

PC magazin

anul I nr. 2 mai 1990

revistă de informatică independentă

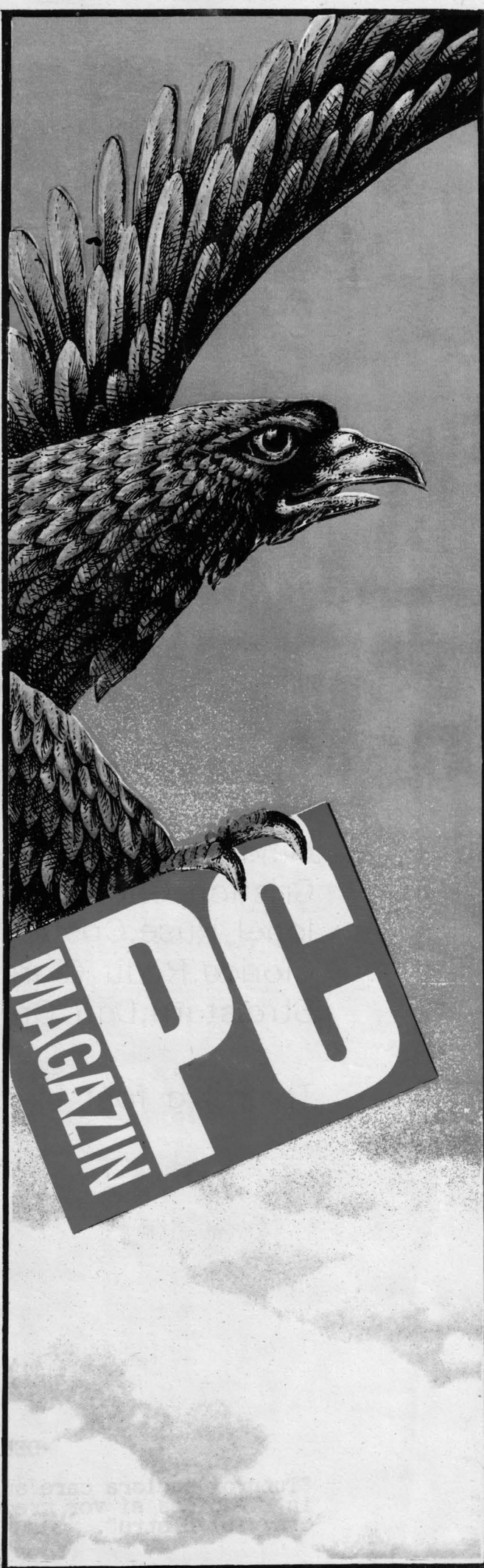
gîndesc calculatoarele ?

limbajul C

inteligență artificială

UNIX ~ sistem al viitorului

strategii, jocuri, enigme



ADISAN

PC-MAGAZIN NR.2 1990

Colectivul de redactie al revistei

Redactor sef fondator: Prof.mat.ADRIAN NEGRU

Redactor sef adjunct: ing.ALEXANDRU BABIN

Redactori stiintifici: Mat.Mihai Constantin

Ing. Marcel Vladescu

Redactor de numar: mat.Catalin Voloseniuc

Grafica si coperta: Grafician Octavian Penda

Ing.dipl. Lucian Misca

Grafician Luminita Ciupitu

La elaborarea acestui numar au colaborat:

Tiberiu Spircu, Ion Paraschiv, Viorel Avram, Irina Negru

Mircea Crutescu, Gheorghe Dumitrascu, Octavian Paiu

Bodosi Imre, Marius Sturzoiu, Adrian Goicea,

Mihaela Olteanu, Mihai Trandafirescu, Irina Babin,

Mihai Unghianu, Carmen Martin, Catalina Stancioiu,

Cristian Gheorghe, Dragos Riscanu, Eugen Ionescu,

Ionel Ruse, Cristian Groza, George Curelet, Bogdan

Ciorica, Radu Ciorica, Daniela Straistaru, Ileana

Straistaru, Dan Cretescu.

Tiparul a fost executat la Tipografia Universul

1990

DEDICATIA NUMARULUI DOI

"Tuturor acelor care au crezut in primul numar al revistei, cred in al doilea si vor crede in cele ce urmeaza ca si in mementoul crezului nostru"

Gindesc oare calculatoarele?

Pentru a putea explora universul inteligentei artificiale (IA) trebuie sa intelegeti mai intii ce inseamna *A GINDI* pentru un calculator. Conceptul de calculator "ginditor" implica faptul ca el executa un program care "gindeste". In aceasta discutie, incercind sa raminem pe terenul termenilor traditionali, un astfel de program il vom numi program inteligent. Oricum, problema ramine deschisa - daca exista sau nu programe inteligente (si deci calculatoare "ginditoare"). Si problema ramine deschisa atit timp cit definitia inteligentei este interpretabila. Exista argumente (chiar de ordin emotional) pentru a sustine fiecare varianta. O intrebare care se iveste in aceasta dezbatere este: in ce mod difera un program inteligent de unul "neinteligent"? Acest articol comenteaza citeva din aceste argumente pro si contra, dar in continuare, discutia poate ramine deschisa.

Pentru a determina ce este un program inteligent, trebuie sa definim *inteligenta*. Intr-un dictionar gasim: "Capacitatea de a acumula fapte, asertiuni si relatii si de a rationa asupra lor". Dar aceasta explicatie ne duce imediat la alta intrebare: "Ce se intelege prin *ratione*?" In acest context, a *rationa* inseamna a gindi - si aici zace problema. Cu citva timp in urma, se spunea ca oamenii nu pot explica *cum* gindesc, dar pot explica *ce* gindesc! (Daca ar fi putut, nu ar fi fost imposibil sa punem si calculatorul sa o faca!)

Daca raminem la stricta interpretare a definitiei de mai sus, se poate argumenta ca toate programele sint inteligente. Iata: prima parte a definitiei stipuleaza "capacitatea de a acumula fapte, propozitii si relatii". Calculatoarele sint extraordinar de dotate in acest domeniu. De exemplu un program de gestiune a bazelor de date pe model relational poate stoca informatii, accepta interogari si, dupa cum o arata numele, reprezinta relatii. Bineinteles, anumite tipuri de informatie (de exemplu imaginile vizuale) sint mult mai greu de acumulat de catre un calculator, dar definitia inteligentei nu specifica categorii de informatii ci doar cere ca acumularea sa aiba loc. Acum e evident ca ordinatoarele satisfac prima cerinta pentru a dispune de inteligenta.

Poate sistemul de gestiune al bazelor de date sa rationeze cu faptele acumulate? Oare? Depinde de ce acceptam ca definitie a ratiunii. Daca manipularea informatiilor din baza de date - cautarea, sortarea, procesarea interogarilor,... - poate fi denumita ratiune, atunci aplicatia in discutie este un program inteligent! (Greu de inghitit.) De aici deducem ca majoritatea calculatoarelor sint inteligente. Trebuie sa va aduceti aminte ca masinile de calcul proceseaza informatia intr-un mod rational, logic.

Pentru multa lume, aceasta concluzie este falsa, deoarece, virtual toate programele ar intra in domeniul IA - o implicatie care nu poate fi adevarata. Intuitia dumneavoastra si programele de IA va spun ca totusi exista o diferentiere. Dar care e? Daca incercati sa explicati neacceptarea faptului ca sistemul de gestiune este un program care "gindeste", veti spune ca programul nu urmareste modul de gindire al unui om. Dar acum puteti spune despre un functionar oarecare (care face acelasi lucru cu programul sus-amintit) ca nu este inteligent sau ca nu are nevoie de inteligenta. Apare deci un paradox: daca programul face ceva, atunci nu gindeste; acelasi lucru facut de o fiinta

umana denota gîndire. Oamenii cred ca faptul de a detine un creier ii face mai specialii si ca au monopolul gîndirii cognitive. Putem admite ca anumite mamifere pot gîndi la un nivel primitiv, dar o mîsina care gîndeste, chiar primitiv, este o idee total neconfortabila (ne aducem aminte de sindromul Frankenstein?). Daca un programator scrie un program "deștept", tendinta este de a spune: "E! Nu este *chiar* deștept. Doar *pare!*". Daca nu se spune asa ceva atunci monopolul oamenilor asupra gîndirii este pierdut.

Exista si un alt mod de a privi problema. Se poate spune ca un ciine este inteligent daca este in stare sa aduca ziarul de la poarta in fiecare dimineata. Dar nu este greu sa construiesi un robot care sa faca exact acelasi lucru, controlat de un calculator. Si totuși, foarte multi ar spune ca robotul nu este inteligent daca aduce ziarul. Aceasta datorita faptului ca robotul este o masina care este programata, care nu *gîndeste* actiunile pe care le executa. (Poate programatorul gîndeste pentru robot, la distanta in timp si spatiu.)

Exemplul cu robotul si ziarul ne aduce din nou la faptul ca oamenii nu stiu cum gîndesc. Deoarece programul care face ca ziarul sa fie transportat este ușor de interpretat, tendinta este de a spune ca programul nu poate fi inteligent, intrucit poate fi inteles. Deoarece procesul gîndirii nu poate fi exprimat, se afirma *incorect* ca orice proces ce poate fi inteles nu poate fi inteligent. In esenta - creatia este intotdeauna mai prejos fata de creator.

Apare chestiunea "dorintei". In decursul istoriei, gîndirea a fost intotdeauna corelata cu conceptul de dorinta libera: numai o entitate care *vrea* sa gîndeasca, va gîndi. Descartes si-a demonstrat propria sa existenta prin faptul ca gîndea, scriind: "Gîndesc, deci exist". Astfel se poate spune ca animalul dresat aduce ziarul fiindca asa *vrea* el, pe cînd robotul face acelasi lucru neputind sa faca altceva din cauza programului. Oare un computer, un aparat care lucreaza strict determinist, poate *alege* sa faca un anumit lucru? (Iata o intrebare realmente filozofical!) Aceasta intrebare poate ingramadi o gramada de programatori care vor afirma ca "o masina nu este decit o masina". Un calculator nu poate avea dorinte deoarece nu poseda creier - doar circuite. Computerul neputind sa "aleaga" ne arata ca nu poate gîndi. Asta ar fi un argument convingator. Cei care impartasesc parerea contrara pot imagina exemple ca: un calculator monitorizeaza greutatea unui camion care se umple de catre un incarcator automat. Cînd aceasta greutate depaseste o anumita limita, el decide ca incarcarea sa fie oprita. A ales calculatorul acest lucru? Da! Calculatorul a controlat clar situatia si a ales oprirea incarcarii. Daca nici aici nu a ales computerul, atunci cine a facut-o? Inseamna ca posibilitatea calculatorului de a procesa expresii conditionale il face sa poata alege, pus intr-o anumita situatie.

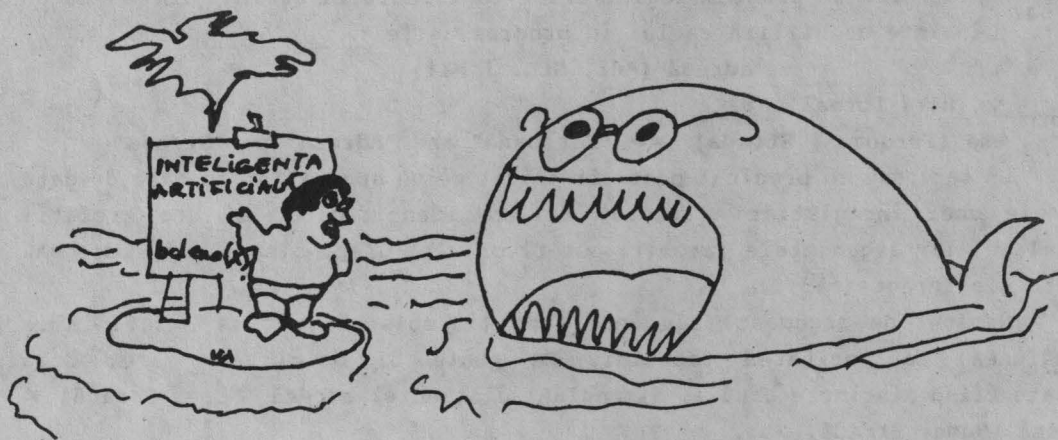
Este un calculator capabil de gîndire? Dupa cum exemplele au ilustrat, sint opinii puternice de ambele parti. Cel mai sigur este sa afirmam ca dezbaterile continua si ca opinia dumneavoastra este sigur cea corecta.

Pina acum am convins pe multi ca este imposibil de determinat daca un calculator gindeste sau nu si daca un program este inteligent sau nu. Totusi se pot da exemple de programe care au comportari similare cu cele umane. In mod clar, anumite programe *par* inteligente (si astea sint subiectul IA). Toate dificultatile aparute mai devreme decurg din definitia *inteligentei*. Ce lipseste din dictionar este ca termenul in discutie este inteligenta *umana*!

Deci, pentru ca un program sa fie inteligent, este suficient ca acesta sa *actioneze* inteligent - deci sa *simuleze* comportamentul uman. Procesele interne programului nu trebuie sa fie aceleasi cu procesele interne creierului uman. Acum putem da o definitie a programului inteligent:

Un program este inteligent atunci cind acesta are o comportare similara cu cea a unui om confruntat cu o aceeaasi problema. Nu este necesar ca programul sa rezolve, sau sa incerce sa rezolve problema in acelasi mod cu fiinta umana.

Concluzia poate fi ca programele inteligente sint cele care denota o comportare inteligenta, asemanatoare cu cea umana, comportare pe care programele "neinteligente" (vezi sistemul de gestiune al unei baze de date) nu o au.



Fundamente ale programării logice în inteligența artificială (II)

Componente de bază în programarea logică

1. Fapte, întrebări, reguli

Elementele de bază ale programării logice sînt determinate natural printr-o singură structură de date: termenul logic și prin trei tipuri de instrucțiuni fundamentale: faptele, regulile și întrebările cunoscute sub denumirea de "clauze Horn".

Vom încerca în continuare o prezentare individuală a acestor elemente constitutive. Astfel, cea mai simplă instrucțiune sau afirmație este "faptul", care nu are altă misiune decît de a arăta că anumite obiecte sînt legate între ele printr-o relație, pe care s-o numim predicativă sau predicat. De exemplu:

scad (7, 3, 4)

este un fapt ce are ca suport operatorul predicativ **scad** care, aplicat primilor 2 operanzi, îl produce pe al treilea, altfel scris:

scad=-

scad (7,3,4) \iff 7 scad 3 = 4 \iff 7 - 3 = 4

În cele ce urmează vom numi obiectele simple atomi, pentru a le deosebi de alte categorii de obiecte cu care vom lucra și anume: liste și mulțimi, iar predicatul (faptele) care acționează asupra obiectelor simple le vom numi predicate de ordin 1, pentru a le deosebi de cele ce operează asupra mulțimilor sau a altor predicate, cunoscute ca predicate de ordin 2.

Orice relație predicativă complexă are ca suport inițial unui sau mai multe fapte, de aceea putem substitui faptele cu condiții inițiale pentru orice program logic, o înșiruire de fapte fiind ea însăși un program logic, și anume cea mai simplă formă de program logic. Orice fapt poate fi definit formal ca un comentariu înainte de utilizarea lui în program astfel:

adresă (Adi, Str. 1 Mai)

poate fi definită formal ca:

adresa (Persoana, Strada) \iff "Persoana" are "Adresa" pe "Strada"

Un fapt sau un predicat poate fi privit de un specialist în baze de date ca numele unei înregistrări (record) ori ca identificator al unei relații particulare, iar argumentele (atomii) pot fi priviți drept câmpuri (fields) sau attribute ale înregistrării.

Numărul de argumente ale unui fapt (înregistrări) o să-l definim ca dimensiunea sau aritatea faptului. Ea poate fi oricît de mare, două predicate fiind distincte dacă au dimensiuni distincte: adresă (Nume, Stradă) \neq adresă (Nume, Stradă, Nr.)

De obicei numele faptelor încep cu literă mică în timp ce numele argumentelor cu literă mare, convenție utilizată în multe implementări ale limbajelor programării logice tocmai pentru a face distincția între două unități lexicale: variabile și constante. Numele variabilelor pot fi oricît de lungi dar trebuie să înceapă cu literă mare. Dintre ele:

Variabilă-de-bază;

X;

- :publică;

-.

Componente de bază în programarea logică

Ultima variabilă, "-", este cunoscută în programarea logică drept "variabilă anonimă". De obicei, ea este pusă în locul unui argument de a cărui valoare nu avem nevoie în program, un fel de "?" în reconstituirea unui nume dintr-un director pe calculator a cărui literă pe prima poziție nu ne interesează:

>dir ?or

va da, dacă aceste nume sînt în director:

Nor, Gor, Dor, Cor

Ca o distincție importantă, în programarea logică, variabilele țin locul unor entități din domeniul obiectelor definite în fapte mai degrabă decît locații de memorie în programarea convențională.

Constantele din programarea logică sînt de 4 tipuri:

- nume: scrise ca niște variabile, oricît de lungi, fără a începe cu "-" sau cu literă mare și fără a fi constituite numai din cifre;
- întregi: diferă de nume prin aceea că sînt formați exclusiv din cifre și pot participa ca operanzi în calcule matematice;
- operatori: sînt atomi lexicali cu semnificație specială dintre care: operatorii matematici "+", "-", "\", operatori de paginare "\n", "\f" etc.
- nume marcate: sînt șiruri de caractere de lungime arbitrară marcate de ghilimele simple: 'șir' ce servesc, în general, pentru delimitarea șirurilor de caractere drept constante, altfel putînd fi considerate ca variabile, ori pentru a delimita comentarii și mesaje pentru utilizatorul programului.

Astfel, orice fapt are ca argument o constantă, argumentele formale, pe de altă parte, putînd fi scrise cu literă mare, fiind considerate variabile sau valori constante luate ca variabile.

A doua formă de afirmație în programarea logică o constituie "întrebarea" sau "cererea". O cerere pretinde răspuns dacă o anumită relație are loc între obiecte. Astfel, cererea adresa (Adi, 1 Mai)? are răspuns la întrebare dacă "adresa" este predicat valid între cei 2 operatori constanți Adi și 1 Mai. Considerînd anterior existența faptului adresa (Adi, 1 Mai) răspunsul este afirmativ la întrebare.

Să presupunem că avem date următoarele informații:

adresa (Adi, 1 Mai).	adresa (Cornel, 1 Mai).
adresa (Doru, Turda).	adresa (Irina, Turda).
adresa (Ioana, 1 Mai).	adresa (Cristi, Dorobanți).

Din punct de vedere sintactic "întrebările" și "faptele" au aceeași structură, fiind distincte prin context. Orice pericol de confuzie este înlăturat dacă se păstrează convenția ca, după orice fapt să punem "." iar după orice întrebare semnul "?". Orice afirmație fără "." sau "?" o s-o numim realizare (goal).

Să denotăm o întrebare prin I?. O astfel de cerere verifică dacă realizarea I este adevărată. Vom numi întrebare simplă o cerere formată dintr-o singură realizare iar prin răspuns la întrebare vom înțelege dacă o cerere este o consecință logică a unui program.

În cele ce urmează, prin reguli de deducție, vom înțelege consecințele logice ale unui program.

Ca o primă regulă de deducție vom marca identitatea:

{ din I se deduce I }.

Dacă un fapt al programului este identic cu o întrebare atunci punerea cererii are răspuns afirmativ (da), altfel, dacă faptul (I.) dedus din întrebarea

Componente de bază în programarea logică

(I?) nu e consecință logică, răspunsul la cerere este negativ (nu).

În regulile deducției, o variabilă pusă într-o cerere concentrează o mulțime de întrebări ale căror fapte sînt consecințe logice ale programului. Astfel, întrebarea adresă (X, 1 Mai)? unifică mulțimea de realizări: {adresă (Adi, 1 Mai), adresă (Ioana, 1 Mai), adresă (Cornel, 1 Mai)} avînd nu numai un răspuns afirmativ ci o mulțime de atomi care fac adevărate toate substituțiile lor cu X în mulțimea faptelor, și anume:

X = Adi; X = Ioana; X = Cornel.

Spunem că atomii Adi, Ioana, Cornel sînt substituții ale variabilei X sau, mai general, putem formula definiția: o substituție este o mulțime finită sau vidă de perechi de forma $X_i = s_i$, unde X_i este variabilă iar s_i este un termen (singura structură de date din programarea logică, înțelegînd prin conceptul de termen un element al mulțimii formate din variabile, constante ori compuneri de variabile și constante (cunoscuți ca termeni compuși sau functori)). Astfel, un functor este o generalizare naturală a noțiunii de "fapt" în aceea că functorul este caracterizat prin nume care este un predicat sau un atom și prin dimensiune sau număr de argumente. Cîteva exemple de functori: adresă(X, 1 Mai), număr (2), scad(5, 1, 4), scad(5, X, 2), scad(6, X, X,), scad(scad(2, 1, 1), 1, X) etc.

Observăm că spațiul functorilor conține spațiul faptelor, ceea ce impune o delimitare calitativă și anume:

Vom numi "finale" alte întrebări, realizări, fapte etc. în care nu apar variabile ca argumente iar acei termeni ce conțin variabile îi vom denumi prin "nefinale". Astfel adresă (Adi, 1 Mai) este un termen final în timp ce adresă (Adi, X) este nefinal.

Ca o continuare a definiției unei substituții va trebui să impunem și condițiile ca $X_i \neq X_j$ pentru $i \neq j$ iar $X_i \in \{s_j\}$, $(\forall) i, j$.

Funcțional, vom nota prin f un termen iar prin $f(s)$ termenul obținut prin aplicarea unei substituții s putînd defini, astfel, o funcțională $T: \text{TXS} \rightarrow T$, dată de

$$T(f, s) = f(s)$$

ce realizează substituția variabilei X prin s_i pentru fiecare pereche ($X=s_i$) în s. [T = spațiul factorilor; S = spațiul substituțiilor].

Obținem astfel o nouă noțiune, și anume, aceea de "instantare", în sensul că un termen T este o "instantare" a lui f dacă există o substituție s pentru care $T = f(s)$.

Putem spune acum că adresă (Adi, 1 Mai) este o instantare a lui adresă (X, 1 Mai) sau adresă (Cornel, 1 Mai) este o instantare (nu unică) a lui adresă (X, Y) realizată sub substituția $s := \{X = \text{Cornel}; Y = 1 \text{ Mai}\}$.

/* Program 1*/

domains

 persoană, hobby = symbol

predicates

 pasiune(persoană, hobby)

clauses

 pasiune(elena, citit).

 pasiune(ion, computere).

 pasiune(ion, badmington).

 pasiune(leonard, badmington).

 pasiune(maria, înot).

 pasiune(maria, citit).

Componente de bază în programarea logică

Declarația - domains - va cuprinde toate declarațiile de termeni, liste sau orice alte structuri ce intervin în corpul programului. Astfel, declarația - symbol - ține locul unui termen (variabilă sau constantă). Declarația - predicates - va cuprinde lista functorilor formali, deci, ai acelor functori ce nu sînt fapte. Declarația - clauses - va cuprinde lista tuturor functorilor precum și a regulilor (pe care le vom discuta în cele ce urmează).

Avem astfel un program logic. Cererile de realizare ale programului pot fi puse implicit în corpul programului prin declarația - goal - urmată de cererile de realizare (sub formă de întrebări, fără simbolul "?" la sfîrșit), ca de exemplu:

```
goal
    pasiune(maria,X)
    pasiune(X,computere)
    pasiune(X,călărie)
```

cereri care vor fi tratate prin execuția programului, răspunsurile ce se vor afișa pe ecran sau la imprimantă, fiind:

```
X = înot; X = citit;
    ready
X = Ion;
X = false.
```

Dacă cererile nu sînt date implicit, în momentul reluării programului, pe ecran va apare mesajul:

- goal?

unde, ca răspuns, se poate formula orice cerere. Astfel, tastînd - pasiune(Ion,X) ca răspuns ne va apare:

```
X = computere;
X = badmington;
- goal?
```

urmînd ca, dacă nu mai avem nici o altă cerere să abandonăm programul tastînd <ESC>.

Existența cererilor "nefinale" conduce natural la definirea unor fapte sau functori universali, cuantificați în sensul că o definire de forma:

adresă(X,lMai)

este realizare pentru orice instantare a lui X sau realizarea adresă(Nume,lMai) este adevărată oricare ar fi numele <Nume>. Acest fenomen de instantare poate fi clarificat drept a treia regulă de deducție logică în sensul definiției - <Dintr-o afirmație universal cuantificată I, se deduce o instantare a ei, I(s), oricare ar fi substituția s>.

Astfel, afirmația scad (X,X,0) are loc oricare ar fi substituția lui X cu un număr real. La întrebarea scad (5,5,0) răspunsul este <da> în baza faptului scad (X,X,0), de unde apare conceptul de "instantare comună" a doi termeni. Vom numi, astfel, "instantare comună" a termenilor P și Q un termen R care este o instantare a lui P și a lui Q, în sensul că există două substituții s1 și s2 simultane pentru care R = P(s1) este sintactic echivalent cu R = Q(s2).

Interesant de remarcat că realizările scad (5,Y,0) și scad(X,X,0) au amîndouă aceeași instantare, și anume, scad (5,5,0), deoarece aplicînd substituția {Y = 5} lui scad (5,Y,0) și substituția {X = 5} lui scad (X,X,0) obținem aceeași realizare: scad (5,5,0).

În general, a răspunde unei întrebări folosind un fapt înseamnă a căuta o instantare comună atît faptului cît și întrebării, răspunsul fiind instantarea

Componente de bază în programarea logică

comună dacă există, altfel răspunsul fiind <nu>.

Să trecem, în continuare, la o extindere naturală a noțiunii de întrebare, și anume, la conceptul de întrebări compuse, realizate prin conjugarea <și> logică a mai multor întrebări simple.

Ele pot apare sub formă de:

a) **conjuncții de fapte** ca:

adresă (Adi, 1 Mai), bărbat (Adi)?

Înțelegînd virgula drept conjuncție " \wedge " între întrebări, realizarea lor înșemnînd realizarea simultană a celor 2 întrebări componente.

b) **conjuncții de functori cu variabile identic distribuite:**

adresă (X, 1 Mai), bărbat (X)?

Înțelegînd întrebarea: "Există entitatea X pentru care are loc atît adresă(X, 1 Mai) cît și bărbat (X)", variabila X fiind evident existențial cuantificată.

Generalizînd rezolvarea întrebării conjunctive:

$I_1, I_2, \dots, I_n?$

Într-un program logic revine la găsirea unei substituții s pentru care $I_1(s), \dots, I_n(s)$ sînt fapte existente în program.

Să trecem, în continuare, la studiul relațiilor ce sînt derivate din clauze existente în program, relații cunoscute în programarea logică drept reguli. Regulile sînt afirmații de forma:

$R \leftarrow R_1, R_2, \dots, R_n$

unde R se numește **capul regulei** iar $R_i, i=1, n$ - **corpul regulei**. Atît R cît și R_i sînt realizări în programul logic. Avem deci definiția completă a clauzei Horn ca o mulțime de fapte, întrebări și reguli. Pentru $n = 1$ avem o clauză specială cunoscută ca regulă iterativă ($R \leftarrow R_1$).

Să facem o observație importantă. Pe tot parcursul expunerii, prin relația $P \leftarrow Q$ sau $P :- Q$ vom înțelege că P este o consecință a lui Q sau P are loc dacă și numai dacă Q are loc. Simbolurile " \leftarrow " și " $:-$ " sînt echivalente, primul fiind folosit în programe logice universal valabile, cel de-al doilea fiind specific compilatorului TURBO-PROLOG implementat pe microcalculatoarele personale profesionale sub sistemele MS-DOS, PC-DOS, XENIX, OS/2 (deci " \leftarrow " și " $:-$ " sînt implicații logice echivalente cu \Rightarrow în sensul că " $P \leftarrow Q$ " \iff " $Q \Rightarrow P$ ").

Să ne amintim că în capitolul I definisem predicatul:

drum (Oraș1, Oraș2, Distanță)

prin care vom înțelege că ruta este adevărată dacă (\exists) o compunere de drum-uri care să unească Oraș1 cu Oraș2. Să formulăm, în limbajul programării logice, această condiție:

rută (Oraș1, Oraș2, Distanță) \leftarrow drum (Oraș1, Oraș2, Distanță) (1)

**rută (Oraș1, Oraș2, Distanță) \leftarrow drum (Oraș1, X, Dist.1),
rută (X, Oraș2, Dist.2), (2)**

Distanță = Dist.1 + Dist.2

Înțelegînd că rută (Oraș1, Oraș2, Distanță) are sens, adică (\exists) un drum între Oraș1 și Oraș2 dacă drumul este direct în domeniul faptelor (regula 1) sau dacă există un drum de la Oraș1 la un OrașX cu distanța Dist.1 și o rută de la OrașX la Oraș2 cu Dist.2, astfel ca distanța între cele 2 orașe să fie Dist.1 + Dist.2. Regulile (1) sînt disjunctive, realizarea uneia determinînd căutarea în cealaltă. Să definim, în continuare și mulțimea faptelor acestui program prin:

drum (București, Brașov, 173).

drum (București, Ploiești, 60).

drum (Constanța, Iași, 600).

Componente de bază în programarea logică

drum (București, Constanța, 300).

drum (Brașov, Constanța, 400).

Ne interesează dacă are loc realizarea:

rută (București, Iași, X)?

urmînd ca, dacă există ruta, să aflăm și distanța între cele 2 orașe.

Cu regula rută, definită anterior, avem:

- clauza (1) - Nu este satisfăcută pentru că nu există drum direct București, Iași;

- clauza (2) - admite descompunerea:

rută (București, Iași, X) ← drum (București, X1, Dist.1)

rută (X1, Iași, Dist.2)

drum (X1, Iași, Dist.2)

Da! pentru că pentru $X1 = \text{Constanța}$ și avem instantarea: drum (X1, Iași, Dist.2) = drum (Constanța, Iași, 600) realizată cu substituția $\{X1/\text{Constanța}, \text{Dist.2}/600\}$. Deci rută(X1, Iași, Dist.2) = rută (Constanța, Iași, 600) iar drum (București, X1, Dist.1) se instantează prin substituirea anterioară cu drum (București, Constanța, Dist.1) care, la rîndul ei, se instantează cu faptul existent drum (București, Constanța, 300) cu substituția $\{\text{Dist.1}/300\}$. Astfel, cum $X = \text{Dist.1} + \text{Dist.2}$, cererea

rută(București, Iași, X)

se instantează în realizarea rută (București, Iași, 900) cu substituția $\{X/\text{Dist.1} + \text{Dist.2}\} = \{X/300 + 600 = 900\}$. Această ultimă realizare poate fi adăugată mulțimii faptelor, îmbogățind baza de date a programului. Vom vedea că în orice sistem PROLOG există un predicat de sistem ce permite adăugarea realizărilor nou obținute ca fapte la baza de cunoștințe (sau de date), predicat cunoscut ca assert (clauză) sau assert (realizare).

Exemplul prezentat folosește o tehnică larg uzitată în programarea logică în general și în limbajul Prolog în particular, și anume, recursivitatea. Acest procedeu este o tehnică puternică în programarea nenumerică și poate fi folosită în două moduri, și anume:

- utilizată pentru descrierea structurilor care au alte structuri în componență;
- utilizată în programe care conțin în corpul lor apeluri la ele însele sau copii ale lor, de a căror realizare depinde realizarea însăși a programului.

Vom vedea cum recursivitatea folosește în mod natural procedeele a) și b) în manipularea datelor și a programelor, aflîndu-și expresia ideală în prelucrarea arborilor, a listelor și a calculului formal, un spațiu de două capitole fiind afectat studiului recursivității și al structurilor de date complexe în programarea logică. Pentru fixarea cunoștințelor să introducem acum noțiunea de listă, pentru a putea exemplifica cîteva tehnici de aplicare a recursivității în scrierea regulilor clauzale.

Astfel, lista este o structură de date comună în programarea nenumerică, putînd fi definită ca o mulțime ordonată de elemente despărțite prin virgulă, care poate avea orice lungime, elementele listei putînd fi formule atomice, structuri sau alți termeni ce pot include, la rîndul lor, liste.

Deși listele sînt singura structură de date alături de constante în limbajul LISP, putînd reprezenta gramatici, grafuri, formule etc., ele sînt doar o formă de structură a datelor în Prolog. Fiind mulțimi ordonate, scrierea elementelor în listă este esențială, notația folosită pentru reprezentarea lor

Componente de bază în programarea logică

fiind înșiruirea elementelor între două paranteze drepte:

(1) listă = $[X_1, X_2, \dots, X_n \dots]$.

Sînt legitime următoarele exemple:

$[\]$ (lista vidă), $[a_1, a_2, \dots, a_n]$ (lista elementelor $a_i, i=1, n$), $[a_1, a_2, [a, X, Y], a_4]$ (lista $[a, X, Y]$ este tratată ca un element - a_3 - al listei care o conține).

Procesarea unei liste presupune o anumită modalitate de delimitare conceptuală a elementelor ei, notația uzuală folosită în limbajul programării logice fiind cea care delimitează elementele unei liste în **capul** listei (vezi 1) format din primul element (X_1) al listei și **corpul** sau **coada** listei reprezentat de mulțimea $[X_2, \dots, X_n \dots]$, ce se constituie ca o sublistă a lui (1) sau:

$[X_1, X_2, \dots, X_n, \dots] = [\text{Cap} \mid \text{Corp}]$, $\text{Cap} = X_1$, $\text{Corp} = [X_2, \dots, X_n \dots]$.

Înțelegerea reprezentării listelor o considerăm esențială în scrierea regulilor clauzale ale programelor logice, de aceea devine necesară prezentarea cîtorva exemple descriptive, astfel:

Să considerăm lista:

listă = [mere, pere, prune, caisel].

Prin procesul de instantare și, mai general, al celui de unificare (ce va fi detaliat în capitolul următor), răspunsul la următoarele cereri de satisfacere:

- i) listă = [pere | Coadă];
- ii) listă = [mere, pere, prune | Coadă];
- iii) listă = [X | Coadă];
- iv) listă = [[mere, pere, prune, X] | Coadă];
- v) listă = [mere | [X, Y, Z]];
- vi) listă = [[mere, pere] | [prune, caisel]]
- vii) listă = [[mere, pere] | [X | Y]];

este dat după cum urmează:

	cap	coadă
i	pere	[pere, prune, caisel]
ii	[mere, pere, prune]	[caisel]
iii	X = mere	[pere, prune, caisel]
iv	[mere, pere, prune, caisel] (X = caise)	[]
v	mere	[pere, prune, caisel]
vi	[mere, pere] 	[prune, caisel] (listă cu un element = o nouă listă)
vii	[mere, pere]	X = prune, Y = caise

Prin extensie există o reprezentare a șirurilor de caractere sub formă de listă, sub formă de coduri ASCII sau sub formă lexicală reprezentînd fiecare cuvînt al unei fraze ca element distinct al listei: Astfel, șirul "Adrian" este reprezentat sub formă de listă prin:

[65, 100, 114, 105, 97, 110]

iar fraza <Este o dimineață frumoasă> sub forma:

[Este o dimineață frumoasă].

Întrebarea cea mai uzuală folosită în legătură cu o listă este dacă un element se află sau nu într-o anumită listă sau, altfel spus, dacă elementul X este membru al unei liste Y, relație ce definește formal predicatul membru (X, Y)

Componente de bază în programarea logică

unde X este elementul căutat iar Y lista în care se procesează căutarea. Să observăm că putem face o delimitare a căutării, și anume, putem verifica dacă X este cap al listei sau X se află în coada listei, altfel scris:

membu ($X, [X|_]$)

adică " X este membru al listei dacă X este cap de listă" sau

membu ($X, [_|Y]$): - membru (X, Y)

sau X este membru al listei dacă X este membru al corpului Y al listei.

Utilizarea variabilei anonime "_" s-a folosit marcînd neinteresul nostru pentru variabila care formează corpul listei în primul exemplu și pentru variabila care marchează capul listei în al doilea exemplu. Acest predicat este un prim exemplu de prelucrare recursivă a unei liste. Putem prezenta forma completă a clauzei **membu (element, listă)** cu o notație pe care o vom păstra constantă pe tot parcursul expunerii, și anume, reprezentarea unei liste sub forma:

listă = [Cap | Corp] = [X | X_S]

unde prin X, Y, \dots va fi reprezentată variabila ce formează capul listei iar prin X_S, Y_S, Z_S, l_S corpul sau coada listei. Avem: - membru (Element, Listă) are semnificația:

<Element este membru al listei Listă>

membu ($X, [X | X_S]$).

membu ($X, [Y | Y_S]$): - membru (X, Y_S)

În astfel de predicate recursive apar restricții ce se cer îndeplinite și anume: a) **condițiile de mărginire** și b) **condiția de recursivitate**.

Astfel, condițiile de mărginire sînt dictate de apartenența sau nu a lui X la listă. Astfel, prima regulă clauzală dictează dacă procesul de căutare a lui X în listă este continuat sau stopat prin prezența lui X în capul listei. Neapartenența lui X la listă este dictată de căutarea procesată recursiv în a doua regulă clauzală, pînă la atingerea condiției <membu ($X, []$ >, moment în care procesul încetează, raportînd neapartenența lui X la listă.

În procesul de recursivitate, încercarea de satisfacere a predicatului **membu** prin apel la el însuși produce după fiecare încercare o listă mai scurtă decît cea precedentă, procedeul oprindu-se dacă X devine identic cu capul unei sublistă sau dacă se ajunge la lista vidă $[]$, moment în care recursia se oprește.

Să analizăm un alt program logic, și anume, programul care generează cel mai mare divizor comun a două numere X și Y . El are forma:

- cmmdc (X, Y, D)

cu semnificația: $D = \text{c.m.m.d.c.}(X, Y)$ (cmmdc are semnificația "celui mai mare divizor comun").

(1) cmmdc (X, X, X)

(2) cmmdc (X, Y, D): - $X < Y$, cmmdc (Y, X, D)

(3) cmmdc (X, Y, D): - $X > Y$, $X1$ is $X-Y$, cmmdc ($X1, Y, D$).

Înainte de analiza acestui program să reamintim notațiile folosite:

- 1) conjuncția **"&"** între 2 predicate are semnificația conjuncției "și" logice, următoarele echivalențe fiind imediate: " P, Q " \iff " P și Q " \iff " $P \wedge Q$ ";
- 2) semnul **":-"** sau **"←"** are semnificația lui "if" logic sau a implicației logice "→";
- 3) conjuncția **";"** între 2 predicate are semnificația disjuncției "sau" logice, următoarele echivalențe fiind imediate: " $P; Q$ " \iff " P sau Q " \iff " $P \vee Q$ " \iff " P or Q ";
- 4) conjuncția **","** prezintă după un predicat atestă validitatea faptului predicativ. Astfel, existența lui P în program face să aibă loc realizarea P la întrebarea P ?

Componente de bază în programarea logică

Cele trei clauze prezentate în programul `cmmdc` disting cele trei posibilități între argumentele numerice X și Y :

- 1) egalitatea valorilor, $X = Y$;
- 2) primul argument este mai mic;
- 3) primul argument este mai mare.

Observăm prezența operatorilor relaționali $<$; \leq ; $>$, \geq) în limbajul predicativ, sensul lor fiind același ca și în programarea convențională. Distincție face operatorul echivalenței $=$ ce diferă de ceilalți în sensul că operanzii pot fi termeni nenumerici sau variabile, în ultimul caz, rezultatul aplicării lui $=$ fiind instantarea comună sau unificatorul celor doi operanzi indiferent dacă ei sînt numerici sau nu. Dacă amîndoi operanzii sînt numere, atunci operatorul $=$ funcționează ca test de egalitate între numere.

Operatorul `is` este operatorul de asignare aritmetică în sensul că, dacă membrul stîng al operatorului este o variabilă atunci el va fi instantat cu rezultatul calculului numeric din membrul drept (`X is 2+3` va instanta variabila X cu 5 sau avem substituția $\{X/5\}$).

O altă variabilă mai naturală a programului s-ar putea scrie:

- (1) `cmmdc (X,0,X): - X > 0`
- (2) `cmmdc (X,Y,D): - mod (X,Y,Z), cmmdc (Y,Z,D)`

unde prin `mod (X,Y,Z)` se înțelege predicatul "modulo" care atestă că Z este restul împărțirii lui X la Y sau simbolic:

`mod (deîmpărțit, împărțitor, rest).`

La rîndul său, "mod" poate fi reprezentat prin program astfel:

- `mod (X,Y,X): - X < Y`
`mod (X,Y,Z): - plus (X1,Y,X), - mod (X1,Y,Z)`

unde predicatul "plus" are semnificația:

`plus (Operand1, Operand2, Operand1 + Operand2)`

- Deci `plus (X1,Y,X): - X1 = X - Y;`
`plus (0,X,X).`

Observăm, în final, echivalența celor două programe.

2. Rezoluția în forme clauzale

Am arătat în capitolul I că scrierea unei structuri predicative sub formă normală conjunctivă sau, mai precis, sub formă de conjuncții de disjuncții comportă șase etape. Dacă atunci studiul l-am făcut asupra unor predicate scrise ca rbd-uri formale, mai teoretic, să încercăm acum trecerea unor reguli clauzale concrete prin cele șase etape ale transformării lor din forma compusă în conjuncții de reguli disjunctive.

Să considerăm, astfel, propoziția:

"Toate femeile sînt mame"

sau formal:

oricare $(X, \text{femeie}(X) \Rightarrow \text{mamă}(X))$

- 1) Să eliminăm implicația " \Rightarrow ":

oricare $(X, \neg \text{femeie}(X) \text{ or } \text{mamă}(X))$

înțelegînd prin "or" operatorul "sau", (" \vee ").

- 2) Unicitatea negației în fiecare regulă este asigurată de regulile prezentate în capitolul 1. În cazul nostru există o singură negație în predicatul $\neg \text{femeie}(X)$. O expresie de forma \neg oricare $(X, \text{femeie}(X))$ s-ar putea transforma în există $(X, \neg \text{femeie}(X))$.

Componente de bază în programarea logică

3) Skolemizarea

Dacă propoziția noastră s-ar complica puțin, în sensul că am avea:
"Orice femeie care este mamă are un copil"

sau formal:

$$\text{oricare } (X, \text{femeie}(X)) \Rightarrow \text{există } (Y, \text{mamă}(X, Y))$$

unde mamă (X, Y) are semnificația "X este mama lui Y". Skolemizarea transferă propoziția în:

$$\text{oricare } (X, \text{femeie}(X)) \Rightarrow \text{mamă}(X, f(X))$$

unde f(X) este funcția Skolem cu proprietatea că $Y = f(X)$.

4) Eliminarea cuantificatorilor se face scoțându-i în afara formulelor atomice, fără afectarea conținutului. Relația noastră devine:

$$a) \neg \text{femeie}(X) \text{ or } \text{mamă}(X);$$

sau

$$b) \neg \text{femeie}(X) \text{ or } \text{mamă}(X, f(X));$$

Putem înlocui operatorul "or" cu echivalentul său de punctuație ";", regulile devenind:

$$a) \neg \text{femeie}(X) ; \text{mamă}(X).$$

$$b) \neg \text{femeie}(X) ; \text{mamă}(X, f(X)).$$

5) Distribuția conjuncției "și" (\wedge) peste disjuncțiile "sau" (\vee) - se face cu regulile distribuției ca în exemplul:

$$\text{succes}(X) \vee (\text{scris}(\text{Adi}, X) \wedge (\text{trist}(\text{Adi}) \vee \text{nemulțumit}(\text{Adi})))$$

cu echivalentul "Pentru orice X, ori X este succes, ori atît X este scris de Adi și Adi e trist sau nemulțumit, se transformă în echivalența:

$$((\text{succes}(X) \vee \text{scris}(\text{Adi}, X)) \wedge (\text{succes}(X) \text{trist}(\text{Adi}) \vee \text{nemulțumit}(\text{Adi})))$$

cu semnificația:

"Ori X este un succes ori Adi a scris X și ori X este succes ori Adi e trist și nemulțumit".

6) Punerea sub forme clazale impune scrierea tuturor membrilor conjuncțiilor sub forme modulare, ca elemente ale unei mulțimi conjunctive:

$$\{[(\text{succes}(X) \vee \text{scris}(\text{Adi}, X)), [(\text{succes}(X) \vee \text{trist}(\text{Adi}) \vee \text{nemulțumit}(\text{Adi}))]]\}$$

↑

element 1 conjuncție "și" element 2

Scrierea sub formă clazală nu este totuși foarte comodă în stadiul 6 datorită faptului că orice clază poate fi o expresie de literalii (formule atomice și negații ale lor).

Rezolvarea scrierii convenabile a clazelor se face punînd literalii negați în dreapta celor ne-negați, ei fiind despărțiți de operatorul deducției sau implicației logice ":-" sau "←". Această observație provine din considerentul că există următoarele echivalențe evidente:

$$(P \vee Q) \vee (\neg R \vee \neg S \dots)$$

$$\text{sau: } (P \vee Q) \vee \neg (R \wedge S \wedge \dots)$$

$$\text{sau: } (R \wedge S \wedge \dots) \Rightarrow (P \vee Q)$$

sau încă:

$$P ; Q :- R, S, \dots$$

adică P sau Q au loc dacă R și S și..., au loc.

Sub aceste considerente, exemplele de mai înainte admit scrierea:

$$1) \text{mamă}(X) :- \text{femeie}(X).$$

$$2) \text{mamă}(X, f(X)) :- \text{femeie}(X).$$

Următorul pas în capitolul 1 îl constituia formarea rezolvenților sau analizarea posibilelor reguli noi ce se pot forma din cele prezente scrise s.a.

Componente de bază în programarea logică

formă clauzală. Acest lucru se realiza prin disjuncția convenabilă între reguli clauzale, eliminarea aparițiilor de forma P și $\neg P$ sau: $P:-P$, precum și instantarea variabilelor prin disjuncție.

De exemplu să presupunem că avem clauzele:

(1) mamă($X, f(X)$) ; copil(X) :-.

(2) copil(Z) :- tată(Y, Z).

(3) tată(U, Adi) :- mamă($Ioana, Adi$)

Regula (2) cu (3) ne dă prin disjuncție și instantare regula (4):

(4) copil(Adi) :- mamă($Ioana, Adi$).

Să observăm că tată(Y, Z) a fost instantat cu tată(U, Adi) prin substituția $\{Y/U, Z/Adi\}$, clauzele tată(U, Adi) și \neg tată(U, Adi) eliminându-se din cauza prezenței lor sub forma:

tată(U, Adi) :- tată(U, Adi).

Compunerea prin disjuncție a regulilor (1) și (4) ne dă:

copil(Adi), copil(Adi) :- . sau

(5) copil(Adi) :- echivalent cu faptul că Adi este copil.

Întrebarea firească care se pune acum este dacă aceste două procedee ale rezoluției pot ajuta la demonstrarea vreunei probleme corecte. Răspunsul este afirmativ și demonstrarea acestui rezultat, cunoscut ca **refulare completă** va constitui obiectul complet al unui capitol următor. Cert este că **refularea completă** permite demonstrarea de teoreme și luarea deciziei dacă un set de clauze este **corect pus**, în sensul că **rezoluția** aplicată lor este corectă, dacă din producerea rezolvenților se poate deduce clauza vidă NIL sau, formal, ":-".

În cazul nostru, prezența unei clauze:

(6) :- copil(Adi)

va produce, împreună cu (5), rezolvanta:

:- (sau NIL).

Astfel, consecința rezoluției este că Adi este un copil.

Putem spune acum că, privit în general, un sistem Prolog este un demonstrator de teoreme pe bază de rezoluție pentru clauze Horn. Strategia particulară folosită este cea a rezoluției prin **insertie liniară**. Această metodă începe cu punerea unei realizări și apoi analiza disjuncției ei cu o ipoteză clauzală dată pentru a forma o nouă clauză. Apoi se formează disjuncția dintre această nouă clauză și o altă ipoteză clauzală dată, și așa mai departe.

În acest mod, ultima clauză derivată este reprezentată ca o conjuncție a realizărilor ce mai trebuie satisfăcute pentru demonstrarea problemei puse. Astfel, în fiecare stadiu obținem o clauză al cărei **cap** instantează (unifică) cu o realizare, elimină realizarea satisfăcută și adaugă **corpul** clauzei instantate la mulțimea realizărilor ce mai trebuie satisfăcute.

Pentru fiecare realizare, sistemul alege clauzele într-o ordine fixată, fiecare nouă clauză fiind luată în considerare numai dacă cele precedente nu au condus la satisfacerea realizării.

O altă metodă este aceea prin care sistemul nu ține cont de mai multe căi simultane spre soluții posibile. Ultima metodă are avantajul că, dacă o soluție există, atunci ea va fi găsită, pe cînd prima poate duce la o recursivitate în adîncime care să producă un număr de iterații ce pot umple memoria calculatorului ducînd, astfel, la blocaj de sistem fără un răspuns la stadiul în care s-a ajuns în căutare. Astfel, utilizarea metodelor programării logice în Prolog poate duce la un impas. Esențial este, după cum vom vedea, cunoașterea profundă a tehnicilor de programare ce permit alegerea strategiei convenabile pentru succes.

Limbaje de programare în designul sistemelor de operare și al aplicațiilor

Limbajul C (II)

Variabile, operatori și expresii

Ca și în alte limbaje, obiectele de bază manipulate într-un program C sînt reprezentate de variabile și constante.

Variabilele devin accesibile programului prin declararea lor. Declararea variabilelor se realizează prin:

- precizarea tipului acestora, tip care determină mulțimea valorilor pe care le pot avea variabilele și ce operații se pot executa asupra lor;
- precizarea numelui;
- precizarea eventualelor valori inițiale pe care le pot avea.

1. Numele variabilelor

Numele sînt constituite ca șiruri de caractere formate din litere și cifre și caracterul special "-" (underscore).

Primul caracter din șir trebuie să fie literă. Caracterul "-" (underscore) este tratat ca literă dar este indicat ca el să nu fie utilizat ca prim caracter în nume deoarece este utilizat la formarea numelor funcțiilor din bibliotecă. În C numele definite în programul sursă (inclusiv funcțiile) au prioritate mai mare decît cele din bibliotecă.

Numele pot avea diverse dimensiuni (funcție de implementare) dar uzual sînt tratate numai primele 31 de caractere. Pentru numele de funcții și variabile externe numărul de caractere considerate este mai mic (între 6 și 8; garantat prin standard 6, ca numele funcțiilor **FORTRAN**, de exemplu).

În C se face distincție între literele mari și literele mici. Ca regulă de stil este indicat ca numele variabilelor să fie atribuite cu litere mici iar numele constantelor simbolice cu litere mari.

În C sînt rezervați următorii identificatori:

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

care reprezintă cuvinte cheie ale limbajului.

2. Tipuri de date și dimensiunea lor

În C dispunem de un număr mic de tipuri de date de bază, și anume:
char: un singur byte capabil să manipuleze un singur caracter din setul de caractere acceptat de calculatorul gazdă;

int: un întreg reflectînd în general dimensiunea naturală a întregilor în cal-

Variabile, operatori și expresii

calculatorul gazdă. De exemplu pentru CORAL, în care dimensiunea întregilor este de 16 biți, `int` va avea dimensiunea de 16 biți;

`float`: virgulă mobilă, simplă precizie;

`double`: virgulă mobilă, dublă precizie.

Pentru `int` se mai pot aplica calificatorii:

`short`: dimensiune normală pentru `int` (16 biți);

`long`: dimensiune dublă pentru `int`.

Calificatorul `long` poate fi aplicat și lui `double` obținând astfel variabile în virgulă mobilă precizie extinsă.

Exemple:

`int x,y,z;`

`long int i,j;`

`short int x;`

În declarațiile cu `short` și `long` cuvântul `int` poate fi omis, de exemplu:

`long i,j;` este echivalent cu `long int i,j.`

Declarațiile `long` și `short` permit obținerea unor dimensiuni practice diferite pentru `int` extinzând de fapt gama de tipuri de date.

În tabela 1 este redată dimensiunea variabilelor din C pe diverse tipuri de calculatoare.

Tabela 1. Precizia variabilelor - dependentă de calculator

Calculator	CORAL	Honeywell 6000	IBM 370	Interdata 8132
Tip variabilă	ASCII 8 biți	ASCII 9 biți	EBCDIC 8 biți	ASCII 8 biți
<code>char</code>	8	9	8	8
<code>int</code>	16	36	32	32
<code>short</code>	16	36	16	16
<code>long</code>	32	36	32	32
<code>float</code>	32	36	32	32
<code>double</code>	64	72	64	64

C tratează variabilele `char` (de 1 byte) ca și cum acestea ar conține valori întregi.

Ca și pentru întregi, lungimile pentru virgulă mobilă ale obiectelor sînt dependente de implementări, `float`, `double` și `long double` pot fi reprezentate pe 1, 2 sau 3 lungimi distincte.

Variabilele de tip `int` și `char` pot fi reprezentate cu semn (`signed`) sau fără semn (`unsigned`).

Numerele fără semn (`unsigned int`, `unsigned char`) sînt pozitive sau nule și pot fi considerate, din punct de vedere aritmetic, ca numere modulo 2^n , unde n reprezintă numărul de biți pentru reprezentarea tipului respectiv.

De exemplu dacă `char` are 8 biți atunci variabilele declarate `unsigned char` pot lua valori în intervalul [0,255] iar cele declarate `signed char` pot lua

Variabile, operatori și expresii

valori în intervalul [-127,+127].

Dimensiunile acestor elemente ale limbajului sînt specificate în fișierele standard <limits.h> și <float.h> care conțin numele constantelor simbolice asociate acestora. De asemenea, funcția standard `sizeof()` a lui C returnează dimensiunea elementelor specificate ca argument.

Vom vedea, într-un articol ulterior, modul în care utilizatorul, pornind de la aceste tipuri de bază poate să construiască, cu investiții de programare minime (în general "declarative"), tipuri de date extrem de complexe.

3. Constante

Tipurile de constante admise în C sînt:

- 1) întregi: șir de cifre zecimale precedate eventual de semn și urmate eventual de specificatorul preciziei de reprezentare:

[semn] cifre [precizie]

Exemple: 123, +123, -123L, 1230L, 123ul etc.

Dacă valoarea desemnată de constantă nu poate fi reprezentată pe `int` sau dacă constanta are specificatorul de precizie `l` (el) sau `L`, atunci se va reprezenta ca `long`. De exemplu, valoarea +1234567, care nu încapă în cei 16 biți ai lui `int` (valoarea minimă +32767), va fi reprezentat pe un `long int` ($2^{31}-1$, valoarea maximă).

Constantele fără semn sînt scrise cu specificatorul de precizie `u` sau `U` (constantele `unsigned long` fără semn au specificatorul `ul` sau `UL`);

- 2) caracter: 'car', un singur caracter între '':

Valoarea unei constante caracter este o valoare întreagă reprezentabilă pe un byte, cu sau fără semn.

Exemple: 'A', 'a', 'c', '1', '0' etc.

În acest exemplu constanta caracter '0' are valoarea, în setul de caractere ASCII, 48 care nu are nici o legătură cu valoarea întreagă 0.

Acest tip de constante pot fi utilizate în scrierea expresiilor (cu operații numerice) ca orice alți întregi.

Anumite caractere, destinate în general controlului modului de lucru al perifericelor, se introduc prin secvențe de tipul '\car' (secvențe *escape*).

Aceste secvențe sînt percepute extern ca două caractere dar reprezintă un singur caracter. Prin același tip de secvență putem introduce și configurații de biți astfel:

'\ooo', unde ooo reprezintă trei cifre octale, $o \in \{0,1,\dots,7\}$; sau prin

'\xhh', unde hh reprezintă una sau mai multe cifre hexazecimale (0...9, a...f, A...F).

Setul complet de secvențe *escape* este redat în tabela 2.

De exemplu dacă dorim să reprezentăm form feed ASCII putem să definim o constantă simbolică (cu comanda `#define`) `FF` astfel:

```
#define FF '\014'
```

Tabela 2. Setul complet de secvențe *escape*

Secvența	Semnificație
\a	alert (bell) : alarmă alertă

Variabile, operatori și expresii

Secvența	Semnificație
\b	backspace : spațiu înapoi
\f	formfeed : început de bloc
\n	newline : linie nouă
\r	carriage return : retur de car
\t	horizontal tab : tabulare orizontală
\v	vertical tab : tabulare verticală
\\	backslash : caracterul \ însuși
\?	question mark : semn întrebare
\'	un singur apostrof
\"	apostrof dublu
\ooo	număr octal
\xhh	număr hexazecimal
\0	valoarea zero binar

3) octale: un șir de cifre octale precedat de 0 (zero) cifrele din șir trebuie să aparțină intervalului $[0,7] \cap \mathbb{N}$.

De exemplu dacă dorim să inițializăm variabila unsigned a cu valoarea octală 1777 vom scrie astfel:

```
a = 01777;
```

4) hexazecimale: un șir de cifre hexa precedat de litera x sau X.

Cifra din șir trebuie să aparțină mulțimii $\{0,1,2,3,4,5,6,7,8,9, a,b,c,d,e,f,A,B,C,D,E,F\}$.

De exemplu, prin 48 în hexa, putem atribui variabilei caracter a valoarea '0' astfel:

```
a = x48;
```

5) reale: acest tip de constante pot fi definite utilizând una din notațiile:

- cu marcă zecimală: [semn][cifre].[cifre] ca în exemplele +123.15, -1.3, 0.24, +2.71 etc;

- cu exponent (notația științifică):

```
[semn]{cifre|0}•cifre{E|e}[semn]{cifre}
```

De exemplu: 0.12E3, 0.12e3, 127.432E-6 etc.;

6) șir: un șir de caractere între " ", "șir" de exemplu "Programarea în limbajul C".

Un șir este marcat la sfârșit cu valoarea zero binar (\0 convenția ASCII), reprezentată pe un byte. Practic un șir poate fi considerat (și este) un vector de caractere ale cărui elemente sînt caractere. Numărul de elemente ale vectorului este dat de numărul de caractere cuprinse

Variabile, operatori și expresii

între apostroafe +1 pentru valoarea 0 binar (deci 25 pentru exemplul nostru);

- 7) **constante enumerative:** o enumerare este o listă de constante cu valoare întreagă (listă de valori) în care elementele din listă sînt numerotate începînd cu valoarea 0. De exemplu, prin declarația:

```
enum răspuns {DA,NU,YES,NOT}
```

valorilor din listă li se asociază numerele 0->DA, 1->NU, 2->YES și 3->NOT.

Numele din enumerațiile date într-un program trebuie să fie distincte. La enumerare, numelor li se pot atribui valori, de exemplu: `enum escapes {BELL = '\a', BACKSPACE = '\b', HTAB = '\t', NEWLINE = '\n', VTAB = '\v', RETURN = '\r', FORMFEED = '\f', QUESTION = '\?'}`

Enumerările reprezintă o cale de asociere a valorilor constante la nume simbolice care diferă de asocierea realizată cu `#define` prin faptul că se realizează în programul nostru și nu în afara sa.

- 8) **expresii constante:** o expresie constantă este o expresie care are ca operanzi numai constante. Aceste expresii se evaluează în faza de compilare a programului și în concordanță cu acest lucru expresia constantă poate fi utilizată în orice loc în care sintaxa limbajului C admite utilizarea unei constante.

- 9) **definirea constantelor simbolice:** opțional, la nivelul programului (declarației `main()` sau declarației nume funcție()) se pot atribui nume constantelor obținînd astfel constante simbolice (scrise ca regulă de stil cu litere mari). Declararea constantelor simbolice se face cu directiva pentru macroprocesor:

```
#define nume_constantă valoare
```

Exemple:

```
#define LGLINE      81
#define FORMFEED   '\014'
#define NULL        '\0'
#define HOMEKEY    (256+71) /*expresie constantă*/
#define UPARROW    (256+72) /*expresie constantă*/
#define PGUPKEY    (256+73) /*expresie constantă*/
#define BEGIN      {
#define END        }
#define PAGE        (24*LGLINE) /*expresie constantă*/
```

Aceste constante simbolice sînt înlocuite, în textul programului, cu valorile atribuite lor, după care are loc compilarea efectivă (similar modului de lucru al XPROC la FELIX sau al procesorului de comenzi indirecte de la CORAL, în cazul utilizării parametrilor formali sau al apelurilor imbricate de proceduri).

Constantele simbolice reprezintă o alternativă extrem de comodă pentru adaptarea programelor C la specificul unui calculator gazdă sau terminal (faza de instalare a unui produs). Este suficientă înlocuirea lor cu valorile specifice unui anumit calculator. De exemplu, dacă UPARROW (săgeată în sus - deplasare cursor în sus) are valoarea (256+10) este suficient să schimbăm definiția constantei cu `#define UPARROW (256+10)` și să recompilăm programul în acest context. De regulă este preferabil, pentru aplicații mari, să construim fișiere standard cu astfel de constante simbolice care vor fi utilizate de către realizatorii aplicațiilor prin citarea acestor fișiere cu ajutorul directivei `#include <nume_fișier>`.

Designul și exploatarea foilor de calcul electronice (II)

Tipuri de date Lotus

O celulă poate conține o dată din unul din tipurile următoare:

- număr;
- formulă;
- etichetă.

În funcție de primul caracter tastat, 1-2-3 decide dacă intrarea va fi o valoare, formulă sau etichetă, modificând corespunzător indicatorul de mod (READY -> VALUE, READY -> LABEL).

O dată de tipul număr respectă următoarele reguli:

- începe cu unul din caracterele: 0-9; separatorul zecimal selectat (. sau ,); +; -; !; (
- conține pînă la 240 caractere;
- este în intervalul $[10^{-99}; 10^{99}]$. 1-2-3 lucrează intern cu numere în intervalul $[10^{-303}; 10^{303}]$;
- poate începe cu un simbol monetar, nealfabetic, de exemplu \$;
- poate conține un singur separator zecimal (.,) specificat prin setare;
- se poate termina cu % indicînd un procentaj;
- se poate introduce și în formatul virgulă mobilă (mantisă E exponent).

Dacă numărul de cifre introduse depășește dimensiunea celulei, 1-2-3 va afișa în celulă * * *..., luînd însă în calcule valoarea numărului introdus; la modificarea adecvată a dimensiunii celulei, numărul apare integral.

1-2-3 lucrează cu numere avînd pînă la 15 cifre zecimale; din acestea, pe panoul de control nu vor fi afișate decît 9 cifre, deși calculele se efectuează cu toate cifrele introduse.

O dată de tipul formulă este o instrucțiune care operează cu numere sau șiruri de caractere (text). Ea poate conține numere, texte, operatori, adrese de celule, funcții 1-2-3 (o parte din noțiuni vor fi detaliate ulterior).

O dată de tipul formulă respectă următoarele reguli:

- trebuie să înceapă cu unul din caracterele: 0-9; separator zecimal; +; -; ; " ; !;
- conține pînă la 240 caractere;
- începe cu semnul + dacă prima parte a formulei este o adresă de celulă.

La validarea introducerii unei formule, în foaie va apare rezultatul ei. Vizualizarea formulei nu mai este posibilă decît mutînd cursorul chiar pe celula respectivă sau printr-o modificare a formatului de afișare al celulei.

Operanzii care participă în formulă pot fi:

- Numere;
- Șiruri de caractere (text);
- Adrese de celule care conțin numere sau etichete;
- Nume de blocuri (în particular celule);
- Funcții Lotus.

O mențiune specială trebuie făcută referitor la adresarea celulelor în formule. Adresarea acestora poate fi: relativă, absolută sau mixtă. Adresa unei celule în forma relativă are forma: <literă coloană><număr

Tipuri de date Lotus

linie> (exemplu B7). Dacă cu ajutorul comenzii /copy formula care conține adresa relativă este copiată într-o celulă situată m coloane la dreapta și n linii mai jos, litera noii adrese va fi vechea literă deplasată în mulțimea ordonată a literelor coloanelor cu m litere, iar numărul de linie se va incrementa cu n unități. Introducerea caracterului \$ înaintea literei coloanei sau numărului coloanei inhibă modificarea acestora la deplasarea formulei obținându-se:

- pentru \$<literă coloană>\$<număr linie> adresare absolută;
- pentru \$<literă coloană><număr linie> sau <literă coloană>\$<număr linie> adresare mixtă.

Definim, de asemenea, noțiunea de funcție Lotus. Aceste funcții specifice se numesc @ funcții, apelarea lor făcându-se în formatul: @ nume_funcție (arg₁,...,arg_n).

Aceste funcții permit efectuarea, printre altele, a unor calcule matematice, logice, cu șiruri de caractere.

1-2-3 utilizează 3 tipuri de formule:

- formule aritmetice. Operanzii lor sînt valori numerice iar operatorii sînt cei aritmetici. Exemplu: +A7*3 are ca efect înmulțirea conținutului celulei A7 cu 3;

- formule text. Operanzii lor sînt șiruri de caractere; operatorii sînt specifici. Exemplu: +A18 "Număr" are ca efect concatenarea textului din celula A1 cu textul "Număr", rezultatul aflîndu-se în celula care conține formula;

- formule logice, care compară valorile din două sau mai multe celule sau valoarea dintr-o celulă cu o constantă, rezultînd o valoare 0 sau 1. Exemplu: +Cell>= 10 are ca rezultat 0 dacă conținutul celulei numite Cell<10 respectiv 1 dacă acesta ≥ 10, rezultatul aflîndu-se în celula care conține formula.

Operatorii care pot fi utilizați în formule au fiecare asociat un număr de precedență care reprezintă ordinea în care 1-2-3 efectuează operațiile în formulă. 1-2-3 efectuează o operație cu o prioritate direct proporțională cu numărul de precedență.

La același număr de precedență operațiile se execută secvențial de la stînga la dreapta.

Operator	Operație	Număr precedență
^	Exponențiere	7
-, +	Operatori semn	6
*, /	Înmulțire, împărțire	5
+, -	Adunare, scădere	4
=<>	Egal, diferit	3
<>	Mai mic, mai mare	3
>=	Mai mare sau egal	3
<=	Mai mic sau egal	3
"NOT"	Negare logică	2
"AND"	Și logic	1
"OR"	Sau logic	1
&	Concatenare șiruri	1

Tabelul prezintă deci diversele operații admise, cu operatorii aferenți precum și numerele de precedență corespunzătoare.

Evident, utilizînd parantezele () se pot depăși regulile de precedență. De altfel, se recomandă utilizarea parantezelor ori de cîte ori există cel mai mic dubiu.

O dată de tipul etichetă este orice altă intrare admisă de 1-2-3, care nu

Tipuri de date Lotus

este număr sau formulă.

O dată de tipul etichetă respectă următoarele reguli:

- începe cu unul din caracterele prefix de etichetă: ' ; " ; ^ ;
- poate conține pînă la 240 caractere din mulțimea LICS (Lotus International Character Set).

Caracterele prefix de etichetă au rolul de a determina amplasarea în celulă a etichetei: ' - alinierea la stînga; " - alinierea la dreapta; ^ - centrare.

Caracterele prefix nu apar vizibil în celulă, ele pot fi vizualizate doar la poziționarea cursorului pe celula respectivă.

Dacă se dorește utilizarea unor etichete numerice sau care încep cu numere este necesară prefixarea acestora cu unul din caracterele prefix.

La introducerea unei etichete de dimensiune mai mare decît dimensiunea celulei aceasta se poate extinde și în celula vecină, cu condiția ca aceasta să fie goală; oricum, indiferent de partea afișată, 1-2-3 memorează integrat eticheta introdusă.

Blocuri

Unitatea de bază a foii este celula. Celulele se pot grupa în blocuri; anumite comenzi 1-2-3 nu operează decît cu blocuri. Un bloc poate fi alcătuit dintr-o celulă, o linie, o coloană, mai multe linii sau coloane, alcătuiind în final o structură dreptunghiulară de celule.

Declararea blocurilor poate fi făcută în trei moduri:

- tastînd adresele celulelor care limitează blocul;
- prin expandare, pornind de la celula curentă în cadrul dialogului interactiv la execuția unei comenzi;
- utilizînd un nume de bloc, bloc creat cu comanda /Range Name Create.

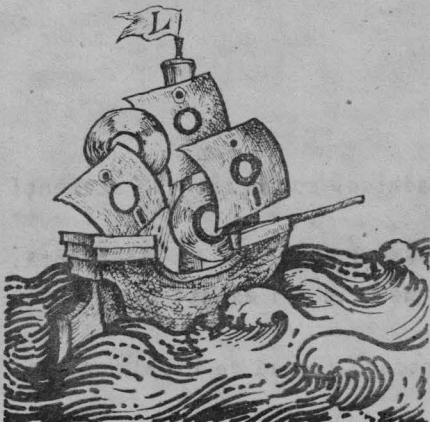
Deci se recurge la:

- tastarea adreselor celulelor diagonal opuse, separate de două puncte.

De exemplu A1..B5 indică blocul care are în stînga su celula A1 și în dreapta jos celula B5;

- în timpul dialogului interactiv, la execuția unei comenzi se tastează., determinîndu-se așa numita celulă de ancorare; apoi se deplasează cu tastele de poziționare cursorul; aria inclusă în bloc este vizibilă pe ecran. În final se apasă RETURN, validîndu-se selecția blocului;

- un bloc poate primi un nume; ulterior toate comenzile care se referă la blocul respectiv îl vor referi prin numele său. De exemplu, blocul A1..B5 poate primi numele B11 utilizînd comanda /Range Name Create.



Tehnici software în PROLOG și ASSEMBLER pentru microsystemele PC (I)

Exploatarea microsystemelor prin limbajul PROLOG și ASSEMBLER

1. Limbajul PROLOG ca limbaj convențional

Dincolo de mediul programării logice, în care tehnicile de lucru se înscriu în sfera principiilor programării logice, avem nevoie de exploatarea metodelor tradiționale ale programării în executarea unor aplicații care cer rutine și apeluri pentru servicii de intrări/ieșiri, control al procesorului, gestiune a ecranului, a tastaturii, avertizării acustice, gestiune a fișierelor și a unităților de discuri, manipulare a șirurilor de caractere etc. Date fiind structura specifică și scopurile precise ale unui limbaj al inteligenței artificiale, aceste servicii nu apar ca parte integrantă a structurii compilatoarelor sau interpretoarelor logice. De aceea s-a impus elaborarea unor subansamble specifice, formate din rutine și proceduri speciale încorporate ca module distincte în structura procesoarelor logice, în speță a limbajului Prolog. Astfel, acesta devine utilizabil în aplicarea resurselor convenționale prin următoarele facilități: a) asigurarea unei interfețe programabile cu limbajele de programare tradiționale Fortran, Pascal, C, Basic, Assembler; b) rutine specifice ale procesorului ce permit executarea apelurilor de sistem și generatoare a întreruperilor software și hardware.

Să încercăm, în continuare, o prezentare detaliată a acestor tehnici argumentând eficiența înțelegerii și utilizării lor prin aplicații și programe care permit utilizatorului să stăpânească și să manipuleze un procesor și un sistem de operare pentru satisfacerea celor mai exigente nevoi. Cele prezentate în continuare se orientează asupra exploatării microcalculatoarelor personale profesionale în compunerea cărora intră microcalculatoare din familia Intel de tip 8086, 8088, 80186, 80286, 80386 și care lucrează sub sistemul de operare MS-DOS (începând cu versiunea 3.0), OS/2 (asupra căruia nu sînt valabile întreruperile software ale sistemului MS-DOS prin funcția 21H), PC-DOS sau XENIX. Date fiind largă sa răspîndire și flexibilitatea în exploatare, vom face apel specific la limbajul **TURBO PROLOG** și **MASM**, cu mențiunea că aceleași rutine, eventual cu mici diferențe sintactice, există în orice procesor logic PROLOG, apelurile și convențiile următoare rămînînd valabile.

2. Întreruperi și apeluri de sistem

Așa cum menționam, limbajul Prolog pune la dispoziția utilizatorului cîteva predicate interne ce permit programatorului accesul la nivelul sistemului de intrări/ieșiri fundamental al calculatorului personal, cunoscut ca **BIOS**. Ceea ce este însă foarte important rezidă în cunoașterea rutinelor (și a funcțiilor acestora) existente în **BIOS** și, mai ales, a convențiilor necesare exploatării lor, apelul prin predicatul Prolog fiind doar o problemă de înlocuire parametrică.

Predicatele la nivel de sistem puse la dispoziție de compilatorul de Turbo Prolog sînt:

- 1) **beep** - a cărui acțiune este trimiterea unui semnal sonor la difuzorul calculatorului;

2) bios - (*Numar-Intrerupere, Registre-de-intrare, Registre-de-iesire*).

Parametrii *Registre-de-intrare, Registre-de-iesire* aparțin unui domeniu intern al compilatorului, numit *regdom*, definit la declarare prin:

$\text{regdom} = \text{reg}(\text{AX}, \text{BX}, \text{CX}, \text{DX}, \text{SI}, \text{DI}, \text{DS}, \text{ES})$

unde AX, BX, CX, DX, SI, DI, DS, ES sînt registre pe 16 biți ai microprocesoarelor Intel din familia 8086, 8088, 80X86 ($X \in \{1, 2, 3\}$) ce sînt declarate de tip întreg. Funcția acestui predicat, cel mai puternic dintre cele cu efect la nivel de sistem este dată de:

Întreruperea BIOS cu numărul *Numar-Intrerupere* se declanșează avînd valorile specifice pentru regiștri dată în parametrul de *Registre-de-intrare* urmînd ca rezultatele execuției întreruperii specificate să fie returnat în *Registre-de-iesire* a căror utilizare ulterioară să ne confere răspunsul la cererea utilizator formulată.

3) *date (An, Luna, Zi)* - parametrii *An, Luna, Zi* sînt de tip întreg. Dacă acești parametri sînt apelul *date (A,B,C)*; - citește data din ceasul intern al calculatorului în variabilele *A, B, C*, altfel, apelul *date (i,j,k)*; - potrivește ceasul intern la anul *i*, luna *j*, ziua *k*.

4) *port-byte (Nr-Port, Valoare)* - parametrii *Nr-Port, Valoare* sînt de tip întreg, avînd acțiunea de a trimite valoarea *Valoare* la portul cu numărul *Nr-Port*.

Acest predicat este echivalentul lui *out port, valoare* în Basic, C, Pascal și limbaj de asamblare.

5) *ptr-dword (Variabila-Sir, Segment, Deplasament)* - parametrii *Segment* și *Deplasament* sînt de tip întreg iar *Variabila-Sir* este de tip șir (string). Dacă se specifică doar *Variabila-Sir*, apelul întoarce în variabilele *Segment* și *Deplasament* adresa de bază a segmentului și poziția în acest segment (deplasamentul față de baza segmentului) a începutului șirului *Variabila-Sir*.

Dacă se specifică un segment de memorie cu adresa de bază *Segment* și o adresă oarecare în acest segment în valoarea *Deplasament*, apelul predicatului întoarce în *Variabila-Sir* conținutul locației de memorie de la adresa $\text{Segment} * 16 + \text{Deplasament}$ și următoarele valori consecutive în format ASCII, sub formă de șir, pînă la locația de memorie ce conține octetul cu valoare 0 (NULL). În fapt, utilizarea ulterioară a șirului se face apelînd adresa sa de bază, *Variabila-Sir*, ce marchează începutul unei succesiuni de caractere ASCII ce se termină cu caracterul NULL (valoare 0); sau în alte versiuni ale Prologului, caracterul terminal este \$ (semnul dolar).

6) *membyte (Segment, Deplasament, Continut)* - parametrii sînt de tip întreg. Acțiunea predicatului este duală: dacă *Continut* este specificat cu o valoare, aceasta va fi implantată la adresa de memorie dată de $\text{Segment} * 16 + \text{Deplasament}$. Dacă valoarea întregului *Continut* este liberă (deci nespecificată), atunci predicatul dă variabilei *Continut* valoarea octetului de memorie citit de la adresa $\text{Segment} * 16 + \text{Deplasament}$. Predicatul are acțiunea lui *poke*, adresă, valoare din BASIC.

7) *memword (Segment, Deplasament, Continut)* - parametrii sînt de tip întreg, acțiunea predicatului fiind similară cu cea a lui *membyte*, deosebirea constînd în faptul că în *Continut* se citește valoarea unui cuvînt de memorie (2 octeți) atunci cînd valoarea acestuia este literă.

8) *system (Comanda-Sir)* - parametrul este șir de caractere de tip *string*. Predicatul trimite sistemului de operare o comandă spre execuție, specificată în *Comanda-Sir*.

Exploatarea microsystemelor prin limbajul PROLOG și ASSEMBLER

9) **sound** (*Durata, Frecventa*) - parametrii sînt de tip întreg. Apelul acestui predicat produce un sunet cu frecvența dată de *Frecventa* și durata specificată în *Durata*, ce reprezintă sutimi de secundă.

10) **storage** (*Stiva, Cap, Coada*) - parametrii sînt de tip real iar acțiunea predicatului constă în specificarea în cele 3 variabile parametrice a mărimii zonelor de memorie folosite de Prolog în timp real, în execuție (zone de memorie aferentei stivei etc.).

11) **time** (*Ora, Minut, Secunde, Zecimi*) - parametrii sînt de tip întreg iar dacă parametrii au valoare anonimă, predicatul citește ora din ceasul intern și returnează valorile aferente în variabile, altfel setează timpul după valorile specificate în parametrii predicatului.

12) **trace** (*Simbol*) - parametrul *Simbol* este de tip symbol; acțiunile de tip **trace** (on), **trace** (off) permițînd trase de program sau nu.

Este necesar a prezenta în continuare cîteva întreruperi BIOS și valorile aferente registrelor sistem care pot genera funcții de sistem utile utilizatorului în gestionarea formei convenționale a aplicațiilor.

Serviciile ROM-BIOS sînt organizate pe categorii, fiecare categorie avînd o întrerupere aferentă, după cum urmează:

Întrerupere (în zecimal)	Funcție realizată
5	Tipărire pe ecran
16	Servicii video
17	Lista echipamentelor sistem
18	Mărimea memoriei
19	Servicii de disc
20	Servicii ale porturilor sîriale (RS-232)
21	Servicii pentru portul de casetă
22	Servicii de tastatură
23	Servicii ale portului paralel
24	Servicii ROM-BASIC
25	Inițializare sistem
26	Servicii pentru dată și oră

Următoarele macroui în limbaj de asamblare prezintă funcțiile întreruperii Bios 10h, cunoscută și ca întreruperea video și a căror utilizare este pe larg explicată în capitolele următoare. Prezentarea se face sub forma unui program denumit BIOS.INC, a cărui apel într-o secvență assembler prin comanda:

```
INCLUDE BIOS.INC
```

permite utilizatorului care lucrează în limbajul de asamblare MASM - versiunea 5.x - a firmei Microsoft să facă apel la aceste macroui prin numele declarat al lor în cele ce urmează, împreună cu parametrii ceruți:

```
.XCREF
```

```
.XLIST
```

```
IF1
```

```
;0Fh
```

```
CitMod          MACRO          ;citește modul video
                mov     ah,0Fh
                int     10h
```

Exploatarea microsystemelor prin limbajul PROLOG și ASSEMBLER

```

                ENDEM
;00h
SetMod          MACRO    mod      ;setez modul video specificat de
                MOV      al,mod    ;parametrul mod
                XOR      ah,ah
                INT      10h
                ENDEM
;0Bh
SetColor        MACRO    culoare  ;pun culoarea pe ecran cu codul
                SUB      bh,bh     ;în "culoare"
                MOV      bl,culoare
                MOV      ah,0Bh
                INT      10h
                ENDEM
;0Bh
SetPaleta       MACRO    culoare  ;pun paleta de culoare cu codul
                MOV      bh,l      ;în "culoare"
                MOV      bl,culoare
                MOV      ah,0Bh
                INT      10h
                ENDEM
;05h
SetPagina       MACRO    pagina   ;aleg pagina video
                MOV      al,pagina
                MOV      ah,05h
                INT      10h
                ENDEM
;03h
GetCur         MACRO    pagina   ;citesc poziția cursorului din "pagina"
                IFNB    <pagina>
                MOV      bh,pagina
                ELSE
                XOR      bh,bh
                ENDIF
                MOV      ah,03h
                INT      10h
                ENDEM
;02h
SetCurPoz      MACRO    coloana,rînd,pagina ;determin o poziție dorită a
                                                ;cursorului
                IFNB    <coloana>
                MOV      dl,coloana
                ENDIF
                IFNB    <rînd>
                MOV      dh,rînd
                ENDIF
                IFNB    <pagina>
                MOV      dh,pagina
                ELSE
                XOR      bh,bh
                ENDIF

```


Exploatarea microsystemelor prin limbajul PROLOG și ASSEMBLER

```

        mov     ah,02h
        int     10h
        ENDEM

;01h
SetCurMărime  MACRO  linie_sup_linie_inf      ;aleg un cursor cu o anumită
                                                ;formă (vezi capitolul 5)

        mov     ch,linie_sup
        mov     cl,linie_inf
        mov     ah,01h
        int     10h
        ENDEM

;08h
GetChAtr       MACRO  pagina                  ;citesc atributul unui
        IFNB    <pagina>                    ;caracter
        mov     bh,pagina
        ELSE
        sub     bh,bh
        ENDIF
        mov     ah,08h
        int     10h
        ENDEM

;09h
ScriuChAtr     MACRO  car,atrib,pagina,rep     ;setez attribute unui caracter
        IFNB    <car>
        mov     al,car
        ENDIF
        IFNB    <atrib>
        mov     bl,atrib
        ENDIF
        IFNB    <pagina>
        mov     bh,pagina
        ELSE
        xor     bh,bh
        ENDIF
        IFNB    <rep     >
        mov     cx,rep     t
        ELSE
        mov     cx,1
        ENDIF
        mov     ah,09h
        int     10h
        ENDEM

;0Ah
ScriuCh        MACRO  car,atrib,pagina,rep     ;scriu un caracter
        IFNB    <car>
        mov     al,car
        ENDIF
        IFNB    <atrib>
        mov     bl,atrib
        ENDIF
        IFNB    <pagina>

```

Exploatarea microsystemelor prin limbajul PROLOG și ASSEMBLER

```

mov      bh,pagina
ELSE
xor      bh,bh
ENDIF
IFNB    <repet>
mov      cx,repet
ELSE
mov      cx,1
ENDIF
mov      ah,0Ah
int      10h
ENDEM

;06h and 07h
Scroll  MACRO      distanța,atrib,suscol,susrînd,joscol,josrînd
                ;fac defilare de pagină între anumite limite
IFDEF    suscol
mov      cl,suscol
ENDIF
IFDEF    susrînd
mov      ch,susrînd
ENDIF
IFDEF    suscol
mov      dl,suscol
ENDIF
IFDEF    josrînd
mov      dh,josrînd
ENDIF
IFDEF    atrib
mov      bh,atrib
ELSE
mov      bh,07h
ENDIF
IF      distanța LE 0
mov      ax,0600h + (-(distanța) AND 0FFh)
ELSE
mov      ax,0700h + (distanța AND 0FFh)
ENDIF
int      10h
ENDEM

;08h, 06h and 02h
Șterg  MACRO      ;șterg ecranul
        GetChAtr
mov      bl,bh
mov      bh,ah
sub      cx,cx
mov      dx,184Fh
mov      ax,0600h
int      10h
mov      bh,bl
sub      dx,dx
mov      ah,02h

```


Exploatarea microsystemelor prin limbajul PROLOG și ASSEMBLER

```
int      10h
ENDEM

ENDIF
.CREF
.LIST
```

Înteruperile dedicate serviciilor grafice, tastaturii, gestiunii discurilor vor fi pe larg discutate în secțiuni aferente special acestor unități.

În continuare, să exemplificăm însă câteva utilizări ale celorlalte servicii ROM-BIOS.

1) Tipul microcalculatorului

Există întotdeauna probleme în legătură cu tipul de microcalculator folosit, în sensul microprocesorului implementat. Pentru ca un program să fie portabil pe diverse tipuri de microcalculatoare este necesară cunoașterea mașinii pe care el este încărcat și aleasă partea din programul sursă compatibilă cu aceasta. Informația despre tipul calculatorului se află la adresa FFFFE (hexa) în ROM-BIOS, valoarea octetului de la această adresă delimitând tipul mașinii după cum urează:

Valoare octet (hexa)	Calculator.
FF	PC (procesor 8086, 80186)
FE	XT (procesor 8088)
FD	PC junior
FC	AT (procesor 80286)

Se face mențiunea că putem scrie orice întreg sub formă hexazecimală cu convenția ca numărul hexazecimal să fie precedat de semnul \$ (dolar). Astfel, numărul 16 zecimal se scrie \$10 în hexazecimal etc.

Apelul:

membyte (\$F000, \$FFE, *Continut*)

întoarce în *Continut* valoarea întreagă aflată la adresa FFFE. Astfel, testând valoarea din *Continut* cunoaștem tipul calculatorului.

2) Tipul de adaptare video

Este întotdeauna necesar să se cunoască pentru un program, în special pentru rutine grafice, dacă adaptorul video al calculatorului este o placă monocrom, placă grafică CGA sau placă EGA. Octetul care conține informația despre tipul adaptorului se află în zona de date BIOS, la adresa 0040:0010. Astfel, dacă biții 5-4 sînt 11, placa monocromă este activă, sînt 10 - pentru placa color cu rezoluție 80X25, 01 pentru placa color cu rezoluție 40X25 și 00 pentru placă EGA.

În cazul unei plăci EGA mai este necesar a se cunoaște cîtă memorie video este aferentă și tipul monitorului atașat. Pentru tipul display-ului se testează bitul 1 la adresa 0040:0087 care, dacă are valoarea 1, avem un display monocrom, altfel, dacă are valoarea 0, avem de-a face cu un display color.

Astfel, apelurile următoare verifică tipul monitorului video, modul video și tipul adaptorului video existent.

Apelul:

Exploatarea microsystemelor prin limbajul PROLOG și ASSEMBLER

membyte (\$40, \$87, *Continut*)

testează valoarea octetului la adresa 0040:0087.

Dacă *Continut* = 0 atunci nu avem placă EGA iar dacă scriem:
true (*Continut* and 8 = 0) (testez bitul 1)

realizarea acestui predicat ne indică existența plăcii EGA.

Elaborarea unor rutine care să simuleze acțiunea predicatului membyte este indicată în limbajul BASIC precum și în cel de asamblare:

- simulare BASIC -

10 REM "Tipul adaptorului video"

20 DEF SEG = &H40 ' segment la începutul zonei BIOS

30 CONȚINUT = PEEK (&H87) ' citesc valoarea la deplasamentul
' 87 hexa în segmentul '

40 IF CONȚINUT = 0 THEN 600

50 IF ((CONȚINUT AND 8) = 0) THEN 800

.

.

600 PRINT "NU AVEM PLACA EGA"

.

800 PRINT "AVEM PLACA EGA" . . .

- simulare limbaj de asamblare -

; Tipul adaptorului video:

MOV AX,40H ; registrul ES va conține adresa

MOV ES,AX ; de început a segmentului BIOS

MOV AL,ES:[87H] ; citest în AL octetul la

; adresa 0040:0087

CMP AL,0 ; AL conține valoarea

JE NU-placă-EGA ; lui *Continut*

TEST AL,00001000B ; efectuez AL AND 8

JZ Da-placă-EGA. . .

3) Determinarea numărului și tipului de periferice din sistem

La pornirea sistemului, BIOS-ul cercetează echipamentele conectate și crează un registru de stare care raportează cele găsite. Acest registru este un cuvânt pe 16 biți aflat la adresa 0040:0010 (40H este adresa de început a zonei de date a BIOS-ului). Convențiile de semnalizare a diverselor periferice în acest registru sînt standardizate, în general, pentru microsystemele profesionale:

Poziție bit	Semnificație
bit 0	valoare 1, există unități de diskete, altfel nu
1	valoare 1, există coprocesor matematic
2-3	valoare 11, memoria RAM ocupă 64 k
4-5	informații adaptor video (valoare 11 - 80X25 monocrom, valoare 10 - 80X25 placă color)
6-7	numărul unităților de diskete (valoare 0 - o unitate)
8	numai pentru PC junior (valoare 0 - există chip DMA)
9-11	numărul adaptorilor seriali
12	valoare 1 - există port pentru joystick
13	numai pentru PC junior (valoare 1 - există imprimantă)

Exploatarea microsystemelor prin limbajul PROLOG și ASSEMBLER

Poziție bit	Semnificație
	serială)
14-15	numărul adaptorilor paraleli

Să observăm că investigarea unităților de discuri este dată atât prin investigarea biților 6-7 cât și a bitului 0 care, dacă are valoare 0, înseamnă că nu există unități de discuri.

Apelul:

`memword (S40, S10, Continut)`

întoarce în cuvântul *Continut* configurația registrului de stare BIOS.

Apelul:

`Continut1 = Continut mod 2`

ne dă informații asupra bitului 0. Astfel, dacă valoarea lui *Continut1* este 0, nu există unități de discuri.

Același efect îl are și întreruperea BIOS-INT11H, care întoarce registrul de stare în AX. Se generează doar întreruperea, fără nici o valoare a registrelor de intrare.

Rutina următoare cercetează existența unităților de discuri din sistem folosind această întrerupere, 11H:

`verif-disc:`

`bios ($11,-,reg(AX,-,-,-,-,-,-,-)),`

`AX AND 65 = 0,!write('Nu există unități discuri')`

`verif-disc:`

`bios($11,-,reg(AX,-,-,-,-,-,-,-)),`

`write('Avem unități de discuri').`

Simularea rutinei de mai sus în limbaj de asamblare ar da o imagine clară a acțiunii predicatului bios ():

`; Test unități-discuri:`

`INT 11H ; generez întrerupere 11H`

`TEST AL,0 ; obțin în AX registrul stare`

`JZ Nu-discuri. . . ; dacă AL = 0 nu sînt discuri`

Pentru utilizatorii sistemului de operare MS-DOS există încă o gamă extrem de variată de funcții și întreruperi pentru controlul sistemului.

Rutinele pe care MS-DOS le folosește în stăpînirea operațiilor de sistem și a resurselor pot fi apelate de către orice program utilizator, folosirea lîr mărind independența aplicațiilor de schimbări și convenții în versiuni noi atât ale sistemului de operare cât și ale compilatoarelor. Categoriile mari de apeluri sistem ale MS-DOS-ului se pot delimita în: operații de intrare-ieșire, managementul memoriei, gestiunea fișierelor, apeluri pentru rețele, diverse funcții sistem.

Aplicațiile invocă serviciilor MS-DOS cu ajutorul întreruperilor software, ale căror limite se încadrează între 20H-27H.

Întreruperea 21H, cea mai utilizată, satisface cererile de funcții sistem.

Fiecare întrerupere sau funcție folosește valori de intrare sau rezultate ale ieșirii în registrele de sistem.

Vom trata, în continuare, întreaga gamă de apeluri la funcții sistem.

Exploatarea microsystemelor prin limbajul PROLOG și ASSEMBLER

oferind cititorului posibilitatea de a stăpîni integral resursele microcalculatorului cu care lucrează și, în final, de a fi convins de flexibilitatea și utilitatea limbajului Prolog nu numai ca un produs tipic al programării logice ci și ca un instrument fiabil în programarea convențională și de sistem.

Așa cum spuneam, sistemul DOS rezervă întreruperile 20H-3FH pentru sine. Asta înseamnă că locațiile de memorie cu adresă absolută între 80H-FFH sînt rezervate de DOS. Pe scurt, întreruperile DOS sînt:

Întreruperea 20H (Terminare program)

Se generează această întrerupere pentru ieșirea dintr-un program. Vectorul întrerupere determină sistemul DOS să restaureze adresa de terminare, adresa CTRL-Break și adresa de ieșire a erorii critice la valorile pe care le aveau la intrarea în lucru a programului. Buffer-ele fișierelor sînt salvate și toate handlerele sînt închise. Înaintea apelului întreruperii 20H, programul trebuie să se asigure că registrul CS conține adresa de segment a PSP-ului (prefixul segment de program).

Întreruperea 21H (Apeluri de funcții sistem)

Este descrisă pe larg prin funcțiile sale în paginile următoare.

Întreruperea 22H (Adresa de terminare)

Controlul se transferă adresei locației acestei întreruperi cînd programul se termină. Această adresă este copiată în PSP-ul de program la momentul creării segmentului program. Este indicat să nu se apeleze direct această întrerupere, funcția EXEC a lui INT 21H făcînd în locul nostru apelul.

Întreruperea 23H (Adresa de ieșire)

În timpul desfășurării unui program, întreruperea 23H este executată la apelul utilizator <Ctrl-Break>

Întreruperea 24H (Vectorul handlerului de eroare critică)

Cînd se întîmplă o eroare fatală sub DOS, controlul este transferat de întreruperea 24H. Registrul AH va avea bitul 7 pe 0 dacă avem o eroare, altfel BP:SI va conține adresa blocului de control al headerului de unitate (device) de unde se pot citi diverse informații după valorile în registrul DI, astfel;

Valoare	Numele erorii
0	Încercare de scriere pe disc protejat la scriere
1	Unitate necunoscută
2	Unitate neoperațională
3	Comandă necunoscută
4	Erori de date (CRC)
5	Cerere greșită
6	Eroare de căutare
7	Mediu necunoscut

Exploatarea microsystemelor prin limbajul PROLOG și ASSEMBLER

Valoare în DI	Numele erorii
8	Sector nedetectat
9	Imprimanta fără hîrtie
A	Greșeală la scriere
B	Greșeală la citire
C	Eșec general

Stiva utilizator este folosită și are componența de sus în jos:

IP conține regiștrii DOS din generarea lui (INT 24H)

CS

Flags

AX Regiștrii utilizator la momentul cererii

BX INT 21H primare

CX

DX

SI

DI

BP

DS

ES

IP Din întreruperea 21H originală

CS de la utilizatorul DOS

Flags

Regiștrii sînt notați astfel că, dacă un IRET este executat, DOS răspunde cu valori în AL, astfel:

(AL) = 0 - ignoră eroarea;

(AL) = 1 - repetă operația;

(AL) = 2 - termină program cu INT 23H;

(AL) = 3 - anulează cererea de sistem activă în acel moment.

Întoarcerea controlului din eroarea fatală la o rutină utilizator de tratare a sa presupune:

- Înaintea execuției lui INT 24H:

a) Aplicația utilizator să redirecteze vectorul INT 24H către un vector ce punctează spre o rutină utilizator de diagnosticare și remediu;

- În timpul lui INT 24H:

b) Cînd controlul este predat rutinei utilizator registrul *Flags* este salvat în stivă și se execută un CALL FAR la vectorul original INT 24H salvat în pasul a);

c) DOS răspunde utilizatorului cu:

> Abort, Retry, Ignore (?)

după care, la răspunsul utilizator (prin A, R sau I), întoarce controlul rutinei de eroare utilizator după instrucțiunea ce urmează lui CALL FAR;

d) Acum rutina de diagnoză a erorii creată de utilizator are control deplin. Întoarcerea din ea la aplicația originală unde s-a produs întreruperea se face printr-o instrucțiune IRET (întoarcere din

Exploatarea microsystemelor prin limbajul PROLOG și ASSEMBLER

întrerupere) în rutina de diagnoză. Altfel, această rutină distruge regiștrii IP, CS, Flags din stivă.

Întreruperea 25H (Citire absolută disc)

Controlul este transferat direct unității de disc (vezi macro-ul de utilizare). Numărul specificat de sectoare este transferat între unitatea dorită și adresa de transfer.

Dacă transferul a fost efectuat cu succes, CX este 0, altfel 1 și AX conține codul erorii.

Întreruperea 26H (Scriere absolută disc)

Are același efect ca INT 25H, cu mențiunea că se scrie pe disc.

Întrerupere 27H (Terminat și rămas rezident)

Vectorul acestei întreruperi este folosit de programe care rămân rezidente când COMMAND.COM reprimește controlul (vezi funcția 31H). După propria sa inițializare, programul pune în DX ultima adresă a sa plus una relativă la valorile lui DS ori ES (deplasamentul la care alte programe pot fi încărcate), executînd apoi INT 27H. DOS consideră acest program ca o extensie a sa și astfel nu este acoperit cînd alte programe sînt în execuție. Există următoarele restricții:

- a) Întreruperea nu este folosită de programe .EXE;
- b) Întreruperea restaurează vectorii 22H, 23H și 24H la fel ca și întreruperea 20H;
- c) Doar programe de pînă la 64 k pot fi făcute rezidente;
- d) Programele rezidente nu închid fișierele.

Întreruperile 28H - 2EH (Rezervate pentru DOS)

Întreruperea 2FH (Întreruperea multiplexoare)

Stabilește o interfață între 2 procese.

Întreruperile 30H - 3FH (Rezervate pentru DOS)

Considerăm utile macro-urile în limbaj de asamblare de utilizare a întreruperilor 25H-27H, care permit apelul lor în secvențe asamblor de sine stătătoare sau încorporate în Prolog.

Întreruperea 25H -

```
ABS_DISK_READ_macro_disk, bufer, num_sector, primul_sector /*citire absolută  
disc*/
```

```
mov al, disk  
mov bx, offset bufer  
mov cx, num_sector  
mov dx, primul_sector  
int 25H  
popf  
endem
```

Întreruperea 26H

```
ABS_DISK_WRITE macro disk, bufer, num_sector, primul_sector /*scriere  
absolută disc*/
```


Exploatarea microsystemelor prin limbajul PROLOG și ASSEMBLER

```
mov  al,disk
mov  bx,offset bufer
mov  cx,num_sector
mov  dx,primul_sector
int  26H
popf
endem
```

Întreruperea 27H

```
RAMÎN_RESIDENT macro ultima_instrucțiune
mov  dx,offset ultima_instrucțiune
inc  dx
int  27H
endem
```

Există o succesiune de operații fixe care permit apelul la serviciile unei funcții sistem:

- a) Mutarea datelor în registre;
- b) Mutarea numărului funcției dorite în AH;
- c) Mutarea codului de acțiune în cadrul funcției AL;
- d) Apelul la întreruperea 21H.

Cînd MS-DOS preia controlul după cererea funcției, el folosește o stivă internă și păstrează registrele nefolosite pentru informațiile returnate de funcția utilizatorului (cu excepția lui AX), astfel că stiva programului utilizator apelat trebuie să fie suficient de mare pentru întreruperea de sistem (cel puțin 128 octeți). Iată, în continuare, lista completă a funcțiilor sistem ce pot fi exploatare cu întreruperea 21H:

Funcție	Descriere
00H	Terminare program
01H	Citește tastatură cu ecou
02H	Afișează caracter
03H	Intrare auxiliară
04H	Ieșire auxiliară
05H	Tipărește caracter
06H	Intrări/ieșiri consolă
07H	Intrare consolă
08H	Citește tastatură
09H	Tipărește șir caractere
0AH	Intrare tastatură cu bufer
0BH	Verificare stare tastatură
0CH	Varsă bufer, citește tastatură
0DH	Reșetează disk
0EH	Selectează disk
0FH	Deschide fișier
10H	Închide fișier
11H	Caută prima intrare (în director)
12H	Caută a doua intrare (în director)
13H	Șterge fișier
14H	Citire secvențială
15H	Scriere secvențială

Exploatarea microsystemelor prin limbajul PROLOG și ASSEMBLER

Funcție	Descriere
16H	Crează fișier
17H	Redenumește fișier
18H	REZERVAT
19H	Citește discul curent
1AH	Setează adresa de transfer disk
1BH	Citește datele din drive-ul curent
1CH	Citește date din drive
1DH-20H	REZERVAT
21H	Citire aleatoare (random)
22H	Scriere aleatoare (random)
23H	Determină mărimea unui fișier
24H	Setează înregistrare relativă
25H	Setează vector întrerupere
26H	Crează noul PSP
27H	Citire nedefinită bloc
28H	Scriere nedefinită bloc
29H	Analiză nume fișier
2AH	Citește data
2BH	Setează data
2CH	Citește ora
2DH	Setează ora
2EH	Setează/resetează bitul verificare
2FH	Citește adresa de transfer disk
30H	Citește versiunea MS-DOS
31H	Continuă proces
32H	REZERVAT
33H	Verificare Control-C (CTRL-C)
34H	REZERVAT
35H	Citește vector întrerupere
36H	Citește spațiul liber disc
37H	REZERVAT
38H	Citește/setează țara
39H	Crează director
3AH	Șterge director
3BH	Schimbă director curent
3CH	Crează driver
3DH	Deschide driver
3EH	Închide driver
3FH	Citește driver
40H	Scrie driver
41H	Șterge fișiere director
42H	Mută pointer fișier
43H	Citește/setează attribute fișier
4400H,4401H	IOCTL date
4402H,4403H	IOCTL caracter
4404H,4405H	IOCTL bloc
4406H,4406H	IOCTL stare
4408H	IOCTL schimbabil
4409H	IOCTL redirectat bloc

Exploatarea microsystemelor prin limbajul PROLOG și ASSEMBLER

Funcție	Descriere
440AH	IOCTL redirectat driver
440BH	IOCTL reluare
440CH	Generic IOCTL (pentru drivere)
440DH	Generic IOCTL (pentru unități periferice)
440EH	Citește hartă drive
440FH	Setează hartă drive
45H	Duplică driver fișier
46H	Forțează duplicare fișier
47H	Citește director curent
48H	Alocă memorie
49H	Memorie liberă alocată
4AH	Setează bloc
4B00H	Încarcă și execută program
4B03H	Încarcă overlay
4CH	Sfârșit proces
4DH	Citește cod retur proces secundar
4EH	Găsește primul fișier
4HF	Găsește al doilea fișier
50H-53H	REZERVAT
54H	Citește starea de verificare
55H	REZERVAT
56H	Schimbă punct intrare în director
57H	Citește/setează ora și data în fișier
58H	Setează strategia de alocare
59H	Citește eroare
5AH	Crează fișier temporar
5BH	Crează fișier nou
5C00H	Blocaj
5C01H	Deblocaj
5DH	REZERVAT
5E00H	Citește nume mașină
5E02H	Setează printer
5F02H	Citește asigurarea listei de intrări
5F03H	Crează asigurarea listei de intrări
5F04H	Anulează asigurarea listei de intrări
60H-61H	REZERVAT
62H	Citește PSP
63H-7FH	REZERVAT

Prezentăm, în continuare, modul de utilizare al fiecărei funcții și apelul cu ajutorul limbajului PROLOG. Funcțiile care au scris rutinele Prolog vor fi explicate în detaliu pe parcursul capitolelor ce urmează.

Terminare Program (Funcția 00H)

Apel

AH = 00H

CS

Adresa segment a PSP

Întoarcere rezultat

Exploatarea microsystemelor prin limbajul PROLOG și ASSEMBLER

Nici o valoare

Registrul CS conține adresa segmentului PSP înainte chemării acestei funcții.

Toate buferele de fișier sînt scrise pe disc.

Utilizare Prolog:

```
terminare - program: - AX = $0000,  
bios ($21, - reg(AX,-,-,-,-,-,-,-,-));
```



Inițiere în conceptele comunicației de date (I)

Această epocă pe care o trăim cu toții, a tehnologiilor și informaticii, este puternic dependentă de calculatoare. "Jargonul" folosit în calculatoare poate, și de fapt creează adeseori confuzii. Multe din noțiuni au definiții neclare și unele au mai multe înțelesuri. Noii sosiți în lumea comunicațiilor de date sînt adesea, pe bună dreptate, depășiți de volumul informațiilor disponibile. Chiar și utilizatorii experimentați nu sînt întotdeauna siguri asupra semnificației unor termeni sau concepte.

Termenii și conceptele vor fi prezentate succint. În cele ce urmează sperăm să se pună la dispoziție elementele primare necesare pentru a înțelege și a evalua necesitățile de comunicații de date pe care le veți avea.

1. Datele manipulate de calculatoare

Calculatoarele sînt sisteme numerice care lucrează în sistemul de numerație binar (baza 2). Acest sistem este cel mai eficient-tehnologic sistem care poate fi folosit la tehnica de calcul.

Deci, datele din lumea exterioară (numere, texte, imagini etc.), pentru a putea fi prelucrate de către un calculator, trebuie să fie transformate în prealabil în echivalentele lor binare. Din fericire, acest proces greoi nu se efectuează de către utilizatorul calculatorului ci de către instrumente care realizează translația cu rapiditate și precizie.

Translația datelor de tip text în forma lor binară are, de obicei, la bază codul ASCII (American Standard Code for Information Interchange). Setul de caractere ASCII atribuie o configurație distinctă de 8 cifre binare (biți) pentru fiecare literă din alfabetul englez, pentru fiecare cifră zecimală (0 la 9), pentru fiecare semn de punctuație uzual (+, - etc.) precum și pentru o varietate de caractere "speciale" (de control). Acest grup de 8 biți este denumit în mod uzual byte (sau octet). Valorile din codul ASCII folosesc de fapt numai 7 din cei 8 biți ai unui byte - primul (cel mai semnificativ) bit are întotdeauna valoarea 0.

Astfel, de exemplu, litera A se reprezintă în cod ASCII prin byte-ul binar 01000001 - adică în zecimal 65, hexazecimal 41 și octal 101.

Valorile de cod ASCII de la 32 la 127 (96 caractere) sînt folosite pentru reprezentarea caracterelor grafice (imprimabile). Valorile de la 0 la 31 sînt caracterele "speciale" (neimprimabile) sau codurile de control. Aceste caractere de control pot fi generate la majoritatea tastaturilor de calculatoare prin apăsarea tastei control (<ctrl>) și a unei taste alfabetice.

Reprezentarea uzuală a tastei <ctrl> este caracterul circumflex (^). De exemplu, codul de control cu valoarea 1 (soh - start of heading) se poate genera cu combinația ^A (circumflex și A).

De regulă datele cu care lucrează calculatoarele sînt "grupate" în fișiere care pot fi, principial, de două tipuri: programe și date pentru programe.

Orice fișier care conține numai caractere ASCII este domeniul fișier de tip text, iar fișierele care conțin și alte date decît caractere ASCII sînt denumite fișiere binare. În mod normal fișierele de programe sînt fișiere binare, iar fișierele de date pot fi atît de tip text cît și binar.

Inițiere în conceptele comunicației de date (I)

Fișierele standard ASCII conțin numai caracterele ASCII imprimabile (codurile 32 la 127) și patru coduri de control - din setul ASCII: <return>, LF (line feed), FF (form feed) și HT (tabulare orizontală). Acest tip de fișiere nu trebuie confundat cu fișierele rezultate din prelucrarea de cuvinte (word processing). Programele de procesare a cuvintelor crează, de regulă, fișiere care conțin în principal text ASCII, dar și coduri binare de 8 biți și o sumedenie de caractere de control ASCII. Aceste fișiere trebuie tratate ca fișiere de tip binar.

Majoritatea terminalelor (tastaturilor de calculatoare) și imprimantelor actuale pot genera și imprima atât litere mari cât și mici (bineînțeles, în afara semnelor de punctuație). Dar, în ceea ce privește software-ul, unele programe sau limbaje de comandă a unor sisteme de operare "recunosc" drept caractere distincte literele mari și cele mici, pe când altele nu.

Programele sau sistemele de operare care tratează în mod distinct caracterele mari și mici se numesc sensibile la tipul caracterelor (case-sensitive). De exemplu sistemul UNIX este sensibil la tip (spre deosebire de MS-DOS). Procesoarele de cuvinte sînt și ele, ca regulă generală, sensibile la tipul caracterelor.

Deoarece, de obicei, comunicațiile de date implică tipuri diferite de calculatoare (deci și de sisteme de operare și programare), problema incompatibilității referitoare la sensibilitatea de tip de caractere este reală.

2. Termeni fundamentali utilizați în comunicația de date

Transmisia paralelă de date constă în disponibilitatea simultană a 8 linii (căi) de date (pe care se vehiculează semnele electrice corespunzătoare biților 1 și 0) pe care se poate transmite - între două calculatoare de exemplu - nu legate complet la un moment dat (figura 1).

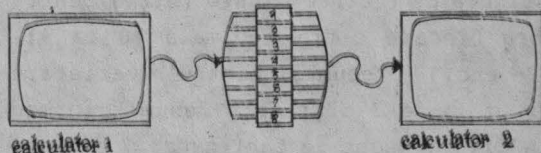


fig. 1

Avantajele evidente ale transmisiei paralele (viteză și acuratețe) sînt contrabalansate pentru distanțe mai mari (zeci de metri) de costul considerabil al căilor fizice de date. Din aceste motive transmisia paralelă se folosește pentru comunicații "locale" (imprimante, legături de mare viteză între calculatoare).

Transmisia serială de date folosește pentru comunicație o singură linie (cale) pe care datele circulă bit cu bit (figura 2).

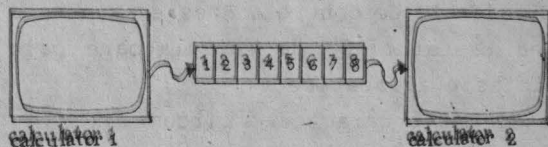


fig. 2

Un avantaj important al acestui tip de transmisie îl constituie costul relativ redus (un singur canal de comunicație). Dar, evident, viteza de transmisie (la același debit al canalului) este mai mică. Un alt avantaj important al transmisiei seriiale îl constituie posibilitatea folosirii circuitelor telefonice obișnuite.

Conexiuni punct-la-punct. Numai două calculatoare sînt implicate într-o sesiune de transfer de date și numai acestea pot folosi canalul (linia telefonică, de exemplu) care le conectează.

Conexiuni multi-punct. Mai multe calculatoare se conectează pe o aceeași linie utilizînd-o pe rînd fiecare dintre ele.

Inițiere în conceptele comunicației de date (I)

Calculatorul stăpîn (master) controlează alocarea canalului (liniei) de comunicații pentru fiecare din calculatoarele subordonate (slave) care nu pot utiliza linia decât la inițiativa calculatorului master.

Viteza de transmisie. Terminologia referitoare la viteza de transmisie este uneori greșit înțeleasă. Viteza de transmisie de date pentru transmisii seriale este exprimată, în mod normal, în biți/secundă (bps). Uneori se mai folosește și unitatea de măsură baud - care însă nu este în mod necesar aceeași cu biți/secundă (bps).

Un baud se definește ca numărul de secunde electrice care se transmit pe un canal de comunicație în interval de o secundă.

Numai în cazul în care se transmite (prin metode de codificare aleasă!) un singur bit/semnal electric, viteza exprimată în baud este egală cu bps.

Dar, în mod uzual, se transmite mai mult de un bit/semnal electric. De exemplu: pentru un canal de 2400 baud - viteza este de 2400 bps (pentru 1 bit/semnal), 4800 bps (2 biți/semnal), 9600 bps (4 biți/semnal).

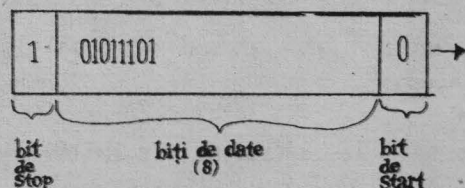
Vitezele uzuale folosite pe canalele seriale pentru comunicații de date sînt: 1200, 2400, 4800, 9600 și 19200 bps.

Ambele calculatoare trebuie să folosească aceeași viteză de transmisie.

Transmisii sincrone. Caracterele (bytes) de date sînt trimise pe linia de comunicații, unul după altul fără intervale de "pauză".

Transmisii asincrone. În aceste transmisii caracterele (bytes) sînt trimise unul după altul, dar pot apare ocazional intervale de "pauză" între caracterele transmise. Acest tip de comunicație este mai simplu și mai ieftin decât cel sincron și este folosit în mod deosebit la transmisii de date în care sînt implicate micro și minicalculatoare.

Datorită pauzelor care pot apare între caractere, acestea trebuie delimitate între ele. Delimitarea se face prin adăugarea unui bit la începutul caracterului (bit de start - are întotdeauna valoarea 0) și un bit (uneori 2!) la sfîrșitul caracterului (bit de stop - are întotdeauna valoarea 1) (figura 3).



Acești "extra" biți sînt "generați" la transmisie și eliminați de către hardware la recepția completă a unui caracter.

Imaginea unui caracter transmis poate fi redusă la 7 biți - în cazul în care se transmit numai date de tip ASCII.

Paritate. În transmisiile de date asincrone se poate folosi un bit suplimentar (de paritate) pentru detectarea unor erori simple de transmisie. Acest bit se plasează automat de către hardware înaintea bitului de stop. Se poate folosi paritate "pară" (număr total de biți de date 1 - par) sau "impară":

Opțiunile de "paritate" sînt deci: fără paritate (nu se generează bitul de paritate), bit de paritate 1 (mark), bit de paritate 0 (space), paritate pară, paritate impară.

Opțiunile de lungime de caracter: 7 sau 8 biți de date. Aceste opțiuni pot fi regăsite la majoritatea sistemelor de comunicație asincrone din micro/minicalculatoare. Formatul cel mai "acoperitor" în transmisiile asincrone este: 1 bit de date start, 8 biți de date, fără paritate, 1 bit de stop.

Semnalul de întrerupere (break). În transmisiile asincrone, atunci cînd se transmit date, cel puțin un bit are valoarea 1 (bitul de stop!) - chiar și pentru caracterul "null".

Inițiere în conceptele comunicației de date (I)

Deci un șir de date 0 constituie un "semnal" care este generat și detectat de hardware și poate fi utilizat pentru suspendarea sau întreruperea prelucrării normale.

Transmisii semi-duplex. Datele pot fi transmise în ambele direcții, dar nu simultan. În protocoalele semi-duplex (de exemplu: BSC, Kermit, XMODEM etc.) calculatorul care emite trebuie să aștepte confirmarea fiecărui bloc de date de la calculatorul corespondent, înainte de a transmite următorul bloc (figura 4).

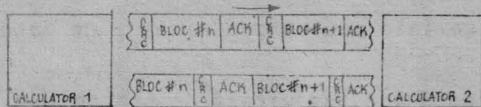
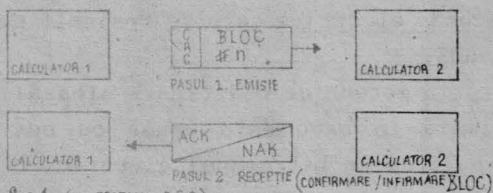


fig. 5 (EX. PROTOCOL SDLC)

Controlul fluxului de date se poate face de regulă prin suspendarea și reluarea traficului de date în canalul de comunicație. Această acțiune este similară cu acțiunea fastelor <ctrl>S (suspendare), și <ctrl>Q (reluare) la un terminal conectat la un minicalculator.

Controlul fluxului prin software. Caracterele ASCII de control flux (DC3 și DC1 corespunzătoare la <ctrl>S și <ctrl>Q sau XOFF/NON), sînt transmise (inserate în fluxul de date) de regulă de către software-ul de control comunicație de date. Controlul fluxului previne pierderea unor informații în cazul în care receptorul nu are capacitate suficientă de prelucrare (sau viteza de comunicație este prea mare!).

Dar aceste caractere ASCII de control flux pot fi generate/detectate și de unele dispozitive hardware (multiplexoare), acțiune care poate interfera cu transferul unor date binare, denaturîndu-se.

Controlul fluxului prin hardware se poate realiza folosind semnalele electrice RTS (request to send)/CTS (clear to send).

Acest control este mai rar utilizat.

Transparența datelor. Circuitele de comunicație de date pot fi alcătuite din mai multe componente hardware/software - modemuri, interfețe seriale, drivere din sisteme de operare etc. Aceste componente pot altera fluxul de date transmis prin circuitul de comunicație datorită: lungimii datelor (7/8 biți), caracterelor de control flux, prelucrarea unor caractere ASCII.

Circuitul care lasă să treacă nealterate prin toate componentele sale toate caracterele (de 8 biți lungime!) vehiculate, se numește circuit de comunicație de date cu transparență totală.

3. Interfețe seriale

Interfața serială este componenta hardware a unui calculator care realizează transmisia/recepția pe canalul de comunicație bit cu bit precum și dezasamblarea/asamblarea biților în caractere (bytes - de 8 biți).

În interfețele seriale asincrone caracterul din calculator se dezassemblează (la transmisie) în biții de date care se transmit pe linie serial împreună cu biții de start/stop precum și cu eventualul bit de paritate (generați în interfață).

Inițiere în conceptele comunicației de date (I)

Hardware pentru interfețe seriale. Conectarea interfețelor seriale la canalul de comunicație (de regulă prin modem!) se face prin conectoare standardizate tată/mamă de 25 contacte sau, uneori (în cazul transmisiilor asincrone), conectoare standardizate tată/mamă de 9 contacte.

Calculatoarele au în componență diverse tipuri de interfețe seriale: porturi seriale încorporate (circuite + conector), plăci care se pot adăuga, multiplexoare sau chiar procesoare specializate (procesoare frontale).

Porturile seriale încorporate se folosesc în special la microcalculatoare. Multiplexoarele sînt utilizate de regulă la minicalculatoare și microcalculatoare multiutilizator.

Procesoarele frontale se folosesc ca intermediar între mini/microcalculatoare și calculatoare medii/mari (mainframe). Sînt cuplate la mainframe prin legături (de regulă paralele) de mare viteză care "concentrează" un număr important de canale de mică viteză la care sînt conectate mini/microcalculatoare.

Interfețele pot fi configurate (formatul datelor și viteza de transmisie) atît hardware (prin comutatoare sau strapuri) cît și software (de regulă prin comenzi pentru sistemul de operare sau programul de comunicație).

Standarde electrice pentru interfețele seriale. Se folosește în mod uzual standardul electric cu nivele de tensiune (EIA RS-232) - Electronica Industry Association Revised Standard 232).

Standardul definește atît nivelele de tensiune cît și alocarea contactelor pentru conectoarele cu 25 sau 9 contacte.

Semnalele RS-232 folosite în mod uzual de interfețele seriale în transmisiile asincrone sînt:

- pentru circulația de date:

Transmit Data	TXD	contact numărul 2
Receive Data	RXD	contact numărul 3
Signal ground	SG	contact numărul 7

- pentru controlul modemului:

Carrier Detect	DCD	contact numărul 8
(Detectie purtătoare)		
Data Terminal Ready DTR		contact numărul 20.

Standardul RS-232 definește două tipuri de configurații de interfețe numite DTE (Data Terminal Equipment) și DCE (Data Communication Equipment). În mod normal prin DTE trebuie înțeleasă interfața serială a terminalelor iar prin DCE trebuie înțeleasă interfața modemului. Dar, la calculatoare, interfața serială poate fi atît de tip DTE cît și de tip DCE.

Standardul RS-232 recomandă conectoare de tip mamă pentru interfețele de tip DCE și de tip tată pentru interfețele de tip DTE.

Semnalele de interfețe sînt denumite cu referință la DTE. Fiecare poziție din interfața DTE corespunde unei anumite funcții care este complementară cu funcția de pe poziția corespunzătoare din interfața DCE.

De exemplu, poziția (contactul) 2 din interfața DTE corespunde funcției de ieșire (transmisie) date (TXD), în timp ce același contact (2) din interfața DCE, deși este denumit la fel (TXD), corespunde funcției de intrare (recepție) date dinspre DTE!

Această observație trebuie avută în vedere la conectarea a două interfețe de același tip, cînd trebuie cuplate contactele cu aceeași funcție - și nu cu același număr.

Inițiere în conceptele comunicației de date (I)

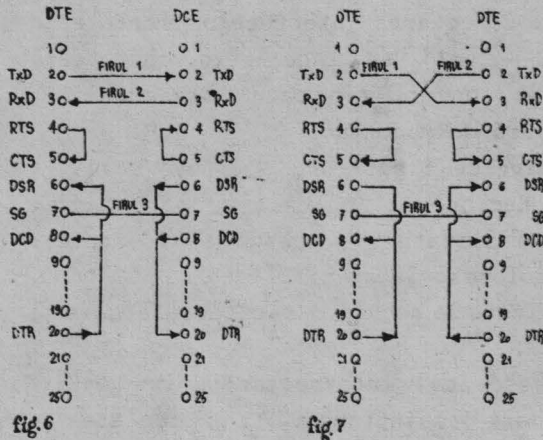


fig. 6

fig. 7

Un exemplu de conectare (cablu) DTE/DCE pentru aplicații în care nu se folosește controlul de modem și nici controlul hardware al fluxului este arătat în figura 6.

În figura 7 este prezentată schema de conectare (cablu) pentru DTE/DTE (sau DCE/DCE!), în aceleași condiții din exemplul precedent.

Conexiunile (strapurile) 4-5, 6-8-20 care trebuie realizate la ambele conectori ale cablurilor prezentate sînt

necesare pentru "simularea" electrică a prezenței modemului.

4. Modemuri

Modemurile sînt dispozitive care convertesc o intrare serială de date în semnal electric analogic care se vehiculează pe linia de comunicație (de exemplu circuit telefonic) și invers. Aceste acțiuni de comunicație sînt denumite modulare și demodulare (MODEM) (figura 8).



fig. 8

Modemuri semi-duplex și duplex.

De regulă modemurile semi-duplex sînt folosite pentru transmisii sincrone (protocolul BSC, de exemplu, este un protocol semi-duplex).

Purtătoarea (carrier) este semnalul electric analogic din linia telefonică care este modulat pentru a transmite datele numerice (binare, seriale). Modemurile semi-duplex au o singură frecvență pentru purtătoare, iar cele duplex dispun de cîte o frecvență purtătoare pentru fiecare direcție.

Modemul apelat își plasează primul purtătoare (purtrătoare de răspuns) în canalul de comunicație. După detectarea purtătoarei de răspuns de către modemul apelator, acesta își plasează propria purtătoare (purtătoarea originală) în canal. Cînd ambele purtătoare sînt prezente în canalul de comunicație (pe cele două direcții) se consideră că s-a realizat conectarea și se poate începe comunicația de date propriu-zisă.

Viteza de transmisie și standarde pentru modemuri asincrone. Vitezele uzuale sînt 1200, 2400, 4800, 9600 bps. Pentru fiecare din aceste viteze de transmisie există standarde care se referă la caracteristicile tehnice ale interfeței cu linia (analogică), inclusiv frecvențele de purtătoare.

Standardul 212 definește modemuri care lucrează la viteza de 1200 bps.

Standardul 224 definește modemuri care lucrează la viteza de 2400 bps cu posibilitate de repliere la 1200 bps (CCITT definește aceste modemuri cu standardul V.22 bis).

Modemuri rapide pentru viteze de 9600 (standardul CCITT V.32) sau 19200 bps. Standardul V.32 definește modemuri (interfețe de linie) duplex sincron/asincron cu circuite simple (două fire) comutate sau particulare, viteza de 9600 bps.

EDITOARE ȘI FONTURI (I)

ION PARASCHIV și TIBERIU SPIRCU

Introducere

Matematicienii (și nu numai ei) obișnuiesc să spună că folosesc pentru a exprima în scris noțiunile, simboluri speciale. Dar ce este oare un simbol ?

Să ne informăm într-o ediție recentă din "The Oxford Illustrated Dictionary"; găsim:

symbol = written character conventionally standing for some object.

Să căutăm deci semnificația cuvântului "character" (caracter). În dreptul său găsim, printre altele:

character = graphic sign / distinctive mark.

Au apărut deci cuvintele "sign" (semn) și "mark" (marcă); să vedem ce înseamnă acestea; anume, în ordine alfabetică:

mark = written symbol / sign

sign = written mark ... / symbol.

Este limpede că pe această cale nu vom ajunge nicăieri. Mai ales dacă ținem seamă că în limba română au fost împrumutate, în decursul vremii, diferite înțelesuri contradictorii pentru anumite cuvinte, ca să nu mai amintim și despre diverse cuvinte cu același înțeles. De aceea vom încerca să adoptăm un punct de vedere rigid, definind (pe cât posibil) în mod strict sensurile cuvintelor folosite.

O primă indicație despre modul în care trebuie să procedăm este dată de apariția, în toate cele patru explicații de mai sus, în mod sistematic, a cuvintelor legate "written" (scris) și "graphic" (grafic). De fapt, scopul nostru este de a transmite sau comunica diverse informații, iar unul dintre modurile de a face aceasta a fost și rămâne încă scrisul, desenul.

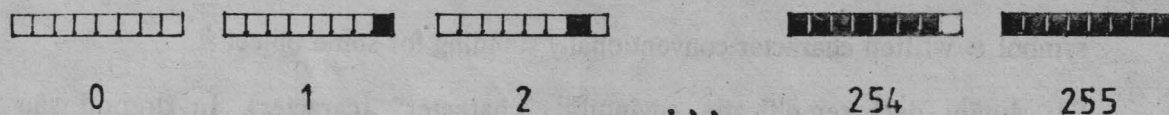
"Comunicare" și "informație" vor fi, pentru noi, noțiuni fundamentale, deci nu vor fi definite, ci doar explicate prin exemple și prin conexiunile cu alte noțiuni. Dar, este important să reținem, aceeași comunicare poate transmite diverse informații, poate fi "interpretată" în mod diferit.

Un caz particular important, asupra căruia vom reveni în mod repetat, va fi "comunicarea" de la un calculator (unitate centrală) spre un periferic (videoterminal, imprimantă, plotter etc.). Fiecare periferic își are propria sa "inteligență" cu care "interpretează" în felul său "comunicările" primite de la unitatea centrală.

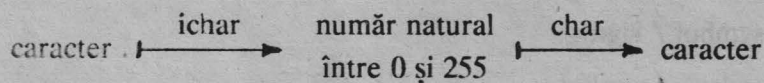
După cum este bine cunoscut, unitatea elementară de măsură a informației este bitul (cuvântul bit fiind o prescurtare de la "binary digit" = cifră binară); acesta poate fi asimilat cu o cifră binară, anume cu 0 sau cu 1.

Alternativa 0-1 poate fi interpretată și ca stins-aprins, gol-plin, fals-adevărat, alb-negru; pentru noi va prezenta importanță în special ultima interpretare, prin culoare (alb - negru, dar uneori negru - alb !).

Bitul fiind o unitate prea mică pentru lucrul efectiv, a fost adoptată, ca unitate practică de măsurare a informației, byte-ul. Vom considera, de cele mai multe ori, un byte ca o secvență de opt biți. Există, evident, $2^8 = 256$ de moduri de a colora (cu alb și negru) un byte, iar aceste moduri de colorare pot fi puse, evident, în corespondență biunivocă cu elementele mulțimii numerelor naturale mai mici decât 2^8 , de exemplu așa:



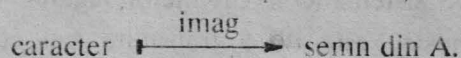
Să precizăm, pentru început, câteva noțiuni importante. Mai întâi, prin caracter vom înțelege oricare dintre cele 256 moduri de a colora un byte. Corespondența biunivocă de mai sus este realizată de cele două funcții, inverse una alteia:



(Acel număr natural între 0 și 255 poartă numele de caracter ASCII.)

Apoi, vom considera o mulțime A, finită și total ordonată (de desene sau imagini grafice), pe care o vom numi alfabet. De exemplu, acest alfabet ar putea fi format din desene ale literelor latine drepte și înclinate, ale semnelor matematice uzuale și speciale, ale literelor grecești. Este necesar să includem în mulțimea A, ca element special, desenul vid. Prin semn vom înțelege oricare dintre elementele acestei mulțimi A.

Ne vor interesa în special funcțiile de forma

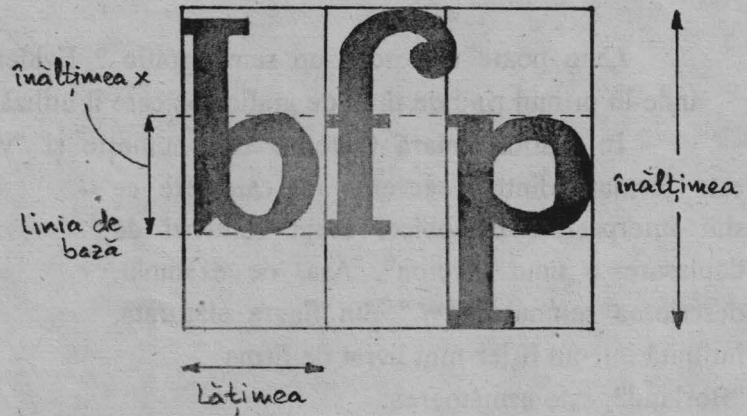


și folosirea lor în editarea de texte.

Semne grafice - caracteristici generale

Vom considera că semnele grafice sînt încadrate în dreptunghiuri și că sînt descrise cu doar două culori, alb și negru.

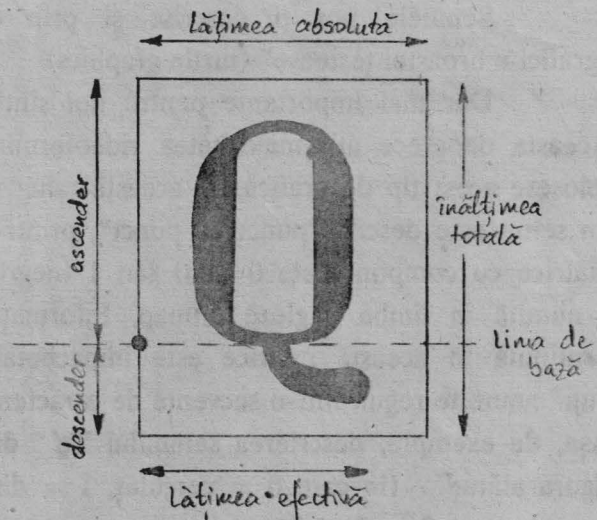
Să prezentăm caracteristicile unui semn, mai întâi așa cum sînt ele recunoscute în arta grafică tradițională. Cel mai bine o putem face prin exemple, dintre care trei sînt prezentate în figura alăturată. Punctele îngroșate marchează "originile" celor trei semne.



Să observăm că, de regulă, părțile "negre" ale semnelor sînt aliniate față de o așa-numită "linie de bază" (unele coboară însă și sub această linie), cea mai mare parte a lor aflîndu-se deasupra acesteia, pînă la așa-numita "înălțime x" (peste care unele mai trec) ceea ce se observă destul de clar în figură.

Să observăm de asemenea că dreptunghiul semnelor din mijloc (litera "f") este depășit în partea din dreapta sus; acesta este așa-numitul fenomen de "kerning", destul de important pentru aspectul estetic al scrisului.

Și în grafica produsă cu ajutorul calculatorului sînt păstrate aproximativ aceleași caracteristici; excepția o constituie fenomenul de "kerning", care este acum obținut prin considerarea a două lățimi: una absolută și alta efectivă. De data aceasta, vom preciza atributele "grafice" ale unui semn (vezi figura alăturată), comune graficii cu segmente și celei cu puncte (toate aceste atribute fiind numerice):



WIDTH = lățimea absolută
 ACTWID = lățimea efectivă (care nu este întotdeauna mai mică sau egală cu lățimea absolută a aceluiași semn !).

ASC = înălțimea peste linia de bază ("ascender")
 DESC = adîncimea sub linia de bază ("descender")
 HEIGHT = înălțimea totală (egală cu suma dintre ASC și DESC).

Să atenționăm asupra faptului că înălțimea peste linia de bază și adîncimea sub linia de bază nu sînt determinate doar de partea "neagră" a semnelor; mai pot interveni considerente legate de construcția concretă a aparatului de redare. De asemenea, lățimea efectivă este determinată și de considerente de "kerning" ale altor semne.

Semne grafice - descriere

Cum poate fi descris un semn grafic ? Evident, răspunsul la această întrebare depinde în primul rînd de tipul de grafică pe care îl utilizăm.

În grafica liniară (care se mai numește și "vectorială") descrierea unui semn este formată dintr-o secvență de caractere ce sînt interpretate de obicei drept comenzi de deplasare a unui "creion". Așa, de exemplu, descrierea semnului "∇" din figura alăturată, întîlnită într-un fișier-font livrat de firma "Borlañd", este următoarea:

80 09 83 80 86 89 85 06 81 86 87 00

Putem recunoaște aici, după o oarecare reflecție, drumul urmat de "creionul" ce descrie semnul; el trece prin punctele

(0,9), (3,0), (6,9), (5,6), (1,6), (7,0).

De asemenea, ne dăm seama ușor care sînt comenzile de coborîre și ridicare a "creionului".

Semnele pot fi descrise și prin deplasări relative, în spiritul așa-numitei "grafici a broaștei țestoase" (turtle graphics).

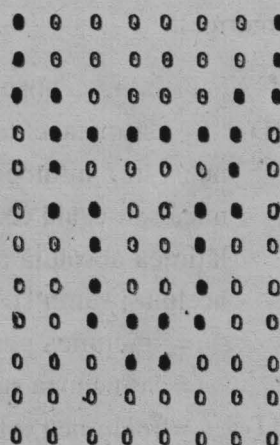
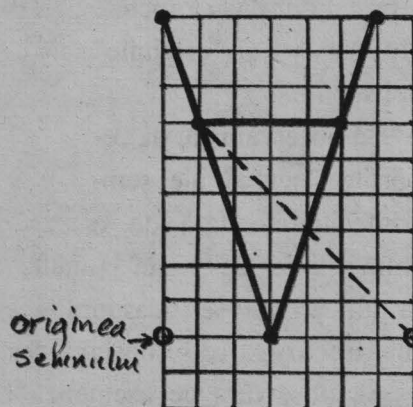
Dar mai importante pentru noi sînt descrierile în grafica raster (prin puncte). Aceasta deoarece atît majoritatea videoterminalelor, cît și imprimantele laser moderne, folosesc acest tip de grafică. În această tehnică, un semn este descris "punct cu punct", printr-o matrice cu componentele 0 (alb) sau 1 (negru) - numită în limba engleză bitmap. Informația conținută în această matrice este împachetată după anumite reguli într-o secvență de caractere. Așa, de exemplu, descrierea semnului "∇" din figura alăturată (în care 0 = cerculeț, 1 = disc negru) poate fi regăsită în secvența

81 81 C3 7E 42 66 24 24 3C 18 00 00

(ca în fișierul-font BIT8X12.FNT din sistemul PAINT_BRUSH) sau în secvența

3F 46 3F 7B 46 67 4B 47 4B 47 46 67 3F 7B 3F

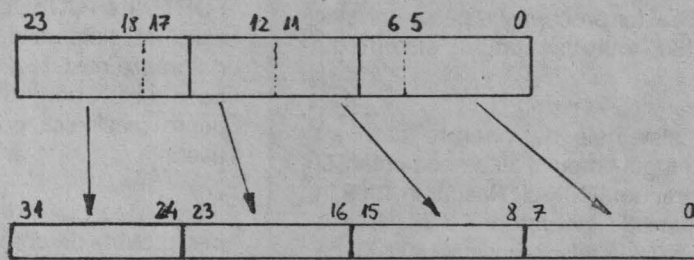
(ca în fonturile din sistemul VENTURA, specifice imprimantelor cu laser "Xerox").



Editoare și fonturi (I)

Dacă semnificația primei secvențe poate fi recunoscută relativ ușor (fiecărei linii a matricii îi corespunde câte un byte), nu același lucru se poate spune despre cea de-a doua. În aceasta semnul este descris, coloană după coloană, prin operația de sixelizare.

Procesul de sixelizare constă în transformarea a 24 de biți consecutivi în 4 bytes (= 32 biți) consecutivi, conform schemei următoare:



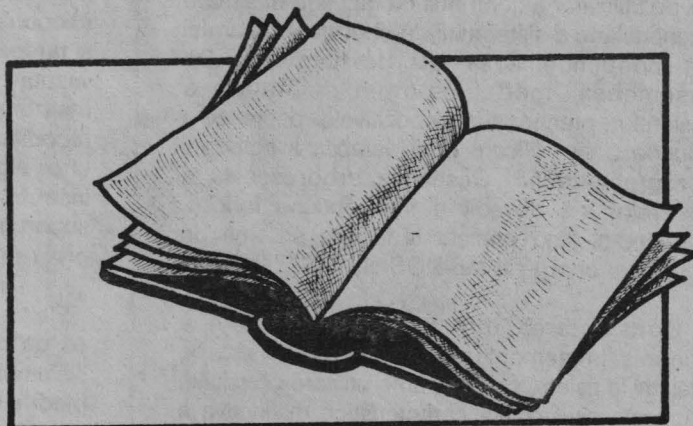
Fiecare grupă de câte 6 biți (de "valoare" v cuprinsă între 00h și 3Fh) se transformă într-un byte de valoare $v + 3Fh$. Astfel, în urma sixelizării se vor obține caractere de "valoare" cuprinsă între 63 (3Fh) și 126 (7Eh).

Să încheiem această primă parte prin a reaminti caracteristicile generale ale unui semn grafic, utilizat în grafica raster:

WIDTH ACTWID ASC DESC HEIGHT BITMAP.

Partea a doua va fi dedicată explicării noțiunii de "font".

Notă. Textul a fost obținut cu ajutorul unei imprimante laser Xerox4045.



Unix-ul, standardul viitorului in sisteme de operare ?

Aparitia in ultima vreme pe scena informaticii mondiale a unor evenimente deosebite relative la sistemul de operare UNIX, cu potentiale efecte asupra viitorului informaticii, ne-au indemnat sa facem citeva considerente asupra cauzelor aparitiei lor precum si a perspectivelor ce se deschid multdiscutatului sistem de operare.

Putine sint sistemele de operare care au generat de-a lungul timpului atitea comentarii si controverse precum UNIX-ul. Nascut in 1969 in S.U.A. la renumitele laboratoare Bell (filiala a gigantului mondial in telecomunicatii AT&T), ca o adaptare a unui proiect nefinalizat (lansat de ageneral Electric MIT, ce viza partajarea resurselor calculatorului PDP-7 intre doi utilizatori), sistemul **cucereste din ce in ce mai multi utilizatori ajungind la finele anului 1988 la respectabila cifra de 1.000.000 instalari. Consideram ca este suficient pentru a ne forma o imagine asupra dinamicii sistemului de operare in anii 1980, sa mentionam ca la inceputul deceniului 8 erau doar in jur de 2.500 de instalari. In prezent nu exista constructor mare care sa nu posede o adaptare proprie a UNIX-ului pe masinile pe care le produce.**

Surprinzator este faptul ca UNIX-ul, din punct de vedere tehnic, nu aduce concepte revolutionare. Care sint totusi elementele care au concurat la succesul acestui sistem de operare? Vom enumera citeva dintre acestea, fara a avea pretentia de a le ierarhiza. Sistemul de gestiune a fisierelor oarecum rudimentar, critica de unii pentru faptul ca structura si continutul fisierelor nu sint controlate de sistem ci de utilizator a contribuit de fapt la o mai mare generalitate a sistemului de operare, precum si la independenta sa fata de hardware. De asemenea, modul omogen de tratare a fisierelor, precum si a dispozitivelor periferice a dus la o simplificare considerabila a activitatii programatorului. Gestiunea arborescenta a fisierelor s-a dovedit a fi o optiune forficata, conceptul fiind prezent si la alte sisteme de operare, cum ar fi MS-DOS si VMS.

Considerarea directoarelor ca fisiere a permis implementarea multor functii sistem, relativ la fisiere, ca programe utilitare obisnuite. Aceasta este o alta caracteristica distinctiva a

UNIX- uli, ducind la efectuarea cu usurinta a modificarilor, asigurindu-se in acelasi timp fiabilitatea.

Un alt element de o importanta deosebita in succesul UNIX-ului il constituie limbajul de programare in care a fost scris. Este vorba de limbajul C (inventat de unul din autorul UNIX-ului) care, prin calitatile sale de expresivitate, versatilitate si eficienta s-a impus rapid ca un standard in domeniul limbajelor de programare, alaturi de mai "virstnicii" FORTRAN, COBOL, BASIC. Limbajul C i-a asigurat sistemului de operare, caruia i se datoreaza nasterea, portabilitatea. Practic orice constructor care dispune de un compilator pentru masina sa, poate sa adapteze UNIX-ul cu usurinta.

Spre deosebire de alte sisteme de operare, interpretorul de comenzi Shell nu este o parte a sistemului, ci un program obisnuit, putind fi in consecinta usor de modificat pentru a raspunde diferitelor cerinte. De asemenea, acesta se remarca prin faptul ca din anumite puncte de vedere, poate fi considerat ca un limbaj de programare de nivel inalt, usor extensibil pentru a raspunde unor cerinte de moment ale utilizatorilor. Prin modularitatea sa, data de faptul ca diferitele elemente ale sale sint implementate relativ independent, Shell-ul permite prototipizarea aplicatiilor.

Facilitatea de pipe, adica de captare a iesirii unui program si directarea ei ca intrare in altul, incurajaza inovatia si conexiunile in proiectarea aplicatiilor.

In sfirsit, nu inasa in cele din urma, multimea bogata de utilitare cu care este livrat sistemul constituie adevarate instrumente software care fac din UNIX un adevarat mediu de programare de mare productivitate. Programatorul este stimulat spre dezvoltarea de noi instrumente soft mai complexe, pornind de la cele simple (disponibile sub UNIX), incurajind dezvoltarea si nu reinventarea. **Munca programatorului este vazuta ca o activitate de creatie, de folosire a instrumentelor soft si nu una rutiniera de recodificare in diferite limbaje de asamblare a unor algoritmi consacrați. De asemenea, prin instrumentele sale soft, UNIX-ul reprezinta un suport pentru intreg ciclul de viata al produselor program.**

Elementele mai sus mentionate au contribuit pe de o parte la o mai mare adaptabilitate a sistemului, acesta putind fi usor de integrat in medille de calcul existente, iar pe de alta parte

au dus la o mare diversitate a adaptarilor elaborate. Livrarea initiala, in cod sursa si gratuit universitatilor, a dus la cresterea popularitatii acestui sistem in mediul universitar. Astfel, noile generatii de utilizatori au fost pregatiti in spiritul UNIX.

Portabilitate si compatibilitate.

Atitudinea producatorilor hard fata de fenomenul UNIX a fost diferita. Noii sositii au vazut in UNIX o sursa bogata de soft fiabil. Lansarea imediata a produselor lor era conditionata doar de existenta unui compilator C si de eforturi minore de adaptare. Consacrati domeniului (vezi IBM) au intrat pe piata UNIX-ului doar din ratiuni economice (si asta destul de tirziu, adica in momentul in care si-au vazut afectate interesele). Utilizatorii au optat pentru UNIX deoarece, chiar daca acesta nu le oferea performantele sistemelor de operare "de casa" ale marilor producatori, le asigura insa o oarecare independenta de producator.

Toate acestea, impreuna cu lipsa unei standardizari, au dus la aparitia unei diversitati de implementari UNIX, fapt care are insa implicatii negative asupra compatibilitatii. Astfel, dupa varianta UNIX V7, AT&T anunta la inceputul anilor 1980 o noua implementare, UNIX SYSTEM III la care renunta in 1983, pentru implementarea UNIX SYSTEM V. In paralel evolueaza implementarea Universitatii Berkeley, UNIX BSD. Noutatile aduse de aceste doua implementari se refera la ameliorarea gestiunii fizice a fisierelor, la dezvoltarea bibliotecii de rutine sistem (mecanisme de comunicare interprocese), precum si a programelor utilitare, evolutiile lor influentindu-se reciproc. Dintre celelalte implementari ale UNIX-ului mentionam: ULTRIX al firmei DEC, HP-XX al firmei Hewlett Packard, foarte popularul XENIX dezvoltat de Microsoft si Santa Cruz Operation, pentru calculatoare personale, A/UX al firmei Apple si nu in ultimul rind, AIX al IBM-ului, care incearca sa supraliciteze facilitatile UNIX-ului in speranta impunerii unui standard (reunind functionalitatile UNIX SYSTEM V cu cele ale UNIX BSD aducind in plus citeva facilitati proprii).

Standardizarea, singura alternativa de progres

Pornind de la premiza ca standardizarea, in conditiile unei adevarate anarhii pe piata UNIX-ului, este singura alternativa de evolutie a propriului sistem, AT&T elaboreaza in 1985

prima versiune a normelor SVID SYSTEM V (Interface Definitions). Aceste norme definesc functionalitatile sistemului UNIX SYSTEM V nu insa si modul lor de realizare; se definesc interfetele cu codul sursa si comportamentul modulelor sistem din punct de vedere al programatorului si utilizatorului. Conformitatea unui UNIX ce respecta SVID nu este decit functionala, recompilarea fiind o etapa necesara pentru a asigura compatibilitatea intre doua sisteme de operare UNIX ce respecta SVID-ul. De asemenea, AT&T incearca sa impuna si un standard la nivel de cod binar prin anuntarea (octombrie 1987) hotaririi de a elabora impreuna cu SUN a normelor ABI (Application Binary Interface). ABI este de fapt un standard destinat utilizatorilor prin care acestia vor putea rula (sub DOS de exemplu) direct aplicatiile lor sub orice masina bazata pe un procesor standard (Intel). In acest fel AT&T a reusit intr-o oarecare masura sa monopolizeze piata in continua crestere a UNIX-ului. Astfel la sfirsitul anului 1987 90% din baza instalata (pentru o piata estimata la mai mult de 9 miliarde) apartineau sistemelor UNIX SYSTEM V si descendentului sau pentru calculatoarele personale.

Crearea in mai 1988 (la numai o luna dupa anuntarea de catre AT&T a versiunii 4 a sistemului UNIX SYSTEM V) de catre marii producatori IBM, DEC, Hewlett-Packard, Apollo, Bull, Hitachi, Nixdorf, Philips, Simens (care vad in UNIX un adevarat pericol ce le afecteaza beneficiile si parti ale pietelor) a societatii Open Software Foundation (OSF) reprezinta o atitudine de contracarare a tendintei de monopolizare a AT&T-ului. Scopul declarat al societatii OSF este de a dezvolta un UNIX intr-adevar standard si deschis pornind de la sistemul AIX care este bazat pe UNIX SYSTEM V versiunea 2 al IBM-ului.

De fapt se urmareste dezvoltarea unui mediu de programare bazat pe UNIX cu eventuale facilitati de retea, baze de date relationale si suport pentru limba nationala. Dezvoltarea produselor OSF fiind bazate pe tehnologiile existente UNIX-ul avea sa fie deci mobilul declansarii mult preconizatului razboi al gigantilor AT&T si IBM.

Privarea AT&T de dreptul de a-si pune propriul sistem ca baza de plecare al proiectului OSF a dus la o riposta pe masura si anume crearea (in octombrie 1988) unei diviziuni AT&T, numita UNIX (oarecum autonoma financiar, cu capital deschis altor firme care sprijina AT&A) dedicata in totalitate dezvoltarii

sistemului UNIX SYSTEM V. Aliatii AT&T-ului, care nu vor sa renunte la standardul UNIX SYSTEM V deoarece doresc sa sustina clientii lor care au investit destul de mult in acest sistem, sint: Control Data, Fujitsu, Gould, ICL, Intel, Motorola, Olivetti, NCR Prime Sun Microsystems. Astfel, comunitatea informatica mondiala s-a scindat in 3 grupari, cele 2 tabere "beligerante" OSF si AT&T cu aliatii sai si partea mai prudenta care se situeaza intr-o pozitie de expectativa.

Aceste eforturi de standardizare au loc in contextul intensificarii activitatilor de acelasi gen a altor organisme internationale. Astfel, puternica organizatie internationala IEEE (The Institute of Electrical and Electronics Engineers Inc.) defineste normele POSIX (Portable Operating Systems Interface for Computing Environment). Specificatiile IEEE 1003.1 ale POSIX-ului au fost aprobate de alte organisme de standardizare (ANSI in 1986 si ISO). POSIX-ul, cu unele exceptii, este identic cu SVID-ul AT&T. Un alt organism international care va avea un rol important in definirea standardelor relative la UNIX este organizatia X/Open. Scopul declarat al organizatiei este de a dezvolta sisteme deschise (independente de producator) care sa devina standarde multi-vinzator. Astfel organizatia apara interesele utilizatorului, eliberindu-l de dependenta de un anumit echipament. Infiintata in 1984 de 5 constructori: Bull, Siemens, Olivetti, ICL, Nixdorf, organizatia reuneste astazi un numar apreciabil de firme (peste 100) reprezentind 15 mari producatori de hard intre care: AT&T, Bull, DEC, Fujitsu, HP, IBM, Olivetti, Philips, Siemens, Sun, Unisys; producatori de soft: Lotus Microsoft, Oracle, Palational Technology etc.), societati de servicii si utilizatori. Normele X/Open relative la portabilitate numite CAE (Common Application Environment) se sprijina in principal pe SVID.

Perspective

Viitorul UNIX-ului va depinde de solutiile pe care le va oferi, celor doua mari deziderate ale momentului in domeniu: integrare si unificare interfete. Relativ la primul element un rol important il are raportul dintre UNIX si sistemele de operare "de casa" (vezi OS/2 pentru IBM, VMS pentru DEC) ale marilor producatori. In acest sens prevedem trei tentinte:

Prima se refera la posibilitatea formarii sistemelor de operare "de casa", insa dupa preluarea controlului pietei UNIX-ului (dat de succesul proiectului OSF). Aceasta posibilitate,

desi ar avantaja consacratii domeniului, este mai putin probabila tinind cont de tendinta de integrare si deschidere.

A doua tendinta este de promovare a unui UNIX deschis conform cu principalele specificatii de standardizare (OSF, SVID, X/Open, POSIX) care insa sa inglobeze facilitatile sistemelor de operare "de casa". Noua versiune (a 3-a) a sistemului de operare Ultrix-32 (lansate de DEC in toamna lui 1988) se incadreaza in aceste tendinta. Astfel, conform declaratiilor DEC-ului, noua versiune este prima implementare UNIX care respecta standardul IEEE, POSIX, SVID-ul, specificatiile de nivel 0 ale OSF si standardul de portabilitate X/Open. De asemenea, prin caracteristicile sale, noua versiune este foarte apropiata de ultima versiune a sistemului de operare.

A treia tendinta ar putea fi enuntata: integrare prin unificarea interfetelor. Aceasta ar consta din coexistenta tuturor sistemelor de operare care insa la nivel utilizator ar avea o singura interfata standardizata. O abordare in acest sens se observa la firma DEC care pe langa incercarea de a apropia (la nivel de facilitati) cele doua sisteme (ULTRIX si VMS) realizeaza si conectarea VMS/Ultrix (printr-un protocol de retea de tip TCP/IP), preconizind implementarea interfetei de tip X-Windows XU (X User Interface) pe sistemul de operare VMS. De asemenea, preocuparile de elaborare a sistemelor duale UNIX/MS-DOS, se inscriu in aceasta tendinta.

Intr-o astfel de analiza nu trebuie desconsiderate aspectele de natura economica. Consideram astfel ca viitorul UNIX-ului va depinde in principal de cele patru orientari de standardizare OSF, AT&T, POSIX, X/Open inasa in mare masura el va fi decis de lupta de influenta a celor doua tabere OSF, pe de o parte si AT&T, pe de alta parte.

In disputa sa cu OSF, AT&T are un atu important de natura economica si anume incheierea cu armata aerului americana (la numai citeva zile dupa anuntarea infiintarii diviziei UNIX Inc.) a unuia din contractele secolului in valoare de 929 milioane dolari. Una din clauzele contractului prevede ca toate masinile beneficiarului sa functioneze sub sistemul de operare UNIX SYSTEM V care sa fie compatibil cu toate sistemele de operare UNIX eterogene, existente in armata americana.

Aplicatii grafice pe calculatoare personale

Aparute in 1981, calculatoarele personale au cunoscut o dezvoltare deosebita atat in ceea ce priveste partea de hardware, cit si programele scrise pentru aceste masini. De la inceput s-a constatat ca prezentarea grafica a rezultatelor programelor prezinta numeroase avantaje, intelegere mai rapida, forma atractiva de prezentare, ceea ce a condus la importante eforturi in realizarea si dezvoltarea adaptoarelor grafice si a dispozitivelor periferice grafice.

Avind in vedere faptul ca performantele aplicatiilor grafice (viteza, rezolutie, numar de culori, etc) sint strins legate de performantele sistemului folosit precum si a dispozitivelor periferice, consideram necesara o scurta prezentare a procesoarelor ce echipeaza calculatoarele personale din familia IBM (PC, XT, AT si PS/2) precum si a tipurilor de echipamente grafice cuplate la acestea: imprimante, plottere, scannere, digitizoare.

Procesoare

8086 echipeaza calculatoarele IBM PC, XT si PS/2 Model 30. Acest procesor lucreaza in mod real, adica este destinat sa execute un singur proces la un moment dat. Nu exista nici o protectie intre o parte a memoriei si alta. Poate adresa direct pina la 1 MB de memorie, iar adresa specificata in limbajul de asamblare corespunde cu adrese fizice de memorie. Viteza maxima de lucru este de 10 MHz.

80286 echipeaza sistemele de tipul IBM AT, PS/2 Model 50 si PS/2 Model 60. Acest procesor poate lucra in mod real ca si 8086, dar are si un mod protejat, in care mai multe procese pot fi executate concurent, memoria utilizata de un proces fiind protejata de cea folosita de un alt proces. Procesorul poate adresa pina la 16 MB de memorie, adresa specificata necorespunzind direct cu adrese fizice de memorie. Este un procesor pe 16 biti, cu viteza maxima de lucru de pina la 16 MHz.

80386 echipeaza masinile din familia IBM PS/2 Model 80. Este un procesor pe 32 de biti, continind toate facilitatile procesorului 80286 si aceleasi moduri de lucru. In plus, poate adresa pina la 4 GB de memorie, iar viteza de lucru ajunge pina la 33 MHz.

Coprocesoarele aritmetice 8087, 80287 si 80387

Sint procesoare specializate, destinate efectuării calculelor matematice mult mai rapid decit procesoarele din familia 86. In plus, aceste calcule sint efectuate simultan cu executia procesorului principal. Coprocesoarele 80287 si 80387 pot lucra si in mod protejat, iar 80387 are un set de instructiuni deosebit de puternic fata de celelalte doua.

Indiferent de masina pe care ruleaza, sistemul de operare DOS (PC-DOS sau MS-DOS) lucreaza numai in modul real. Modul protejat al procesoarelor 80286 si 80386 este folosit de sistemele de operare UNIX, Microsoft XENIX, OS/2 si MOS (Modular Operating System, numai pentru 80386).

Adaptoare grafice

Aparut la IBM in 1984, primul adaptor grafic care s-a impus ca standard a fost cel numit CGA (Color Graphics Adapter). Rezolutia acestui adaptor (320x200 pixeli in 4 culori sau 640x200 in 2 culori) este departe de a oferi utilizatorilor posibilitatea realizarii unor aplicatii grafice complexe. De asemenea, numarul maxim de culori disponibile simultan pe ecran, patru, este mic.

Urmatoarele adaptoare grafice, EGA (Enhanced Graphics Adapter) si VGA (Video Graphics Array), care de asemenea s-au impus ca standarde, au o rezolutie mult mai buna (640x350 la EGA si 640x480 la VGA) si un numar mult mai mare de culori (16 dintr-o paleta de 64 la EGA si 16 dintr-o paleta de 256k la VGA). Desi aceste adaptoare reprezinta un salt calitativ remarcabil comparativ cu CGA, nici ele nu sint inca la nivelul cerut de aplicatiile unde fidelitatea imaginii este foarte importanta, de exemplu la procesare de imagini, simulari de procese complexe. Pentru aceste aplicatii, IBM dezvolta in continuare adaptoarele grafice, un exemplu fiind AGA (Advanced Graphics Adapter) cu o rezolutie de 1024x768.

Alte adaptoare grafice utilizate sint: Color Metric 20 Card, cu o rezolutie de 2000x484 si 100 de culori; Hercules Color Graphics Card, cu 16 culori din 64 si o rezolutie de 720x348 si, in plus, cu posibilitatea definirii de catre utilizator a unui set de 3072 de caractere (o extensie a codului ASCII).

Pentru aplicatii grafice complexe s-au realizat adaptoare grafice dedicate cu o rezolutie exceptionala, cele mai bune rezolutii intilnite depasind nivelul de 4096x4096.

Imprimante

O imprimanta grafica este considerata ca avind o rezolutie buna daca aceasta depaseste 180 dpi (puncte pe inch, in engleza dots per inch). Imprimantele cu laser pot ajunge pina la 300 dpi sau chiar la 600 dpi, tinzind sa devina comparabile cu cele mai moderne masini de tiparit).

Scannere

Acestea sint dispozitive destinate preluarii imaginilor de pe fotografii. Imaginile, rasterizate la o rezolutie care depinde de calitatea scanner-ului, sint introduse in fisiere ce pot fi apoi prelucrate de diverse programe (editoare grafice, pachete de desktop publishing). Imaginile obtinute pot fi vizualizate pe display sau imprimanta. Marea majoritate a scanner-elor au o rezolutie de 300 dpi, diferenta calitativa intre ele fiind data de viteza de parcurgere a imaginii. Un model foarte performant este Page Reader Model 245, ce poate fi utilizat atit pe calculatoare IBM, cit si Macintosh. Acesta poate parcurge o pagina de 27x20 cm in 30 de secunde.

Digitizoare

Calitatea acestora este data de numarul de functii predefinite puse la dispozitia utilizatorului (de catre driver-ul livrat odata cu echipamentul) precum si de rezolutie. Cele mai performante digitizoare au un set de 100-150 de functii predefinite, ceea ce faciliteaza mult lucrul cu acestea, iar rezolutia poate ajunge pina la 100 de puncte distincte de digitizare pe centimetru.

Alte dispozitive folosite in aplicatiile grafice sint tableta grafica, mouse-ul, joystick-ul.

Firmele mari, producatoare de periferice grafice, ale caror produse s-au impus ca standarde, sint: IBM, EPSON, TOSHIBA, Xerox, HEWLETT-PACKARD.

Aplicatii grafice

Produsele software de aplicatii grafice pot fi impartite aproximativ in urmatoarele categorii:

Editoare grafice

In cadrul acestei clase, produsul care se detaseaza net fata de celelalte editoare prin facilitatile deosebite puse la dispozitia utilizatorilor este AutoCAD, produs de firma americana AutoDesk. Ultima versiune a acestui editor este deja foarte utilizata, aceasta raspindire datorindu-se si numarului foarte mare de drivere pentru dispozitive periferice oferite de program (peste 100). Facilitatile acestui editor, destinat in special proiectarii asistate de calculator, sint, in mare:

- trasare in doua si trei dimensiuni, in numeroase sisteme de coordonate si tipuri de proiectii;
- selectarea oricarui sistem de masura si a dimensiunii paginii de desen;
- localizarea obiectelor prin referinta la alte obiecte;
- marirea sau micșorarea desenelor (zoom);
- editarea de simboluri;
- numeroase tipuri de linie de trasare si culori;
- scalari si rotatii ale obiectelor selectate;
- numeroase tipuri de fisiere de desen, fisiere ce pot fi importate si de catre alte programe grafice, precum si posibilitatea importarii de catre AutoCAD de fisiere create cu alte programe;
- umplerea poligoanelor cu diverse stiluri de interior;
- text, cu peste 60 de tipuri de caractere, in alfabetele latin, chirilic, grec, simboluri matematice, meteorologice, astronomice, muzicale, cu facilitati deosebite in ceea ce priveste disponerea acestora pe suprafata de desen;
- cotari automate ale obiectelor selectate;
- atribute de vizibilitate, prioritate, culoare, stil, atasate obiectelor si posibilitatea editarii facile a acestor atribute;

- introducerea unui desen "de mina", cu ajutorul unui dispozitiv grafic de intrare: tableta grafica, mouse;
- salvarea imaginilor editate in fisiere;
- trasari de curbe, suprafete si polilinii;
- facilitati deosebite de trasare in 3D, de exemplu crearea unui obiect 3D dintr-unul 2D, eliminarea liniilor ascunse, simularea fotografierii cu lentile de distanta focala variabila (teleobiectiv sau "fish-eye"), cu specificarea uneia sau mai multor surse de lumina, realism vizual, iluminari, umbriri;
- crearea de macroinstructiuni cu ajutorul limbajului de programare AutoLISP.

Pe langa acest editor, care incorporeaza un volum foarte mare de munca si dispune de un suport matematic puternic, exista si multe alte editoare grafice, de exemplu Freelance 2 Plus, realizat de firma Lotus, care nu dispune de diversitatea mare de comenzi a AutoCAD-ului, dar, in schimb, este usor de folosit, si pentru desene simple sau rezolvarea unor probleme curente de vizualizare grafica a suprafetelor (de exemplu in grafica economica), este preferabil celui editor foarte complex.

Freelance este un editor 2D, care, in ceea ce priveste complexitatea problemelor abordate se apropie de partea 2D a AutoCAD-ului numai in domeniul introducerii si manipularii textului, avind un numar remarcabil de tipuri de text, care poate fi afisat pe ecran sau tiparit intr-o mare varietate de forme. Contine, de asemenea, o biblioteca de desene simple care pot fi inserate in desene mai complexe create de utilizatori.

Alte editoare grafice cu succes comercial sint Microsoft Paint, Paintbrush, Deluxe Paint, MiniCAD 2, Generic CAD, si multe, multe altele.

Desktop Publishing

Sub aceasta denumire, pentru care nu am gasit inca o traducere acceptabila in limba romana, se inscriu programele care utilizeaza calculatoarele pentru realizarea de publicatii (ziare, reviste, reclame, documente personale), parcurgind toate etapele tipografiei clasice:

- scrierea documentului, care se realizeaza de catre autor cu ajutorul unui procesor de texte;
- editarea/revizuirea textului; editorul efectueaza modificari ale textului initial, apoi revizuieste materialul, eventual de mai multe ori, pina cind, de comun acord cu autorul, considera documentul ca ajuns intr-o stare finala;
- inserarea textului in pagina; operatorul introduce textul manuscrisului in calculator, tinind cont de instructiunile marginale ale editorului privind dimensiunea literelor, tipul de text, precum si numarul de coloane de pe pagina si lungimea fiecarei coloane;
- ilustrarea; un desenator sau un artist grafician creeaza diagrame, grafice, desene diverse sau preia o fotografie cu ajutorul unui scanner pentru ilustrarea articolului;
- design-ul paginii se realizeaza prin specificarea cu ajutorul liniilor desenate pe macheta, a locului ocupat de fiecare coloana, lungimea si latimea ei, precum si a zonelor pentru ilustratii. De regula si aceasta faza presupune mai multe revizuri;
- machetare; cind totul este gata, cu ajutorul unor tehnici speciale se realizeaza macheta paginii din viitoarea publicatie, care reprezinta negativul acestei pagini;
- tiparirea; macheta astfel obtinuta este tiparita in tirajul dorit;

Cel mai performant si utilizat produs din categoria desktop publishing este Xerox Ventura Publisher, realizat de firma Xerox pentru microcalculatoare IBM PC.

Acest program realizeaza toate etapele tipografiei clasice, mult mai usor, mai repede si pe echipamente mult mai ieftine. Singura diferenta o constituie tirajul redus comparativ cu metodele clasice. Programul este compatibil cu un numar mare de procesoare de text (WordStar, Word Perfect, Word, Multimate) si editoare grafice (AutoCAD, Paintbrush, Windows Paint).

Alte programe desktop publishing mult utilizate sint: PageMaker, XPress, Super Paint, Publish It.

Worksheet grafic

Produsele worksheet sînt destinate aplicatiilor in domeniul financiar-contabil. Cu ajutorul lor se editeaza de exemplu: tabele de profituri si pierderi, analiza vinzarilor, buget, salarii, date personale si sute de alte aplicatii posibile. Datele sînt introduse de catre operator in celule, o celula fiind la intersectia dintre o linie si o coloana in cadrul unui tabel. Unele celule pot fi definite ca relatii intre doua sau mai multe celule introduse anterior, un exemplu fiind o celula de total obtinuta prin insumarea tuturor celulelor dintr-o coloana.

Aceste produse utilizeaza grafica pentru prezentarea datelor din tabel intr-o forma mai atractiva. Lotus 1-2-3 realizat, de catre firma Lotus, este produsul cel mai reprezentativ din aceasta clasa, realizeaza numeroase tipuri de grafice economice, diagrame, etc. Un alt produs Worksheet foarte raspindit este Quattro, apartinind firmei Borland International.

Alte produse

Un produs foarte utilizat pentru realizarea de prezentari publicitare este Story Teller. Acest produs contine o puternica baza de desene care impreuna cu desenele realizate de catre utilizator pot concura la realizarea unui scenariu de reclama. Produsul contine de asemenea facilitati atractive de schimbare a cadrelor. El pune la dispozitia utilizatorului imaginativ un mare numar de tipuri de text.

Animatia pe calculator este un alt domeniu in care se depun eforturi deosebite datorita modului laborios in care se realizeaza filmele de animatie in prezent, si anume prin desenarea fiecarui cadru in parte.

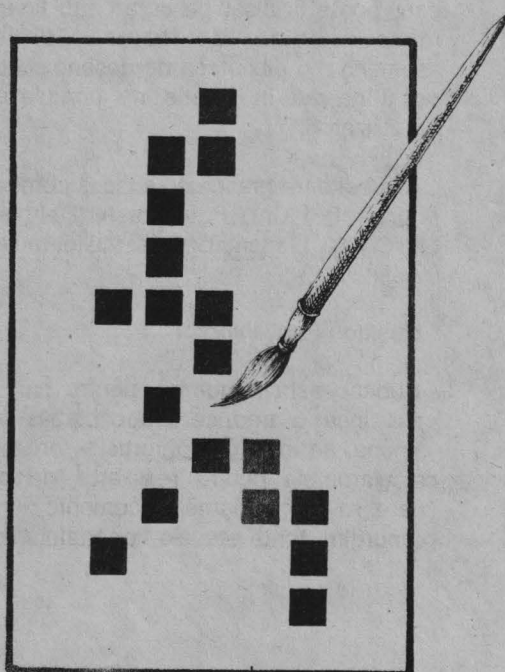
Firma Apple detine un produs pentru realizarea automata a filmelor de desene animate intitulat Video Works. Acest produs preia desene realizate cu un editor grafic si le monteaza intr-un scenariu de film, cu numeroase facilitati din domeniul filmului cum ar fi: selectie de secventa, mixaje, modificarea de obiecte pentru obtinerea efectului de animatie, etc.

Biblioteci grafice

Pe langa aceste aplicatii puse la dispozitia utilizatorilor, orice limbaj de nivel inalt contine si o biblioteca grafica.

Reprezentative in acest sens sînt bibliotecile compilatoarelor Microsoft C V5, Turbo C si Turbo Pascal. Pe langa bibliotecile compilatoarelor s-au realizat si biblioteci independente, ale caror rutine pot fi apelate de regula din mai multe limbaje de nivel inalt. Functiile bibliotecilor grafice pot apartine urmatoarelor categorii:

- rutine pentru configurare: selecteaza modul grafic dorit si stabilesc zonele de memorie pentru scrierea si afisarea imaginilor;
- rutine pentru stabilirea coordonatelor desenului (spatiul utilizator) si zona activa din ecran (spatiul dispozitiv);
- rutine pentru stabilirea paletei de culori;
- rutine pentru stabilirea atributelor: culoare de trasare, stil de linie, grosime linie etc;
- rutine de trasare: linie, arc, elipsa, polilinie, rutine de umplere a unui contur etc;
- rutine pentru afisarea textului;
- drivere si rutine pentru copierea imaginii grafice la imprimanta;
- rutine pentru gestiunea memoriei ecran.



Structuri

Vom incerca in cele ce urmeaza sa abordam impreuna citeva aspecte legate de implementarea grafurilor in programele aplicative.

Problema tratata in acest exemplu nu este nici cea mai complexa si nici cea mai reprezentativa in domeniu. Cu ajutorul ei insa, vrem sa oferim tinerilor informaticieni o imagine a felului in care probleme bine fundamentate din punct de vedere teoretic necesita in practica gasirea unor noi solutii care sa le rezolve. Acest lucru este impus in principal de catre doi factori:

- restrictiile de resurse ale sistemelor de calcul,
- caracterul dinamic al problemelor.

Asemenea probleme concrete sint cele legate de manipularea grafurilor orientate. In intreg domeniul exista numerosi algoritmi si structuri de date corespunzatoare lor. Insa, in cazul unei probleme practice ne putem intilni cu situatii in care graful curent are un numar variabil de noduri si implicit de arce.

Este evident ca in acest caz nu mai putem avea pretentia ca structura de date ce reprezinta graful sa fie matricea sa de adiacenta. Un astfel de obiect este bine sa fie modelat printr-o structura de date dinamica care sa asigure o folosire eficienta a memoriei interne.

Inainte de a trece la problema propriu-zisa si la rezolvarea ei, mentionam ca solutia a fost implementata in limbajul Turbo C (Versiunea 2.0).

Structura pe care o propun este lista de obiecte. Fiecare obiect contine la rindul lui urmatoarele:

- informatia utila a unui nod din graf,
- lista de pointeri catre nodurile atinse de arce care pleaca din nodul curent.

In limbajul C putem defini aceasta structura astfel:

```
struct Node { int info; struct NodeList *node; struct Node *next; }  
struct NodeList { struct Node *node; struct NodeList *next; }
```

Un graf este reprezentat printr-o lista de tip NodeList si va fi indicat printr-un pointer catre inceputul listei.

Problema

Sa se scrie o rutina C care sa furnizeze toate drumurile intre doua noduri ale unui graf.

Rezolvarea

Va propunem o solutie mai putin obisnuita, si anume o rutina care "intoarce" la fiecare apel al ei un alt drum decit cele returnate anterior.

Structura care va retine drumul este o lista dublu inlantuita de forma:

```
struct Path { struct Node *node; struct NodeList *choice; struct Path *back; struct Path *next; }
```

Pentru un pointer de forma: struct Path *curr;

Structuri

curr->choice este un pointer catre elementul din lista back->node->adjacent care corespunde arcului dintre back->node si curr->node.

Pentru rezolvarea problemei s-a adoptat o solutie de tip backtrack.

La sfirsitul cautarilor, dupa epuizarea tuturor posibilitatilor, rutina va intoarce un pointer NULL.

Variabile

flag este un indicator folosit pentru a distinge prima trecere prin rutina de urmatoarele; **begin** si **end** sint inceputul si respectiv sfirsitul listei care va contine un drum intre A si B. Intotdeauna vom avea: begin->node == A si end->choice == B; **new** si **aux** sint variabile auxiliare.

Listiugul functiei **path_between()** este prezentat in continuare. Pe cei intrigati de folosirea instructiunii **goto** ii trimitem la ... celebrul articol "GOTO Considered Harmful" Considered Harmful" apartinind unui programator adevarat, Frank Rubin. De altfel asupra acestui subiect vom mai reveni in paginile revistei noastre.

```
struct Path *path_between (struct Node *A, struct Node *B) {  
  
    static int flag = 0;  
  
    static struct Path *begin, *end;  
    struct Path *new;  
    struct NodeList *aux;  
  
    /*  
     * Am mai gasit drumuri intre A si B ?  
     * flag = 1 --- Da.  
     * flag = 0 --- Nu.  
     */  
  
    if (flag == 1)  
        goto point;  
    new = calloc (1, sizeof (struct Path));  
    new->node = A;  
    new->choice = A->adjacent;  
  
    begin = end = new;  
  
    while (TRUE) {  
        aux = end->choice;  
        if (aux == NULL) {  
            /*  
             * Drumul curent, retinut in lista de la begin la end,  
             * nu mai poate fi continuat  
             */  
  
            if (begin == end) {  
                flag = 0;  
                free (begin);  
                return (NULL);  
            }  
            else {  
  
                /* backtrack */  

```



```

new = end->back;
free (end);
end = new;

point:
end->choice = (end->choice)->next;
end->next = NULL;
}
}
else {
if (aux->node == B) {
flag = 1;
return (begin);
}

/* Aduagam un nou nod pe drumul de la A la B. */

new = calloc (1, sizeof (struct Path));

end->next = new;
new->node = aux->node;
new->choice = (aux->node)->adjacent;
new->back = end;
end = new;
}
}

```



In lumea informaticii de astazi exista foarte multe limbaje de programare si fiecare dintre ele isi au adeptii lor infocati .Dintre ele se distinge insa fara nici o indoiala limbajul de programare de nivel inalt TURBO C, care ofera pe linga fiabilitate si o usurinta deosebita in invatarea si folosirea lui. Aceste facilitati se datoreaza atat celui asa numit Mediu de Dezvoltare Integrata (Integrated Development Environment) care permite realizarea compilarii linkeditarii, rularii si depanarii programelor numai prin folosirea meniurilor cit si datorita bogatiei librariilor de functii de care aceasta versiune a compilatorului le ofera.

Va oferim aici un cod sursa pentru demonstrarea capacitatilor grafice ale limbajului TURBO C pe calculatorul IBM PC sau compatibile.

[TURBO C is a trademark of Borland International]

```

/*  DEMONSTRATIE DE GRAFICA PE CALCULATOR IN
    LIMBAJUL TURBO C

*/

#ifdef __TINY__
#error Programul nu ruleaza in modelul de memorie TINY
#endif

#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>

#include <graphics.h>

#define ESC    0x1b          /* Definesc tasta escape      */
#define OFF    0
#define PI     3.14159

int  GraphDriver;          /* Driverul Grafic           */
int  GraphMode;           /* Valoarea modului grafic   */
double AspectRatio;       /* Cum arata un pixel pe ecran */
int  MaxX, MaxY;          /* Rezolutia maxima a ecranului */
int  MaxColors;           /* Numarul maxim de culori disponibile */
int  ErrorCode;           /* Raporteaza erori grafice  */
struct palettetype palette; /* Folosit ptr. citirea inf. despre paleta */

/*
/*  Functiile prototip
/*
void Initial(void);
void RandomBars(void);
void ColorDemo(void);
void PieDemo(void);
void PutPixelDemo(void);
void PutImageDemo(void);
void Pause(void);
void MainWindow(char *header);
void StatusLine(char *msg);
void changetextstyle(int font, int direction, int charsize);
void DrawBorder(void);
int  gprintf(int *xloc, int *yloc, char *fmt, ...);

/*
/*  Functia main
/*
int main()
(

    Initial();              /* Pune sistemul in modul grafic */
    ColorDemo();            /* Porneste demonstratia         */
    PutPixelDemo();
    RandomBars();
    PieDemo();
    closegraph();          /* Revin la modul text          */

```


Aplicații în TURBO C

```

return(0);
)

/*
/* INITIAL:Initializeaza sistemul grafic si raporteaza erorile */
/* intervenite. */
/* */
*/

void Initial(void)
(
    int xasp, yasp;

    GraphDriver = DETECT; /* Auto-detectie */
    initgraph( &GraphDriver, &GraphMode, "" );
    ErrorCode = graphresult(); /* Citeste rezultatul initializarii*/
    if( ErrorCode != grOk ){ /* A aparut eroare in timpul init*/
        printf(" Eroare de sistem grafic: %s\n", grapherrormsg( ErrorCode ) );
        exit( 1 );
    }

    getpalette( &palette ); /* Citeste paleta de culori */
    MaxColors = getmaxcolor() + 1; /* Citeste numarul maxim de culori*/

    MaxX = getmaxx();
    MaxY = getmaxy(); /* Citeste dimensiunea ecranului*/

    getaspectratio( &xasp, &yasp ); /* Citeste aspect-ul de hardware*/
    AspectRatio = (double)xasp / (double)yasp; /* Citeste factorul de corectie*/
)

/*
/* RANDOMBARS: Afiseaza bare aleatoare */
/* */
*/

void RandomBars(void)
(
    int color;

    MainWindow( "Bare Aleatoare" );
    StatusLine( "Esc aborteaza sau apasa o tasta..." ); /* Pune mesaj jos */
    while( !kbhit() ){ /* Pina la tastare... */
        color = random( MaxColors-1 )+1;
        setcolor( color );
        setfillstyle( random(11)+1, color );
        bar3d( random( getmaxx() ), random( getmaxy() ),
            random( getmaxx() ), random( getmaxy() ), 0, OFF);
    }

    Pause(); /* Pauza ptr. raspunsul utilizat */
)

/*
/* COLORDENO: Afiseaza paleta de culori. */
/* */
*/

void ColorDemo(void)
(
    struct viewporttype vp;
    int color, height, width;
    int x, y, i, j;
    char cnum[5];

    MainWindow( "Demonstratie de culori" ); /* Afiseaza numele dem. */

    color = 1;
    getviewsettings( &vp ); /* Citeste dimensiunea window-ului*/
    width = 2 * ( (vp.right+1) / 16 ); /* Citeste dimensiunea dreptunghi*/
    height = 2 * ( (vp.bottom-10) / 10 );

    x = width / 2;
    y = height / 2; /* Margine */

    for( j=0; j<3; ++j ){ /* Ciclu ptr. rind */
        for( i=0; i<5; ++i ){ /* Ciclu ptr. coloana */

```

Aplicații în TURBO C

```

setfillstyle(SOLID_FILL, color); /* Coloreaza hasurile */
setcolor( color );              /* Aceiasi culoare ptr. margine */

bar( x, y, x+width, y+height ); /* Deseneaza dreptunghiul */
rectangle( x, y, x+width, y+height ); /* Contureaza dreptunghiul */

if( color == BLACK )           /* Daca a fost negru... */
    setcolor( WHITE );         /* Pune conturul alb */
    rectangle( x, y, x+width, y+height );
}

itoa( color, cnum, 10 );        /* Converteste numarul in ASCII */
outtextxy( x+(width/2), y+height+4, cnum ); /* Arata nr. culorii */

color = ++color % MaxColors;   /* Treci la culoarea urmatoare */
x += (width / 2) * 3;
} /* Sfisitul ciclului pe rind */

y += (height / 2) * 3;
x = width / 2;

} /* Sfisitul ciclului pd coloana*/

Pause(); /* Pauza ptr. raspuns */

}

/*
/*      PIEDEMO: Afiseaza o diagrama in "felii de placinta".
/*
/*
#define adjasp( y )      ((int)(AspectRatio * (double)(y)))
#define torad( d )      (( (double)(d) * PI ) / 180.0 )

void PieDemo(void)
{
    struct viewporttype vp;
    int xcenter, ycenter, radius, lradius;
    int x, y;
    double radians, piesize;

    MainWindow( "Felii de Placinta");

    getviewsettings( &vp ); /* Citeste viewport-ul curent */
    xcenter = (vp.right - vp.left) / 2; /* Centreaza orizontal */
    ycenter = (vp.bottom - vp.top) / 2 + 20; /* Centreaza vertical */
    radius = (vp.bottom - vp.top) / 3; /* Va acoperi 2/3 din ecran */
    piesize = (vp.bottom - vp.top) / 4.0;

    while( AspectRatio*radius < piesize ) ++radius;

    lradius = radius + ( radius / 5 ); /* Etichete plasate 20% mai departe*/

    changetextstyle( TRIPLEX_FONT, HORIZ_DIR, 4 );
    setttextjustify( CENTER_TEXT, TOP_TEXT );
    outtextxy( MaxX/2, 6, "Aceasta este o diagrama" );
    changetextstyle( TRIPLEX_FONT, HORIZ_DIR, 1 );
    setttextjustify( CENTER_TEXT, TOP_TEXT );

    setfillstyle( SOLID_FILL, RED );
    pieslice( xcenter+10, ycenter-adjasp(10), 0, 90, radius );
    radians = torad( 45 );
    x = xcenter + (int)( cos( radians ) * (double)lradius );
    y = ycenter - (int)( sin( radians ) * (double)lradius * AspectRatio );
    setttextjustify( LEFT_TEXT, BOTTOM_TEXT );
    outtextxy( x, y, "25 %" );

    setfillstyle( WIDE_DOT_FILL, GREEN );
    pieslice( xcenter, ycenter, 90, 135, radius );
    radians = torad( 113 );
    x = xcenter + (int)( cos( radians ) * (double)lradius );
    y = ycenter - (int)( sin( radians ) * (double)lradius * AspectRatio );
    setttextjustify( RIGHT_TEXT, BOTTOM_TEXT );
    outtextxy( x, y, "12.5 %" );

    setfillstyle( INTERLEAVE_FILL, YELLOW );
    setttextjustify( RIGHT_TEXT, CENTER_TEXT );
    pieslice( xcenter-10, ycenter, 135, 225, radius );
    radians = torad( 180 );

```


Aplicații în TURBO C

```

x = xcenter + (int)( cos( radians ) * (double)radius );
y = ycenter - (int)( sin( radians ) * (double)radius * AspectRatio );
settextjustify( RIGHT_TEXT, CENTER_TEXT );
outtextxy( x, y, "25 %" );

setfillstyle( HATCH_FILL, BLUE );
pieslice( xcenter, ycenter, 225, 360, radius );
radians = torad( 293 );
x = xcenter + (int)( cos( radians ) * (double)radius );
y = ycenter - (int)( sin( radians ) * (double)radius * AspectRatio );
settextjustify( LEFT_TEXT, TOP_TEXT );
outtextxy( x, y, "37.5 %" );

Pause();

}
/*
/*      PUTPIXELDEMO: Afiseaza puncte aleatoare pe ecran
/*      si apoi le sterge
/*
/*
void PutPixelDemo(void)
{
    int seed = 1958;
    int i, x, y, h, w, color;
    struct viewporttype vp;

    MainWindow( "Afisare aleatoare" );

    getviewsettings( &vp );
    h = vp.bottom - vp.top;
    w = vp.right - vp.left;

    srand( seed );
    /* Restarteaza functia random # */

    for( i=0 ; i<5000 ; ++i ){
        /* Pune 5000 pixeli pe ecran */
        /* Genereaza o locatie aleatoare*/
        x = 1 + random( w - 1 );
        y = 1 + random( h - 1 );
        color = random( MaxColors );
        putpixel( x, y, color );
    }

    srand( seed );
    /* Sterge cei 5000 pixeli */

    for( i=0 ; i<5000 ; ++i ){
        /* Citeste pixeli colorati */
        /* Scribe pixelul BLACK */
        x = 1 + random( w - 1 );
        y = 1 + random( h - 1 );
        color = getpixel( x, y );
        if( color == random( MaxColors ) )
            putpixel( x, y, 0 );
    }

    Pause();

}

/*
/*      PAUSE: Pauza pina utilizatorul tasteaza. Daca se
/*      tasteaza ESC atunci programul se termina.
/*
/*
void Pause(void)
{
    static char msg[] = "Esc aborteaza sau apasa o tasta...";
    int c;

    StatusLine( msg );
    /* Pune msg pe ultima linie */

    c = getch();
    /* Citeste un caracter de la */
    /* tastatura */

    if( ESC == c ){
        /* Se doreste terminare? */
        /* Schimba in modul text */
        /* Intoarce la S.O. */
        closegraph();
        exit( 1 );
    }

    if( 0 == c ){
        /* Citeste scan codul de la taste*/
        c = getch();
    }
}

```

Aplicații în TURBO C

```

cleardevice();          /* Sterge ecranul          */
}

/*
/*   MAINWINDOW: Deseneaza o fereastră principală ptr. demonstrație */
/*   și un viewport ptr. mesaje.                                     */
/*
void MainWindow( char *header )
(
    int height;

    cleardevice();          /* Sterge ecranul grafic      */
    setcolor( MaxColors - 1 ); /* Pune culoarea curentă alb */
    setviewport( 0, 0, MaxX, MaxY, 1 ); /* Deschide port-ul la tot ecranul */

    height = textheight( "H" ); /* Citeste înălțimea textului */

    changetextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
    setttextjustify( CENTER_TEXT, TOP_TEXT );
    outtextxy( MaxX/2, 2, header );
    setviewport( 0, height+4, MaxX, MaxY-(height+4), 1 );
    DrawBorder();
    setviewport( 1, height+5, MaxX-1, MaxY-(height+5), 1 );

}

/*
/*   STATUSLINE: Afiseaza pe ultima linie . */
/*
void StatusLine( char *msg )
(
    int height;

    setviewport( 0, 0, MaxX, MaxY, 1 );
    setcolor( MaxColors - 1 );

    changetextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
    setttextjustify( CENTER_TEXT, TOP_TEXT );
    setlinestyle( SOLID_LINE, 0, NORM_WIDTH );
    setfillstyle( EMPTY_FILL, 0 );

    height = textheight( "H" );
    bar( 0, MaxY-(height+4), MaxX, MaxY );
    rectangle( 0, MaxY-(height+4), MaxX, MaxY );
    outtextxy( MaxX/2, MaxY-(height+2), msg );
    setviewport( 1, height+5, MaxX-1, MaxY-(height+5), 1 );

}

/*
/*   DRAWBORDER: Deseneaza o linie solida in jurul viewport-ului */
/*   curent                                                         */
/*
void DrawBorder(void)
(
    struct viewporttype vp;

    setcolor( MaxColors - 1 );

    setlinestyle( SOLID_LINE, 0, NORM_WIDTH );

    getviewsettings( &vp );
    rectangle( 0, 0, vp.right-vp.left, vp.bottom-vp.top );

}

/*
/*   CHANGETEXTSTYLE: similar cu setttextstyle dar verifica */
/*   aparitia erorilor la incarcarea fontelor                */
/*
void changetextstyle(int font, int direction, int charsize)
(
    int ErrorCode;

    graphresult();          /* Sterge codul de eroare    */

```


Aplicații în TURBO C

```
settextstyle(font, direction, charsize);
ErrorCode = graphresult();          /* Verifica rezultatul */
if( ErrorCode != grOk ){            /* Daca a aparut vre-o eroare */
    closegraph();
    printf(" Eroare de sistem grafic: %s\n", grapherrormsg( ErrorCode ) );
    exit( 1 );
}

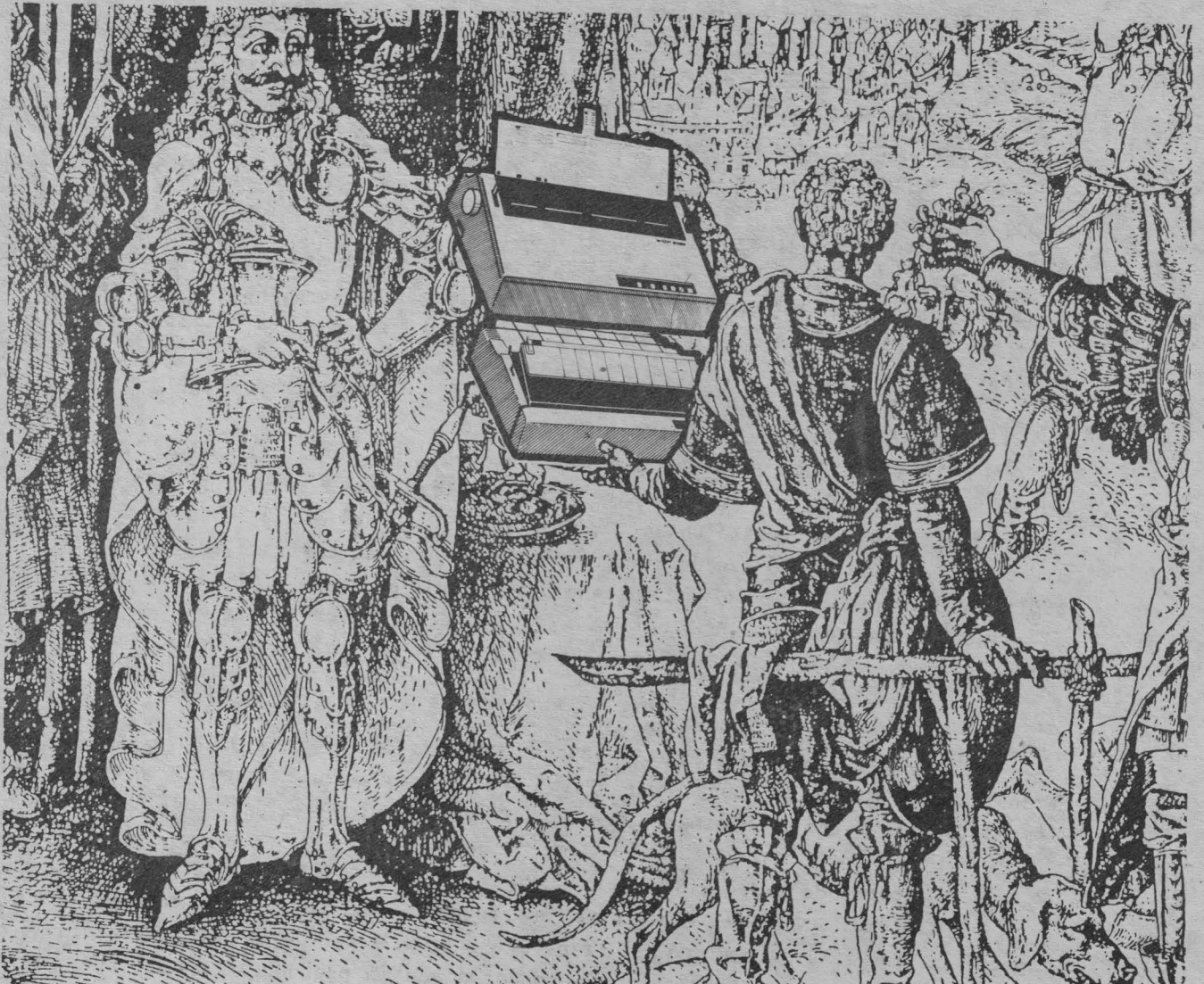
/*
/*   GPRINTF:Folosit ca PRINTF cu exceptia ca output-ul este
/*   trimis in ecranul grafic la coordonate specificate
/*
int gprintf( int *xloc, int *yloc, char *fmt, ... )
(
    va_list argptr;                 /* Pointer la lista de argumente*/
    char str[140];                  /* Buffer ptr sir */
    int cnt;

    va_start( argptr, format );    /* Initializeaza functia va_ */

    cnt = vsprintf( str, fmt, argptr ); /*Scrie sirul in buffer */
    outtextxy( *xloc, *yloc, str );  /* Trimite sirul in modul grafic */
    *yloc += textheight( "H" ) + 2; /* Inainteaza la linia urmatoare */

    va_end( argptr );              /* Incheie functiile va_ */

    return( cnt );
)
```



Inedit, noutăți, recenzii

Casetă back-up de 2,2 GB

Firma Archive UK Ltd a realizat sistemul MaxStream MS2200 pentru calculatoare personale MacIntosh care salvează pe o singură casetă de 8 mm pînă la 2,2 GB. Viteza de transfer este de 13,4 MB/min. iar sistemul este compatibil cu Appleshare, TOPS, 3COM și Novell 2.15. Software-ul de salvare oferă utilizatorului posibilitatea de a include orice număr de fișiere și grupuri de fișiere.

PC portabil

Firma sud-coreană Samsung a realizat PC-ul portabil, construit în jurul procesorului 80L286 de 12 MHz, cu pînă la 4MB RAM, 64KB EPROM. Oferă un ecran LCD cu rezoluție 480X480. Suportă standardul VGA - are hard disc de 20MB și floppy disc. Poate fi dotat și cu procesor matematic 287, cu cartelă modem de 2400 bps și cu software de transfer de fișiere Laplink III.

Stație de lucru cu procesor

Stația de lucru Microsystem 4025T a firmei Intel Corp: construită pe baza procesorului 486 μ P cu unitate de virgulă mobilă încorporată, memorie de 8MB. Sistemul de operare Unix V compatibil cu Xenix și sistemul X-Window. Au încorporate capacități de rețea cu Ethernet, TCP/IP și NFS. Rezoluția grafică este de 20 Megapixeli/s.

Monitor de înaltă rezoluție

Monitorul color CT-20 al firmei Ikegami Electronics oferă o rezoluție de 1280X1024 puncte. Are o frecvență de scanare orizontală de 48-60 KHz și 59-70 Hz pe verticală. Lățimea benzii video este de 100 MHz și aria de afișare activă este de 380X290 mm.

Printer/Plotter compatibil HP-GL

Firma olandeză Advanced Matrix Technology Inc a realizat un printer/plotter color cu matrice în puncte care emulează plotterul HP Desktop precum și o mare varietate de imprimante Epson IBM și Diablo. El poate primi desenele HP-GL direct din aplicații CAD sau de grafică fără a se utiliza software sau hardware suplimentar. Poate imprima pînă la dimensiuni A2 - din programe rulate pe PC, mini și mainframe sau pe stații de lucru CAD/CAM dedicate. Au o varietate de fonte, culori și grosimi de linii la alegere.

Translator din Pascal în C

Produsul P2C al firmei germane Laner & Wollvitz GmbH realizează translatarea modulelor sursă din Turbo Pascal în module sursă C pentru compilatoarele Turbo C, Microsoft C sau ANSI C.

Captator de cadre video

Sistemul SCI DASM-FGM al firmei americane Analogic Corp. este capabil să capteze în timp real date video, să transfere imagini la și de la calculatoarele gazdă și să afișeze aceste imagini în monocrom sau în pseudoculoși RGB. Acest

Inedit, noutăți, recenzii

sistem apare calculatorului gazdă ca un disc RAM conectat printr-o poartă standard SCSI a gazdei. Unitatea conține un modul de captare a cadrelor (Frame Grabber Module), alcătuit dintr-un digitizor video, componente de prelucrare și un software specific.

Dezvoltarea de sisteme expert

Produsul Joshua al firmei germane Symbolics GmbH conține un limbaj și un mediu de dezvoltare pentru construirea de sisteme expert. Joshua permite modificarea dinamică, incrementală în timp, a unei aplicații, pe măsură ce mediul se schimbă. Este o mașină inferențială cu părți reînlocuibile, permițând utilizatorului să redefinească comportarea sau să modeleze orice parte a sistemului. Joshua este simplu de modificat și extins, este rapid și compact.

Coprocessor paralel

Utilizând sistemul de operare MS-DOS sau PC-DOS astfel încât sistemul de fișiere și interacțiunea operator să fie transparentă pentru utilizatori, ProTran caracterizează un sistem de prelucrare paralelă bazat pe transputerul T800 al firmei americane Yarc Systems Corp. cu pînă la 40 MB RAM pe o singură placă AT și de la unul pînă la patru transputere pe placă, cu adaptor de legătură. Pot fi interconectate pînă la 16 transputere într-o configurație hipercuș modificată.

Conectare între PC IBM și MAC

Produsul MacTwin al firmei americane Emerald Technology Inc., reprezentat printr-o combinație puternică de hardware și software, permite, printr-o acțiune de mouse, acces la un PC S/3X sau AS/400. Printre funcțiile sale se remarcă un control mărit al sesiunii, ferestre, autocheie și o opțiune puternică de transfer de fișiere.

Unitate de disc optic reinscriptibil

Firma japoneză Mitsubishi Electric a realizat unitățile de disc optic reinscriptibile de 5.25" - ME-5E1. Unitatea de disc operează la viteza de 2400 rot/min cu o viteză de transfer de 7.4 Mbps. Discurile au o capacitate de 600 MB.

Unități pentru discuri optice

Proiectat pentru discuri optice de 5.25", unitățile VDS 5160 de la firma Vision Data System pot conține 20 de cartridges cu o capacitate de 16 GB. Utilizînd 14 unități interconectate se poate ajunge la o capacitate de memorie disc de 224 GB. Interfața integrată SCSI permite conectarea unității la PC/AT, VME sau MAC.

PS/2 portabil

Firma americană Leading Edge Products a realizat un PS/2 portabil cu următoarele caracteristici: 1-4 MB memorie principală, procesor 80386 SX, frecvența 16 Mhz, ecran cu LCD, VGA, hard disc de 40 MB, unitate floppy de 1,44 MB de 3.5", port paralel, serial și mouse precum și interfață pentru unitate floppy externă.

Accelerator pentru baze de date

Acceleratorul relațional al firmei americane Charles Riker Data Systems este o mașină de baze de date dedicată, proiectată pentru SCBD-ul relațional ORACLE pentru a ridica performanțele acestuia atît pentru prelucrarea on-line a

tranzacțiilor cât și în aplicațiile de suport a deciziilor.

Intrare prin voce

Produsul firmei americane Speech System Inc., Phonetic Engine 200, reprezintă o unitate periferică care acceptă intrarea în vorbire naturală continuă, independent de vocabular și furnizează o ieșire fonetică codificată corespunzătoare. Unitatea se conectează la o stație de lucru pe linie de comunicații.

PC stereo

Filiala americană a firmei Philips a realizat o adaptare a unităților CD-Rom la PC permițînd nu numai utilizarea discurilor compacte la stocarea de date și programe ci și ascultarea de muzică și nu oricum, ci stereo.

PC portabil de 2 kg

Firma Sharp a elaborat calculatorul personal PC6220, care cuprinde un hard disc de 20 MB cu dimensiunea de 6,35 cm, un ecran LCD de 25,4 cm diagonală, VGA, cu 16 nivele de gri. Format A4 cu 3,4 cm lățime.

Calculatoare Motorola

Firma americană Motorola și-a declarat intenția de a construi calculatoare pe baza procesorului Risc 88000. Aceste servere multiutilizator destinate aplicațiilor în rețea vor concura direct oferta R/6000 a firmei IBM. Puterea mașinii va fi cuprinsă între 17 și 60 MiPS. Microprocesorul are pentru moment frecvența de 25 MHz, dar versiunea de 33 Mhz va fi disponibilă în curînd. Mașinile vor lucra sub SO UNIX și vor fi interfațate în rețea TCP/IP și ETHERNET.

Dicționar de termeni

A

ACRTC - Advanced CRT Controller. Controler grafic care respectă specificațiile video de tip CRT.

C

CD-ROM - Compact Disc Read Only Memory. Născut în 1987, acest suport servește la difuzarea și căutarea de informații care necesită o mare cantitate de memorie (imagini, scheme, cataloage, enciclopedii etc.). Capacitatea sa de peste 600 MB permite construirea de baze de date voluminoase dar fără posibilitatea de a le actualiza. Principalii producători: Hitachi, Philips, Pioneer, Sanyo, Sony etc.

Client/Server - Un model de arhitectură în context rețea. Se disting aplicațiile de tip "client" și "serverul" de date. Serverul este un sistem de gestiune a bazei de date precum ORACLE, INGRES, INFORMIX. El se află pe un PC cu procesor 386 sau 486, sau pe un mini sau mainframe. Aplicațiile client sînt situate pe stații de lucru. Cînd un utilizator interoghează o bază de date dintr-o aplicație client cererea este transmisă într-o manieră transparentă la sever care o tratează și apoi răspunde clientului.

CISC - Complexed Instruction Set Computer. Procesor cu set complex de instrucțiuni.

CGA - Color Graphic Adapter. Rezoluție 640X200 cu 16 culori.

Cîmp calculat - Cîmp al cărui conținut depinde de cele ale altor cîmpuri.

Inedit, noutăți, recenzii

D

Disc optic reinscriptibil - Acest tip de disc, născut în aprilie 1989, este cunoscut pentru salvarea temporară a informației pe un disc dur. Discul optic poate fi șters și rescris utilizând o tehnologie laser. Capacitatea discului se ridică la 600 MB. Se poate scrie de circa un milion de ori în cadrul aceleiași zone. Principalii constructori sînt: Ricoh, Hitachi și Sony.

Disc optic neinscriptibil - Tehnologie apărută în 1985 și destinată arhivării de volume mari de date. Acest tip de disc nu permite decît o singură scriere dar care este garantată între 10 și 30 de ani. Capacitatea sa este de 800 MB.

E

Ediție Extinsă a OS/2 - Versiune a sistemului de operare OS/2 vîndută de IBM care integrează un motor de bază de date SQL (DB) și un modul de comunicație (DC).

EGA - Enhanced Graphic Adaptor. Rezoluție 640X350 cu 16 culori.

EISA - O specificație de bus de 32 biți realizată de Compaq care se depărtează de la norma MCA realizată de IBM. Specificația a fost adoptată de mai mulți constructori precum Zenith, Tandy, Goupil, Normerel etc.

P

PIXEL - Picture Element. Cel mai mic element al unei imagini informatice.

POUCE - 2,54 cm.

Q

QBE - Query by Example. Interogare prin exemplu. Sistemul permite extragerea de informații dintr-o bază de date indicînd un exemplu așteptat. Limbajul QBE a fost dezvoltat de Moshe Zloof la IBM. Limbajul are, în prezent, o largă răspîndire fiind implementat de PARADOX, QMF, d'Base IV, STAR etc.

R

Risc - Reduced Instruction Set Computer. Procesor cu set redus de instrucțiuni.

S

SQL - Structured Query Language. Limbaj structurat de interogare. SQL - limbaj standard pentru lucrul cu baze de date, dezvoltat inițial în cadrul proiectului R la IBM. Implementat de foarte multe sisteme de gestiune a bazelor de date relaționale: ORACLE, INGRES, INFORMIX, DB2, SQL/DS, d'Base IV etc.

V

VGA - Virtual Graphic Adaptor. Rezoluție 640X480 cu 256 culori. Modul a fost inaugurat de calculatoarele PS/2 IBM.

Pregatirea Hard Disk-urilor in MS-DOS

O discheta sau un hard disk noi nu sint pregatite pentru utilizare. Inainte de aceasta, ele trebuie supuse procesului de formatare, care inregistreaza pe disc anumite informatii DOS. Exista doue tipuri de formatari: *fizica* (de nivel scazut) si *logica* (de nivel inalt).

Formatarea fizica a unui disc inregistreaza pe acesta semnale magnetice care impart fiecare fata a discului in piste si sectoare. Aceste informatii vor fi utilizate de catre controlorul de disc pentru regasirea sectoarelor de pe disc. Sectoarele defecte sin marcate ca atare intr-o zona rezervata de pe disc. Firma IBM numeste acest tip de formatare *formatare absoluta*. Despre asta in numarul viitor, cind ne vom referi pe larg la formatarea fizica. Este recomandabila, uneori obligatorie, formatarea fizica a hard discului la schimbarea orientarii spatiale a acestuia.

Al doilea tip de formatare este *formatarea logica* (de nivel inalt). Prin formatarea logica, DOS inregistreaza pe disc informatia necesara pentru regasirea (evidenta) fisierelor, a spatiului liber/ocupat pe acel disc. IBM numeste acest tip de formatare *formatare relativa*.

Cunoasteti desigur comanda DOS FORMAT, care pregateste o discheta sau un hard disk pentru utilizare. Ceea ce multi nu stiu insa, este ca FORMAT trateaza dischetele si hard disk-ul in maniere total diferite. Pe o discheta, FORMAT executa atat formatarea fizica cit si formatarea logica. Pe un hard disk, FORMAT nu executa decit formatarea logica.

Hard disk-urile sint, in general, formate fizic de catre fabricant. Toate discurile IBM sint formate inainte de livrare. Alte firme ofera discurile (formate sau nu) impreuna cu programe care pot executa formatarea fizica. IBM nu ofera in general astfel de programe pentru discurile sale. Exista si programe comercializate separat, care suporta o gama larga de discuri (de genul Disk Manager). Despre toate acestea in numarul viitor.

In acest numar ne vom referi pe larg la formatarea logica. Formatarea fizica este rareori necesara; ea constituie un proces destul de complicat, pentru care este nevoie de programe speciale si experienta (sau cel putin asistenta) tehnica. Aproape toate hard disk-urile moderne sint formate fizic de catre fabricant. In schimb, formatarea logica este relativ simpla si poate fi facuta de oricine fara programe sofisticate. Redactia noastra ofera celor interesati asistenta tehnica in orice fel de probleme de acest gen.

Instalarea logica

Dupa formatarea fizica, un disc trebuie *partitionat*, inainte de a fi formatat logic. Partitionarea inseamna impartirea discului in mai multe zone fizice, nu neaparat toate de aceeasi dimensiune, fiecare vazuta ca un disc logic separat. Hard disk-urile pentru IBM-PC pot avea cel mult patru astfel de zone (impropriu spus, dar deja consacrat, patru *partitii*). Aceste partitii sint independente, ele pot avea lungimi diferite, iar in fiecare poate fi instalat un alt sistem de operare. Cel mult una dintre aceste partitii este activa. La pornirea calculatorului, se va lansa sistemul de operare continut in *partitia activa*. Pentru lansarea unui alt sistem de operare (daca exista) trebuie schimbata *partitia activa* si resetat calculatorul. Fiecare sistem de operare care poate rula pe IBM-PC are un program de partitionare; una dintre optiunile acestor programe este schimbarea *partitiei active*. Programul de partitionare in MS-DOS este FDISK.COM

Partitionarea cu FDISK

In continuare vom prezenta sumar modul de utilizare a programului FDISK. Vom presupune ca dorim sa partitionam numai primul disc fizic instalat in sistem (comenzile care trebuie tastate de catre operator sint subliniate).

(1) Porniti calculatorul si incarcati sistemul MS-DOS de pe discheta A. Asigurati-va ca ora si data sint corecte.

(2) Lansati FDISK:

A> FDISK

Programul va afisa un ecran de forma:

Fixed Disk Setup Program
FDISK Options

Choose one of the following:

1. Create DOS Partition
2. Select Active Partition
3. Delete DOS Partition
4. Display Partition Information

Enter choice: [1]

Press **Esc** to exit FDISK

(3) Pentru a crea o partitie DOS, selectati [1] si apasati ENTER.

In continuare, procesul de partitionare este destul de simplu. Urmariti indicatiile si mesajele afisate de program. Daca aveti intrebari, va stam la dispozitie.

Formatarea logica

Ultimul pas in pregatirea unui hard disk este formatarea sa logica. Aceasta se realizeaza cu programul `FORMAT.COM`, inclus in sistemul de operare DOS. Exista si alte programe care pot realiza acelasi lucru, dar recomandam cu tarie utilizarea programului `FORMAT` al DOS-ului. Nici un alt program (de genul Norton, PcTools, Disk Manager etc) nu indeplineste sarcini de sistem mai bine si mai sigur decat un program de sistem si insistam mult sa intelegeti acest lucru. Am observat aceasta tendinta de a folosi exagerat de mult programe "frumoase" de genul Norton Tools si exista persoane care, din aceasta cauza, nu stiu sa utilizeze comenzi elementare de sistem, precum `DIR` sau `COPY`. Este un lucru grav si vom reveni in paginile revistei.

Trebuie inteles ca `FORMAT` este un program DOS care formateaza logic partitii DOS. Daca discul contine si partitii pentru alte sisteme de operare, `FORMAT` nu are nici un efect asupra lor. In cazul in care sint mai multe partitii DOS, acestea se pot formata selectiv. Partitiile DOS sint "vazute" de catre sistem ca discuri logice separate, deci `FORMAT C:` va formata prima partitie DOS, `FORMAT D:`, a doua partitie si asa mai departe. Formatarea logica a unei partitii nu afecteaza in nici un fel datele din celelalte partitii, spre deosebire de formatarea fizica a intregului disc, care distruge irevocabil orice informatie de pe disc. `FORMAT` nu face decat sa initializeze zona sistem a partitiei specificate, zona ce contine: sectorul boot, FAT, directorul radacina. In plus, transfera sistemul de operare si marcheaza partitia respectiva drept *activa*, daca este folosita optiunea `/S`.

Zona de date a partitiei ramine intacta si exista chiar posibilitatea de a o recupera dupa o formatare accidentala. Pentru aceasta trebuie cunoscuta organizarea partitiei (continutul FAT si al directorului radacina) inainte de formatare. Exista multe programe utilitare care pot fi folosite pentru aceasta recuperare; nu este cel mai placut lucru, dar se poate face.

Revenind la formatare, aceasta se face prin comanda:

A> FORMAT C: /S /V

Desigur, în loc de C: putem folosi D: sau altceva, conform discuției de mai sus. Opțiunea /S determină copierea sistemului de operare (fișierele IO.SYS, MSDOS.SYS și COMMAND.COM) de pe discheta de pe care a fost dată comanda în partiția specificată și marcarea acesteia ca partiție *activă*.

Opțiunea /V va da posibilitatea plasării unei etichete pe partiția (discul logic) specificată. Această etichetă este utilă în identificarea discului (apare la comanda DIR); acest lucru nu pare prea util la prima vedere, dar într-o rețea de calculatoare este esențial, eticheta fiind folosită efectiv la identificarea discului respectiv. În terminologia DOS, o discheta sau o partiție pe un hard disk reprezintă un *volum*. Volumele sunt identificate în comenzile operator prin litere (A, B, C, ...).

În DOS 3.3, eticheta este folosită și pentru prevenirea reformatării accidentale a unui volum. Astfel, dacă volumul are deja o etichetă, FORMAT afișează mesajul:

Enter current label for volume C:

Operatorul trebuie să introducă eticheta curentă a volumului respectiv. Aceasta este o protecție în plus oferită de program.

FORMAT afișează apoi încă un mesaj, solicitând confirmarea operatorului înaintea unei operații periculoase:

WARNING!

All data on fixed disk C: will be lost!

Proceed with format? [Y/N]:

După ce răspundeți "Y" începe procesul de formatare. Deși nu modifică decât zona sistem, FORMAT testează întregul volum pentru a marca în tabela de alocare a fișierelor (FAT) sectoarele defecte; de aceea, procesul poate dura câteva minute.

După terminarea formatarei se afișează două mesaje:

Formatting complete

System transferred

Al doilea mesaj apare, desigur, numai dacă s-a folosit opțiunea /S. După câteva secunde apare mesajul:

Enter volume label (11 characters) ENTER for none:

Introduceți un șir de cel mult 11 caractere, apoi apăsați tasta ENTER. Caracterele permise în etichetă sunt alfanumerice, spații, sau oricare din următoarele:

()- _ ~ {} [] , ! @ # \$ % &

În sfârșit, este afișat un mesaj de informare privind spațiul liber de pe volumul respectiv.

Procesul de formatare este încheiat. Se poate acum testa dacă totul este în regulă, resetând calculatorul (cu Ctrl+Alt+Del sau apăsând butonul RESET) fără discheta în unitatea A:. Dacă totul este în regulă, sistemul de operare trebuie să se lanseze de pe hard disk.

După cum vedeți, prepararea unui hard disk nu este un lucru foarte complicat. În numărul viitor vom prezenta formatarea fizică și un program mult utilizat în acest scop, Disk Manager.

PRETURI ORIENTATIVE

pe piata microcalculatoarelor
la inceputul anului 1990

RETELE

Novell Advanced V 2.15	\$1799
Novell NE 1000 Ethernet Workstation Board	179
Novell Netpro	1095
ARC Net PC 130	135
ARC Net PC500WS	279
Western Digital Ethernet Card	229

COMPAQ

386 ZDE 40mb	\$4395
286 Desk Pro 12 mhz	4995
Compaq SLT w/20	3895
Compaq SLT w/40 mb	4349

IBM

PERSONAL SYSTEM/2
intre \$2829 si \$7999

COPROCESOR

8087-3	99	80387-16	339
8087-2	125	80387-20	379

MONITORE

NEC Multisync II A	\$499
IBM 8513 VGA	579
PRINCETON MAX12	149

COMPATIBILE PC

10 MHZ TURBO	
1 Floppy, 640k, tastatura tip AT, placa monografica, monitor mono	\$699
plus:	
pentru sistem color	\$150
pentru w/20	\$249
386 20MHZ Portabil	
1 mb RAM, 40 meg hard disc	\$2799

LAP TOP

Panasonic CF150	\$645
Toshiba 1000	639
Toshiba 1600	3250
Zenith Super Sport 286	3025

IMPRIMANTA

EPSON	\$189-\$899
DICONIX	339
OKIDATA	229-1069
TOSHIBA	379-939
HEWLETT-PACKARD	239-2895
CANON	719-1795

MICE & SCANNER

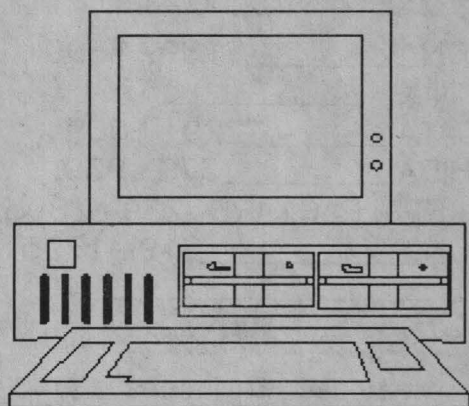
Microsoft Mouse w/windows	\$145
Mousesystems Mouse	89
Logitech Scan Man	185

PRETURI ORIENTATIVE

pe piata microcalculatoarelor
la inceputul anului 1990

SOFTWARE

Aldus Pagemaker 3.0	\$489
Clarion ProDeveloper	389
Clipper	419
D Base IV	419
Enable O/A	449
Freelance Plus	329
Framework III	449
Harvard Graphics	279
Lotus Networker	1595
Microsoft Word 5.0	215
Super Calc V 1.0	299
Symphony 2.0	419
Ventura Dektop Publ.	529
Word Perfect 5.0	239
Wordstar Pro 5.5	209



A simula? Nimic mai simplu!

În prezent un număr din ce în ce mai mare de jocuri sunt dezvoltate de către informaticienii începători. În timp ce unii entuziaști dezvoltă sisteme computerizate dintre cele mai sofisticate, mulți alții continuă să-și folosească calculatoarele ca pe o unealtă de recreere. Vom încerca prin intermediul acestei rubrici să venim în întâmpinarea dorințelor lor. Rămâne de văzut în ce măsură vom face din aceasta un obicei.

Pentru început vă invităm "La pescuit". Vom încerca să simulăm împreună scenariul următor: Totul începe într-o frumoasă zi de vară la mare. Aceasta este reprezentată printr-o grilă 8x8, cu 64 de locații de pescuit ca în figura 1.

Deplasarea bărcii se face începând cu locația (1,1) care reprezintă docul de plecare. Scopul dumneavoastră este de a prinde o cantitate cât mai mare de pește. La un moment dat vă puteți deplasa orizontal sau vertical cu o singură casuță printr-una din direcțiile: nord (N), sud (S), est (E) sau vest (V). Tastând (P) obțineți dreptul de a pescui din nou în același loc, iar comanda (O) vă permite să începeți o nouă sesiune de ... pescuit.

Dacă selectați o direcție care vă va scoate în afara grilei, barca dumneavoastră va naufraga. Sînteți obligat să vă întoarceți la doc în 60 de minute, echivalentul a 60 de ore de pescuit. Dacă nu vă întoarceți la timp pierdeți jumătate din încărcătură. Șansa de a prinde pește este diferită pentru fiecare pătrat și este determinată la începutul excursiei pe mare. Aceasta probabilitate rămâne fixă pe parcursul călătoriei sau va scădea dacă peștele este speriat de rechini.

Numărul maxim de pești care poate fi prins în fiecare pătrat (densitatea) este de asemenea determinat la începutul simulării. Acest număr este cuprins între 1 și 5. Numărul maxim de pești pe care-i puteți prinde într-un pătrat va scădea doar dacă unii dintre ei sînt mincați de pescarusi. Greutatea maximă a unui pește într-un anumit pătrat este produsul dintre rînd și coloană, de aceea cu cât vă deplasați mai departe, cu atît peștele este mai mare.

Pe de altă parte, pe măsură ce trece timpul, cresc șansele de apariție ale unei furtuni de amiază, frecventă în marile în care dumneavoastră obișnuiți să pescuiți. Dacă veți întâlni o furtună, veți pierde 0,5 ore. Una dintre cele mai dificile manevre ale călătoriei este de a pescui suficient de mult, evitînd furtuna.

De asemenea, există șansa de 4% de a vi se întimpla un eveniment neașteptat în timpul oricărei mișcări. Asigurați-vă întoarcerea la doc înainte de scurgerea celor 6 ore.

Indicele care vi se va atribui ca pescar, este dat de numărul kilogramelor de pește pescuit împărțit la 5. Pentru durata jocului vă sugerez folosirea unei grile desenate pe hîrtie, pentru a înregistra cele mai bune locuri de pescuit. Puteți marca de asemenea locația în care vă găsiți la un moment dat.

Iată cum va arăta o sesiune de lucru:

RUN

NU MUSCA,
ÎN LOCATIA 1 1,
TOTAL KG ESTE 0.
PESCUITI DE 0 ORE.
DEPLASARE (N,S,E,V,P,O)? E

NU MUSCA,
ÎN LOCATIA 1 2,
TOTAL KG ESTE 0.
PESCUITI DE 0.1 ORE.
DEPLASARE (N,S,E,V,P,O)? S

ATI PRINS 2 PESTI,
FIECARE CINTARIND 2 KG.
ÎN LOCATIA 2 2,
TOTAL KG ESTE 4.
PESCUITI DE 0.2 ORE.
DEPLASARE (N,S,E,V,P,O)? S

NU MUSCA,
ÎN LOCATIA 3 2,
TOTAL KG ESTE 4.
PESCUITI DE 0.3 ORE.
DEPLASARE (N,S,E,V,P,O)? E

ATI PRINS 4 PESTI,
FIECARE CINTARIND 2 KG.
ÎN LOCATIA 3 3,
TOTAL KG ESTE 12.

PESCUITI DE 0.4 ORE.
DEPLASARE (N,S,E,V,P,O)? E

...

NU MUSCA,
IN LOCATIA 4 6,
TOTAL KG ESTE 12.
PESCARUSII MANINCA DIN MOMEALA,
ASTAZI VETI PRINDE MAI PUTIN.
PESCUITI DE 0.8 ORE.
DEPLASARE (N,S,E,V,P,O)? S

...

ATI PRINS 4 PESTI,
FIECARE CINTARIND 15 KG.
IN LOCATIA 4 8,
TOTAL KG ESTE 155.

...

ATI PRINS 1 PESTE,
FIECARE CINTARIND 3 KG.
IN LOCATIA 3 3,
TOTAL KG ESTE 208.
VALURILE V-AU DEPLASAT.
ACUM SINTETI IN LOCATIA 4 5.
PESCUITI DE 2.6 ORE.
DEPLASARE (N,S,E,V,P,O)? V

...

NU MUSCA,
IN LOCATIA 1 2,
TOTAL KG ESTE 211.
PESCUITI DE 3.2 ORE.
DEPLASARE (N,S,E,V,P,O)? V

ATI REVENIT LA DOC,
DUPA 3.2 ORE DE PESCUIT,
CU 211 KG DE PESTE.
RATA DE PESCUIT ESTE 42.

Programul "La pescuit". Variabile

P (I,J) Probabilitatea de a prinde peste in careul I, J.

D (I,J) Numarul maxim de pesti in patratul (I,J), intre 1 si 5.

G Greutatea fiecarui peste, intre 1 si Rx C.

K Numarul total de kilograme de peste prins.

R Rindul curent.

C Coloana curenta.

N Numar de pesti prins la un moment dat.

T Timpul in zecimi de ora, maxim 6 ore.

M\$ Miscarea (N,E,S,V,P,O), unde N, E, V, S sint directii, P va permite sa pescuiti din nou in acelasi patrat si O reia jocul de la inceput.

Modificari posibile

Modificari minore

- Marimea grilei - liniile 10, 20, 440, 630, 720, 810 si 820.
- Probabilitatea maxima de a prinde peste intr-un patrat - linia 30.
- Densitatea maxima a pestelui intr-un patrat - linia 40.
- Timpul maxim de pescuit - linia 150.
- Probabilitatea unei furtuni - linia 330.
- Rata de pescuit - linia 540.

Modificari majore

- Puteti prinde alte tipuri de pesti: rechini, balene, etc.
- Schimbati scopul in acela de a prinde pestele cel mai mare.
- Aducati motor la barca.
- Succesele la pescuit sa depinda de ora din zi.
- Aducati alte tipuri de hazard, cum ar fi balene, OZN-uri etc.
- Fixati conditii de vreme si pescuit la inceputul excursiei pe mare.
- Utilizati sonare care sa va ajute in localizarea bancurilor de pesti.
- Permeteti barcii sa se deplaseze pe diagonala.

In continuare, va prezentam listingul programului *La pescuit*.

A simula? Nimic mai simplu!

```
5 REM INITIALIZARE PROBABILITATI SI DENSITATE
10 DIM P(8,8),D(8,8)
20 FOR I=1 TO 8: FOR J=1 TO 8
30 P(I,J)=.7*RND(1)
40 D(I,J)=INT(RND(1)*5+1)
50 NEXT J,I
60 P(1,1)=0: K=0: R=1: C=1

145 REM BUCLA PRINCIPALA
150 FOR T=0 TO 6 STEP .1
160 IF RND(1) > P(R,C) OR D(R,C) < 1 THEN PRINT "NU MUSCA.": GO TO 220
170 N=INT(RND(1)*D(R,C)+1)
180 G=INT(RND(1)*R*C)+1
190 K=K+N*G
200 PRINT "ATI PRINS"; N; "PESTI,"
210 PRINT "FIECARE CINTARIND"; G; "KG.,"
220 PRINT "IN LOCATIA"; R; C; ","
230 PRINT "TOTAL KG ESTE"; K; "."

325 REM EXPERIENTE NEASTEPTATE
330 IF RND(1) < T/60 THEN PRINT "FURTUNA -- ATI PIERDUT 1/2 ORA": T=T+.5
340 J=INT(100*RND(1))+1
350 IF J > 4 THEN 370
360 ON J GO SUB 600,700,800,900

370 PRINT "PESCUITI DE "; T; "ORE."
380 INPUT "DEPLASARE (N,S,E,V,P,O)"; M$
390 IF M$="E" THEN C=C+1
400 IF M$="N" THEN R=R-1
410 IF M$="V" THEN C=C-1
420 IF M$="S" THEN R=R+1
430 IF M$="P" THEN RUN
440 IF R < 1 OR R > 8 OR C < 1 OR C > 8 THEN PRINT "ATI ESUAT!": GO TO 550

450 IF R=1 AND C=1 THEN GO TO 500 460 NEXT T

470 PRINT "A TRECUT TIMPUL. SOARELE E LA APUS."
480 PRINT "JUMATATE DIN PRADA VI SE CONFISCA."
490 P=P/2

495 REM SUMARUL EXCURSIEI
500 IF T=0 THEN PRINT "SINTETI INCA LINGA DOC": GO TO 10
510 PRINT "ATI REVENIT LA DOC,"
520 PRINT "DUPA"; T; "ORE DE PESCUIT,"
530 PRINT "CU "; K; "KG DE PESTE"
540 PRINT "RATA DE PESCUIT ESTE "; INT(K/5)
550 INPUT "DORITI SA PESCUITI DIN NOU(D,N)"; X$
560 IF X$="D" THEN RUN
570 END

595 REM SUBRUTINE
600 IF R+C < 9 THEN RETURN
610 PRINT "RECHINII AU SPERIAT PRADA."
620 PRINT "CONTINUA SA NU MUSTE."
630 FOR I=1 TO 8: FOR J=1 TO 8
```

A simula? Nimic mai simplu!

```
640 P(I,J) = P(I,J) - 1
650 NEXT J,I
660 RETURN
```

```
700 PRINT "PESCARUSII MANINCA DIN MOMEALA."
710 PRINT "ASTAZI VETI PRINDE MAI PUTIN."
720 FOR I = 1 TO 8: FOR J = 1 TO 8
730 D(I,J) = D(I,J) - 1
740 NEXT J,I
750 RETURN
```

```
800 PRINT "VALURILE V-AU DEPLASAT."
810 R = INT(8*RND(1) + 1)
820 C = INT(8*RND(1) + 1)
830 PRINT "ACUM SINTETI IN LOCATIA"; R; C
840 T = T + .2
850 RETURN
```

```
900 PRINT "ATI PRINS UN RECHIN DE 50 KG."
910 K = K + 50
920 PRINT "CU "; K; "KG DE PESTE."
930 RETURN
```

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

Figura1

C U P R I N S

INTELIGENȚA ARTIFICIALA	G R A F I C A
<p>Gîndesc oare calculatoarele? 1-3</p> <p>Fundamente ale programării logice în inteligența artificială (II). Componente de bază în programarea logică 4-14</p>	<p>Editoare și fonturi (I) 45-49</p> <p>Inedit, noutăți, recenzii. 50-52</p> <p>Aplicații grafice pe calculatoare personale 53-56</p>
LIMBAJE DE PROGRAMARE	G H I D
<p>Limbaje de programare în designul sistemelor de operare și al aplicațiilor. Limbajul C (II) 15-19</p> <p>Designul și exploatarea foilor de calcul electronice (II). Tipuri de date Lotus. 20-22</p> <p>Tehnici software în PROLOG și ASSEMBLER pentru microsistemele PC (I). Exploatarea microsistemelor prin limbajul PROLOG și ASSEMBLER 23-38</p>	<p>Ghid de probleme rezolvate pentru elevi și studenți.</p> <p>Structuri 57-59</p> <p>Aplicații în limbajul TURBO C 60-65</p>
C O M U N I C A Ț I I	I N E D I T, N O U T A Ț I
<p>Inițiere în conceptele comunicației de date (I) 39-40</p>	<p>Inedit, noutăți, recenzii.</p> <p>Noutăți 66-69</p> <p>Pregătirea hard disc-urilor în MS-DOS 70-72</p> <p>Prețuri orientative la începutul anului 1990 73-74</p>
C O M U N I C A Ț I I	A M U Z A M E N T E
<p>Inițiere în conceptele comunicației de date (I) 39-40</p>	<p>Jocuri și amuzamente. A simula? Nimic mai simplu! 75-78</p>

ADISAN

INTREPRINDEREA ASOCIATIEI SPECIALISTILOR
IN BAZE SI BANCII DE DATE
STR.TURDA NR.114 SC.B AP.71 TEL.660476
BUCAREST/ROMANIA

CURIER EDITORIAL

1. ABONAMENTE

La acest prim punct al curierului editorial, ne anuntam cititorii ca actiunea de inscriere pentru abonamente la revista "PC MAGAZIN" a fost sistata temporar din considerente economice. Acest fapt nu afecteaza abonamentele deja existente care ramin in vigoare.

2. PROBLEME EDITORIALE

Din cauza dificultatilor tipografice, inerente acestei perioade, apartinea urmatorului numar din "PC MAGAZIN" poate fi intirziata cu 10-14 zile. Ne cerem scuze pe aceasta cale pentru acest lucru independent de vointa noastra. In acelasi timp, colegiul nostru redactional depune toate eforturile pentru ca numerele viitoare ale revistei sa apara intr-o tinuta grafica exemplara.

Incepind cu numarul viitor revista noastra va contine alaturi de numeroase articole cu un bogat continut stiintific, un larg spatiu publicitar.

Oferim pe aceasta cale cititorilor nostri posibilitatea de a-si face cunoscute propriile produse, realizari, servicii informatice si proiecte de dezvoltare software.

Asteptam sugestiile si ofertele dumneavoastra.

SEDIUL NOSTRU PROVIZORIU:
str. ION MINCU nr.11, sector 1, BUCURESTI.

We provide services of installation, technical assistance and maintenance for the following software products:

- o XEROX Ventura Publisher, desktop publishing software
- o Word Perfect Version 5.10, word-processor
- o WordStar Version 3.00, 4.00, word-processors
- o Deluxe Paint II, paint program by Electronic Arts
- o Paintbrush, paint program by Z-Soft
- o AutoCad version 10.00, drawing program
- o Microsoft Windows, operating environment
- o MS-DOS, OS/2, PC/MOS 386, operating systems
- o DBMSs (dBase III+, Clipper, SQL Server, FoxBase etc)
- o Lotus 1-2-3, Quattro, spreadsheets
- o MathLab, MathCad, muMath, Eureka, REDUCE, scientific software
- o INSET, text/graphics integrator
- o compilers (Borland, Microsoft etc), DOS utilities etc
- o others, please specify.

We provide assistance in using existing applications and/or developing software for the following peripherals/printers/plotters:

- o CGA, Hercules, EGA, VGA, MCGA or compatible video adapters
- o All types of IBM, Epson, Okidata and Toshiba printers
- o Hewlett-Packard laser and ink-jet printers
- o Hewlett-Packard and compatible plotters



grifon

trei publicații în trei domenii de
maxim interes
în curînd triunghiul

PANORAMIC

panoramic
RC

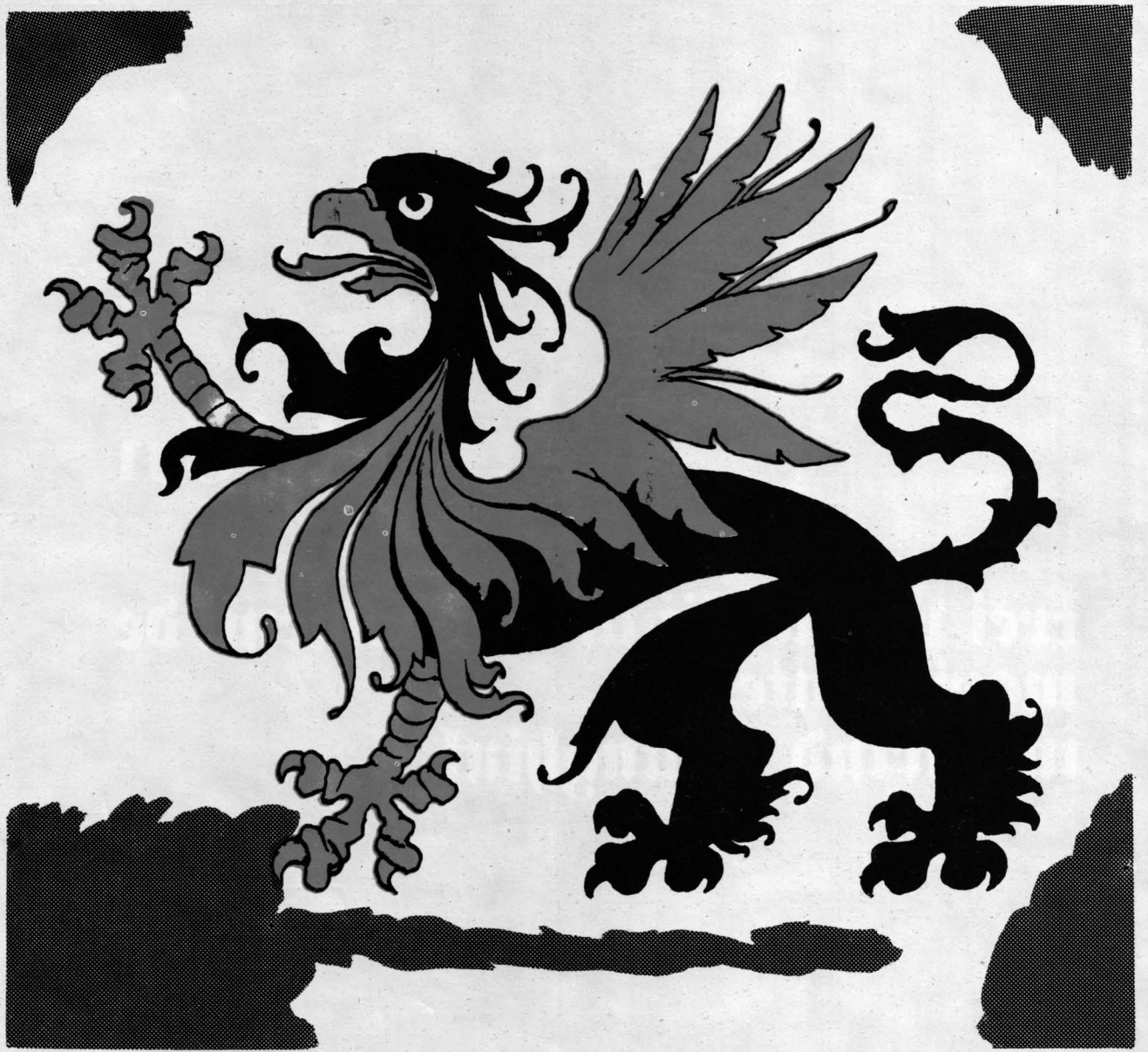
reclamă &
consulting
pentru toți
utilizatorii
de computere

panoramic
A

artistic
o imagine
a
artei plastice
românești

panoramic
LF

linii & forme
în
moda
de ieri
și de azi



g r i f o n

advertising & consulting agency
tel. 79 97 67 ; 11 19 20

reclamă, afis, pliant, ambalaj, siglă, gravură, editură, advertisement, posting, folder, publishing house