ms	803.250	831.500	851.750
11386201801X0	MULTI DESKTOP 386/20 1MbRAM;1,2MbFD;180MbHD;25ms	930.750	958.000	978.500
11386201802X0	MULTI DESKTOP 386/20 1MbRAM;1,44MbFD;180MbHD;25ms	927.500	954.750	975.000
11386201803X0	MULTI DESKTOP 386/20 1MbRAM;1,2/1,44MbFD;180MbHD;25ms	962.250	990.250	1.010.750
11386330401X2	MULTI TOWER 386/33,64Kb Cache 1MbRAM;1,2MbFD;40MbHD;28ms	775.250	803.250	823.750
11386330402X2	MULTI TOWER 386/33,64Hb Cache 1MbRAM;1,44MbFD;40MbHD;28ms	775.250	803.250	823.750
11386330403X2	MULTI TOWER 386/33,64Kb cACHE 1MbRAM;1,2/1,44MbFD;40MbHD;28ms	811.000	838.250	858.500
11386330801X2	MULTI TOWER 386/33,64Kb Cache 1MbRAM;1,2MbFD;80MbHD;19ms	932.500	960.500	981.000
11386330802X2	MULTI TOWER 386/33,64Kb Cache 1MbRAM;1,44MbFD;80MbHD;19ms	932.500	960.500	981.000
11386330803X2	MULTI TOWER 386/33,64Kb Cache 1MbRAM;1,2/1,44MbFD;80MbHD;19ms	968.250	995.500	1.015.750
11386331001X2	MULTI TOWER 386/33,64Kb Cache 1MbRAM;1,2MbFD;100MbHD;19ms	956.250	984.500	1.004.000
11386331002X2	MULTI TOWER 386/33,64Kb Cache 1MbRAM;1,44MbFD;100MbHD;19ms	956.250	984.500	1.004.000
11386331003X2	MULTI TOWER 386/33,64Kb Cache 1MbRAM;1,2/1,44MbFD;100MbHD;19ms	992.000	1.019.250	1.039.750
11386331801X2	MULTI TOWER 386/33,64Kb Cache 1MbRAM;1,2MbFD;180MbHD;25ms	1.115.250	1.143.250	1.163.000
11386331802X2	MULTI TOWER 386/33,64Kb Cache 1MbRAM;1,44MbFD;100MbHD;25ms	1.115.250	1.143.250	1.163.000
11386331803X2	MULTI TOWER 386/33,64Kb Chase 1MbRAM;1,2/1,44MbFD;100MbHD;25ms	1.151.000	1.178.250	1.198/500
11386333001X2	MULTI TOWER 386/33,64Kb Cache 1MbRAM;1,2MbFD;320MbHD;16ms	1.681.500	1.709.500	1.729.000
11386333002X2	MULTI TOWER 386/33,64Kb Cache 1MbRAM;1,44MbFD;320MbHD;16ms	1.681.500	1.709.500	1.729.000
11386331003X2	MULTI TOWER 386/33,64Kb Cache 1MbRAM;1,2/1,44MbFD;100MbHD;16ms	1.717.000	1.744.250	1.764.750
11386336501X2	MULTI TOWER 386/33,64Kb Cache 1MbRAM;1,2MbFD;650MbHD;16ms	2.205.750	2.233.000	2.253.500
11386336502X2	MULTI TOWER 386/33,64Kb Cache 1MbRAM;1,44MbFD;650MbHD;16ms	2.205.750	2.233.000	2.253.500
11386336503X2	MULTI TOWER 386/33,64Kb Cache 1MbRAM;1,2/1.44MbFD;650MbHD;16ms	2.240.750	2.268.750	2.289.250

OFERTĂ

ADISAN PRO486

COD ARTICOL	DENUMIRE ARTICOL	HERCULES	VGA 256	VGA 512
11486250401X3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,2MbFD;40MbHD;28ms	1.381.250	1.409.500	1.429.750
11486250402X3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,44MbFD;40MbHD;28ms	1.381.250	1.409.500	1.429.750
11486250403X3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,2/1,44MbFD;40MbHD;28ms	1.417.000	1.445.000	1.465.500
11486250801X3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,2MbFD;80MbHD;19ms	1.538.500	1.566.750	1.587.000
11486250802X3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,44MbFD;80MbHD;19ms	1.538.500	1.566.750	1.587.000
11486250803X3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,2/1,44MbFD;80MbHD;19ms	1.574.250	1.602.250	1.622.750
11486251001X3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,2MbFD;100MbHD;19ms	1.582.500	1.590.500	1.610.750
11486251002X3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,44MbFD;100MbHD;19ms	1.562.500	1.590.500	1.610.750
11486251003X3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,2/1,44MbFD;100MbHD;19ms	1.598.000	1.626.250	1.645.500
11486251801X3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,2MbFD;180MbHD;25ms	1.721.250	1.749.500	1.769.500
11486251802X3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,44MbFD;180MbHD;25ms	1.721.250	1.749.500	1.769.750
11486251803X3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,2/1,44MbFD;180MbHD;25ms	1.757.000	1.785.000	1.805.500
11486253001X2	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,/1,44MbFD;180MbHD;25ms	2.290.000	2.318.000	2.338.500
11486253001X2	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,2MbFD;320MbHD;16ms	2.290.000	2.318.000	2.338.500
11486253002X3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,44MbFD;320MbHD;16ms	2.290.000	2.318.000	2.338.500
11486253003X3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,2/1,44MbFD;320MbHD;16ms	2.325.750	2.353.750	2.373.250
11486253001X3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,2MbFD;650MbHD;16ms	2.814.500	2.842.500	2.862.000
11486253002X3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,44MbFD;650MbHD;16ms	2.814.500	2.842.500	2.862.000
11486253003x3	MULTI TOWER 486/25,128Kb Cache 1MbRAM;1,2/1,44MbFD;650MbHD;16ms	2.850.250	2.877.250	2.897.750

IC ICP—Unitate economică integrată de cercetare şi producţie;

ICECP rezolvă aplicaţii în domeniile:
— sisteme de calcul multi/mini/micro procesor pentru funcţionare în medii severe;
— aplicaţii "la cheie" cu sisteme de calcul din producţia proprie sau de la alţi furnizori;
— interfeţe universale şi/sau specializate;
— simulatoare de antrenament pentru maşini şi autovehicule grele;
— livrări prompte de echipamente şi subansamble;
— minicalculatoare compatibile PDP pentru medii severe;
— calculatoare personale CP/M; IBM PC;
— familie de surse de comutaţie în gama 5/10/25 W cu ieşiri variabile;

ICECP asigură asistenţă tehnică şi servicii la un înalt nivel profesional pentru toate produsele cercetate şi aflate în fabricaţie proprie.

ICECP vă poate oferi echipamente şi soluţia tehnică corectă pentru aplicaţiile dumneavoastră în care sistemul de calcul trebuie să funcţioneze în medii mecanoclimatice severe.

R.I.C.E.P. — An economic integrated entreprise for research and production

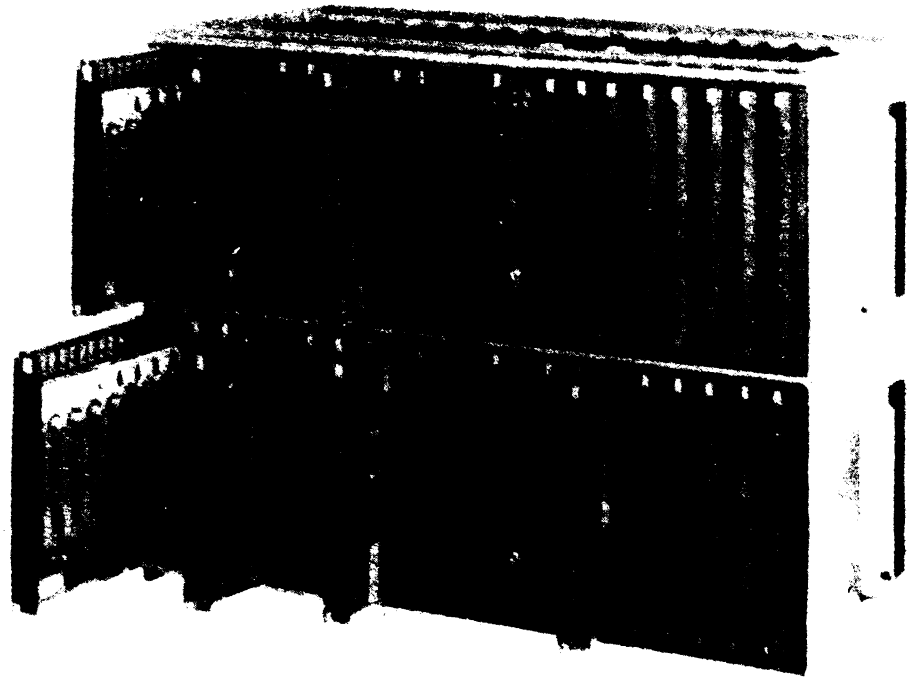
Field of activities:
— computer systems for multi/mini/micro processors working in heavy environments;
— turn-key computer systems from indigene or foreign production;
— universal and/or specialized interfaces;
—training simulators for trucks and other heavy vehicles;
— prompt delivery of equipments and spare parts;
— minicomputers PDP compatible for heavy environments;
— personal computers CP/M; IBM PC;
—a broad set of commutation sources for 5/10/25 W, with variable outputs.

R.I.C.E.P. offers technical assistance and services at high professional level for all research products manufactured by it.

R.I.C.E.P. offers equipments and the right technical solution for your applications using computer systems in heavy mechanical and weather conditions.

ICECP

INSTITUTUL DECERCETĂRI ECHIPAMENTE DE CALCUL ŞI PERIFERICE



BUCUREŞTI
167 B Calea Floreasca ST. Sect.1
Tel. 79.77.97
Telex 11846 itcr