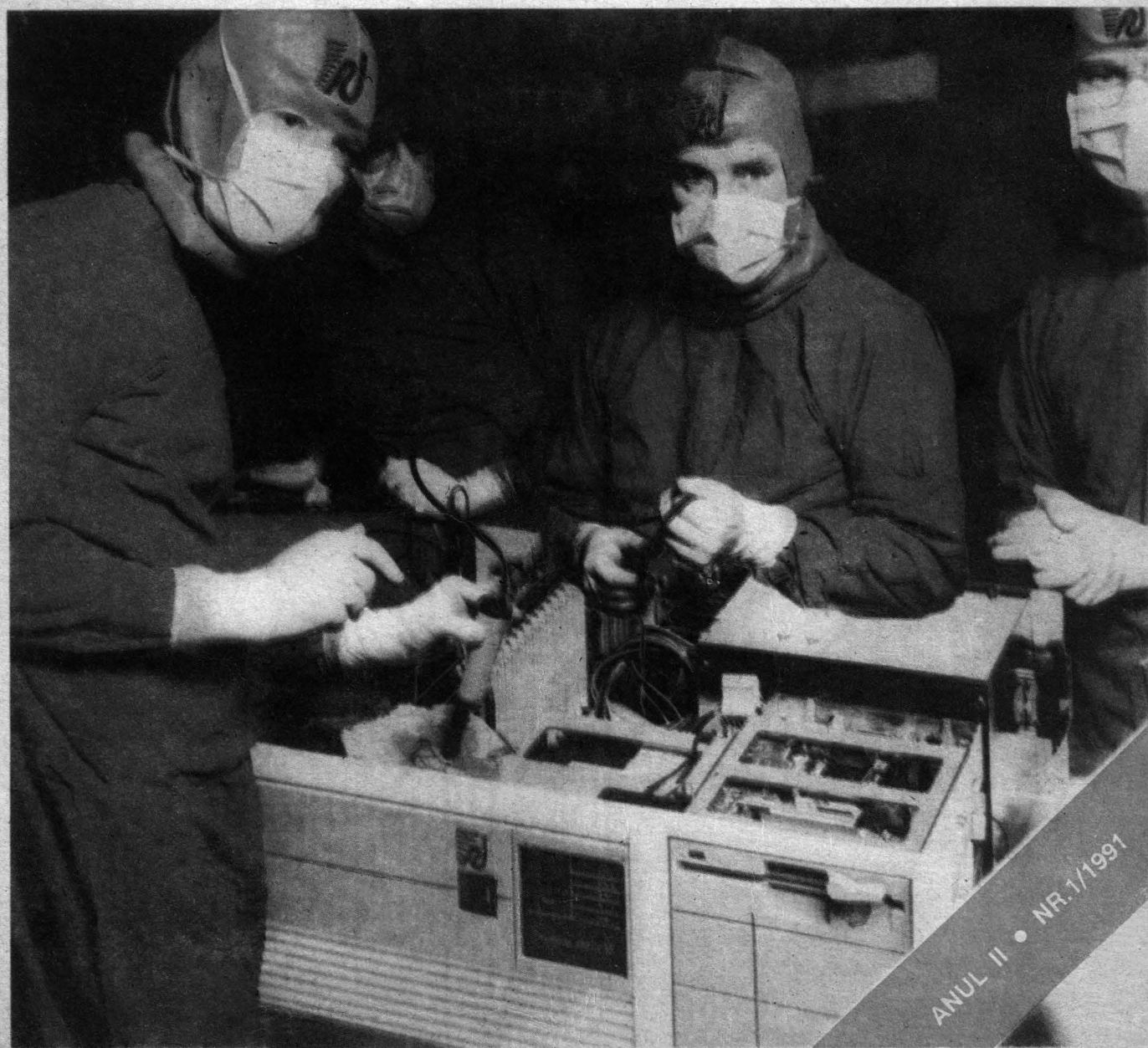
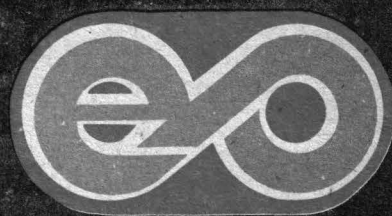
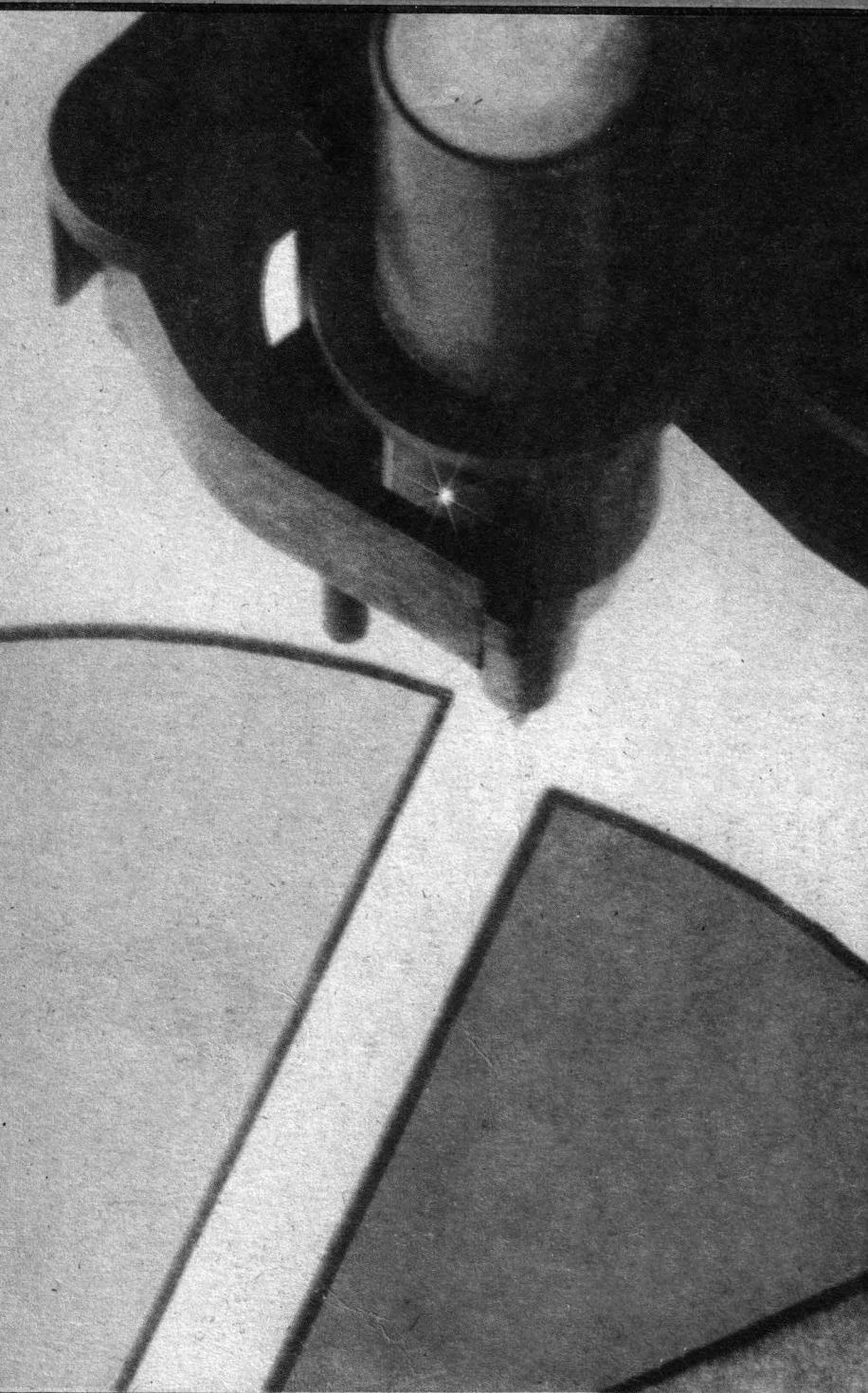


REVISTĂ INDEPENDENTĂ  
DE INFORMATICĂ  
EDITATĂ DE FIRMA ADISAN



- CULTURA INFORMATICĂ  
Rețele de calculatoare (I)  
Rețele neuronale
- INFORMATICA FĂRĂ PROFESOR  
Limbajul C (V)  
Inițiere în programare - Limbajul PASCAL
- WINDOWS  
Interfațarea Turbo PASCAL - MASM  
PC-uri și protecție
- HOME COMPUTER  
Limbajul Z80 (II)  
Despre ecrane și memorarea lor
- PERSPECTIVE





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

**PLOTTERS**

# Cuprins

## CULTURA INFORMATICĂ

Rețele de calculatoare (I) _____	3
Bazele logice ale inteligenței artificiale _____	7
Rețele neuronale _____	11

## INFORMATICA FĂRĂ PROFESOR

Limbajul C (V) _____	16
Editoare și fonturi (V) _____	20
Inițiere în programare — Limbajul Pascal (I) _____	24

## WINDOWS

PC-uri și protecție _____	28
Interfațarea Turbo Pascal - MASM _____	30

## HOME COMPUTER

Limbajul Z80 pe calculatoarele compatibile SPECTRUM (II) _____	35
Despre ecrane și memorarea lor _____	38

## PERSPECTIVE

Programarea eficientă _____	41
Sosesc produsele CASE _____	43
Noutăți _____	44

## 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. Ionuț Stolca  
ing. Florin Dobrian  
Ing. Florin Sultan  
ing. Cureleț-Bălan Gheorghe  
Ing. Călin Șandru  
Ion Paraschiv  
dr. Tiberiu Spircu  
Ion Mușlea  
Bogdan Georgescu  
Bogdan Iordănescu  
Cornel Negru  
Floriceș Popescu  
Anca Dumitru  
Șerban Silvinschi

### GRAFICĂ

Octavian Penda  
Lucian Mișcă  
Luminița Ciupitu

### ADRESA REDACȚIEI

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

Tiparul a fost executat la  
Tipografia UNIVERSUL



# ADISAN

11, Arh. Ioh. Mincu str.  
Bucurest-ROMANIA  
Phone: 17.74.30

COMERCIAL DEP.

---

**Adisan SRL** este o societate înființată în martie 1990 de un grup numeros de specialiști de vîrf în domeniul tehnicii de calcul și comunicație de date, pentru a oferi tuturor specialiștilor în acest domeniu o alternativă la numeroasele oferte primite din străinătate. Societatea are în prezent 16 filiale în București și în întreaga țară și întreține legături cu un mare număr de parteneri străini dintre care cel mai important este **IBM** (prin **SOCODIAG ASYST - IBM**, Franța, prim agent național în România).

Prin expertii proprii **ADISAN** prospectează în permanență piața mondială de microcalculatoare depistînd cea mai avantajoasă ofertă intermediînd contracte între producătorii de microcalculatoare și beneficiari, negociînd achiziționarea cu plata în lei sau valută în funcție de posibilități.

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

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

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

**Departamentul Comercial** vă pune la dispoziție o ofertă completă în lei și valută de echipamente de calcul și telecomunicații (vezi oferta de la sfîrșitul acestui număr al revistei), începînd cu cele mai pretențioase echipamente din punct de vedere al dinamicii, prețului și calității. Se asigură garanție 1 an și service postgaranție prin contract.

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

**SERIOZITATE—COMPETENȚĂ—PROMPTITUDINE**

## REȚELE DE CALCULATOARE (I)

Ing. Ion Stoica  
Ing. Florin Dobrian  
Ing. Călin Șandru

Unul dintre cele mai ample fenomene observate în ultimul deceniu în domeniul tehnicii de calcul îl constituie dezvoltarea rapidă a rețelilor de calculatoare. Această spectaculoasă evoluție se explică prin importanța pe care o are schimbul informațional și avantajele pe care rețelele le oferă în această direcție.

Informația reprezintă, după cum probabil se cunoaște, o înlăturare a nedeterminării, vehicularea ei rapidă și sigură apărând din ce în ce mai mult ca o necesitate a acestui sfârșit de secol. Calculatoarele pot realiza transferuri informaționale cu performanțe ce asigură prelucrări eficiente. Ele pot fi interconectate rezultând rețele de calculatoare. Aceste rețele au o serie întregă de caracteristici, putând fi însă distinse trei mari categorii și anume:

- **rețele locale (LAN)** - sînt funcționale pe o distanță de cîteva sute de metri și sînt amplasate în instituții sau întreprinderi pentru a implementa suportul informatic al sistemului informațional al respectivei unități economice;
- **rețele metropolitane (MAN)** - sînt amplasate la scara unui oraș și interconectează principalele obiective economice ale orașului respectiv;
- **rețele pe arii întinse (WAN)** - interconectează centrele informatice la nivel continental sau planetar.

Evoluția cea mai rapidă a avut loc în cadrul rețelilor 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 existenței unui singur calculator care poate introduce timpi morți în procesul de rulare în cazul apariției unei defecțiuni. Modelul este înlocuit treptat cu un altul în care mai multe calculatoare cu performanțe 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 relație de tip master-slave ci sînt echivalente și autonome în funcționare. Din punct de vedere al fiabilității se poate observa că aceasta crește în cazul modelului rețea deoarece la defectarea unuia sau mai multor calculatoare rămîn totuși o parte în stare funcțională iar rețeaua în ansamblul ei poate continua rulare diverselor aplicații.

Și din punct de vedere al raportului cost/performanță o rețea locală este avantajoasă față 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 rețea. În același timp prețul este de 1000 de ori mai mare. Rezultă de-

ci clar un raport cost/performanță favorabil rețelilor de calculatoare. Desigur, nu trebuie contestat rolul unui mainframe într-o aplicație specifică în care caracteristicile acestuia sînt puse în valoare.

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

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

Pentru a fixa ideile, să considerăm un exemplu care, prin analogie, să facă înțelese o mare parte din conceptele utilizate în descrierea structurii și arhitecturii rețelilor. Să presupunem că într-o țară de dimensiuni mici, s-o numim Liliput, se pune problema urmării spațiului aerian. Pentru aceasta sînt necesare trei stații radar în punctele A, B, C ca în figura 1. Fiecare stație are un punct (centru) de comandă unde există o hartă a întregii țări pe care se notează poziția avioanelor observate. Apare astfel necesitatea unui sistem de comunicație între cele trei centre de comandă. Poziția avioanelor din sectorul propriu de observare va fi comunicată de stația locală în timp ce a avioanelor din afara spațiului local va fi

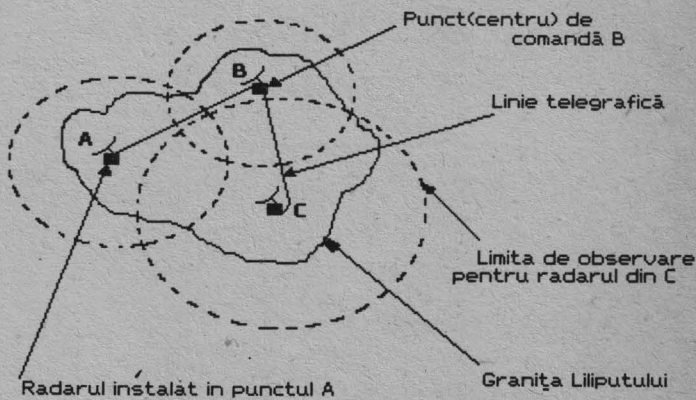


Figura 1.

obținută de la celelalte puncte de comandă. Presupunem că suportul fizic pentru comunicație este o linie telegrafică care conectează punctele de comandă A, B și C. Informațiile care vor fi transmise constau în poziția și eventual traiectoria și viteza avioanelor care sînt observate pe radarul local. Pentru a asigura circulația corectă a fluxului informațional, în fiecare punct de comandă există o structură organizată formată din mai mulți operatori, fiecare cu atribuții specifice. Astfel, un operator urmărește ecranul radarului, un operator notează poziția

pe hartă a avionului, un operator interpretează poziția de pe hartă și formează mesajul ce trebuie transmis și în sfârșit 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 cât mai clari în explicarea conceptelor de bază ale rețelelor de calculatoare.

### STRUCTURA REȚELOR

Atunci când 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 sunt 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 sunt calculatoare specializate și poartă numele de procesoare de mesaje (*IMP - Interface Message Processors*). Liniile de comunicație sunt 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).

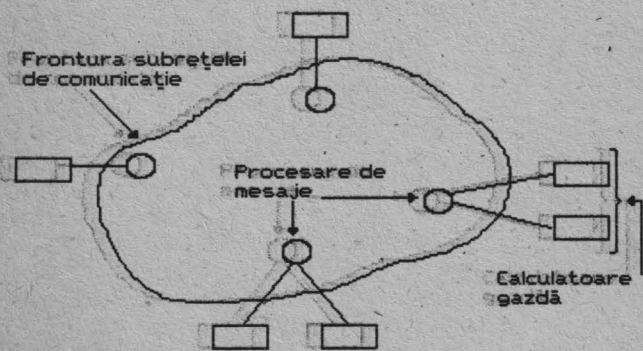


Figura 2.

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 câte o pereche de IMP-uri. Dacă două IMP-uri nu sunt 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 legătură 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.

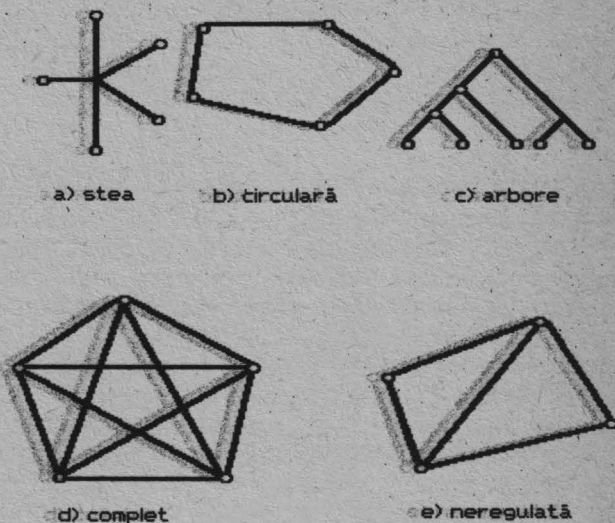


Figura 3.

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.

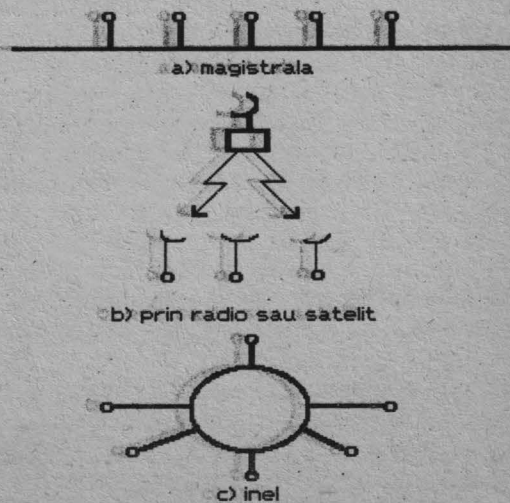


Figura 4.

### ARHITECTURA REȚELOR

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

Rețelele moderne sînt proiectate într-o manieră puternic structurată în vederea reducerii complexității. Această structurare are ca scop împărțirea serviciilor asigurate de rețea în mai multe categorii începînd cu funcțiile de bază privind transportul fizic al unităților de informație, împachetarea acestora pentru formarea unor pachete, rutarea acestor pachete și terminînd cu aplicațiile cele mai generale. Astfel, funcțiile cele mai simple stau la baza altor funcții care le apelează și care sînt la rîndul lor apelate de funcții 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 funcții apelabile nivelelor de deasupra sa, scutindu-le pe acestea din urmă de amănunte privind implementarea acestor servicii.

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

#### La transmisie:

- operatorul 4 urmărește ecranul radarului și comunică coordonatele avionului (într-un sistem de referință local);
- operatorul 3 face transformarea coordonatelor locale în coordonate globale (la nivelul întregii țări) și notează poziția avionului pe hartă;
- operatorul 2 citește poziția de pe hartă și, eventual pe baza pozițiilor anterioare calculează viteza și traiectoria. Apoi transmite într-un mesaj codificat toate datele necesare la operatorul 1;
- operatorul 1 transmite efectiv mesajul pe linia telegrafică.

#### La recepție:

- operatorul 1 preia mesajul recepționat și îl comunică operatorului 2;
- operatorul 2 interpretează mesajul primit (il decodifică) și comunică poziția avionului;
- operatorul 3 notează poziția avionului pe hartă și dacă avionul urmează să intre în spațiul aerian local transmite poziția în coordonate locale operatorului 4;
- operatorul 4 preia urmărirea avionului (eventual modifică parametrii stației pentru a putea observa avionul în condiții optime).

Un nivel de pe o anumită mașină poate conversa cu același nivel de pe o altă mașină. Regulile utilizate în conversație sînt cunoscute sub numele de protocolul nivelului respectiv. Acest protocol folosește serviciile furni-

zate 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 același protocol (adică aceleași scheme de codificare și de decodificare a mesajelor). Dar, ei nu comunică direct, ci se folosesc de protocolul de comunicație de pe nivelul 1. În figura 5 este prezentată o rețea cu șapte nivele.

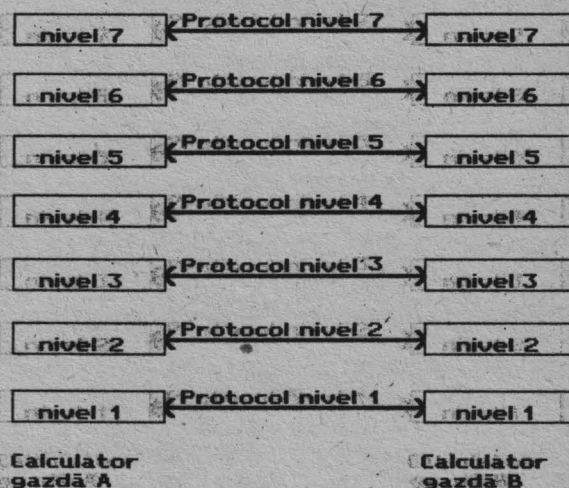


Figura 5

Trebuie înțeles faptul că un astfel de nivel este constituit dintr-un set de funcții implementate, fie în hardware, fie în software pe un anumit calculator din rețea. De exemplu, nivelul 1 poate executa funcții de transportare a biților prin mediul de comunicație, nivelul 2 poate împacheta acești biți în octeți și asigura un anumit grad de siguranță a transmisiei iar nivelele cele mai de sus pot implementa funcții de transfer de fișiere, apeluri de funcții situate pe un alt calculator din rețea etc. Un nivel superior apelează funcțiile puse la dispoziție de un nivel inferior. Structurarea este impusă de complexitatea funcțiilor utilizate. Comunicarea între două nivele adiacente se face printr-o interfață bine stabilită, astfel că este posibilă înlocuirea totală a unui anumit nivel cu condiția respectării riguroase a interfețelor 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 și transmise au exact aceeași formă.

Datele nu se transferă direct între nivelele echivalente de pe două calculatoare gazdă, cu excepția celui mai de jos nivel. La stația 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 comunicație; la recepție același nivel de jos preia datele de pe mediu și le furnizează către nivelele superioare. Astfel, între nivelele de jos are loc o comunicație fizică în timp ce între oricare alte nivele are loc o comunicație virtuală. Comunicația între două nivele adiacente aparținînd aceleiași mașini este realizată cu ajutorul unei interfețe ce definește operațiile primitive și serviciile pe care nivelul inferior le oferă celui superior.

Mulțimea nivelelor și protocoalele asociate acestora formează arhitectura rețelei.

În figura 6 se poate observa fluxul informației între două calculatoare gazdă în cadrul rețelei structurate.

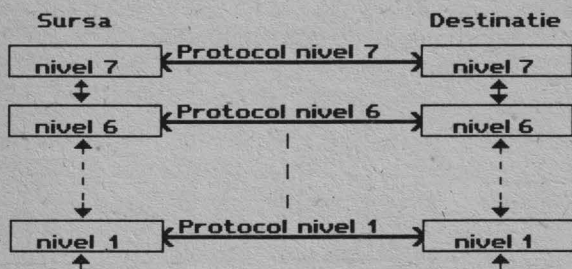


Figura 6

Protocolul unui nivel are la bază o serie de algoritmi care asigură o anumită calitate a comunicației. Pentru ca aceasta să fie ridicată se folosesc diferite metode cum ar fi: împărțirea mesajelor în pachete, numerotarea acestora în cadrul unei secvențe pentru păstrarea ordinii datelor, transmiterea mesajelor pachet cu pachet, confirmarea recepției fiecărui pachet de date, retransmisia pachetelor în cazul pierderii lor pe mediul de comunicație sau în cazul nerecepționării acestora de către stația destinație. 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 recepționează mesajul poate confirma recepția lui dacă l-a primit în bune condițiuni sau poate cere retransmisia lui dacă mesajul este neînțeles. De notat că, recuperarea erorilor de transmisie se poate face la mai multe nivele. Astfel, telegrafistul (nivelul 1) poate cere retransmisia mesajului dacă primește litere sau cuvinte incorecte, sau operatorul 2 poate cere retransmisia mesajului dacă conținutul mesajului nu este inteligibil (lipsesc cuvinte, simbolii etc.).

### INTERCONECTAREA REȚELOR

O dată cu creșterea numărului de rețele locale (LAN) a apărut firesc necesitatea de a conecta mai multe LAN-uri între ele. Problema interconectării nu are o rezolvare generală datorită marii diversități de tipuri de rețele existente. Astfel, fiecare mare producător de calculatoare are propria lui rețea. Acestea diferă între ele atît ca protocoale utilizate, cît și ca viteză de transmisie și 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 rețele locale cu aceleași caracteristici. Pentru a reprezenta acest tip de interconectare, în exemplul nostru vom face unele modificări. Presupunem că avem o țară cu o întindere mai mare împărțită în regiuni. Spațiul aerian al fiecărei regiuni este supravegheat de mai multe radaruri conectate ca în Liliput. Comunicarea între două centre de comandă aparținînd la două regiuni diferite se face printr-un punct de comuni-

cație care în principiu va avea caracteristicile unui bridge. Acesta este format dintr-un singur telegrafist care are acces la rețelele de comunicație ale celor două regiuni. Acesta recepționează 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 și îl retransmite pe cealaltă rețea pentru a ajunge la centrul de comandă destinație.

#### 2. Router

Router-ul se folosește la interconectarea a două rețele care respectă același standard, dar cu implementări care pot fi diferite. De exemplu, pot diferi prin lungimea și formatul mesajelor trimise. Router-ul trebuie să asigure transformarea mesajelor între rețelele pe care le interconectează. Deci, mesajele primite de la o rețea sînt modificate pentru a putea fi recunoscute și manipulate corect în cealaltă rețea. Revenind la exemplul nostru, să presupunem că, Liliput împreună cu o țară vecină, s-o numim Blefuscu, organizează supravegherea spațiului aerian comun. În Blefuscu observația este organizată similar ca în Liliput. În acest caz, se folosește un centru de comunicație 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 rețele cu protocoale care se conformează aceluiași standard. Aceasta nu se poate dacă există o rețea deja instalată cu o arhitectură proprie. Datorită investițiilor deja făcute în softul de comunicație nu se poate pune problema schimbării acestuia. Această problemă este rezolvată printr-un dispozitiv numit Gateway care permite coexistența rețelelor bazate pe standarde de comunicație diferite. Pentru a clarifica ideile putem considera exemplul anterior, dar cu anumite amendamente. Să presupunem că Blefuscu are un alt mod de supraveghere a spațiului aerian (sisteme de coordonate diferite, modalități de transmisie diferite etc). Nu vom detalia aici acest sistem de organizare deoarece este lipsit de importanță. În acest caz centrul de comunicație va avea o organizare mai complexă: el va cuprinde toată organizarea ierarhică regăsită într-un centru de comandă din Liliput și, în paralel, într-un centru de comandă din Blefuscu. O informație transmisă din Liliput către Blefuscu va fi recepționată de telegrafist (operatorul 1), care o va comunica operatorului 2 și așa mai departe pînă la operatorul 4. La acest nivel, informația va fi transmisă către nivelul superior al structurii corespunzătoare organizării din Blefuscu. Informația preluată de acesta este prelucrată și apoi transmisă în modul specific protocoalelor și modalităților de comunicație din Blefuscu către nivelele inferioare.

### ÎN LOC DE ÎNCHEIERE

Acest articol se adresează tuturor aceluia care doresc o inițiere în lumea rețelelor de calculatoare. Intrarea în această lume depinde în ultima instanță de gradul de cunoaștere și 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 rețeaua publică, avînd acces la diverse baze de date, putînd transfera programe și avînd posibilitatea de a lucra acasă.



## BAZELE LOGICE ALE INTELIGENȚEI ARTIFICIALE

Prof.univ.dr. Cornel Popa

**D**acă silogistica aristotelică a fost un organon sau un instrument pentru analiza unor raționamente din viața cotidiană, logica matematică modernă este un organon pentru analiza limbajelor științifice și a bazelor de cunoștințe din programele expert și de inteligența artificială.

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

### 1. Formalizare, capacitate inferențială și inteligență

Orice ființă umană își prețuiește puterea de judecată. Fiecare dintre noi acceptăm mult mai ușor că, într-o împrejurare sau alta, nu a deținut informația necesară, decât că a deținut-o, dar nu am avut capacitatea logic-rațională de a degaja din ea concluziile necesare. Sîntem ființe orgolioase ce nu acceptă nici o ofensă adusă competenței intelectului propriu, capacității lui de a prelucra adecvat informația dată.

Aptitudinea de a deduce și de a calcula, capacitatea de a degaja consecințe cît mai îndepărtate și neașteptate este indubitabil un atribut al inteligenței.

În afară de capacitatea de a infera sau deduce, inteligența presupune chiar înainte de aptitudinea deductiv-calculatorie, abilitatea de a reprezenta intern în imagini subiective sau configurații lexicale, relații și structuri din mediul extern. Înainte de a calcula și de a infera, trebuie să știm să codificăm realitatea în sisteme de ecuații sau structuri semiotice.

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

- R1.** 1. Dacă ninge și plouă, mă duc la munte.  
2. Dacă mă duc la munte nu citesc  
3. Citesc sau joc șah.  
4. Dar eu nu joc șah.  
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 iubiți de toți.  
2. Cei ce sînt păziți se tem de cineva.  
3. Șefii Mafiei sînt păziți.  
4. Șefii Mafiei nu sînt iubiți de toți.

- R4.** 1. Ion e fericit, dacă îl iubește cineva.  
2. Ana iubește pe oricine o iubește.  
3. Ion o iubește 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 obținute la punctul a);  
c) Să decidem prin metoda rezoluției asupra raționamentelor date.

Admitem, deci, că pentru a putea întreprinde un calcul logic sau aritmetic, trebuie mai întîi să reprezentăm și să simbolizăm o problemă într-un anumit limbaj formal, trebuie să transpunem obiectele și relațiile sesizate într-un domeniu empiric, în simboluri, relații și operații proprii unei discipline formale. Rezolvitorii de probleme știu că alegerea cadrului de reprezentare, simbolizarea și determinarea ecuațiilor sînt momentele inițiale, mai dificile, în drum spre găsirea soluției. Pentru a putea face o traducere dintr-o limbă în alta, trebuie să cunoști ambele limbi. Dar mai ales limba în care traduci.

În ceea ce ne privește, în rîndurile care urmează, limbajul în care vom face traducerea unor fapte, enunțuri 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 definește prin alfabet și reguli de formare. Mai ortodox, acestea din urmă, pot fi redată și ca reguli de rescriere sau producțiuni.

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

$A_C = \{a, b, c, \dots\}$	constante individuale
$A_V = \{x, y, z, x_1, \dots\}$	variabile individuale
$A_F = \{f, g, h, f_1, \dots\}$	semne pentru funcții sau operații sau semne funcționale
$A_P = \{P, Q, R, P_1, \dots\}$	semne predicative.
$A_{CL} = \{\neg, \&, \vee, \supset, \equiv, \dots\}$	conective logice
$A_Q = \{\forall, \exists\}$	cuantificatori
$A_G = \{(, ), [, ], \{, \}\}$	semne de grupare

Este util, probabil, să amintim că alfabetul constantei individuale  $A_C$ , va servi pentru redarea numelor proprii sau a descrițiilor singularizatoare. Variabilele individuale vor fi folosite pentru a desemna obiecte individuale oarecare dintr-un domeniu determinat, iar semnele funcționale din alfabetul  $A_F$  vor servi pentru a codifica operații și funcții matematice.

Prin semnele din alfabetul  $A_P$  vom codifica proprietăți și relații instituite între diferite mulțimi de obiecte individuale.

Demn de menționat este faptul că simbolurile din  $A_F$  și  $A_P$  denotînd operații sau relații au întotdeauna un număr  $0 \leq n < \infty$  de argumente ce descriu aritatea funcției sau relației în cauză. Dacă  $f \in A_F$  și  $P \in A_P$ , atun-

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

Conectivile logice nu caracterizează obiecte din lumea empirică, ci sînt instrumente de asamblare și analiză a formulilor ce descriu propoziții declarative, predicate sau formule deschise.

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

Definirea limbajului logicii: predicatelor presupune introducerea succesivă a conceptelor de termen, formulă atomară și 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  $\alpha \in T$ , mulțimea termenilor.

**R2.** Variabilele individuale sînt termeni.

Dacă  $\alpha \in A_{VI}$ , atunci  $\alpha \in T$ .

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

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

*Exemple.*  $a, x$  și  $y$  vor fi termeni în virtutea regulilor **R1** și **R2**. Dacă  $f, g, h \in A_F$  și  $\delta(f)=1$ ,  $\delta(g)=2$ ,  $\delta(h)=2$  și  $a, x$  și  $y$  sînt termeni, atunci vor fi termeni, pe baza regulii **R3**, și 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. $a$                | <b>R1</b>     |
| 2. $x$                | <b>R2</b>     |
| 3. $y$                | <b>R2</b>     |
| 4. $f(x)$             | <b>R3,2</b>   |
| 5. $g(a, y)$          | <b>R3,1,3</b> |
| 6. $h(f(x), g(a, y))$ | <b>R3,4,5</b> |

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

Dăm mai întîi două reguli.

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

Semnul '=' este un element privilegiat din  $A_P$  și desemnează relația de identitate referențială sau egalitate.

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

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

Atomi se obțin, așadar, din aplicarea unor semne predicative de aritate  $n$  la șiruri de  $n$  termeni.

Ecuatiile matematice sînt predicate binare formate în concordanță cu regula **R4**.

În manualele clasice de logică semnele predicative sînt redade 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 redade prin cuvinte scrise cu inițială majusculă; variabilele sînt redade 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 conține numai constante, atunci vom spune că a fost *instanțiatizat* sau interpretat într-un domeniu oarecare. Un atom instanțiatizat poate fi adevărat sau fals.

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

Pornind de la atomii 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 inițializarea definiției inductive a noțiunii de formulă în logica predicatelor. Regulile următoare extind mulțimea formulilor predicative.

**R7.** Dacă  $\alpha$  este o formulă predicativă, atunci  $\neg\alpha$  este și ea o formulă predicativă bine formată.

**R8.** Dacă  $\alpha$  și  $\beta$  sînt formule predicative și  $*$  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 și fără negație, se numesc *literali*.

**DC.** Clauzele sînt mulțimi de literali legați prin disjuncție.

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

Ultima regulă ne va permite formarea unor formule cu cuantificatori.

**R9.** Dacă  $\alpha$  este o formulă bine formată în logica predicatelor și  $Q$  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  $\exists y \exists x (S(x, y, z) \& R(x, u))$  va fi o formulă bine formată cu cuantificatori, care descrie un predicat  $P(z)$ .

**DL. Pred.** Numim *limbaj al logicii predicatelor* formulele obținute 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. Reguliile R1-R3 conduc la formarea termenilor sau a expresiilor denotative.

Obs.2. Reguliile R4 și R5 permit construirea atomilor sau a formulelor predicative elementare.

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

Obs.4. Când un semn funcțional 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, când un semn predicativ  $P$  este de aritate zero, i.e.  $\delta(P)=0$ , atunci  $P$  devine o variabilă proporțională ce poate lua valorile de 1 sau 0 (adevărat sau fals).

Obs.5. Limbajul logicii predicatelor și cu atât mai mult sublimbajele termenilor și al logicii propozițiilor pot fi captate în cadrul unei gramatici generative independente de context sau în cadrul unei gramatici analitice de același tip.

Predicatul este o formulă deschisă în care cel puțin o variabilă individuală apare nelegată de vreun cuantificator. Predicatul mai este numit și funcție propozițională predicativă sau scheme cuantificatoriale. Ele pot fi privite ca structuri sau cadre generatoare de propoziții 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 propoziții complexe din limbile naturale și din teoriile științifice.

### 3. Formalizarea unor raționamente în limbajul logicii propozițiilor

Să ne întoarcem la raționamentele R1-R4 date la început. Primul raționament este formalizabil în limbajul logicii propozițiilor, 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 restrâns prin postularea absenței oricărui argument pentru variabilele  $P, Q, R, S$  i.e.  $\delta(P) = \dots = \delta(S) = 0$ . Astfel  $P, Q, \dots$  decad din statutul de variabile predicative la statutul de variabile propoziționale; vor ține locul unor propoziții oarecare ce pot fi adevărate sau false;  $A_{CL}$  vor desemna conectivele logice  $\neg, \&, \vee, \supset, \equiv$ , iar  $A_G$  vor fi semne de grupare.

Reguliile de formare utilizate vor fi R6, R7 și R8, cu "degenerarea" provocată căderea variabilelor predicative la starea de variabile propoziționale.

Propozițiile "și" și "sau" din limba română vor fi redată prin semnele conjuncției "&" și corespunzător al disjuncției " $\vee$ "; sintagma "dacă..., atunci" va fi redată prin implicație.

Pe această cale R1 devine:

1.  $(P \& Q) \supset R$
2.  $R \supset \neg S$
3.  $S \vee T$
4.  $\neg T$

5.  $P$

6.  $Q$

Este ușor de văzut că propozițiile "ninge", "plouă", "mă duc la munte", "citesc", "joc șah" au fost redată, în ordine, prin variabilele  $P, Q, R, S, T$ .

Cerința a) a fost satisfăcută pentru R1. Întrucât clauzele sînt disjuncții de literali, va trebui să înlocuim formulele ce conțin implicații cu formule echivalente cu ele ce nu conțin implicații. Vom înlocui, de asemenea, formulele ce conțin semnul negației în fața unor conjuncții sau disjuncții, cu altele echivalente cu ele ce conțin semnul negației pe variabilele. Pentru aceasta vom folosi echivalențele:

$$\begin{aligned} A \supset B &\equiv \neg A \vee B \\ \neg(A \& B) &\equiv \neg A \vee \neg B \\ \neg(A \vee B) &\equiv \neg A \& \neg B && \text{legile lui De Morgan} \\ \neg\neg A &\equiv A && \text{legea dublei negații} \end{aligned}$$

Obținem astfel clauzele:

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

R1 și R1b sînt transpuneri clauzele a raționamentului R1. Am satisfăcut astfel cerința b).

Pentru a satisface cerința c), adică a decide asupra unei formule prin metoda rezoluției, trebuie să știm mai întîi ce este principiul rezoluției și în ce constă metoda rezoluției.

### 4. Principiul rezoluției

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

$$res_l(k_1, k_2) = k_1 \setminus \{ l \} \cup k_2 \setminus \{ \neg l \} \quad (4.1)$$

Rezolventul în  $l$  din  $k_1$  și  $k_2$  este clauza ce conține literalii din  $k_1$  și literalii din  $k_2$ , după ce s-au omis  $l$  din  $k_1$  și  $\neg l$  din  $k_2$ .

Exemplu.

$$\begin{aligned} k_1 &= \{ P, Q \} \\ k_2 &= \{ \neg P, R \} \\ \text{resp } res_l(k_1, k_2) &= \{ P, Q \} \setminus \{ P \} \cup \{ \neg P, R \} \setminus \{ \neg P \} = \{ Q \} \cup \{ R \} \\ &= \{ Q, R \} \end{aligned}$$

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

Ori de câte 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  $\neg l \in k_2$ . Dacă  $k_1$  și  $k_2$  sînt adevărate, atunci și rezolventul lor,  $res_1(k_1, k_2)$ , va fi adevărat.

*Demonstrație*

- |   |                |
|---|----------------|
| 1. $k_1 \& k_2$   | ip.            |
| 2. $l \in k_1$  | ip.            |
| 3. $\neg l \in k_2$                                     | ip.            |
| 4. $l = 0$  | ip. supl.      |
| 5. $k_1$  | (E&, 1)        |
| 6. $k_2$  | (E&, 1)        |
| 7. $k_1 \setminus \{l\}$                                | (5, 4, v)      |
| 8. $k_1 \setminus \{l\} \cup k_2 \setminus \{\neg l\}$  | (v, 7)         |
| 9. $res_1(k_1, k_2)$                                    | (RE, 8, 4, l)  |
| 10. $l = 1$   | ip. supl.      |
| 11. $\neg l = 0$  | (def. neg.)    |
| 12. $k_2 \setminus \{\neg l\}$                          | (6, 11, v)     |
| 13. $k_1 \setminus \{l\} \cup k_2 \setminus \{\neg l\}$ | (v, 12)        |
| 14. $res_1(k_1, k_2)$                                   | (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. Trecherile 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:

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

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

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

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*

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

*modus tollens*

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

*tranzitivitatea*

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

### Exerciții de logica propozițiilor

I. Se dau formulele:

- $(p \supset (q \supset r)) \supset (p \supset q) \supset (p \supset r)$
  - $(p \supset r) \supset ((q \supset r) \supset ((p \vee q) \supset r))$
  - $((q \supset s) \& (p \supset q) \& (p \vee r) \& (r \supset t) \& (t \supset u)) \supset (s \vee u)$
- 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 aceluiași 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*

- Dacă alerg, transpir.
  - Daçă joc șah, devin sedentar.
  - Dacă transpir, răcesc.
  - Alerg sau joc șah.
  - Dacă devin sedentar, mi se atrofiază mușchii.
6. Prin urmare, răcesc sau mi se atrofiază mușchii.

B) *Dilemă 2*

- Dacă fac sport, cheltuiesc energie.
  - Dacă cheltuiesc energie, nu devin obez.
  - Fac sport sau privesc opere de artă.
  - Dacă nu devin obez, mă plac fetele.
  - Dacă privesc opere de artă, mă cultiv artistic.
  - Dacă mă plac fetele, sînt fericit.
7. Prin urmare, sînt fericit sau mă cultiv artistic.

## REȚELE NEURONALE

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

Deși 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 atât progreselor tehnologice cât și spectaculoasei dezvoltări a software-ului înregistrate în această perioadă. Puternica renaștere a acestei direcții de vîrf a cercetării a stîrnit vii dezbateri și controverse printre specialiști. Ce sînt aceste neurocalculatoare? Sînt ele capabile să concureze creierul uman? Vor înlocui oare calculatoarele tradiționale? Iată cîteva întrebări la care încercăm să răspundem printr-o serie de articole, deschisă în acest număr al revistei.

### SURSA DE INSPIRAȚIE ESTE MODELUL BIOLOGIC

Întrebat la ora actuală despre creierul uman, nici un psiholog sau neurochirurg nu ar putea da o imagine globală despre funcționarea acestuia. Problema este atât de complexă încît rezolvarea ei completă mai trebuie să aștepte încă foarte mult timp. Anumite lucruri se cunosc totuși. Se știe în general că sistemul nervos este alcătuit din celule numite neuroni între care există o rețea de conexiuni. Neuronii conțin un anumit număr de dendrite și 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 ieșirea. Așadar, un neuron are mai multe intrări și o singură ieșire. Cu toate acestea, axonul se ramifică, astfel că neuronul poate conecta ieșirea sa la mai multe intrări ale altor neuroni. O sinapsă se realizează între axonul unui neuron și 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 ieșire, către ceilalți neuroni cu care este conectat. Condiția de excitare a neuronului este ca impulsurile de la intrare să depășească un anumit prag de sensibilitate specific celulei nervoase.

Determinant pentru modul de funcționare a creierului este gradul foarte ridicat de paralelism în procesarea informațiilor. Astfel, creierul uman conține în jur de zece miliarde de neuroni, fiecare cu peste o mie de sinapse. Această organizare îi permite să prelucreze informații cu o viteză foarte ridicată, deși 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).

### REȚELE NEURONALE ARTIFICIALE

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

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

Prin analogie cu neuronul biologic, un neuron artificial are un anumit număr  $N$  de intrări (corespondente dendritelor) și o singură ieșire (corespondentă axonului), dar care se poate conecta la intrările mai multor neuroni. Fiecare intrare are asociată o pondere care reprezintă importanța pe care un impuls prezent la intrarea respectivă o are în activarea neuronului. Conexiunile se realizează prin legarea ieșirilor 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 și în neuronul artificial. El este notat  $\theta$ . O altă caracteristică a neuronului este comportarea acestuia, definită printr-o funcție 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  $\theta$  pentru a determina dacă combinația de la intrare a depășit această valoare. În funcție de rezultatul calculului precum și de tipul funcției de activare caracteristice neuronului, acesta generează un semnal  $Y$  la ieșire. În figura 1 este ilustrat un neuron, cu notațiile corespunzătoare.

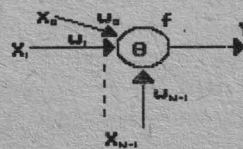


Figura 1

Ieșirea se generează conform formulei:

$$Y = f\left(\sum_{i=1}^{N-1} W_i * X_i - \theta\right)$$

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

- funcția treaptă;
- funcția rampă;
- funcția sigmoidală;

Variația acestor trei modele de funcții neliniare de activare este prezentată în graficele din figura 2.

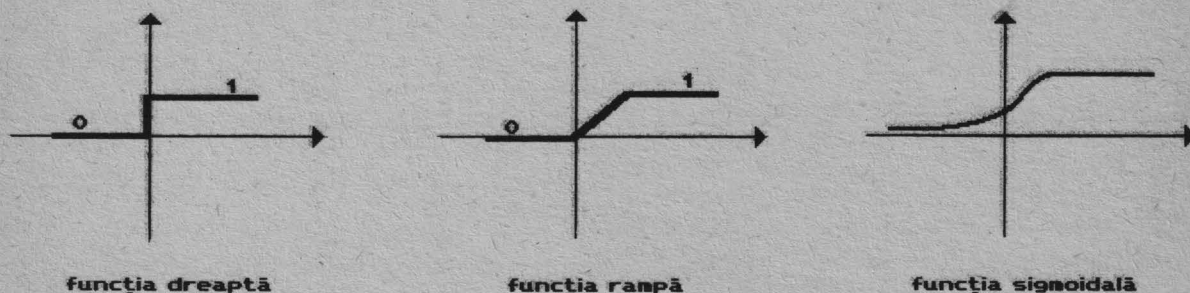


Figura 2

În cadrul unor rețele neuronale complexe se pot întâlni variații temporale ale parametrilor precum și operații matematice mai complicate în locul simplei sumări.

Fiecare element de procesare dintr-o rețea neuronală are așadar un număr de intrări și o ieșire care este funcție de intrări și care este utilizată ca intrare de către alte elemente. Aceste elemente de procesare sînt organizate în mai multe straturi (nivele). Introducerea informațiilor în rețea se face prin intermediul unui strat de

neuronii de ieșire. Între aceste două straturi care asigură comunicarea cu mediul înconjurător este posibilă existența unor straturi intermediare numite și straturi ascunse. Legăturile între neuronii din diversele straturi sînt specificate printr-o matrice de conexiuni ponderate. În funcție de modurile de interconectare se pot deosebi o serie întreagă de topologii neuronale. Structura generală a unei rețele neuronale este ilustrată în figura 3.

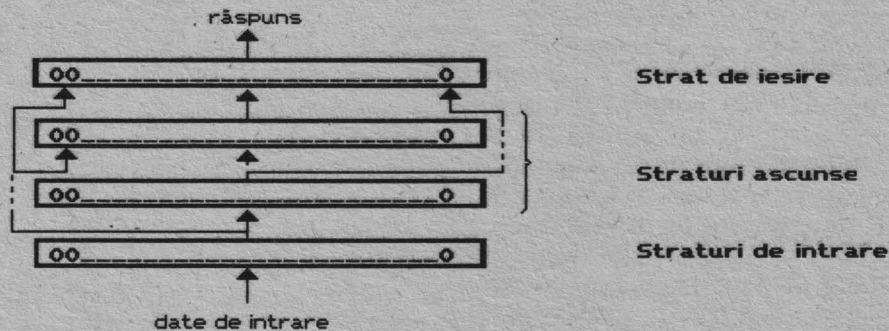


Figura 3.

Primele modele de rețele neuronale studiate conțineau numai unul sau două straturi de neuroni. În primul caz, singurul strat existent ținea loc atât de strat de intrare cît și de strat de ieșire. Aceste prime implementări limitau rolul respectivelor rețele. Ulterior s-au construit modele cu mai multe straturi. Adăugarea de straturi ascunse permite sistemelor concepute să-și formeze o reprezentare internă a problemelor analizate. Aceasta înseamnă că în urma prezentării unor date la intrare, o rețea neuronală își organizează intern informațiile într-un anumit mod.

Rețelele cu un singur strat rezumau această reprezentare internă numai la forma sub care erau prezentate informațiile la intrare. Rețelele 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 rețele, cu cîte unul, două și trei straturi.

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

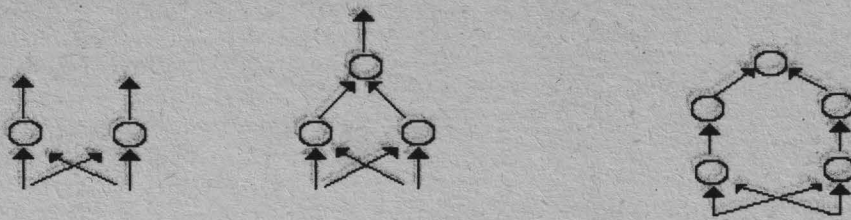


Figura 4.

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 corepondent 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 dezvoltate o serie întregă 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 toate 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 rețelilor 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

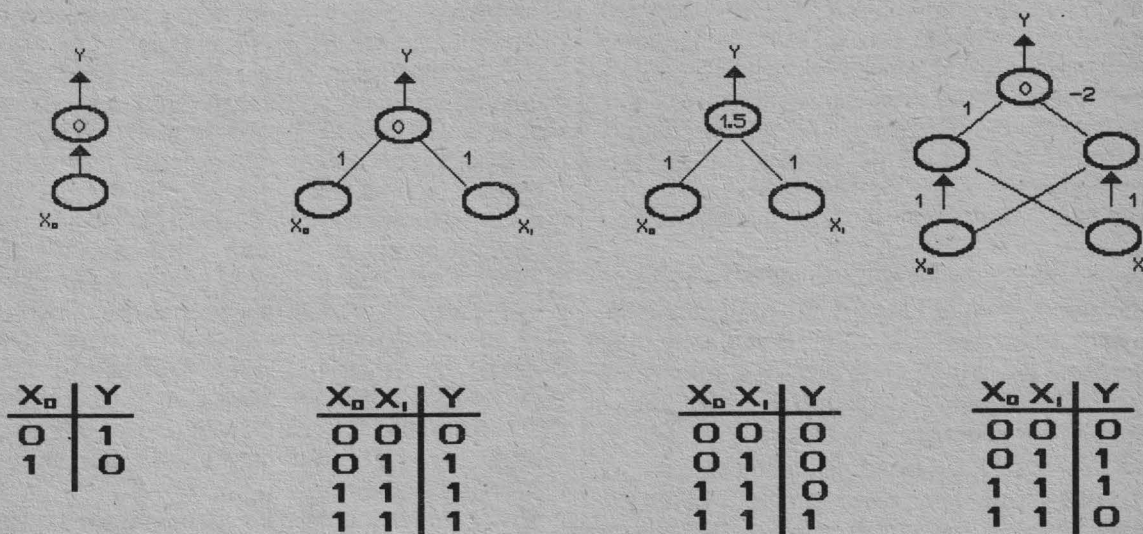


Figura 5.

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:

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

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.

**Funcție de activare** — Funcție matematică după care se calculează valoarea ieșirii unui element de procesare.

**Învățare** — Etapa în care unei rețele neuronale i se introduc date primare în vederea adaptării ponderilor.

**Neuron** — Unitate structurală și funcțională a sistemului nervos.

**Pondere** — Valoare reală care exprimă importanța unei intrări a neuronului.

**Prag** — Nivel energetic de excitație minim.

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

**Strat ascuns** — Strat de neuroni situat între stratul de intrare și cel de ieșire.

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

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

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

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

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

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

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

## 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 avansați care au cunoștințe de programare (eventual în alte limbaje de programare) și 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 sfătuiți să învețe să lucreze întii în limbajul Pascal pentru a căpăta o educație corectă în domeniul programării.

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

### Utilizarea pointer-ilor în limbajul C

Înțelegerea și utilizarea corectă a pointer-ilor este una dintre cheile realizării unor programe C corecte și eficiente. Utilizarea pointer-ilor este necesară pentru a realiza transferul parametrilor funcțiilor prin referință și 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 existența mecanismului de adresare prin pointer-i poate să conducă la apariția 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 ușura explicarea programului am numerotat în comentarii liniile din corpul acestuia. Să urmărim acest program. În linia 1 sînt declarate două variabile reale x și y, și se realizează inițializarea 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 primește valoarea 3.14 (valoarea variabilei x). În linia 5 se afișează valoarea variabilei y. Dacă veți încerca să executați acest program (după ce veți ignora avertismentul referitor la linia 3, în care se atribuie un pointer către un int adresa unei valori de tip float), veți constata că valoarea afișată nu este corectă (se va afișa valoarea -2621.000000). Explicația este simplă: în linia 4 datorită modului diferit în care sînt reprezentate valorile de tip int față de cele de tip float conținutul 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 neinițializat nu pot să fie diagnosticate la fel de simplu la nivelul compilatorului.

Să considerăm și un exemplu tipic de utilizare a pointer-ilor: un program pentru afișarea în baza 16 a conținutului unei zone de memorie :

```
/*
dump.c
Program pentru afișarea unei zone de memorie
începînd de la o adresa specificată de către utilizator.
Se afișează 256 de octeți.
*/
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 < 256; 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 primește ca valoare adresa începînd de la care se dorește preluarea octeților pentru afișare. 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 neinițializat. În linia 8 variabila p primește 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 afișarea pe rând în baza 16 a 256 de octeți începînd de la adresa indicată de utilizator. Pe fiecare linie se afișează 16 valori.

### Utilizarea pointer-ilor pentru transferul parametrilor

În limbajul C mecanismul de transfer al argumentelor pentru funcții este așa numitul transfer prin valoare. Adică, ceea ce se transmite unei funcții pentru fiecare argument actual este valoarea acestuia pe care funcția îl copiază în contextul său. Din acest motiv orice modificare suferită de un astfel de argument în cadrul funcției se referă numai de la copia sa și nu are nici un efect în afara funcției. Avînd în vedere că de multe ori este necesară realizarea unei actualizări pentru un argument în cadrul unei funcții este necesar un mecanism prin care funcția să aibă acces la argumentul propriu-zis și nu numai la valoarea sa. În alte limbaje de programare (**PASCAL**, **FORTTRAN**) acest mecanism se realizează prin transferul argumentelor prin referință. În acest caz ceea ce se transmite unei funcții este adresa argumentului, pe baza căruia funcția are atît posibilitatea să determine valoarea curentă a argumentului cît și 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 demonstrație pentru transferul argumentelor funcțiilor
*/
/* 1 */ void test(int *ip);
void main()
{
/* 2 */ int i = 50, *ip = &i;
/* 3 */ printf("\ni = %i", i);
/* 4 */ test(ip);
/* 5 */ printf("\nDupă execuție test \ni = %i", i);
}
/* 6 */ void test(int *int_p)
{
/* 7 */ *int_p = 100;
}
```

Dacă veți executa programul **test.c** veți obține :

i = 50

După execuție test

i = 100

Să urmărim puțin programul. În linia 6 au fost declarate: tipul funcției **test** și tipul argumentului. Se observă că argumentul este un *pointer* către o valoare de tip *int*. În linia 7 valoarea argumentului (a cărui adresă este conținută în argumentul *int\_p* transmis funcției) este modificată, aceasta primind valoarea 100. În cadrul programului principal în linia 2 se face inițializarea și declararea variabilelor *i* de tip *int* cu valoarea 50 și *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 funcției **test**, valoarea variabilei *i* se modifică conform efectului funcției **test**. Să modificăm programul **test.c** renunțînd la argumentul de tip *pointer*. Se obține programul **test1.c**

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

Pe ecran se va afișa :

i = 50

După execuție test

i = 50

deoarece în acest caz modificarea valorii variabilei *i* în funcția **test1** nu are nici un efect în afară funcției.

O altă situație 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 demonstrație pentru transferul argumentelor vectori
*/
/* 1 */ void copystr(char *destinație, char *sursă);
void main()
{
/* 2 */ char *s1 = "primul șir";
/* 3 */ static char s2[] = {"al doilea șir"};
/* 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 șir");
/* 10 */ printf("%s\n", s3);
}
/* 11 */ void copystr (char *destinație, char *sursă)
{
/* 12 */ while((*destinație++ = *sursă++) != '\0')
/* 13 */ ;
}
```

Ca efect al execuției acestui program se afișează :

```

primul șir
al doilea șir
al treilea șir
    
```

Funcția **copyst** este definită ca avînd două argumente de tip *pointer* către valori de tip *char*. În ciclul conținut în liniile 11 și 12 se face copierea caracterului curent a cărui adresă este valoarea curentă a variabilei sursă la adresa care este valoarea curentă a variabilei destinație. După referirea (obținerea valorii) fiecareia dintre aceste variabile se face incrementarea valorilor variabilelor. Fiind vorba de valori de tip *pointer*, incrementarea se face conform spațiului de memorie ocupat de tipul de date către care indică variabila *pointer* respectivă. Ciclul se repetă pînă cînd se ajunge la sfîrșitul șirului indicat de terminatorul de șir de caractere (un caracter cu valoarea \0). În cadrul programului principal sînt realizate trei apeluri ale funcției **copyst**. 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 funcției. Al doilea apel utilizează variabila *s2* care ca și *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 șir". Valoarea acestei expresii este un *pointer* către șirul constant "al treilea șir".

### 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];
/* 8 */     ++int_ptr)
/* 9 */     printf("%d ", *int_ptr);
}
    
```

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

```

Parcurgere fără pointeri
1 2 3 4 5
    
```

Parcurgere utilizînd pointeri  
1 2 3 4 5

Apărent, cele două parcurgeri sînt echivalente. Dar să urmărim puțin care sînt operațiile care se execută de fapt pentru ciclurile descrise în liniile 4, 5 și respectiv 7, 8. Pentru reprezentarea întregilor limbajul C utilizează 2 octeți. Pentru a realiza accesul la valoarea unui element din vectorul *data* se realizează următoarele operații :

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

Deci, pentru fiecare element de vector se realizează două citiri din memorie, o înmulțire (eventual deplasare) și 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 obținerea valorii curente a *pointer*-ului *int\_ptr* și o citire utilizînd această valoare pentru a obține 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 inițializare este adresa primului element din vector. În loc de instrucțiunea:

```
int_ptr = data;
```

se putea utiliza instrucțiunea:

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

Valoarea de comparație finală este adresa ultimului element. Deoarece și numele vectorului este considerat ca fiind de tip *pointer* o notație echivalentă pentru valoarea de comparație este

```
int_ptr data + 4,
```

și nu

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

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

### Operații asupra pointerilor

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

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

```

/*
lungime.c
Program pentru ilustrarea operațiilor cu pointeri
*/
/* 1 */ int lungime (char *sir);
/* 2 */ void main()
{
/* 3 */ printf ("Lungimea șirului : '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 execuției programului **lungime.c** se afișează textul :

Lungimea șirului : 'Asta este ' este 9.

Funcția **lungime** primește ca argument un *pointer* către un șir de elemente de tip *char*. În cadrul procedurii utilizând *pointer*-ul *ptr* (care de asemenea este un *pointer* către *char*) se parcurge șirul de caractere primit ca argument căutându-se sfârșitul acestuia. În linia 8 se calculează diferența a doi *pointer*-i. Pentru ca rezultatul să fie corect *pointer*-ii trebuie să indice către același tip de date. Valoarea diferenței 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 împărțit automat la valoarea *sizeof(tip)* obținându-se rezultatul corect.

## Pointer-i către funcții

În cazul unui *pointer* către o funcție compilatorul trebuie să știe nu numai că *pointer*-ul adresează o funcție dar și tipul acesteia. De exemplu, declarația

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

declară un *pointer* către o funcție care întoarce un rezultat de tip *int* și nu are argumente. Parantezele în care este inclus *pointer*-ul *(\*fp)* sînt necesare pentru a diferenția această declarație de declararea unei funcții 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 **comparație.c**.

```

/*
comparație.c
Program pentru ilustrarea utilizării pointerilor către
funcții
*/
/* 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("\nIntroduceți două șiruri :\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 șiruri sînt ");
/* 12 */ if (!(*cmp)(a,b))
printf("egale");
/* 13 */ else printf("diferite");
}
    
```

Declararea în linia 2 a funcției **strcmp** anunță compilatorul că numele **strcmp** este un nume de funcție care întoarce o valoare de tip *int*. Actualizarea valorii *pointer*-ului *p* se face prin instrucțiunea:

```
p = strcmp
```

(atenție: numele funcției apare fără paranteze, altfel apariția în membrul drept al operației de atribuire a numelui funcției urmat de paranteze este interpretată ca un apel de funcție). Se observă că nu se utilizează notația:

```
p = &strcmp;
```

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

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

Din exemplul considerat nu rezultă poate foarte clar care sînt avantajele utilizării *pointer*-ilor către funcții. Să ne imaginăm însă scenariul (cuvînt la modă) în care în același program doream să realizăm și compararea unor vectori avînd elemente de alte tipuri. În acest caz funcția **compara** poate să rămînă neschimbată și se definesc funcții de comparație corespunzătoare tipurilor elementelor vectorilor.

## EDITOARE ȘI FONTURI (IV)

Ion Paraschiv și 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 corespondență biunivocă pixel ecran — pixel hârtie.

Sistemul de editare **Mattex** a fost conceput în urma studierii experienței dobândite de către operatoarele mașinilor de editat "Xerox 860" în realizarea concretă de texte matematice complexe. (Un text complex necesită multe semne grafice distincte — de regulă câteva sute — deci mai multe fonturi, precum și posibilități de a realiza exponenți, indici și suprapuneri de semne; toate acestea pe lângă cerințele "clasice": centrări de rânduri, sublinieri sau bolduri de semne, tabulări.)

Odată cu schimbarea mașinilor "Xerox 860" cu aparatură IBM PC compatibilă, prin acest sistem de editare s-a încercat, pe cât posibil, păstrarea obiceiurilor precum și abordarea în același stil "clasic" a activităților specifice aparatului existente.

Având la dispoziție o configurație **Compact AT** + monitor **VGA** + imprimantă laser **Xerox 4045**, sistemul de editare a fost creat special pentru această configurație, deci în prezent nu este portabil. Poate fi adaptat pentru monitoare **EGA** sau **CGA** (dar în acest ultim caz finețea 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 atașează familia de fonturi pentru ecran și pentru imprimantă (o familie de fonturi conține un font-imprimantă, precum și fonturile-ecran reduse 1/2, 1/3, 1/4 corespunzătoare, și ocupă în medie 40 KB). Crearea de noi fonturi este o operație relativ ușoară, iar recunoașterea lor de către sistem este ușor de realizat.

### Descrierea funcționării.

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

Lansând în execuție programul **Mattex**, vom obține în primul rând un ecran asemănător celui prezentat în figura

1. În acesta ni se permite selectarea între discuri rigide C: și discuri floppy A: . Să admitem că am selectat C: ; vom observa sus și jos câte 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 posibilitățile de folosire a tastaturii: tastele și pentru selectare, tasta **Enter** pentru acceptarea subregistriului selectat în cazul nostru **RODICA**), tasta **ESC** pentru revenire la starea anterioară. După ce am selectat (ca subdirectory) **RODICA**, ecranul se modifică. Se observă apariția conținutului acestui subdirectory (lista fișierelor intermediare existente) precum și alternativa de selecție **Fișier existent** — **Fișier nou**. Pentru moment, dacă selectăm "Fișier existent", vom trece instantaneu la un ecran în care linia de jos de mesaje are un conținut mai bogat, iar deasupra ei apare o nouă linie de mesaje, care ne indică tastele cu care putem efectua selecția fișierului dorit. Acest fișier selectat poate fi: editat (tasta **F1**), tipărit (ceea ce înseamnă prelucrarea lui, crearea fișierului final corespunzător și trimiterea spre imprimantă — tasta **F2**), redenumit (tasta **F3**), duplicat (tasta **F5**), distrus (tasta **F8**). Vom prefera însă, pentru moment, să apăsăm tasta **ESC** și să revenim la etapa anterioară.

Selectând "Fișier nou" ni se va cere tastarea unei denumiri. În figura 1 este surprins momentul în care am reușit deja să tastăm numele "nelu" pentru acest nou fișier. (Să observăm că tasta **Back** ne permite să co-

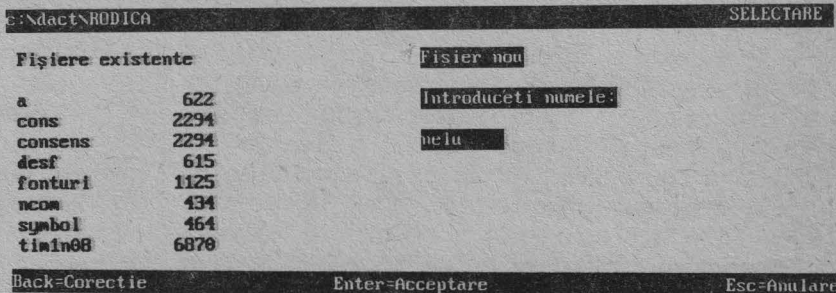


Figura 1

rectăm eventualele greșeli, ceea ce va constitui o regulă generală.) Acceptând acest nume, vom trece la o nouă etapă, cea de selectare a stilului, prezentată în figura 2.



Figura 2

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

Odată cu selectarea stilului ieșim din regimul de **SELECTARE** și intrăm în regimul de **EDITARE**. În acest moment linia de mesaje de sus va conține:

a) denumirea completă a fișierului intermediar în lucru



(drive + path + name, în cazul nostru C:\DACT\RODI-CA\nelu);

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

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

d) rândul curent și pagina curentă;

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

f) abscisa curentă și lățimea paginii (în pixeli).

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

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

Astfel, Alt + B înseamnă schimbarea caracteristicii de bolduire a textului; Alt + C înseamnă 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, ținînd seama că au fost selectate inițial, 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 obține semnele grafice ale fontului curent în cazul nostru symbol-P), așa ca în figura 4.

Utilizînd comenzile prezentate în figura 3, și bineînțeles tastele obișnuite, se realizează de fapt "dactilografierea" brută a textului. Avantajul principal al acestui sistem constituie faptul că, în fiecare moment, operatorul "vede" pe ecran situația exactă a paginii "dactilografiate", așa cum ar ieși ea din imprimantă. Subliniem faptul că pe ecran nu apar secvențe liniare de semne (combinații de texte și comenzi) așa 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 ieșim din regimul normal de "dactilografiere" și intrăm într-un regim de "prelucrare" a textului introdus. Acesta se materializează în primul rând în

aparitiia unui cursor de alt tip și în modificarea liniei de jos de mesaje.

C:\dact\RODI\CA\nelu		symbol-P	c j	2/ 1	236/3428	516/2400	
F1	= Comenzi	+11+	= deplasări cursor	Ctrl++	= stînga rapid	Ctrl++	= dreapta rapid
F2	= Salvare	Home	= inceput de rând	End	= sfîrșit de rând	PgUp	= sus rapid
F3	= Marcare I	PgDn	= jos rapid	Alt+P	= altă pagină		
F4	= Zoom						
F5	= Copiere						
F6	= Mutare						
F7	= Căutare						
F8	= Înlocuire						
F9	= Tipărire						
Insert	= Inserare text scurt						
Del	= Stergere text						
Esc	= Anulare						

EDITARE: COMENZI:

Esc=Anulare

Figura 6

Să apăsăm din nou tasta F10. Vom constata apariția unui mesaj "Se salvează? Da/Nu". Alegînd "Nu" se pierde, bineînțeles, tot ceea ce s-a lucrat. Alegînd "Da" se va cere un nou fișier intermediar, avînd numele dorit (în cazul nostru "nelu"), nume ce va fi adăugat listei fișierelor

C:\dact\RODI\CA\consens		ron1n12-P	j	9/ 1	798/1850	655/2000
<b>C O N S E N S</b>		<b>CON</b> sulting				
		<b>Str</b> uctural				
		<b>EN</b> gineering				
		<b>Systems</b>				
The <b>CONSENS</b> (registered mark) association brings together a g of experts in mathematics, aviation, civil and industrial construction, nu research. B of many points of interest, especially in using nume calculus and computer aided design we desire a fast orientati international standards.						
<b>CONSENS</b> has juridical personality being authorized for consu by contracts with <b>Marcati textul de mutat</b> units for:						
<ul style="list-style-type: none"> <li>• studies of resistance for mechanical engineering, civil and indu construction, equipments;</li> <li>• studies for structures of resistance based on finite element moc using finite element method, boundary element method, finite differ</li> </ul>						
EDITARE: MUTARE		F3=Marcare II			Esc=Anulare mutare	

Figura 7

existente (vezi figura 1).

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

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" obținută prin apăsarea tastei F4.

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



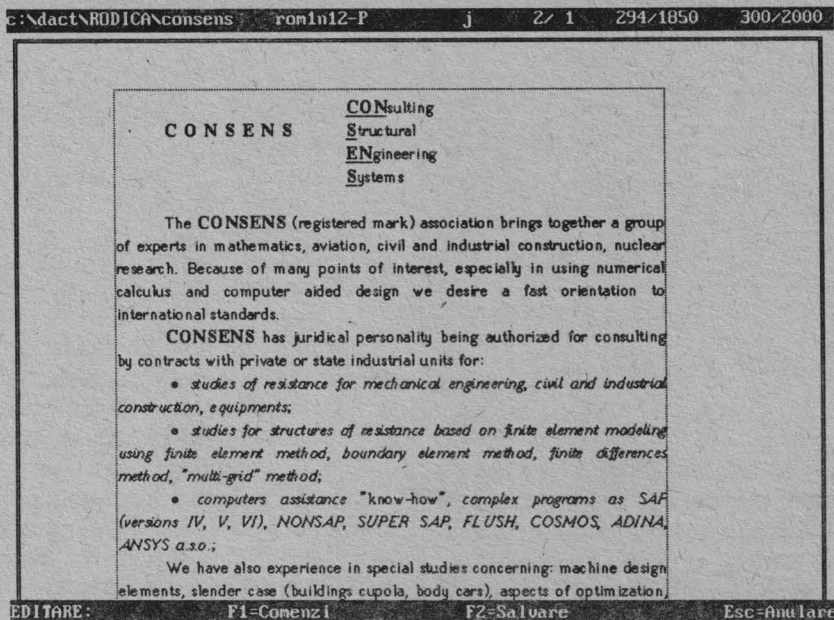


Figura 8

luçgu EDITARE : ZOOM și că avem de ales între trei posibilități de prezentare a paginii pe ecran: mare (vezi figura 5), mijlocie (vezi figura 7) și mică (vezi figura 8).

Încheiem aici prima parte a prezentării sistemului de editare **Mattext**, cu mențiunea că figura 7 prezintă o situație ce va fi explicată ulterior.

## ANUNȚ IMPORTANT

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

## INIȚIERE 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 inițieze în programare se poate pune întrebarea de ce a fost ales ca limbaj de inițiere tocmai limbajul Pascal, când există limbaje mai "simple" cum este limbajul BASIC ori limbaje mai puternice și mai sofisticate cum este limbajul PROLOG. Răspunsul este destul de simplu : este singurul limbaj de programare larg răspândit și care a fost proiectat special pentru învățarea programării. Autorul acestui limbaj (Niklaus Wirth - profesor la Universitatea Tehnică din Zürich) s-a străduit să realizeze un limbaj de programare cât mai pur, care să "silească" programatorul "să se poarte frumos", nepermițându-i acestuia să scrie programe dezordonate. Evident însă fantezia și puterea omului poate să învingă orice bariere și se pot scrie programe la fel de proaste în Pascal ca și 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 numește "limbajul Pascal standard", a cărui descriere apare în așa-numitul "Raport Revizuit" publicat de către Niklaus Wirth în 1974). Implementările existente pentru diferite calculatoare au fost realizate prin extinderea limbajului cu facilități legate de sistemul de operare și hardware-ul specific calculatoarelor respective.

Implementarea cea mai spectaculoasă a acestui limbaj aparține firmei BORLAND, care prin compilatorul TURBO Pascal a revoluționat pur și simplu domeniul, atât prin performanțele intrinseci ale compilatorului (timp de compilare, memorie ocupată de către compilator) cât și prin ușurința 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 și în versiunile de la 3.0 până la 5.5 sub sistemul de operare MS-DOS pe microcalculatoare compatibile IBM. Din acest motiv pentru prezentarea inițială a limbajului vom considera pentru exemplificare limbajul TURBO Pascal versiunea 3.0.

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

### Elementele de bază ale limbajului Pascal

Cum arată un program?

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

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

În primul rând ar fi bine să încercați să executați acest program pentru ca să vă convingeți ca stăpâniți operațiile de "bucătărie" legate de compilatorul TURBO Pascal, aceasta însemnând că știți să introduceți textul programului, să îl compilați și să îl lansați în execuție.

Să urmărim puțin programul prezentat. Se observă că programul apare sub forma unui text care se termină cu punct. Programul are câteva 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" și "end" fac parte din categoria cuvintelor cu semnificație predefinită, denumite cuvinte rezervate (cheie) ale limbajului și sînt utilizate în acest context pentru a delimita corpul executabil al programului;

- *corpul programului* este conținut între cuvintele cheie begin și end și este, pentru exemplul considerat, format dintr-o singură instrucțiune.

Un program real va conține și alte componente pe care le vom prezenta mai târziu.

După cum sperăm că ați ghicit, efectul execuției acestui program constă din afișarea pe ecran a textului :

Felicitari, ati reusit !

Să ne propunem acum să modificăm acest program pentru a afișa 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 instrucțiuni. Amîndouă instrucțiunile realizează afișarea textului conținut între caracterele apostrof. Prima instrucțiune realizează în plus și trecerea

la începutul liniei următoare, astfel încât execuția următoarei instrucțiuni să determine afișarea pe o linie nouă.

În exemplul considerat apare și un nou element de punctuație și anume caracterul ";", care este utilizat pentru separarea instrucțiunilor. În primul exemplu considerat acest caracter nu a fost necesar deoarece în corpul programului nu există decât o singură instrucțiune.

Programul doi poate să fie scris și sub forma :

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

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

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

```
{ Program de demonstrație pentru
afișarea 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 informații suplimentare față de ceea ce rezultă imediat din textul programului. Ca regulă de comportare, ori de cîte ori avem impresia că am fost "deștepți" probabil că este bine să introducem un comentariu care să explice notația, construcția sau algoritmul utilizat.

### Tipuri simple de date

În general se spune că un program realizează o prelucrare de informație. 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 informația) și constă din afișarea acestora.

Informațiile pe care le prelucrează un program sînt în general numere, texte, valori logice. Prelucrările cele mai simple suferite de către aceste informații sînt reprezentate de operații definite asupra acestora. Evident, în funcție de tipul informației, regulile de execuție a operațiilor pot să fie foarte diferite. De exemplu, dacă pentru toata lumea este clar ce înseamnă operația de adunare definită asupra numerelor, operația de adunare definită asupra textelor poate să însemne de exemplu concatenarea acestora, dar pot să fie imaginate și alte definiții.

De asemenea operația de adunare asupra valorilor logice adevărat și fals poate să corespundă unei operații de tip "sau logic", dar poate să corespundă și operației "și logic". Din acest motiv în majoritatea limbajelor de programare se utilizează noțiunea de tip de date. Specificarea unui tip de date presupune descrierea valorilor pe care le pot lua datele de tipul respectiv și specificarea operațiilor și semnificațiilor acestora.

Limbajul Pascal a fost prevăzut cu o serie de tipuri de date standard (predefinite) dar și 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 și tipul **byte**.

Tipurile **integer**, **real** și **byte** sînt tipuri de date care descriu numere, deci le putem denumi tipuri de date numerice. Diferențele 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** și precizia reprezentării acestor valori. Modul de reprezentare și particularitățile ce decurg din acesta sînt funcție de compilatorul utilizat. Astfel, pentru limbajul TURBO Pascal datele de tip **integer** sînt numere întregi 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 și  $123,45678901 \times 10^{20}$  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 și următorul program :

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

Ca efect al execuției acestui program pe ecran se afișează :

```
Ce părere aveți despre rezultatele afișate ?
3
-77
32767
3.140000000E+00
1.2345678901E-03
1.2345678901E-03
```

Se observă că modul de afișare depinde de tipul datelor ce se afișează. Astfel, datele întregi apar în notația matematică uzuală, cu condiția să se încadreze în domeniul tipului respectiv. Dacă se încearcă utilizarea într-un program a unei instrucțiuni

```
write(60000)
```

compilatorul va semnala eroarea corespunzătoare utilizării unei constante întregi prea mari. Datele de tip real se afișează în așa numita formă normalizată (cu mantisă și exponent), cu numărul maxim de cifre semnificative (11). Evident, o astfel de notație nu este potrivită în orice situație. Limbajul TURBO Pascal oferă posibilitatea de a specifica cu puțin efort și alte forme de afișare 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 execuției pe ecran se afișează :

```
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
```

Ați remarcat efectul notațiilor de forma " : intreg" și respectiv " : intreg1 : intreg2" ?

Notația " : intreg" precizează lungimea zonei în care se afișează numărul, aliniat la dreapta, cu condiția ca acea-

stă zonă să fie suficientă pentru afișarea integrală 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 afișării.

Notația " : intreg1 : intreg2" se utilizează pentru afișarea fără exponent a numerelor de tip real. Această notație precizează atât lungimea zonei în care se afișează numărul ("intreg1"), cât și lungimea zonei destinate afișării părții subunitare ("intreg2"). În cazul în care lungimea zonei de afișare este insuficientă, ea este extinsă automat, astfel încât să poată fi afișată integral partea întregă a numărului, urmată de numărul de zecimale cerut.

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

Pentru a ușura identificarea numerelor afișate, în exemplele anterioare s-a făcut afișarea acestora pe linii separate. Aceasta nu este însă unica posibilitate de afișare. Vă rugăm să încercați să executați și următorul program:

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

Ar trebui să obțineți următorul rezultat :

Afișare urita :12-3 -4.1200000000E+00

Afișare normala : 1 2 -3 -4.12

Se observă din nou diferența dintre writeln (afișare urmată de trecere la începutul liniei următoare) și write (afișare cu păstrarea poziției). De asemenea se remarcă posibilitatea realizării unei spațieri între numerele afișate pe același rând, prin specificarea unei lungimi mai mari decât cea necesară.

Dacă veți executa și programul șase veți constata că are același efect ca și programul cinci, deși este scris mai compact.

```

program șase;
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 instrucțiune write sau writeln se poate specifica o listă de informații (numere sau texte, separate prin virgule) care se vor afișa pe același rând.

Asupra datelor numerice se pot executa operații aritmetice. În limbajul Pascal operațiile aritmetice sînt specificate cu ajutorul următoarelor notații :

+	adunare
-	scădere
*	înmulțire
/	împărțire cu rezultat de tip real
div	împărțire cu rezultat de tip întreg (operație definită în cazul limbajului TURBO Pascal pentru tipurile <b>integer</b> și <b>byte</b> )
mod	restul împărțirii întregi (operație definită în cazul limbajului TURBO Pascal pentru tipurile <b>integer</b> și <b>byte</b> )

Efectele execuției acestor operații sînt ilustrate de următorul program :

```

program șapte;
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 execuției acestui program se va afișa 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 operații fără probleme nu necesită comentarea rezultatelor obținute (nici măcar pentru operația 2 / 3). Să urmărim însă puțin cazul operațiilor aritmetice "cu probleme". Primele trei cazuri ilustrează faptul că depășirea domeniului de valori pentru numerele întregi, ca rezultat al unor operații aritmetice, conduce la obținerea unor rezultate incorecte.

Este interesant să se compare valorile expresiilor :

$6000. * 6000 / 6000$  și respectiv  $6000 * 6000 / 6000$

În primul caz rezultatul este cel corect deoarece rezultatul înmulțirii este un număr real a cărui valoare se încadrează în domeniul tipului de date real. În al doilea caz rezultatul înmulțirii depășește domeniul tipului întreg și din acest motiv rezultatul obținut este incorect.

În afară de operațiile aritmetice asupra tipurilor de date numerice, în limbajul Pascal pot să fie efectuate și alte tipuri de operații. De asemenea asupra celorlalte tipuri simple de date (char, boolean) sînt definite operații 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 și 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 totuși 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 notațiile : **true**, respectiv **false**. Modul de reprezentare în calculator depinde de compilator, dar se respectă regula **false** < **true**.

## PC-uri ȘI PROTECȚIE

Ion Mușlea

Conceperea PC-urilor ca sisteme de calcul "deschise" (programatorului), ce nu îngrădesc libertatea de acțiune a utilizatorului, a dus la o reacție promptă și oarecum previzibilă: producătorii de soft au fost obligați să-și ia măsuri de precauție pentru a nu fi "furați" (mai bine zis copiați pe gratis) sistematic. Altfel spus, au început să-și protejeze programele.

Una din metodele curențe 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 configurații, copierea fișierelor corespunzătoare,...) se va iniția o acțiune de scriere pe dischete sursă (fie în scopul de a incrementa un contor, fie pentru a memora o anumită configurație de biți). În momentul în care s-a depășit

numărul permis de copii se va da un mesaj de avertizare, iar instalarea va fi oprită. Chiar dacă piratul "soft" a fost precaut și și-a protejat discheta sursă la scriere, "furtul" nu va reuși pentru că producătorul are grijă să sisteze operația de instalare în caz de nereușită a scrierii pe disc.

Deși în aparență octetul "buclucaș" 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 și debusolarea autorului unei eventuale trasări. De asemenea putem avea mai multe condiții de testat, octeții de test puțin avea la rîndu-le poziții variabile în fișier (după niște algoritmi deosebit de alambicați); ca trucuri clasice putem adăuga și 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 își 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 veți completa cu partea de instalare și-i veți adăuga cîteva trucuri ingenioase, veți obține poate un INSTALL de "primă mîna".

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 aparține la acest capitol o constituie programele "documentație" de tipul BIOS-HELP sau NORTON-GUIDE. Un astfel de produs are de regulă două părți distincte: fișierul de date și programul ce îl face accesibil. Particularitatea lor? Pentru a fi ferite de priviri indiscrete, fișierele de date au fost concepute de așa manieră încât la citirea cu WORDSTAR sau NEDIT vor fi vizibile doar câteva rînduri ce conțin numele produsului și al firmei producătoare. Astfel se ajunge la o situație inedită: DOS-ul indică un fișier cu o lungime de câteva sute de Kó, iar editorul de texte îl tratează ca și cum ar avea câteva zeci de octeți.

Dacă se încearcă vizualizarea conținutului aceluiași fișier cu TYPE sau funcția VIEW din NORTON COMMANDER (apelată cu F3) se va remarca un fapt interesant: cartușul 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) și 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 și al celui ce încearcă să-l "fure". În esență amîndoi au de făcut același lucru: să înlocuiască un caracter cu un altul (piratul un "->" cu un blank, iar "producătorul" un cracter oarecare cu un "->"). Acest lucru se poate realiza foarte ușor folosind spre

exemplu funcțiile DOS (se poate lucra atît din MASM cî și din limbaje evaluate de tip TURBO C sau TURBO PASCAL). În esență structura de program înainte prezentată rămîne valabilă așa că în continuare vom trata alte modalități de a realiza același lucru.

Cum din editoarele de text nu putem șterge acel caracter "->" pentru că ele îl tratează ca pe un EOF, am putea încerca același lucru cu funcția EDIT din NORTON COMMANDER. Vom remarca însă că din cauza lungimii fișierelor, ele nu pot fi decît vizualizate, nu și modificate. Deci din punct de vedere al piratului se pare că soluția este una singură. Pentru producător ar mai fi însă încă o posibilitate: să scrie "cartușul" cu un editor oarecare, după care să continue editarea cu funcția EDIT (se apelează cu tasta F4) din NORTON COMMANDER. La acest apel se va remarca faptul că putem "continua" fișierul și după acel caracter "->". După cum se vede această a doua soluție este oarecum restrictivă prin faptul că nu se permite crearea unor fișiere la fel de lungi ca și "sub" WORDSTAR sau NEDIT, și în plus "spargerea" se poate face mult mai comod de către un eventual "hobbyist" care știe trucul. Practic această modalitate de ascundere a informației ține mai mult de domeniul lui "Știați că...", dar oricum este o "ciudățenie" ce merită a fi reținută. Tot ca fapt divers ar mai fi de amintit faptul că fișierele create cu NEDIT se încheie cu un singur "->", pe cînd cele WORDSTAR cu o suită de "->".

## ÎN ATENȚIA PRODUCĂTORILOR DE CALCULATOARE!

*Firma "ASTERIX", prin specificul ei — inițiere în programarea calculatoarelor personale — vă oferă cea mai eficientă reclamă punînd față în față calculatorul dumneavoastră cu virtualul cumpărător.*

*Relații suplimentare se pot obține la sediul firmei:*

*Str. B.P. Hasdeu, nr.93, Constanța*

*Telefon: 916-42753, după ora 16.*

## Interfațarea Turbo Pascal - MASM

Ion Mușlea

### Noțiunea de UNIT

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

Un UNIT se definește ca fiind o colecție de constante, tipuri de date, variabile, proceduri și funcții. Altfel spus un UNIT este o bibliotecă ce poate fi folosită de orice program.

Un UNIT are două secțiuni: "INTERFACE SECTION" și "IMPLEMENTATION SECTION". Prima va conține toate structurile accesibile utilizatorului (de notat faptul că aici vor figura doar declarațiile de proceduri și funcții, nu și

corpul acestora), iar cea de a doua va conține structurile invizibile utilizatorului UNIT-ului împreună cu corpurile procedurilor și funcțiilor descrise în prima secțiune.

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

```
{ $L filename }
```

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

O altă observație ar fi legată de faptul că în "INTERFACE SECTION" nu se vor utiliza declarații 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 fișierul MYUNIT.PAS care va avea următorul conținut:

```
unit MyUnit;
interface
procedure increment (var variabila: integer);
implementation
procedure view (oVariabila: integer);
{această funcție 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 opțiunea "Compile to disk" și vom compila fișierul astfel obținut tastând Alt-F9. Rezultatul acestei acțiuni va fi obținerea în directorul curent a unui fișier cu numele de MYUNIT.TPU.

În final vom mai crea încă un fișier, cu numele UNIT\_USR.PAS, care va conține 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 și executa normal, obținând în cele din urmă programul executabil UNITUSR.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 declarație de tipul

```
uses MyUnit;
```

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

Aceste proceduri și funcții vor fi scrise în limbaj de asamblare și compilate cu MASM, obținându-se astfel niște fișiere .OBJ ce-i vor fi indicate compilatorului Pascal prin directive de tipul \$L.

### Observație:

Pentru fiecare fișier .OBJ vom avea nevoie de câte o directivă \$L.

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

Toate rutinele declarate externe în cadrul UNIT-ului trebuie să apară în segmentul de cod ("CODE SEGMENT") al unui fișier .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 același număr de parametri și în plus tipurile parametrilor și ale rezultatelor trebuie să coincidă.

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

Pentru ca Turbo Pascal-ul să poată reuși formarea UNIT-ului ce conține directive \$L e necesară respectarea câtorva reguli:

- orice procedură sau funcție 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 definiție pot fi alinate BYTE sau WORD și nu trebuie să specifice vreo clasă de memorie.
- toate variabilele declarate în DATA SEGMENT vor fi neinițializate (se va folosi "?"), căci Turbo Pascal-ul va ignora oricum inițializările făcute la acest nivel.

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

### Proceduri și funcții externe scrise în asamblor

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

### Exemplu:

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

- procedurile și funcțiile declarate cu EXTRN nu pot fi apelate folosindu-ne de un OFFSET. Deci, dacă avem o declarație de tipul EXTRN PROC1:NEAR, un apel în stilul CALL PROC1+8 este încorect;
  - această restricție nu e valabilă și pentru variabile.
- la variabilele externe (definite cu EXTRN) nu se admit referiri la nivel de byte (deci nu pot fi folosiți operatorii HIGH și 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 octeți) cu semn pentru LONGINT se vor folosi 2 cuvinte
- CHAR: 1 octet fără semn
- BOOL: 1 octet conținînd 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 octeți
  - single 4 octeți
  - double 8 octeți
  - extended 10 octeți
  - comp 8 octeți
- POINTER: 2 cuvinte (OFFSET în LOW, SEGMENT în HIGH)
- STRING: pentru N caractere vom avea N+1 octeți, dintre care primul va conține lungimea STRING-ului (deci N)
- SET: cel mult 256 octeți
- ARRAY: locații succesive de elemente componente; la cele multidimensionale cea mai din dreapta dimensiune crește prima.
- RECORD: locații succesive de elemente componente
- FILE: se reprezintă ca RECORD-uri.

### Convenții de apel

Transmiterea parametrilor procedurilor și funcțiilor se va face prin stivă. Înainte de apelul rutinei parametrilor vor fi puși în stivă în ordinea declarării.

Există două tipuri de parametri: transmiși prin referință (în stivă va fi pus un *pointer* la acea variabilă sau transmiși 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 referință;

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

- INTEGER: 1,2,4 octeți (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 EXCEPȚIE; 6 octeți puși pe stivă (cel mai semnificativ primul);
- POINTER: 2 cuvinte (SEGMENT-ul e primul);
- STRING: pointer la o valoare;
- SET: pointer la 32 de octeți;
- ARRAY/RECORD: dacă au 1,2,4 octeți se pun pe stivă, altfel prin pointer la valoare.

Rezultatele funcțiilor vor fi returnate după cum urmează:

- dacă sînt de tip ordinal (INTEGER, BOOL, ENUMERATED) vor fi returnate în regiștri (1 octet în AL, 1 cuvînt în AX, un dublu cuvînt în DX(HIGH-ORDER):AX(LOW-ORDER));

- dacă sînt de tip REAL în regiștri DX(HIGH-ORDER), BX(MIDDLE-ORDER) și AX(LOW-ORDER);

- POINTER-ii vor fi returnați în DX (SEGMENT) și AX (OFFSET);

- STRING-urile: apelantul va transmite la apel un *pointer* la o locație de memorie temporară, la care adresa va regăsi apoi valoarea de string "returnată"; de remarcat că funcția nu trebuie să schimbe *pointer*-ul.

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

De subliniat și faptul că orice procedură sau funcție "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, transmițînd astfel un parametru adițional. Astfel rutina "încuibărită" va putea folosi variabilele locale ale apelantului.

### Observație

Rutinele "încuibărite" nu pot fi declarate externe!

### Exemplu:

Se începe prin a se crea cu ajutorul unui editor de text fișierul **proc\_asm.asm** care va avea următorul conținut:

```

data      segment      word      public
extrn VARREAL:word    ;e o variabilă reală externă
data      ends
code      segment      byte      public
          assume      cs: code
extrn TIPREAL:far,NUMĂR:near ;declarare de rutine externe
subr1     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 corecție cu 2 octeți;
;se va folosi procedura TIPREAL (scrisa în Pascal) pentru a vizualiza valoarea parametrului transmis
          push        cx
          push        dx
    
```

```

                push    ax
                call    TIPREAL
;se apelează funcția NUMAR (scrisă tot în Pascal care va returna un număr real (se folosesc
;registrii DX, BX, AX)
                call    NUMĂR
;variabilei VARREALA i se atribuie valoarea returnată de funcția NUMAR
                mov     ds:VARREALA,ax
                mov     ds:VARREALA+2,bx
                mov     ds:VARREALA+4,dx
;se vizualizează valoarea returnată de funcția 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 fișier se va compila cu MASM (testați MASM și apoi RETURN; la prima întrebare testați PROC.ASM, iar la următoarele doar RETURN).

Apoi, se va crea fișierul MYUNT2.PAS care va conține:

```

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, obținându-se astfel fișierul MYUNIT2.TPU.

În sfârșit se va crea fișierul TEST1.PAS cu conținutul:

```

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

```

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

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

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

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

ORGANIZEAZĂ PERMANENT

# CURSURI

CALIFICARE, FORMARE, SPECIALIZARE,

PERFECTIONARE

PENTRU PERSOANELE CU STUDII MEDII

PE MINI-MICROCALCULATOARE (8-16 BIȚI)

- Formare operatori echipamente culegere, transmitere, prelucrare primară a datelor.
- Formare analiști-programatori asistenți.
- Operatori inițiere în informatică, lucrări secretariat și dactilografie.
- Perfecționare pe microcalculatoare (prelucrare de texte și documentații).
- Programare în limbaj BASIC.
- Programare în limbaj dBASE.
- Utilizare microcalculatoare compatibile IBM-PC.

La cursuri pot participa și persoanele cu studii superioare.

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

**Informații la telefon: 988/73534.**

## LIMBAJUL MAȘINĂ Z80 PE CALCULATOARELE COMPATIBILE SPECTRUM (II)

Bogdan Georgescu  
Bogdan Iordănescu

Așa cum am promis în numărul trecut, vom prezenta un program care folosește întreruperile Z80 și care are un efect apropiat de cel al unui virus informatic, lipsindu-i evident protecția și capacitatea de autoreproducere. Pe SPECTRUM, de altfel, nu poate exista un virus în adevăratul sens al cuvântului, datorită lipsei discurilor și a conectării în rețea, deci a căilor de transmitere (acest lucru ar fi posibil pe variantele extinse, 48K cu Microdrive și Interfața 1 sau +2, respectiv +3). Am asemănat programul pe care îl vom prezenta cu un virus deoarece, odată inițializate întreruperile, se îngreunează

lucrul cu calculatorul. Astfel, la fiecare 32 de întreruperi apărute în sistem și detectate de program (32 de baleieri ale ecranului efectuate pe sincrogenerator), se efectuează câte 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 ș.a.m.d. Datorită faptului că se deplasează numai punctele pe ecran, fără informația de culoare (atribute), efectul este foarte derutant, scrierea unei linii în **BASIC** devenind aproape imposibilă, deși între două întreruperi succesive, deci în majoritatea timpului, interpretorul lucrează, așteptînd introducerea instrucțiunilor. Menționăm că este afectată numai imaginea, respectiv memoria ecran, în rest instrucțiunile 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 față de prezentarea din numărul trecut) rulează pentru a da rezultatul care se vede pe ecran.

Așadar...

	ORG	#FF10	; 65296 zecimal.
	LD	HL,#FE00	; se creează tabela conținînd octeți cu valoarea
	LD	A,#FC	; #FC, de lungime 257 baiți 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 și număr scroll-uri.
	LD	(#FFFF),A	
	LD	A,#FE	; în 1 octetul cel mai semnificativ al adresei tablei.
	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 și salvează regiștrii 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, citește tastatura.
	POP	AF	; restabilește situația inițială și revine acolo unde
	POP	BC	; a fost întrerupt programul principal.
	POP	DE	

	POP	HL	
	EI		
	RET		
SUB	LD	A,(#FFFE)	; încrementează contor scroll-uri.
	INC	A	
	AND	7	; modulo 7.
	LD	(#FFFE),A	
	CP	4	; dacă este între 4 și 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 rânduri pixeli.
	LD	B,0	; în BC valoarea 31.
LOOP1	LD	C,31	
	LDDR		; deplasează un rând.
	INC	HL	; HL la începutul rândului.
	LD	(HL),C	; pune 0 acolo.
	DEC	HL	; HL și DE la sfârșitul rândului următor.
	DEC	HL	
	DEC	DE	
	DEC	A	; face asta pentru 192 de rânduri.
	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 încât 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 și se apelează cu RAND USR, se scrie programul, se assemblează și apoi se rulează subrutina de inițializări din BASIC, cu RAND USR 65296. Stiva a fost deplasată atât de jos din două motive: primul, pentru a nu fi afectată de execuția programului în cod mașină, care rulează la adrese "înalte", acolo unde în mod normal este stiva, și 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 și prin reintrarea în GENS și scrierea următoarei mici rutine:

```

ORG      61000
DI
IM      1
    
```

```

EI
RET
    
```

Este esențial ca instrucțiunea CLEAR să fie executată înainte de prima apelare a asamblorului, deoarece o greșeală în scrierea GENS-ului face ca acesta să funcționeze incorect dacă între două apelări succesive se modifică poziția stivei.

Și acum, sîntem datori cu cîteva explicații cu privire la modul cum rulează subrutinele de scroll. Vom încerca să explicăm, cît mai puțin pretențios posibil, cum se stochează în memorie punctele și culorile de pe ecran.

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

În fișierul display, fiecărui punct de pe ecran i se asociază un bit din memorie, punctul avînd culoarea *paper* dacă bitul este 0 și *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 câteva convenții.

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

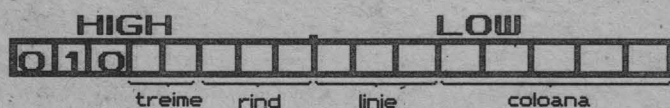
Cele 24 de linii, sînt grupate în 3 zone a câte 8 linii, zone numerotate de sus în jos de la 0 la 2 și numite, cum credeți? — treimi. În aceste zone comportarea memoriei ecran este identică, deci să considerăm una din ele. Într-o treime se află 8 linii numerotate 0-7 de sus în jos și 32 de coloane numerotate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

Foarte bine, veți 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 și 192 ( $8 \times 24$ ) linii, și avem același rezultat. Pentru a vedea motivele care ne-au împins la o asemenea activitate sinucigașă, vă propunem să rulați următoarea rutină:

	LD	HL, 16384	
	LD	BC, nr. octeți	
LOOP1	LD	(HL), #FF	
	LD	DE, 700	
LOOP2	DEC	DE	} ciclu de tempo- rizare
	LD	A, D	
	OR	E	
	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 așteptat ca punînd în loc de "nr. octeți", valori ca 32, 256, 512, 1024, 2048, 4096, 6144 (nu mai mult), să vedem cum rîndurile de pixeli sînt înnegrite unul după altul, și totuși nu se întîmplă așa: se ocupă întîi primul rînd din fiecare din cele 8 linii din treimea 0, apoi al doilea ș.a.m.d., iar cînd se ocupă toată treimea se trece la următoarea și se face același lucru.

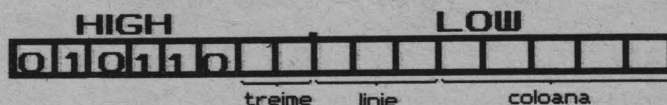
Pornind de la aceste observații, se poate scrie forma generală, binară, a adreselor din memoria display, și anume:



unde fiecare căsuță reprezintă un bit.

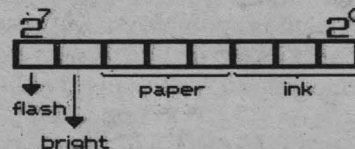
Această formă respectă regulile de incrementare stabilite mai sus și este utilă în programare. Dar ce se întîmplă cînd în cei doi biți rezervați treimii se scrie 11 binar, adică 3 zecimal? Adresele care rezultă nu mai fac parte din fișierul display, ci din zona atributelor, care stochează informația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

Relația dintre memorie și ecran în cazul atributelor nu este așa complicată, pentru că nu mai intervin rîndurile de pixeli, attributele referindu-se numai la caractere. Forma binară a adreselor de attribute este:



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

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



Bitul de *bright* are semnificație pe SINCLAIR, dar nu și pe HC 85, datorită hardware-ului care nu are prevăzut un circuit pentru controlul acestui bit. Acesta este motivul pentru care instrucțiunea BRIGHT din BASIC nu funcționează pe HC 85.

Mai trebuie să facem o observație și cu privire la înteruperi: pe HC85, în ciclul de acceptare a înteruperii, octetul primit de microprocesor este întotdeauna #FF (codul instrucțiunii RST #38), deci ar putea fi evitată tabela de 256 de valori. Nu dispunem însă de scheme detaliate ale SPECTRUM-ului, pentru a ști sigur dacă acest lucru este valabil și la acest calculator. Am preferat să urmărim exemplul majorității producătorilor de programe și să adoptăm soluția tablei, pentru mai multă singuranță.

În numărul următor vom prezenta sistemul de input/output al SPECTRUM-ului, împreună cu un program aplicativ cu efecte mai puțin distructive și puțină animație.

## CALCULATOARE COMPATIBILE SPECTRUM — DESPRE ECRANE ȘI MEMORAREA LOR

Sorin Ciupa

Imaginea afișată pe monitor (sau televizor) corespunde cu informațiile conținute într-o zonă din memoria calculatorului, începând cu adresa 16384 și care are o lungime de 6912 bytes. Afișarea se face prin citirea continuă a acestor 6912 bytes, cu o frecvență de 50 de imagini/secundă și transmiterea spre monitor (sau după modulare în înaltă frecvență spre televizor) împreună cu indicațiile de sincronism specifice imaginii TV.

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

Pentru a înțelege semnificația adreselor imaginii, să ne amintim că în câmpul afișajului există 32 de coloane (0...31) și 24 de linii (0...23), fiecare linie fiind constituită din 8 rînduri (0...7) de pixeli, puncte.

Definirea completă a câmpului unui caracter se face prin 9 locații de memorie, 8 corespunzînd celor 8 rînduri de puncte pentru desen și al nouălea pentru atribute. Transformarea în număr binar al byte-ului conținut într-o adresă ce definește un număr binar al byte-ului conținut într-o adresă ce definește un rînd, are ca rezultat grafică 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 câmpul celor 8 × 8 puncte. Ultima adresă de memorie, a 9-a, ce se referă la câmpul unui caracter, definește atributele, precizînd culoarea fondului, culoarea scrisului, dacă caracterul este mai luminos (BRIGHT), sau dacă pîlpîie (FLASH). De aici apare evident că tot câmpul unui caracter, adică toate cele 8 rînduri de puncte, au aceeași culoare de fond, de scriere, sunt la fel de luminoase și pîlpîie sau nu.

Cele 9 locații de memorie care definesc câmpul 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 și rînduri și pe coloane, pentru grafică și 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), și 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 și 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 și va începe în același mod stocarea sectorului următor.

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

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

Un byte ce reprezintă un atribut, are următoarea semnificație pentru cei 8 biți ai săi:

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

Biții FLASH și BRIGHT pot fi 1 pentru activat sau 0 pentru dezactivat.

Cei trei biți pentru PAPER (6,5,4) și pentru INK (3,2,1) pot avea următoarele valori:

0 0 0	pentru culoarea neagră
0 0 1	pentru culoarea albastră
0 1 0	pentru culoarea roșie
0 1 1	pentru culoarea magenta (purpuriu)
1 0 0	pentru culoarea verde
1 0 1	pentru culoarea azurii (cyan)
1 1 0	pentru culoarea galbenă
1 1 1	pentru culoarea albă

De exemplu, pentru a scrie cu cerneală galbenă pe hîrtie roșie, cu pîlpîie, vom determina valoarea byte-ului 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. (Operația se face ușor prin funcția 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 așa:

```

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 și de cronometrat timpul său de execuție.



În cod mașină, 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 MAȘINĂ	MNEMONICA	SEMNIFICAȚIE
....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	.DIR	Transferă conținutul adresei HL în adresa DE, crește HL și DE cu o unitate, scade BC cu o unitate și repetă pînă cînd BC=0
....11	201	RET	reîntoarcere din subrutină

Avînd ca bază această rutină în cod mașină, 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 și CIRCLE). După desenare de pe bandă, se memorează în RAM și pot fi chemate în afișaj în cadrul unui program.

PROGRAM : SCREEN STORAGE	OBSERVAȚII
<pre> 10 CLS : PRINT "MEMORAREA ECRANE- LOR""Acest program permite stocarea în memoria calculatorului a unor desene, realizate prin programe de desen și memorate pe bandă ca ecrane, pentru a fi chemate în timpul rulării unui program." 20 PRINT #1; "Cite ecrane doriți sa memorați ?(1...5)": PAUSE 0: LET n=CODE INKEY-48: IN- PUT:;IF n&lt;1 OR n&gt;5 THEN GO TO 20 30PRINT AT 8,3;"Se va memora un ecran": IF n&lt;&gt;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 conti- nuăm comanda"; BRIGHT 1;"CLEAR ";sb 50 PRINT ""Programul BASIC + variabilele pot avea maximum";sb-23755;"bytes.";#1;"Apasați ori- care clapă.": 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;"RAN- DOMIZEUSR";ie:LETh+255-j*27:LETl=100-j*12 70 RESTORE: FOR i=0 TO 11: READ x: POKE ie+i,x: NEXT i 80 PRINT""Porniți 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 și acest program poate fi sters.""Oricînd puteți rechema ecranele din memorie cu comanda indicată."" Pentru salvarea pe bandă a zonei de memorie care conține ecranele va trebui să dați comanda: "" BRIGHT 1;"SA- VE""nume""CODE";65368-n*6924;" ";n*6924 100 PRINT ""Acum apăsați NEW pentru a șterge 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 </pre>	<p>Se pot memora maximum 5 ecrane; dacă se încarcă al șaselea, nu rămîne decît foarte puțin pentru programul BASIC.</p> <p>Se păstrează n într-o adresă ce nu va fi afectată de comanda CLEAR ce urmează.</p> <p>Se păstrează neafectată zona de memorie 65368...65535 care conține graficele definite de utilizator.</p> <p>Se rezervă memorie pentru ecrane.</p> <p>Se recuperează variabila n.</p> <p>Se determină adresa de început.</p> <p>Adresa de început poate fi scrisă <math>256 \times h + l</math></p> <p>Se memorează rutina de mutare de pe ecran în memorie.</p> <p>Mută ecranul în memorie.</p> <p>Transformă rutina de mutare pentru a muta din memorie pe ecran.</p> <p>NOTĂ. Pentru a realiza desenul prin BASIC, se înlocuiește LOAD ""CODE 16384,6912 DIN LINIA 80 cu o comandă GO SUB desen.</p>

Informații despre ecran sau despre elementele afișate pe ecran se pot obține prin cîteva comenzi BASIC sau din variabilele de sistem.

COMENZI BASIC :  
— SCREEN (linie, coloană) unde  $0 \leq \text{linie} \leq 23$  și  $0 \leq \text{coloană} \leq 31$ , are ca rezultat caracterul aflat la linia și

coloana respectivă.

— ATTR (linie, coloană) unde  $0 \leq \text{linie} \leq 23$  și  $0 = \text{coloană} = 31$ , are ca rezultat atributul de la linia și coloana respectivă, exprimat conform celor prezentate anterior.

— POINT (x,y) unde  $0 \leq x \leq 255$  și  $0 \leq y \leq 175$ , analizează punctul de pe ecran și este 1 dacă punctul are culoarea cernelii (INK) sau 0 dacă punctul are culoarea hîrtiei.

VARIABLE DE SISTEM, se obțin prin PEEK adresa.

— BORDCR din adresa 23624 specifică atributele comune ale părții de jos a ecranului și a marginii (border), cu codificarea știută.

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

— COORDS din adresele 23677 și 23678 conțin coordonatele x și respectiv y ale ultimului punct plotat cu co-

menzi grafice.

— SPOSN din adresele 23688 și 23689 conțin coordonatele poziției PRINT în partea de sus a ecranului, respectiv la PRINT AT linie, coloană, adresa 23688 conține cifra 33-coloană iar adresa 23689, 24-linie.

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

— ATTR P din adresa 23693 conține atributele globale, codificate obișnuit, conform celor prezentate anterior.

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

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

— DFCC din adresele 23684 și 23685 conține adresa în display a poziției PRINT în partea de sus a ecranului.

— DFCLL din adresele 23684 și 23685 conține același lucru pentru partea de jos a ecranului.

PARTEA			GRAFICĂ				ATRIBUTE				
COLOANA			0	1	...	31	0	1	...	31	
SECTOR	LINIE	RIND									
1	0	0	16384	16385	.....	16515					
		1	16640	16641	.....	16671					
		2	16896	16897	.....	16927					
		3	17152	17153	.....	17183					
		4	17408	17409	.....	17439	22528	22529	.....	22559	
		5	17664	17665	.....	17695					
		6	17920	17921	.....	17951					
	7	18176	18177	.....	18207						
	1	0	16416	16417	.....	16447					
		1	16672	16673	.....	16703					
		2	16928	16929	.....	16959					
		3	17184	17185	.....	17215					
		4	17440	17441	.....	17471	22560	22561	.....	22591	
		5	17696	17697	.....	17727					
		6	17952	17953	.....	17983					
	7	18208	18209	.....	18239						
	2	7	0	16608	16609	.....	16639				
			1	16864	16865	.....	16895				
			2	17120	17121	.....	17151				
			3	17376	17377	.....	17407				
			4	17632	17633	.....	17663	22752	22753	.....	22783
5			17888	17889	.....	17919					
6			18144	18145	.....	18175					
7		18400	18401	.....	18431						
8		0	18432	18433	.....	18463					
		1	18688	18689	.....	18719					
		2	18944	18945	.....	18975					
		3	19200	19201	.....	19231					
		4	19456	19457	.....	19487	22784	22785	.....	22815	
		5	19712	19713	.....	19743					
	6	19968	19969	.....	19999						
7	20224	20225	.....	20255							
3	15	0	18656	18657	.....	18687					
		1	18912	18913	.....	18943					
		2	19168	19169	.....	19199					
		3	19424	19425	.....	19455					
		4	19680	19681	.....	19711	23008	23009	.....	23039	
		5	19936	19937	.....	19967					
		6	20192	20193	.....	20223					
	7	20448	20449	.....	20479						
	16	0	20480	20481	.....	20511					
		1	20736	20737	.....	20767					
		2	20992	20993	.....	21023					
		3	21248	21249	.....	21279					
		4	21504	21505	.....	21535	23040	23041	.....	23071	
		5	21760	21761	.....	21791					
6		22016	22017	.....	22047						
7	22272	22273	.....	22302							
23	0	20704	20705	.....	20735						
	1	20960	20961	.....	20991						
	2	21216	21217	.....	21247						
	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 aplicații pentru PC au fost sau mai sînt trecute în C din limbajul în care au fost inițial implementate, iar reclamele pentru echipamentele orientate către C ca și bibliotecile lor în revistele de programare depășesc ca număr pe cele în toate celelalte limbaje la un loc. Dar, este de așteptat ca în relativ puțini ani să discutăm despre clasicul C cu nostalgie, așa cum deja mulți programatori privesc acum limbaje precum, COBOL, FORTRAN, BASIC. Această predicție se bazează pe două motive:

Primul, odată cu standardizarea unui limbaj, evoluția sa este blocată de cerințele stricte ale consensului și compatibilității cu practica existentă. Limbajul nu mai poate fi adaptat rapid la schimbări ale metodologiei de programare sau la cerințe ale noilor echipamente de operare. Într-un anumit moment, cînd cerințele realității s-au depărtat prea mult de facilitățile oferite de limbaj, devine mai ușor pentru programatori să treacă la alt limbaj sau să inventeze unul nou.

Al doilea, și cel mai important este faptul că are loc o radicală schimbare în optica programării. Lumea limbajelor tradiționale este gata să fie revoluționată, iar câștig de cauză vor avea limbajele de programare orientate pe obiect - OOPLs (Object Oriented Programming Languages). O asemenea revoluție nu este fără precedent - vezi înlocuirea limbajului FORTRAN cu limbaje moderne, structurate în blocuri, ca PASCAL, C și MODULA II în ultimul deceniu. Dar trecerea la OOPLs va cunoaște 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 tradiționale, vizualizezi datele ca niște elemente fixe, în timp ce diverse rutine intervin pentru a executa operații cu acestea. În cadrul unui echipament și a softului asociat orientate pe obiect, rutinele sînt vizualizate ca statice — ascunse în interiorul unor mici cutii negre numite obiecte — și datele sînt cele care acționează: trec de la un obiect la altul sub formă de mesaje. Acest fapt are o corespondență interesantă cu software-uri grafice, ca Microsoft, Windows ș.a.

În ciuda agitației de dată recentă în articole și cărți de specialitate, OOPLs nu sînt tocmai o nouă invenție. Prototipul SIMULA a fost pentru prima oară descris în 1967 - chiar a precedat limbajul C - fiind inventat de Ole-Johan Dahl și Krysten Nygaard de la Universitatea din Oslo și Centrul de Calcul Norvegian. Mai cunoscutul SMALLTALK a fost inventat în mijlocul anilor '70. Au urmat altele. Evenimentul cheie în acest sens a fost apariția

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 susținătorii printre academicieni și unii programatori pe sisteme C. Caracteristic față de OOPLs care l-au precedat este compatibilitatea cu codul C existent permițînd programatorilor să învețe și 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 același timp cu originalul lui Stroustrup. Dar **Objective C** era mai apropiat de SMALLTALK și mai puțin intuitiv pentru programatorii C. Unele caracteristici ale limbajului SMALLTALK, ca și originalului C++, arată de ce OOPLs au fost privite ca niște curiozități de către marea majoritate a programatorilor ațîția ani. Erau necesare o mașină puternică cu un adaptor grafic cu o înaltă rezoluție, megabați de RAM (Random Acces Memory) și un disc fix mare și rapid. Numai foarte recent au reușit asemenea aparaturi să fie la îndemîna posibilităților financiare ale programatorului obișnuit. De asemenea, anumite caracteristici ale SMALLTALK s-au dovedit a fi slăbiciuni cînd limbajul este folosit pentru proiecte mari ce implică mulți programatori.

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

Inițial, deși creștea numărul susținătorilor, programarea orientată pe obiect a servit numai seminariilor academice și tezelor studenților pentru absolvire, pînă în mai 1989. Atunci atît BORLAND cît și MICROSOFT au trecut de partea OOPLs introducînd echipamente și utilități cu extensii orientate pe obiect. Coincidența în timp și asemănarea între MICROSOFT QUICKPASCAL și BORLAND TURBO PASCAL 5.5 a fost remarcabilă. Ambele erau compatibile cu TURBO PASCAL 5.0 și includeau documentație excelentă, materiale de referință pentru extensii, editoare multifîșier în cadrul ferestrelor și "debugger" integrat (sistem de facilități destinate depanării programelor). Și ambele au fost oferite programatorilor la prețuri sub 100 \$. Aceasta a pus efectiv OOPLs în mîna milioanei de programatori serioși și capabili care se considerau gata să facă încercarea. În mai 1990 BORLAND a scos **TURBO C++**, o evoluție naturală a produsului său deja popular, **TURBO C**. Apariția acestui nou produs cu toate facilitățile aferente ne întărește convingerea că am intrat în era limbajelor orientate pe obiect.

### Caracteristici ale OOPLs

Reticența 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 ș.a. Ei sînt obișnuiți 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 și imprevizibil bombardat cu evenimente nesolicitate variind de la conectori, pînă la cerințe ale sistemului de operare de a reface părți 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 acționează niciodată, el întotdeauna reacționează.

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 și INHERITANCE (traducerea e nepotrivită: încapsulare, polimorfism și moștenire). Dacă ții minte aceste trei cuvinte "la modă", veți impresiona cînd veți dori să cumpărați echipamente informatice la petreceri sau cînd vreți să derutați interlocutorul cu privire la ceea ce faceți.

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

**POLYMORPHISM** — este similar cu supraîncărcarea operatorului din C sau PASCAL, unde poți folosi același operator pentru manipularea mai multor tipuri de date: compilatorul C sau PASCAL este suficient de inteligent pentru a examina tipul datelor și 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 complexității și trăsătura care le apropie cel mai bine de aplicațiile în medii cu WINDOW. O nouă

clasă de obiecte derivă dintr-una existentă, prin schimbarea uneia sau mai multor metode aferente clasei vechi și adăugîndu-i una sau mai multe. Acest proces se mai numește uneori și *subclasare*. Noua clasă trebuie să conțină numai codul metodelor adăugate sau schimbate; codul pentru metodele "moștenite" neschimbate rămîne în clasa de referință. Este ușor de imaginat cît simplifică aceasta codificarea unui program care trebuie să controleze mai multe diferite tipuri de ferestre care sînt mici variațiuni ale aceleiași.

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

### TS — PCB

Pachet de programe pentru proiectare asistată de calculator a plăcilor cu circuite imprimate pentru calculatoare personale compatibile IBM PC/XT/AT sau PS2, cu memorie minimă 512 Ko, interfață grafică EGA/VGA, CGA sau Hercules (Felix PC, Junior XT, Robotron 1834).

Funcții realizate:

- Introducerea schemelor electronice și extragerea automată a listei de semnale.
- Amplasarea componentelor și trasarea semnalelor în mod automat, semiautomat și interactiv.
- Verificări în timp real cu generarea automată a documentației de fabricație, incluzînd clișee pentru fiecare strat, mască de găurire, bandă de găurire, mască serigrafică, mască de lipire, desene de test, rapoarte etc.

TRISOFT s.r.l.

Str. Tunari nr. 62, Bl. 24D, Ap. 13

72121 București, ROMÂNIA

Tel. (90) 102004

## SOSESC PRODUSELE CASE

Cureleț-Bălan Gheorghe

În ultimii ani piața 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. Apariția domeniului se justifică prin aceea că elaborarea de *software fiabil* (robust) a ajuns în prezent într-un moment de criză.

Apariția calculatoarelor personale și revoluția pe care a declanșat-o au dus la o largă răspândire a calculatoarelor în cele mai diverse domenii, și în cele mai diverse categorii socio-profesionale. Astfel s-a estimat că la sfârșitul anului 1988 numărul calculatoarelor personale răspândite pe întregul mapamond era de aproximativ 20 milioane. Evident că în prezent acest număr este mult mai mare.

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

Soluția acestei crize o reprezintă folosirea însăși a calculatorului în activitatea de elaborare software. Mulți ar putea să ne reproșeze că el a fost folosit încă de la apariția primelor sisteme de operare, asamblatoare și compilatoare și că actualele produse de succes destinate calculatoarelor personale și minicalculatoarelor (editoare, spreadsheet-uri, sisteme de gestiune baze de date, etc.) sînt de un real ajutor elaboratoarelor de software. Acestora le putem răspunde că produsele **CASE** se ridică prin destinație și facilități deasupra categoriei de produse sus menționate.

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

La început, prin elaborarea software-ului se înțelegea activitatea de codificare într-un limbaj de programare a cerințelor beneficiarului. Această activitate avea un caracter artizanal, nedisciplinat, fără a respecta anumite norme, motiv pentru care au început să apară o sumă întregă de inconveniente, cum ar fi : software-ul elaborat era nefiabil, necesitînd o întreținere continuă a sa, codul (sursa) era ilizibil și greu accesibil pentru alți programatori, termenele de livrare erau depășite iar costurile subestimate, etc. Toate aceste inconveniente corelate cu alte elemente cum ar fi continua ieftinire a hardware-ului, creșterea ponderii software-ului, etc. au dus la necesitatea disciplinării activității de elaborare a software-ului. S-a ajuns astfel la concluzia că elaborarea

software-ului nu înseamnă doar codificare ci ea presupune parcurgerea unui șir 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 științifice, ingineresti, care să constituie limitele între care să se desfășoare actul programării fără însă a impieta asupra aspectului creativ, prin excelență, al său. S-a stabilit astfel că activitatea de elaborare a software-ului constă din parcurgerea următoarelor etape : specificare cerințe de realizare, analiză/proiectare, codificare, testare, implementare și întreținere. Această înșiruire de etape formează ciclul de viață a software-ului. Aspectul ciclic derivă din aceea că elaborarea unei noi versiuni a software-ului presupune reparcurgerea etapelor sus amintite.

Cercetările în această direcție au dus la conturarea la sfârșitul 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 fiecărei etape. Acestea reprezintă metodologiile. Respectarea metodologiilor presupune folosirea anumitor metode și tehnici. Putem defini deci ingineria programării ca domeniul care se ocupă de totalitatea metodologiilor, metodelor, tehnicilor care să conducă în condiții de rentabilitate și eficiență la elaborarea unui software care să satisfacă toate cerințele beneficiarului.

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

Fără a avea pretenția unei definiții exhaustive putem defini **CASE** ca folosirea tehnicilor de inginerie software precum și 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 viață a software-ului. În plus, ele oferă echipei de elaborare suport metodologic precum și facilități de asigurare a calității ( verificare și asigurare a consistenței, completitudinii și conformității cu anumite standarde de elaborare). Putem considera deci, că **CASE** aduce dezvoltării de soft aceleași beneficii ca și **CAD/CAM** producției.

Deși pe plan internațional există unele tendințe de contestare a statutului de sine stătător al domeniului, considerăm că **CASE** și-a conturat frontierele aflîndu-se în prezent în curs de maturizare. O dovadă a viabilității **CASE** o reprezintă adoptarea sa de principalii "actori" de pe scena tehnicii de calcul și informaticii mondiale (*IBM, DEC, Hewlett-Packard, Bull, Tektronix, Sun, Texas Instruments, Oracle, Relational Technology, Software AG* etc.). Menționăm de asemenea că piața produselor **CASE** cunoaște în prezent una din cele mai mari rate de creștere

din industria de software. Conform unui studiu publicat în 1988, de CASE Consulting Group, la sfârșitul anului 1987 existau peste 150 de firme concurente pe piața produselor CASE, estimându-se pentru 1992 o piață 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. și Japonia. Menționăm astfel programele de cercetare: **Esprit** (al comunității europene), **Alvey** (Marea Britanie), **STARS** (*Software Technology for Adaptable Reliable Systems*) - finanțat de DOD (300 milioane \$), **Leonardo** (al corporației *Microelectronics and Computer Corporation (MCC)* - SUA) și **Sigma** - *Software Industrialized Generation and Maintenance Aids* (proiect de

100 milioane \$, lansat în 1985 și finanțat de MITI, care își propune să automatizeze cu 80% concepția de soft și 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 - menționăm că primul produs (CASE Analyser) destinat rezolvării acestei probleme a fost deja prezentat de firma *Parallax/P-Cube Corp.* cu ocazia simpozionului CASE de la începutul anului 1989 la Dallas și Boston.

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

## NOUTĂȚI • NOUTĂȚI • NOUTĂȚI

### 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 ușurință în întrebuițare (interfața utilizator dotată cu menu simplu), este silențioasă. MT 906 este superioară modelelor precedente MT 905 și 910 prin rezoluția 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 și 2 interfețe standard (RS232 și paralela CTX). Imprimanta MT 906 se distinge și prin modul de gestionare a hîrtiei: spre deosebire de alimentarea tradițională poate să dispună opțional de 2 casete cu o capacitate de 200 coli fiecare.

### Page Maker - Windows 3.0

Utilizatorii de Page Maker vor putea să beneficieze de performanțele aduse de versiunea 3.01 față de Windows 3.0. Aceasta asigură o creștere a vitezei de 40-50% față de versiunea precedentă, un răspuns instantaneu la anumite funcții și posibilitatea de a beneficia de buna funcționare a rețelelor Windows 3.0. Page Maker 3.01 a fost îmbogățită cu noi formate de test, ca Test Format, Word Perfect 5.0 și 5.1.

### New Wave 3.0

Hewlett-Packard își întărește propria strategie prin New Wave. Această software permite utilizatorilor să trieze cele mai bune produse disponibile pe piață, fie ale Hewlett-Packard ca și ale altor producători, și să le integreze într-o rețea deschisă și cooperativă, cu scopul de a simplifica diferitele operațiuni de gestionare a informațiilor. Noua versiune 3.0 folosește M.S.Windows 3.0, exploataînd într-un mod special interfața utilizator și asigurînd o mai eficientă gestionare a memoriei, reducînd de la 3 la 2 MB capacitatea de memorie necesară ca suport al numeroaselor aplicații. Cu această nouă versiune e posibilă realizarea de operațiuni complexe și repetitive cu o singură comandă, și împărțirea resurselor rețelei printr-o simplă manipulare a "icoanelor" vizualizate pe ecran.

### Un 68040 pentru NeXT

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

### O alternativă pentru mouse

Se numește PC Trackball și e produsul lui Mouse System. La prima vedere este la fel, dar mișcarea cursorului pe ecran nu se mai obține prin deplasarea mouse-ului pe masa de lucru, ci pur și 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 și o versiune de MS-DOS 2.1 sau superioară. Rezoluția variază între 400 și 6400 CPI și asigură o deplină compatibilitate cu standardul Microsoft Mouse. PC Trackball ar putea să fie o reală alternativă pentru mouse, mai ales în cazul unui spațiu limitat, și mai ales în cazul calculatoarelor portabile, folosite în condiții de spațiu precare.

### Recunoaștere 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 și trimite testul la sintetizatorul incorporat pentru audierea și controlul imediat. Micro-Intro Voice e furnizat cu Host-software și poate funcționa în mediu DOS, OS/2, UNIX și DEC. Sistemul e disponibil în două versiuni: prima versiune completă, cu recunoaștere și sinteză vocală, a doua doar cu sinteza vocală.

## OFERTA

## 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 DESKTOP 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

## OFERTA

## 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

ICECP—Unitate economică integrată de cercetare și producție;

ICECP rezolvă aplicații în domeniile:

- sisteme de calcul multi/mini/micro procesor pentru funcționare în medii severe;
- aplicații "la cheie" cu sisteme de calcul din producția proprie sau de la alți furnizori;
- interfețe universale și/sau specializate;
- simulatoare de antrenament pentru mașini și autovehicule grele;
- livrări prompte de echipamente și subansamble;
- minicalculatoare compatibile PDP pentru medii severe;
- calculatoare personale CP/M; IBM PC;
- familie de surse de comutație în gama 5/10/25 W cu ieșiri variabile;

ICECP asigură asistență tehnică și servicii la un înalt nivel profesional pentru toate produsele cercetate și aflate în fabricație proprie.

ICECP vă poate oferi echipamente și soluția tehnică corectă pentru aplicațiile dumneavoastră în care sistemul de calcul trebuie să funcționeze în medii mecano-climatice severe.

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

Field of activities:

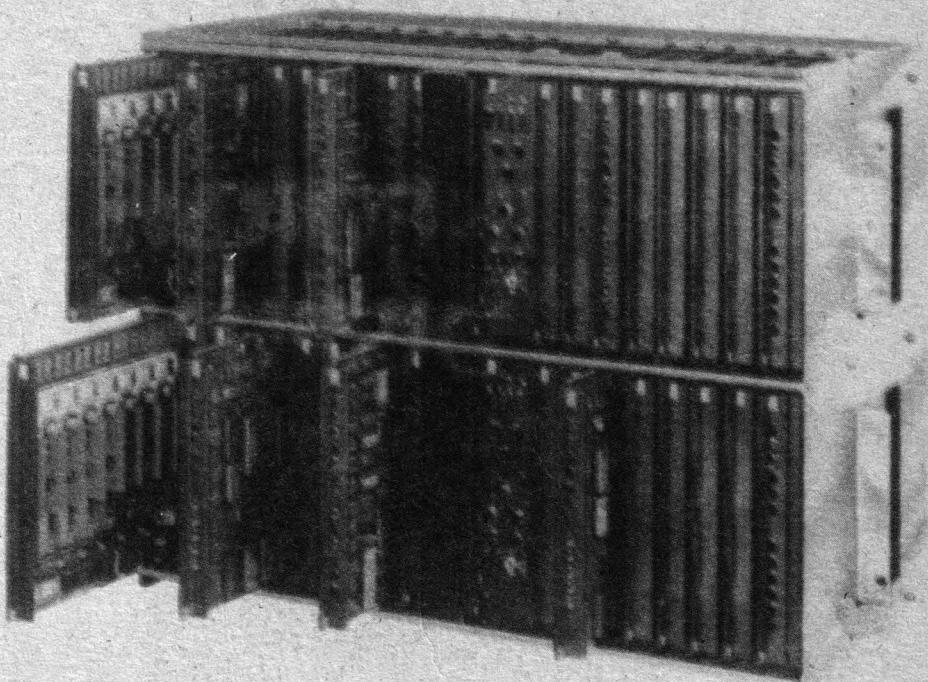
- computer systems for multi/mini/micro processors working in heavy environments;
- turn-key computer systems from indigene or foreign production;
- universal and/or specialized interfaces;
- training simulators for trucks and other heavy vehicles;
- prompt delivery of equipments and spare parts;
- minicomputers 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.

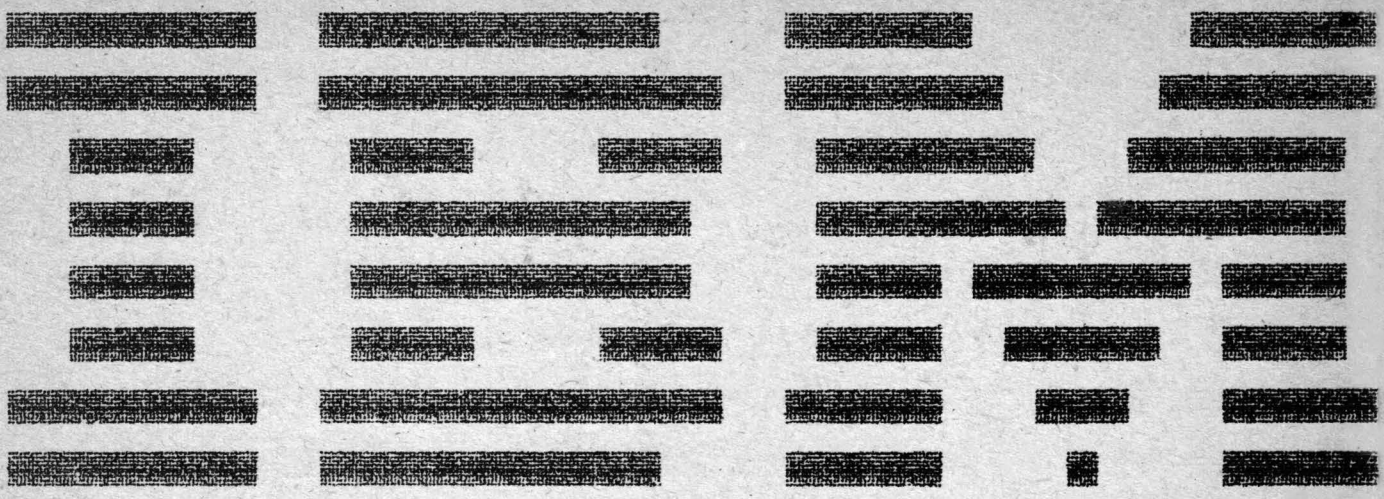
R.I.C.E.P. offers equipments and the right technical solution for your applications using computer systems in heavy mechanical and weather conditions.

# ICECP

## INSTITUTUL DE CERCETĂRI ECHIPAMENTE DE CALCUL ȘI PERIFERICE



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



prin  
SOCODIAG — ASYST  
INFORMATICA  
DISPONIBILA



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