

if

**CALCULATOARE
PERSONALE**

4 / 92

(15)

150 Lei

ISSN 1220-1529

Revistă editată de Micro ATCI S.R.L. Tîrgu Mureş



Ultima oră!

S-a înființat POLICOM S.A., societate comercială multinațională cu capital privat, avînd principal acționar Ab Mod S.N.C.

Noua firmă vă oferă un sortiment larg de produse:

- Calculatoare personale compatibile IBM
- Imprimante matriciale
- Dispozitive grafice
- Floppy - dischete
- Produse software originale
- Copiatoare Toshiba
- Truse și accesorii pentru întreținere

Nu uitați!

POLICOM S.A. un partener potențial pentru afacerile dumneavoastră.

CLUJ: B-dul 22 Decembrie nr. 135 tel. 95-156350

Noutăți		Actualizări distribuite	30
		Transparență	31
		Cele douăsprezece porunci pentru baze de date distribuite	31
Noutăți AST			6
Club BORLAND			6
Noutăți DELL			6
Noutăți EPSON			8
Borland C++ 3.0			8
OS/2 V 2.0			8
Vă prezentăm			
IBM Personal System/1 Pro 386SX			9
Descriere			9
Cerințe de programare			9
Compatibilitate			9
Limitări			10
Produse suportate			10
Compatibilitate software			10
Multitasking			
Multitasking Cooperativ în C++			11
Multitasking cooperativ și multitasking forțat			11
Programul			11
Comutarea contextului			11
Obiecte task			12
Comutare			12
Programul demonstrativ			13
Concluzie			13
Rețele			
PC-urile ca sisteme de achiziție de date în rețea			18
Rețele distribuite			19
Genesis - în rețele distribuite			19
LT/Control în rețele distribuite			19
Rețele locale ierarhice			20
LT/Control în rețele ierarhice			20
Genesis în rețele ierarhice			20
Baze de date			
FOXPRO 2.0 - Tehnologia Rushmore			24
Impactul tehnologiei Rushmore asupra stilului			25
Pe ce dăm banii?			25
Cîteva obiecții			27
Rushmore în Rețea			27
Indecși în general			28
Concurență			28
Concluzii			28
BAZE DE DATE DISTRIBUITE			29
Citiri și actualizări la distanță			29
Baze de date distribuite			29
Interogări distribuite			30
Grafică			
O privire asupra graficii animate			32
Imagini fotorealiste			32
Umbrirea costă timp			33
Ray-tracing-ul nu este un leac universal			34
Este necesară răbdarea			34
Laborator			
Programare VGA - Animație			36
BANNER			40
Cursuri			
Cîteva probleme de logică rezolvate și comentate în limbalul TURBO PROLOG V2.0			42
PROBLEMA 1:			42
PROBLEMA 2:			44
PROBLEMA 3:			45
Fundamente			
Metode de regăsire cu multiatribut:			48
1. Modalități de folosire a metodei semnăturii			48
2. Metode de proiectare a fișierului semnătură și de extracție a semnăturilor			50
3. Comparație între metode.			51
Hardware			
MODULELE MULTICIP			53
Introducere			53
Noua soluție - MCM			54
Perspective			56
MCM avansate			56
Practică			
PLAY.ASM			58
Program autoreproducător			59
Meniuri verticale în PARADOX			60
Mai multe comenzi într-o linie DOS			62
Mulțime Mandelbrot			63
Rubrici			
SHARE - if			65
Editorial			5

... nu-*é* nimic, zic eu. Încerc sistemul de pe drive-ul A, la rece", asta pentru siguranță, după care încerc din nou o citire pe hard. Iarăși eșuează. E clar, nu e virus. Atunci ce e, mă întreb. E, bine, iubito, știi pînă la urmă cine era de vină? CMOSUL!, dragă, CMOSUL!!



if

revistă de informatică
editată de firma Micro ATCI

Director: ing. Dumitru Dunca

Redacția:

Redactor:

ing. Cristian Nagy

Secretariat redacție:

Doina Ceșa

Colaboratori:

Antonescu Valentin - caricatura
mat. Cozac Nelu

ing. Iacob Ioan

ing. Kallo Tibor

ing. Guțică Mirela

ing. Maier Romulus

ing. Malide Cristian

ing. Pantea Mircea

Tiparul: Tiporex S.R.L.

Tiraj: 8.000 ex.

Preț: 150 lei

Manuscrise originale sau listin-guri de programe sînt primite cu plăcere de redacție, cu condiția să nu fi fost publicate și în altă parte. Prin expedierea unui manuscris pe adresa redacției, autorul consimte implicit la publicarea materialului său în cadrul revistei. Onorariul se negociază cu redacția. Materialele nepublicate nu se înapoiază și nu se rețin.

Revista noastră vă oferă spațiu pentru reclamă și publicitate. Pentru amănunte vă rugăm să luați legătura cu redacția.

Cei care doresc să anexeze revistei pliante publicitare tipărite în regie proprie, sînt rugați și ei să se adreseze redacției.

Adresa și telefonul redacției:

Micro ATCI, redacția "if",

RO-4300 Tîrgu Mureș,

C.P. 64, O.P. 1,

tel./fax 954/31660

Despărțire

Îi anunțăm pe cititorii noștri fideli că în redacția revistei "if" s-a produs o importantă schimbare. Colectivul redacțional care s-a ocupat de conceperea și editarea acestei reviste, adică doamna Ingrid Maier, domnul Romulus Maier și domnul Iosif Fettich, au fondat o nouă firmă - HotSoft și nu mai lucrează în cadrul firmei Micro ATCI, nici la redacția "if".

Cei care doresc să mențină legătura cu ei o pot face la adresa: HOTSOFT C. P. 172, O.P. 1, Tîrgu Mureș sau la unul din telefoanele: 954 - 41882, 41417.

Noul redactor le mulțumește pentru ținuta și prestigiul pe care le-a dobîndit revista "if" datorită eforturilor depuse de ei încă de la primul număr.

Mesaj

"Grea misie, misia de ... redactor" - mai ales atunci cînd este vorba de un nou redactor (nou numai pentru cititorii vechi ai revistei). Grea, mai ales privit prin prisma prestigiului dobîndit de "if" de-a lungul timpului. Ar trebui ca acest prestigiu să nu aibă de suferit, iar cel care se va ocupa de acum de revistă este decis să îl mențină.

Chiar dacă acest număr nu pare a fi chiar cea mai fericită ilustrare a bunelor mele intenții, vă rog să nu vă descurajați. Eventual puteți încerca să-l priviți ca pe un număr de "tranziție". Și cum orice tranziție originală se face cu unele mici sacrificii, mai amînați puțin aluziile la demisie.

În altă ordine de idei, o revistă care să fie citită de mulți informaticieni trebuie să fie scrisă de mulți informaticieni, adică și de către cei care o citesc. Sînt convins că mulți dintre cei care urmăresc revista noastră ar avea ceva important de adus la cunoștința colegilor de breaslă. Nu ezitați să o faceți! Dacă bănuieți că eforturile dumneavoastră de a pune la punct o aplicație sau de a face să funcționeze un program rebel v-au condus la rezultate ce i-ar interesa și pe alții, scrieți-ne.

O notă specială pentru cei nemulțumiți: dacă există ceva care vă deranjează în revista noastră, nu ezitați să ne scrieți. Critica este foarte utilă, chiar dacă nu constituie totdeauna cea mai plăcută surpriză.

În general, am fi foarte interesați să știm ce vă place și (eventual) ce nu vă place în "if", ce ați mai dori să găsiți în paginile noastre.

Cristian Nagy

Noutăți AST

Firma americană AST Research anunță două noi apariții:

- ▣ Un notebook AST Premium Exec 386SX/25C realizat în tehnologie FSTN (Film Supertwist Nematic Technology).
Caracteristici: procesor AMD 386SXL-25, 4MB RAM, floppy 3,5", hard disk de 60 sau 80 MB, display VGA color (640x480 puncte cu 16 culori), preț orientativ 6000\$. Noul notebook a primit premiul "Innovation of the Year" în martie 1992, la Hanovra.
- ▣ Un sistem de transmisie - pe linie telefonică normală - a imaginilor video fixe numit "Photophone Image Communication System". Sistemul este deja utilizat: completarea și actualizarea bazelor de date complexe (Royal Canadian Mounted Police), asistență tehnică complexă la distanță (Pratt & Whitney, Hughes Aircraft).

CLUB

O știre de ultimă oră:

Firmele LOGIC din Sibiu și ROMSOFT din București au format un grup de inițiativă având ca scop fondarea primului Club al utilizatorilor de produse Borland.

Cei interesați sînt rugați să adreseze întrebările sau sugestiile privind organizarea și funcționarea clubului, problematica și statutul, cota de participare pe adresa domnului Gavrilescu Gavril, Centrul Național de Difuzare a produselor Informatică - ROMSOFT S.A., bul. Mareșal Averescu nr. 8-10, sect. 1, București, tel/fax: 90 - 666229.

(LOGIC News)

DELL COMPUTER

A lansat, în luna mai 1992, calculatorul 450 DE/2 DG. Noul sistem este echipat cu procesor 80486 la 50 MHz, procesor grafic, 96Mb RAM, 1.4 Mb HDD, 2 Gb tape backup. Acest sistem este realizat în tehnologia numită "Processor - Direct Graphics" dezvoltată de DELL împreună cu INTEL. Avantajul cuplării directe a sistemului grafic constă în evitarea "gîturirii" transferurilor pe magistrala EISA, oferind semnificative performanțe grafice la un cost incredibil de redus. De asemenea, realizarea cu ASIC (Application Specific Integrated Circuit) a controloarelor de memorie și I/O îmbunătățește atât performanța cît și fiabilitatea calculatorului.

EPSON- Tehnologie care impune

Cuplarea imprimantelor la Host prin cablu coaxial

EPSON pune la dispoziția noilor imprimante matriciale, cu jet de cerneală și laser, interfețele din seria B, care pot fi montate într-un singur conector al imprimantei. Interfața pentru cablu coaxial permite emularea imprimantelor IBM 3287, 3262, 3268, 4214 și 4224 și dispune de un test al funcțiilor logice ale imprimantei, de intrare de la PC pentru Intelligent Printer Sharing și de funcții pentru tipărirea automată a șirurilor de caractere.

Pentru o adaptare optimă la sistem, interfața mai dispune de seturi de caractere specifice pentru 24 de țări, posibilitate de configurare de la Host sau de la PC, Transmission - și Printer - Dump, precum și de patru funcții Bypass (utilizarea directă a proprietăților imprimantei).

Cuplare imprimantelor la Host prin cablu Twinax

Interfața Twinax #C82D42 permite cuplarea noilor imprimante EPSON la Host-uri IBM/3X și AS/400. Sînt emulate imprimantele IBM 5256, 5224, 5225 și 4210/14/-24. Interfața oferă aceleași facilități ca și cea pentru cablu coaxial, prezentată mai sus.

(EPSON - Germania)


Port paralel mai rapid

După părerea lui Bill Cott, directorul programului de cercetare a pieței de imprimante pentru InfoCorp/Computer Intelligence, viteza sporită a portului paralel de imprimantă este rezultatul unui utilitar soft care "sare" peste BIOS-ul PC-ului. Utilitarul va fi oferit de Microsoft ca un produs de sine stătător, dar Gott prezice că fabricanții de PC-uri îl vor include în viitoarele lor versiuni de BIOS.

Driver-ele software bidirecționale vor elimina o sursă de necazuri ale utilizatorilor - trimiterea de caractere la imprimantă și descoperirea, mult mai târziu, a faptului că imprimanta nu are hîrtie sau toner. Un astfel de driver va permite imprimantei să afișeze informația pe ecranul utilizatorului în loc să trimită numai un mesaj criptic de genul "Printer error".

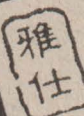
Mesajele de eroare a imprimantei mai lungi și mai detaliate îi vor ajuta pe utilizatori să diagnosticheze problemele de tipărire "deoarece trăim în rețea, (și) imprimantă nu este amplasată fizic (aproape de) persoana care efectuează listarea".

(PC Week)



We draw on
your
imagination

A Bold New Shape.
A Fast New Pace.

**A & C**
INTERNATIONAL S.A.
AUTHORIZED DEALER
Tel: 40-0-53.53.15
Service - "Hot Line": 40-0-12.77.73
Fax: 40-0-12.77.74
Telex: 10674 MAXIM R
Constantin Tanase Str.
No. 15, Sect. 2,
Bucharest, 73299
Romania

...smooth. Our
...thanks to
...ty and
...d
Take a bold first step to reshape
plotting at your firm. Find out more
today: Call **800-932-1212**. In Canada,
call 416-635-9010. We're expecting to
hear from you.

**CalComp**
A Lockheed Company

Microsoft și HP se unesc pentru îmbunătățirea listării în LAN

Microsoft Corp. și Hewlett - Packard Co. au format o echipă comună într-un efort de a da formă viitorului listării în rețea.

Cele două firme dezvoltă un software ce va "împinge" rata de ieșire a unui port paralel de la 150 Kb/sec. la 1 Mb/sec., precum și driver-e care permit imprimantei să comunice anumite situații de eroare, de exemplu că a rămas fără hîrtie.

În plus, Microsoft definitivează un cartuș pentru imprimantele HP Laserjet seriile II și III care va include un driver de imprimantă GDI (Graphics Device Interface) pentru Windows.

Cartușul va permite aplicațiilor Windows să tipărească documentele utilizînd comenzi GDI în loc să le convertească în limbajul de comandă "nativ" al imprimantei. Se estimează că eliminarea acestui pas va mări viteza de listare a lui Windows de mai mult de zece ori.

Ideea GDI a Microsoft - ului este cea mai nouă tentativă de a-și așeza tehnologia pe poziția de rival al PostScript-ului lui Adobe Systems Inc, urmînd eșecului înregistrat de limbajul de descriere a paginii True-Image al lui Microsoft.

Atît porturile paralele bidirectionale de mare viteză cît și driver-ele soft bidirecționale sînt așteptate să apară în viitoarea generație de imprimante laser de la HP.

Prima imprimantă din această familie, care va fi lansată spre sfîrșitul verii, va folosi o nouă "mașinărie" de la Canon Corp. pentru a lista 8 pagini pe minut la o rezoluție de 600 puncte pe inch. Ea va include Nivelul 1 Post Script, cu Nivelul 2 ca opțiune.

Mașina va costa între 1995\$ și 2500\$ în funcție de cantitatea de memorie pe care o include.

(PC Week)

OS/2 Versiune 2.0

Viena, 27 mai 1992 - Distribuitorii de IBM Personal Systems din Europa răsăriteană au instalat și pot face demonstrații cu OS/2. Versiunea 2.0 - un sistem de operare puternic și totuși, ușor de utilizat.

OS/2 2.0 oferă trei medii de operare: DOS, Windows și OS/2 într-un singur pachet, oferind o flexibilitate unică. În acest moment există, pentru 16 biți, peste 17000 de aplicații DOS, 4900 pentru Windows și 2500 pentru OS/2.

Sub OS/2, utilizatorii au acces la un spectru imens de programe.

Pentru a se asigura că OS/2 2.0 va rula pe platforme compatibile IBM PC, firma IBM a creat un important

laborator de testare a echipamentelor la Boca Raton, SUA, pe lîngă cel existent la Basingstoke, Marea Britanie. Pînă acum, testele de compatibilitate OS/2 2.0 au fost trecute de peste 100 de modele, inclusiv echipamente provenind de la Compaq, Dell, CompuAdd, AST și Tandy.

(Serviciul de presă IBM - Eastern Europe)

BORLAND C++ 3.0

A apărut versiunea 3.0 a compilatorului de C++ produs de BORLAND International.

După afirmațiile firmei, această versiune are cîteva noutăți importante:

- ❑ Optimizări globale pentru C și C++ - compilatorul oferă acum optimizări globale. Optimizatorul dispune de facilități ca eliminarea codului mort, sub-expresii comune și globale, compactarea buclelor, transmiterea parametrilor prin regiștri, permițînd minimizarea mărimii aplicațiilor sau maximizarea vitezei lor de execuție.
- ❑ Mediu de dezvoltare Windows - se pot edita, compila și rula aplicații de sub Windows. Mediul de dezvoltare integrat Windows (IDE) include un puternic editor multi-window, ObjectBrowser care permite vizualizarea grafică a relațiilor între obiecte și permite "navigarea" rapidă prin codul sursă, SpeedBar pentru accesul rapid la funcțiile uzuale și un help "online" care include referințe complete pentru Windows API.

Cerințe minime pentru sistem:

- ❑ Familia IBM sau Compaq de PC-uri și echipamente compatibile 100%.
PC-DOS (MS-DOS) 3.0 sau versiuni ulterioare compatibile 100%.
Microsoft Windows 3.0 sau versiuni ulterioare compatibile 100% pentru programare sub Windows.
Opțional - Microsoft SDK.
Este necesar hard disc.
- ❑ Borland C++: Intel 286 sau superior cu 1 Mb de memorie extinsă. Pentru programarea sub Windows sînt necesari 2 Mb de memorie extinsă.

(Borland International)

Vă prezentăm:

IBM Personal System/1 Pro 386SX

La sfârșitul lunii aprilie 1992, "big blue" a anunțat calculatorul IBM PS/1 Pro 386SX Model 2123.

Acesta este menit să lărgescă spectrul produselor IBM cu sisteme mai puternice și mai ușor de extins. Modelul se adresează aplicațiilor de tip:

- lucru adus acasă de la birou
- conducerea unor firme mici
- școlarizarea pentru PC
- învățămînt

Calculatorul IBM PS/1 Pro 386SX Model 2123 este livrat în trei variante:

- Model E41 - dischetă 1,44 MB, disc fix 40MB
- Model E81 - dischetă 1,44 MB, disc fix 80 MB
- Model E31 - dischetă 1,44MB, disc fix 129 MB.



Descriere

Configurația hardware constă din:

- Sistem cu:
 - procesor 80386SX la 20 MHz.
 - memorie de 2MB (85) pe placa de bază
 - două socluri suplimentare SIMM pe placa de bază
 - circuit ceas

- interfață grafică VGA pe 16 biți
- dischetă de 1,44MB
- disc fix de 40MB, 80MB sau 129MB
- trei conectori de bus AT
- interfață paralelă
- interfață serială
- port tastatură
- port mouse

- Monitor color VGA de 14":

- rezoluție maximă
 - APA 640x480
 - AIN 720x400
- frecvență verticală cadre maximă: 70Hz, neîntrețesut
- frecvență de baleiaj vertical maximă: 31,5KHz
- număr maxim de nuanțe de gri: 64
- număr maxim de culori: 256
- număr maxim de culori ale paletelor: 262 144
- moduri standard
 - text: 25 linii x 80 coloane
 - grafic: 480 linii x 640 coloane.

- Tastatură cu 102 taste

- Mouse cu 2 butoane

Pe discul fix sînt instalate, de către producător, următoarele produse software:

- IBM DOS 5.0
- Microsoft Works pentru DOS 2.0
- Microsoft Windows 3.0
- Microsoft Productivity Pack for Windows.

Cerințe de programare

Sistemul de operare inițial pentru IBM PS/1 Pro 386SX este DOS 5.0. Echipamentul este compatibil cu:

- PC/DOS 5.00
- PC/DOS+WINDOWS 3.0
- Operating System/2 2.0.

ROM-ul suportă Compatibility BIOS (CBIOS), POST și BASIC.

Compatibilitate

Calculatorul IBM PS/1 Pro 386SX Model 2123 este compatibil cu sistemele IBM Personal Computer, Personal Computer XT, Personal Computer AT și PS/2 cu bus AT la nivelul interfeței BIOS și cu cele mai multe interfețe hardware. Datorită nivelului înalt de integrare, nu pot fi utilizate extensiile de memorie și adaptoarele grafice pentru IBM PC-XT, AT și PS/1.

Limitări

Spațiul maxim de memorie care poate fi adresat pe 24 de biți (386 SX) pe IBM PS/1 Pro Model 2123 este de 16MB.

Din acest spațiu, sistemul are nevoie de 1MB pentru utilizare proprie, permițând beneficiarilor să utilizeze un spațiu de memorie de maximum 15MB.

Produse suportate

Produsele hardware compatibile cu IBM PS/1 Pro 386SX Model 2123 sînt:

- module memorie PS/2
- adaptoare de rețea IBM
- adaptoare Token Ring IBM
- disc fix de 129 MB IBM
- ecrane color 8513, 8514, 8515, 8518 sau monocrom 8503
- imprimante IBM Personal Printer, Laser Printer, Pro Printer
- imprimante Epson LQ 2550, HP Laserjet III sau Paintjet XL
- adaptoare 3Com Ethernet II, X25/Communication

Compatibilitate software

Testele de compatibilitate software au fost efectuate de către EMEA Compatibility Test Group de la Basingsstoke, Scoția în perioada 6.02.92 la 26.03.92.

Următoarele produse sînt compatibile cu IBM PS/1 Pro și vor funcționa foarte apropiat de specificațiile din documentații:

- Sisteme de operare
 - DOS 5.00
 - DR DOS 6.0
 - OS/2 ES\$ 1.0, LS 2.0 și SE 2.0
- Produse de comunicații
 - 3270 Emulation Program Entry Level 2.00
 - Novell Netware Lite 1.0
- PC 3270 2.00
- Windows
 - PC 3270 Windows 2.00
 - Procom Plus 1.1
- Produse pentru business/productivitate
 - Aldus PageMaker (Windows) 4.00 și (OS/2 PM) 3.01
 - Corel Draw (Windows) 2.01 și (OS/2 PM) 2.00
 - Lotus 123 (Windows) 1.0 și 123/G(OS/2 PM) 1.00
 - Lotus Freelance (DOS) 4.0 și (Windows) 1.0
 - Microsoft Excel (Windows) 3.00
 - Microsoft Windows 3.0
 - Microsoft Word (Windows) 1.10 și 1.11
 - Microsoft Flight Simulator 4.0
 - Word Perfect (DOS) 5.1

Următoarele produse soft au fost testate pe modele S.U.A. cu sisteme de operare S.U.A.

- Enhanced 52550 Emulation Program 2.2
- IBM Personal Communications /3270 2.0
- IBM CAD și CAD/Plus (DOS și OS/2) 3.0
- IBM System Manager 2.0
- Story Board Plus 2.0

- Software de rețea IBM
 - IBM PC Lan Program 1.34
 - IBM OS/2 Lan Server Entry 2.0
 - IBM OS/2 Lan Server Advanced 2.0
- Software de rețea Novell
 - NetWare Lite 1.0
 - NetWare 2.2 și 3.11
 - Novell Print Server 1.21
 - NetWare Link/X.25 1.0

Următoarele produse soft au fost testate pe IBM PS/1 Pro de un laborator independent de testare utilizînd modele și sisteme de operare S.U.A.

Firma	Descriere	Vers.	DOS OS/2	
			5.0	2.0
Aldus	PageMaker	4.0	x	
Ashto-Tate	dBASE III Plus	1.10	x	
	dBASE IV	1.10	x	
Autodesk, Inc.	AutoCad	10C	x	x
	AutoSketch	2.0	x	
Borland International, Inc.	PARADOX	3.5	x	
	PARADOX for OS/2	3.0		x
	Quattro Pro	3.0	x	
	TURBO C++	2.0	x	
Central Point Software	FastBack Plus	2.1	x	
	PC Tools Deluxe	7.1	x	
Fox Software, Inc.	Fox Pro	1.02	x	
Lotus Development Corporation	Freelance Plus	3.01	x	
	Lotus 1-2-3	3.0	x	x
Micro Focus	COBOL/2 Compiler + Workbench + Toolset	2.4		x
	Basic Compiler	7.1		x
Microsoft Corporation	Excel	3.0	x	
	Macro Assembler	5.1		x
	Windows	3.0	x	
	Word for Windows	1.1	x	
Oracle Corporation	ORACLE for OS/2	5.1C		x
Peter Norton Computing	Norton Utilities Advanced	6.0	x	
QuarterDeck, Inc. Software Publishing	DesqView 386	2.3	x	
	Harvard Graphics	2.301	x	
Word Perfect Corporation	WordPerfect	5.1	x	
Wordstar International	WordStar 2000 Plus	6.0	x	
	WordStar	6.0	x	

(IBM - Eastern Europe)

Multitasking Cooperativ în C++

Programarea sistemelor incluse în instalații complexe ne pune în fața unor probleme interesante. O aplicație tipică poate implica "jonglarea" - pe un sistem foarte mic - cu o varietate de funcții de eșantionare, procesoare, control și comunicație ce se desfășoară simultan. Astfel de aplicații sînt, inevitabil, paralele și se implementează cel mai bine prin mai multe taskuri mici care lucrează împreună. Din nefericire, pentru un multitasking normal este necesar un suport din partea sistemului de operare, comutarea contextelor poate fi lentă, iar partajarea resurselor poate fi complicată. Pentru a evita aceste neajunsuri se poate adopta soluția creerii unor obiecte de multitasking cooperativ în C++.

Multitasking cooperativ și multitasking forțat

În multitasking-ul forțat, o întrerupere de ceas lansează periodic un program executiv. Acesta determină dacă taskul curent a rulat prea mult timp și - dacă este așa - forțează comutarea contextului la următorul task din lista de așteptare. Dacă taskul mai dispune încă de timp, el actualizează un contor și revine din întrerupere. Deoarece o întrerupere poate să survină și să forțeze o comutare de context în orice punct al programului, sistemul trebuie să salveze și să refacă starea completă a procesorului. Partajarea resurselor de către taskuri impune rutine care să blocheze și să deblocheze accesul la structurile de date comune. Mai mult, apelurile la sistemul de operare trebuie să fie reentrante (ceea ce nu se întâmplă la MS-DOS).

De cealaltă parte, multitaskingul cooperativ nu are un executiv și

nu se bazează pe întreruperi. De câte ori un task este gata să cedeze controlul, el apelează o subrutină (*pause*) care comută contextul la următorul task. Deoarece punctul de comutare este întotdeauna același (rutina *pause*) este necesar să se salveze numai indicatorul vârfului stivei și câțiva regiștri. Aceasta determină o comutare de context simplă și - potențial - foarte rapidă. Multitaskingul cooperativ simplifică mult partajarea resurselor deoarece nu se pot "strecura" alte taskuri în timpul actualizării structurilor comune de date. Apelurile de sistem pot fi efectuate sub orice sistem de operare deoarece, în timpul apelului, nu poate surveni nici o comutare neașteptată de context.

Programul

Nucleul de multitasking cooperativ prezentat mai jos este implementat utilizînd un obiect task din C++. Obiectul task și fișierele "header" sînt prezentate în listingurile Unu și Doi. A mai fost inclus și un program demonstrativ (Listing Trei) care prezintă utilizarea nucleului de multitasking. Programul demonstrativ are nevoie de obiectul fereastră text și de fișierele "header" prezentate în Listingurile Patru și Cinci. Interfața cu nucleul constă din rutinele *InitTasking*, *fork* și *pause*. Rutina *InitTasking* inițializează sistemul multitasking și ea trebuie apelată înainte de orice alt apel. Rutina *fork* generează noi taskuri, iar *pause* efectuează comutarea contextului la următorul task. Obiectul task în sine are o metodă de a modifica ceea ce execută un obiect task - *Activate* și o metodă *Show* pentru afișarea variabilelor stării curente în scopul depanării. Deși programul este scris în Bor-

land C++, el ar trebui să fie portabil și în alte medii, dar cu următoarele neajunsuri:

Programul demonstrativ utilizează câteva funcții de lucru cu ecranul specifice bibliotecii Borland standard, iar obiectele task fac acces direct la regiștrii procesorului via "pseudovariabilele" Borland. Referințele la aceste pseudovariabile se pot înlocui ușor cu secvențe scrise în limbaj de asamblare. Se poate folosi orice tip de procesor dar, desigur, trebuie modificată partea de salvare a regiștrilor. Deși aceste obiecte au fost concepute pentru modelul de memorie *small*, este necesară numai o mică modificare pentru modele mai mari.

Comutarea contextului

Convenția de apel din C, utilizată și în C++, realizează apelul unei proceduri depunînd pe stivă toți parametrii transmiși, urmați de adresa de întoarcere. Cînd procedura apelată se termină, ea revine la adresa preluată din stivă lăsînd programului apelant sarcina de a îndepărta din stivă parametrii transmiși. Regiștrii BP, SI, DI și indicatorul vârfului stivei (SP) păstrează starea completă a unui program Turbo C++ atunci cînd se apelează o funcție. De aceea, pentru a salva starea unui task, este necesar să se depună numai acești trei regiștri pe stivă și să se memoreze indicatorul vârfului stivei. Următorul obiect task începe cu preluarea indicatorului salvat al vârfului stivei, extragerea regiștrilor din stivă, și revine la adresa de unde a fost lansat noul task. Acum noul task rulează iar apelantul pune stiva în ordine. Un exemplu de diagramă de execuție pentru două taskuri poate fi observat în Figura 1. Se

Multitasking

remarcă faptul că instrucțiunea *pause* din interiorul buclei *while* a taskului 1 se execută de două ori. Fiecare execuție abordează taskul 2 într-un alt punct, în funcție de locul în care taskul 2 a cedat controlul la execuția anterioară.

Obiecte task

Rutina *fork* din Listingul 2 crează un nou obiect task și inițializează obiectul pentru a executa funcția ce i-a fost transmisă. Fiecare obiect conține propria sa zonă de stivă, indicatorul vârfului stivei ce a

stivă. Valoarea zero pentru mărimea stivei determină constructorul să asigneze noul obiect stivă la stiva sistemului. Numai un singur obiect task poate să dețină stiva sistemului. Programul crează acest task în timpul executării funcției *InitTasking* apelată la începutul programului. În cazul modelului de memorie *small*, funcția *malloc* (care alocă zona de stivă) utilizează indicatorul curent al vârfului stivei pentru a determina dacă are spațiu disponibil. Pentru ca procesele fiu să poată genera taskuri suplimentare, funcția *fork* "împrumută" stiva siste-

tare de context la taskul următor. Fiecare obiect task are câte un pointer înainte și unul înapoi, astfel încât metoda *destructor* poate să extragă ușor un task din lista înlănțuită.

Comutare

Atunci când taskul aflat în execuție este gata să cedeze controlul execută funcția *pause*. Această rutină trimite un mesaj *Switch* către obiectul task aflat în execuție curentă. *Switch* depune pe stivă cei trei regiștri, salvează indicatorul vârfului stivei, și se mută la următorul task din lista înlănțuită. Metoda *switch* utilizează o mică "șmecherie" pentru salvarea regiștrilor. Pentru fiecare metodă a unui obiect, compilatorul generează automat o instrucțiune *PUSH BP* la începutul și instrucțiunea complementară *POP BP* înainte de instrucțiunea de întoarcere.

În mod normal compilatorul utilizează *SI* și *DI* pentru variabile regiștru și generează instrucțiuni *PUSH* numai dacă se utilizează variabile regiștru. Asignând, la începutul rutinei, pe *SI* lui *DI*, vom forța compilatorul să salveze regiștrii *SI* și *DI* generând *PUSH SI*, *PUSH DI*, precum și instrucțiunile *POP* corepondente la sfârșitul rutinei. După executarea instrucțiunilor *PUSH*, toți regiștrii necesari sînt pe stivă, astfel încît indicatorul vârfului stivei este memorat în obiect. Funcția extrage indicatorul vârfului stivei pentru noul task din lista înlănțuită și, la ieșirea din rutină, extrage din stivă regiștrii noului task și se întoarce la adresa anterioară de execuție a noului task. Codul, în limbaj de asamblare, generat de *switch* este prezentat în Figura 3.

Această tehnică de a forța compilatorul să salveze regiștrii importanți funcționează în multe alte compilatoare. Deși rutina ar putea fi optimizată în limbaj de asamblare, a fost mai ușor să se păstreze codul la nivelul înalt și să nu recurgă la programare direct în limbaj de

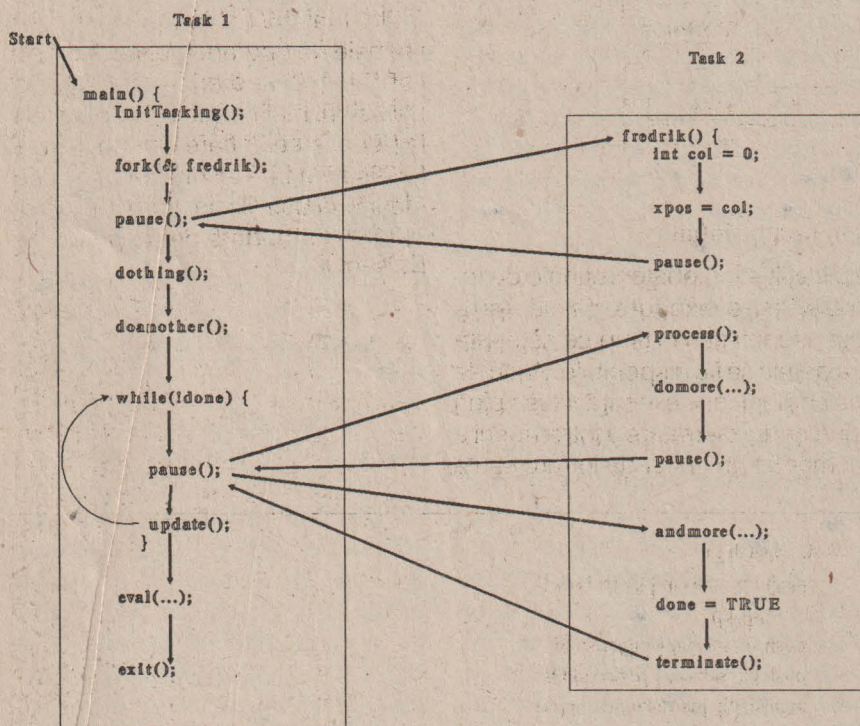


Figura 1: Execuția pentru un sistem cu două taskuri

fost salvat, un număr de identificare pentru depanare și se leagă cu alte obiecte task. Numărul de obiecte task nu este limitat decât de memoria disponibilă. Procedeeul de construire a taskurilor leagă noul obiect într-o listă circulară de alte obiecte task (Figura 2). Mărimea zonei de stivă proprie este transmisă constructorului. Mărimea implicită a stivei este, în acest program, destul de mare din cauza unor funcții de bibliotecă de genul lui *printf* care utilizează un spațiu apreciabil din

mului înainte de noua operație. Acum oricare din taskuri pot genera taskuri suplimentare după necesitate, utilizînd funcția *fork*. Metoda *Activate* inițializează stiva taskului astfel încît să refacă valori fictive în *SI*, *DI*, *BP* și inițializează adresa de întoarcere astfel încît să execute funcția transmisă lui *fork*. De asemenea, *Activate* pune pe stivă adresa rutinei *terminate*. Dacă funcția transmisă se termină și se întoarce, execuția trece la *terminate*, care șterge obiectul task și face o comu-

terminat o transmisie.

Concluzie

Multitaskingul cooperativ oferă unele din avantajele unor sisteme de operare multitasking puternice, în absența multor complicații. Pentru sisteme înglobate în alte echipamente, când resursele sînt adesea reduse, lipsa unui executiv și a supervizorului asociat fac multitaskingul cooperativ să fie ideal. Pentru sisteme mici, se poate elimina chiar și alocarea dinamică de memorie pentru a se obține cod compact și rapid. Mai mult chiar, pînă cînd sistemele de operare multitasking se vor răspîndi la o arie mai largă de utilizatori, multitaskingul cooperativ le permite celor care sînt nevoiți să folosească taskuri multiple să scrie efectiv aplicațiile în timp ce așteaptă cu nerăbdare detronarea MS-DOS-ului.

C. N.

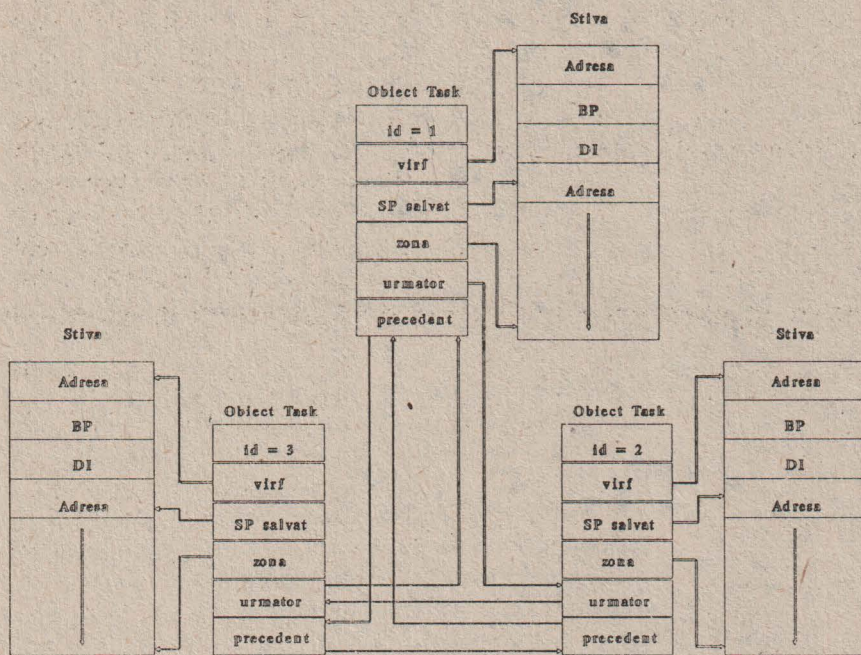


Figura 2: Obiecte task în lista înlănțuită

asamblare.

Programul demonstrativ

Programul demonstrativ crează șase taskuri care rulează fiecare în fereastra sa de text (programul este compatibil cu orice display standard IBM). Programul crează cinci dintre taskuri executînd *fork*; al șaselea este taskul "principal", care utilizează stiva sistemului. Cînd programul demonstrativ rulează se pot crea taskuri temporare suplimentare apăsînd tasta M. Chiar și cu un 8088 programul poate lansa un mare număr de taskuri înainte de a se încetini apreciabil. La programarea acestui nucleu multitasking cooperativ este important să nu uităm să plasăm o instrucțiune *pause* ori de cîte ori o funcție este pasibilă de un consum mare de timp. De exemplu, rutina de intrare de la tastatură a fost scrisă în așa fel încît să cicleze periodic prin *pause* în timp ce așteaptă apăsarea unei taste, iar întîrzierile sînt realizate prin monitorarea ceasului BIOS și executarea instrucțiunii *pause* pînă la scurgerea numărului dorit de tacturi. În

aplicațiile "înglobate" rutinele de intrare/ieșire execută *pause* între eșantionări sau în timp ce așteaptă ca datele să fie disponibile. Rutinele de comunicație execută *pause* pînă cînd codul de tratare a întreruperii a complrtat un buffer de intrare sau a

```

Task:: Switch ()
    push bp; salveaz { cadrul stiv }
    mov bp, sp
    push si; salveaz { registrul SI }
    push di; salveaz { registrul DI }
    mov si, di; instrac'iune dummy
    ...
    _SI = _DI; //instruc'iune pentru a for'a salvarea SI, DI
    saved = (int *)_SP; //memoreaz { SP }
    Current Task = next; //fixare pe noul task
    _SP = (int) Current Task - saved; //utilizeaz { SP al noului task }
    ...
    pop di; reface DI din noul task
    pop si; reface SI din noul task
    pop bp; reface cadrul stiv { din noul task }
    ret; revine la adresa de execu'ie a noului task

```

Figura 3: o parte din codul în limbaj de asamblare generat pentru Task:: Switch.

Multitasking

Listing Unu

```
// Fisier: TASK.H -- Obiecte task
// Fiecare obiect task contine propria sa stiva, legaturi cu obiectele task
// urmator si precedent din lista inlantuita, indicatorul salvat al
// virfului stivei si informatii pentru depanare

class Task {
    int *area; // baza zonei stiva
    int id; //numarul task-ului - pentru depanare
    Task *next; // task urmator
    Task *prev; // task precedent
    int *saved; // locatia stivei salvata in timpul lui pause()
    int *top; //virful stivei

public:
    void Activate( void (*)() ); // functie de lansare
    int GetId() // intoarce numarul task-ului
        { return id; }
    Task *GetNext() // intoarce pointer la task-ul urmator
        { return next; }
    int *GetSP() // intoarce SP salvat
        { return saved; }
    void SetNext(Task *t) // fixeaza pointer la task-ul urmator
        { next = t; }
    void SetPrev(Task *t) // fixeaza pointer la task-ul precedent
        { prev = t; }
    void Show(); // afiseaza date pentru depanare
    void Switch(); // comuta contextul la task-ul urmator
    Task(int); // constructor
    ~Task(); // distrugator
};

Task *fork(void (*)());
Task *InitTasking(); // initializeaza nucleul multitasking
void pause(); // comuta la task-ul urmator

extern int totalTasks; // numar total de task-uri - pentru depanare
```

Listing Doi

```
// Fisier: TASK.CPP // Multitasking cooperativ in C++
#include <conio.h>
#include <stdlib.h>
#include "task.h"

// Prototipuri
void terminate(); // iesire din task la terminare

// Definiri
#define STACKSIZE 0x200 // marimea stivei fiecarui task

// Variabile globale
Task *CurrentTask = 0; // task-ul in executie
Task *SystemTask; // task-ul principal care utilizeaza stiva sistemului
int totalTasks = 0; // contor de task-uri -- pentru depanare

//Task.Aactivate - pregateste stiva unui obiect task astfel incit, atunci
```

```
// cind se comuta la acest task sa se execute functia transmisa
void Task::Activate( void (*)() )
{
    saved = top; // reset stiva
    *(--saved) = (int) &terminate; // "ucide" functia taskului la iesire
    *(--saved) = (int) f; // plaseaza functia pentru adresa de revenire
    *(--saved) = _BP; // salveaza toti registrii pentru comutare
    *(--saved) = _SI; // salveaza SI
    *(--saved) = _DI; // salveaza DI }

// Task.Show - afisarea informatiilor pentru depanare
void Task::Show()
{
    cprintf("Task: %4i area: %04X\n\r", id, area);
    cprintf(" top: %04X saved: %04X\n\r", top, saved);
    cprintf("prev: %04X next: %04X", prev, next); }

// Task.Switch - comuta contextul la noul obiect task
// din lista inlantuita. Salveaza indicatorul curent al virfului
// stivei, preia indicatorul virfului stivei pentru task-ul urmator
// (dupa ce l-a facut task curent) si revine
void Task::Switch()
{
    _SI = _DI; // forteaza compilatorul sa salveze SI,DI
    saved = (int *) _SP; // memoreaza indicatorul virfului stivei
    CurrentTask = next;
    _SP = (int) CurrentTask - saved; // refaca SP pentru noul task
}

// Task.Task - initializeaza noul obiect task. Plaseaza obiectul in lista
// inlantuita cu celelalte task-uri. Daca marimea este 0, nu alocati spatiu
// pentru stiva ci foloseste stiva sistemului.
Task::Task(int size)
{
    static int newid = 0; // identificator unic pentru fiecare task
    id = newid++;
    totalTasks++;
    if (size)
        { // se creaza un task operator ?
            if ((area = (int *) malloc(size * 2)) == 0) //Nu, deci alocati
                {
                    cprintf("Memorie insuficienta pentru a crea task-ul %i\n", id);
                    exit(1);
                }
            top = area + size; // fixeaza virful absolut al stivei
            saved = top;
            next = CurrentTask - GetNext(); // se leaga in lista task-urilor
            prev = CurrentTask;
            prev - SetNext(this);
            next - SetPrev(this);
        } else { // nu alocati stiva
            top = (int *) _SP; // in schimb, preia stiva sistemului
            saved = top;
            next = this; // deoarece este primul task, il fac sa
            prev = this; // puncteze spre mine
        }
}
```

```
// Distrugatorul de task-uri - reda sistemului toata memoria alocata
Task::~Task()
{
    totalTasks--;
    prev-SetNext(next); // scoate acest task din lista inlantuita
    next-SetPrev(prev);

    CurrentTask = next; // noul task curent
    if (area) // nu elibereaza daca nu este alocata stiva
        free(area); // elibereaza stiva obiectului
}
// fork - creaza un task nou care sa execute functia transmisa.
// La terminarea/ functiei task-ul este distrus automat.
// fork intoarce un pointer la noul obiect task sau NULL
// daca nu are spatiu de memorie
Task *fork( void (*f)() )
{
    Task *newtask; // pointer la noul obiect task
    // In modelele de memorie "small", malloc utilizeaza indicatorul
    // virului stivei pentru a determina daca exista memorie libera.
    // Pentru a permite ramificarea din sub-task-uri, se "imprumuta" stiva
    // sistemului pentru operatia malloc.
    int temp = _SP; // salveaza SP curent
    _SP = (int)SystemTask-GetSP() - 20; // imprumuta stiva sistemului
    // creaza noul obiect task
    if ( (newtask = (Task *) new Task(STACKSIZE)) )
        newtask-Activate(f); // pregateste noul task sa execute functia
    _SP = temp; // reface stiva initiala return newtask
    return newtask;
}

// InitTasking - Initializeaza totul inainte de inceperea
// multitasking-ului. Aceasta functie trebuie apelata inainte
// de ramificarea oricarui alt task.
// Ea creaza task-ul "sistem" cooptind stiva sistemului
// intr-un obiect task. (task nr. 0) si face CurrentTask sa punteze
// la noul task operator.
Task *InitTasking()
{
    CurrentTask = (Task *) new Task(0); // creaza stiva
    sistem SystemTask = CurrentTask; // fixeaza indicatorul
    // virului stivei
    return SystemTask; }

// pause - interfata non-obiect cu contextul comutat
void pause()
{
    CurrentTask-Switch(); // comuta contextul in afara task-ului
}

// terminate - elimina task-ul curent la terminarea lui fork.
// Acesata nu // nu este o metoda, dar adresa lui este plasata
// pe stiva initiala si daca functia task-ului se intoarce, terminate
// va fi urmatoarea adresa de executie
void terminate()
{
    _DI = _SI;
    delete CurrentTask; // "ucide" task-ul
```

```
curent _SP = (int) CurrentTask-GetSP(); // fixare pe stiva task-ului
// urmator si revenire in noul task
}
```

Listing Trei

```
// Fisier: DEMO.CPP
// Program demonstrativ pentru obiecte de multitasking cooperativ

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h> ,h
#include <time.h>
#include <conio.h>
#include <string.h>
#include <bios.h>
#include <ctype.h>
#include <dos.h>

#include "task.h"
#include "twindow.h"

//Prototipuri pentru functiile demonstrative
void endlessCount();
void fiveseconds();
void funwindow();
void msdelay(int);
int newgetch();
void periodic();
void quicky();
void status();
void wallclock();

main()
{
    /* Initializeaza nucleul multitasking. Creaza task-ul tata sistem */
    InitTasking(); // initializeaza task, coopteaza stiva sistem
    /* -- initializare ecran -- */
    textattr(WHITE); // text "normal - alb pe negru
    clrscr(); // sterge ecranul
    _setcursortype(_NOCURS); // fara cursor

    /* -- lanseaza citeva task-uri -- */
    fork( &endlessCount ); // lanseaza task-ul de numarare infinita
    fork( &wallclock ); // lanseaza task-ul ceas
    fork( &periodic ); // lanseaza lansatorul periodic de task-uri
    fork( &funwindow ); // lanseaza o fereastra stranie
    fork( &status ); // lanseaza functia de numarara a task-urilor

    /* -- creaza fereastra principala -- */
    TextWindow myWindow(1,20,80,25, (LIGHTGRAY < 4) + BLUE);
    gotoxy(20,1);
    cputs("**** Multitasking Cooperativ - Demonstratie ***\r\n");
    cputs(" Fiecare fereastra este un obiect task separat ");
    cputs("care executa o functie C++.\r\n");
    cputs("Toate ruleaza 'concurrent' utilizind\r\n");
    cputs("rutina de comutare a contextului pause().");
```

Multitasking

```
gotoxy(2,6);
cputs("Comenzi: [C]rearea unui nou task, [I]esire");

/* -- asteapta o tasta si o proceseaza -- */
for(;;)
    switch ( toupper( newgetch() ) )
    {
        case 'I': // iesire - sterge ecranul si iese
            window(1,1,80,24);
            textattr(WHITE);
            clrscr();
            _setcursortype(_NORMALCURSOR);
            return(0);
        case 'C': // genereaza un nou task
            fork(&quicky);
            break;
        default: // caracter ilegal
            sound(500);
            msdelay(160);
            nosound();
            break;
    }

// endlessCount - deschide o fereastră și numara fara oprire
void endlessCount()
{
    TextWindow myWindow( 40,7,80,9,(CYAN<4) + RED);
    cprintf("Acest task nu se termina niciodata!");
    long count = 0;
    for(;;)
    {
        // numara, dar nu uita sa faci pause()
        myWindow.Activate();
        gotoxy(2,2);
        cprintf("Numarator: %li", count + +);
        pause(); // lasa si alte task-uri sa ruleze
    }
}

// fiveseconds - deschide o fereastră, numara 5 secunde si iese
void fiveseconds()
{
    TextWindow myWindow(5,5,25,8,(GREEN<4) + RED);
    cprintf(" Task temporar care");
    gotoxy(2,2);
    cprintf("dureaza 5 secunde");

    time_t t; // citeste ora curenta
    t = time(NULL);

    int i = 10000; // decrementeaza de la 10000
    while( difftime(time(NULL), t) < 5 ) {
        // continua decrementarea
        myWindow.Activate(); // pina cind difftime este 5 secunde
        gotoxy(6,3); // sau mai mult
        textcolor(BLACK);
        cprintf("%5i",i--);
    }
}
```

```
pause(); // lasa si alte task-uri sa ruleze
}

// funwindow - afiseaza un caracter mobil in fereastră
void funwindow()
{
    TextWindow myWindow(60,12,79,12, (CYAN<4) + YELLOW);
    for( int i = 0; i = ++i % 20 )
    { //muta i de la 0 la 19
        myWindow.Activate();
        cputs(" ");
        gotoxy( abs( ((i/20) * 20) + i) + 1, 1);
        textcolor( rand() % 15 );
        cputs("$");
        msdelay(100); // asteapta 100 ms
    }
}

// msdelay - intirziere cu un numar de milisecunde
// cu rezolutie de 55 ms
void msdelay(int delay)
{
    long ticksplus = biostime(0,0L) + delay / 55;
    while (biostime(0,0L) - ticksplus) // asteapta pina trece timpul
        pause(); // lasa si alte task-uri sa ruleze
}

// newgetch - face acelasi lucru ca si getch, cu exceptia asteptarii
// unei taste
int newgetch()
{
    while ( !kbhit() )
        pause();
    return getch();
}

// periodic - lanseaza periodic un alt task
void periodic()
{
    TextWindow myWindow(1,10,65,2,(LIGHTGRAY<4)
        + MAGENTA);
    cputs("Lanseaza un task temporar la fiecare 10 secunde");
    for(;;)
    {
        for( int i = 0; i 10; i + + )
        { // bucleaza de 10 ori inainte de ramificare
            myWindow.Activate();
            gotoxy(20,2);
            cprintf("%i",i);
            msdelay(1000); // intirziere de o secunda
        }
        fork(&fiveseconds); // lanseaza noul task ce moare dupa 5 secunde
    }
}

// quicky - deschide ferestre, ramine activ citeva secunde si iese
void quicky()
{
}
```

```

static int xpos = 3; // pozitia x a ferestrei noului task
static int ypos = 3; // pozitia y a ferestrei noului task
TextWindow myWindow(xpos + 1,ypos + 1,xpos + 8,ypos + 6,
    (RED < 4) + WHITE);
xpos = (xpos + 3) % 64;
ypos = ++ypos % 7;

for( int i = 0; i 10; i++ )
    { // decrementeaza 10 secunde
    myWindow.Activate();
    cprintf("Mort in 10 sec.: %i",i);
    msdelay(1000); // intirziere de 10 secunde
    }
}

// status - afiseaza numarul de taskuri care ruleaza
void status() { TextWindow myWindow(1,1,18,1, (MAGENTA < 4) +
YELLOW);
for(;;) { myWindow.Activate();
Tasks);
    msdelay(2000); // asteapta 200 milisecunde } }

// wallclock - afiseaza permanent ora void wallclock()
{ TextWindow myWindow(54,1,80,1,(RED < 4) + BLUE);
time_t t; // va mentine ora
char buf[40]; // buffer temporar pentru a elimina \n
for(;;)
    { // actualizarea orei se face permanent
    myWindow.Activate();
    t = time(NULL); // preia adresa sirului de caractere cu ora
    strcpy(buf,ctime(&t)); // copiaza sirul in temp
    buf[24] = "\0"; // elimina \n
    cprintf("%s",buf);
    msdelay(1000); // asteapta o secunda
    }
}

```

Listing patru

```

// Fisier: TWINDOW.H -- obiecte fereastră pentru programul
// demonstrativ
// Presupune functiile de biblioteca Borland C++

class TextWindow {
    int attrib; // atribut pentru modul text
    int left, top; // pozitii x,y de inceput
    int right, bottom; // pozitii x,y finale

public:
    void Activate(); // activare
    TextWindow(int,int,int,int,int); // constructor
    ~TextWindow(); // distrugator
}

```

Listing cinci

```

// Fisier: TWINDOW.CPP - obiecte fereastră text

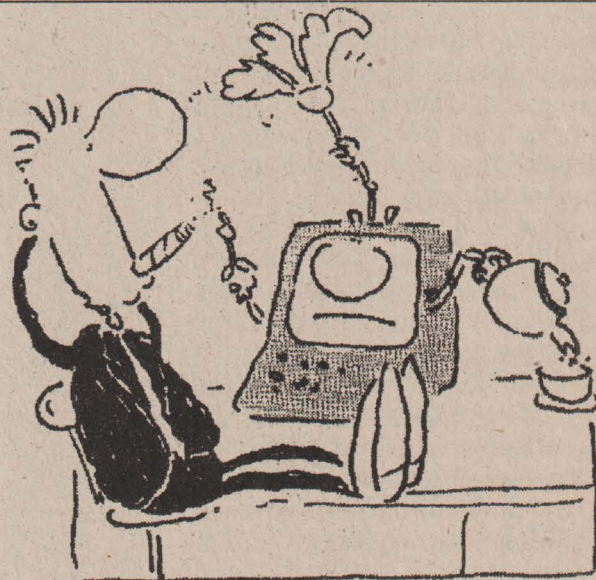
#include "twindow.h"
#include <stdio.h>
#include <conio.h>

// activarea unei ferestre
void TextWindow::Activate()
{
    window(left,top,right,bottom); // utilizeaza biblioteca C++
    textattr(attrib);
}

// constructor de ferestre - memoreaza coordonatele si sterge fereastră
TextWindow::TextWindow(int l,int t,int r,int b,int a)
{cp
    left = l;
    top = t;
    right = r;
    bottom = b;
    attrib = a;
    Activate(); // activeaza fereastră
    clrscr(); // sterge fereastră
}

// distrugatorul de ferestre - sterge ferestrele si le face fundalul negru
TextWindow::~TextWindow()
{
    Activate();
    textattr( (BLACK << 4) + WHITE);
    clrscr();
}

```



PC-urile ca sisteme de achiziție de date în rețea

Rețelele locale de calculatoare (Local Area Network - LAN), au început să capete o mai largă răspândire doar în ultimii ani. Cuvinte "cheie" ca Workstation, File Server, Token Ring, NETBIOS, ETHERNET, ARCNET etc., au început să facă parte din jargonul zilnic al utilizatorilor de PC-uri. Calculatorul personal (PC), este "forța majoră" care singură, este responsabilă pentru aceste fapte.

PC-ul, a fost conceput inițial pentru a fi un înlocuitor ieftin al minicalculatoarelor, deoarece era mult mai accesibil majorității utilizatorilor. Pe măsură ce numărul de utilizatori creștea, aceștia simțeau tot mai mult nevoia de a-și comunica datele între ei.

Datorită cantității PC-urilor IBM XT/AT și compatibile existente, tot mai mulți producători au început să dezvolte soft și hard pentru a-și putea realiza aplicațiile lor în rețea. Astfel, un număr mare de companii au produs interfețe montabile în PC, ce realizau toate funcțiile controllerelor de rețea (Ethernet, Arcnet, Token Ring) și tot mai multe versiuni de soft de rețea apăreau în mod regulat.

PC-urile erau inițial utilizate în birouri pentru a realiza task-uri (sarcini) cum ar fi: editare și prelucrare de texte, gestionarea de baze de date și calcul tabelar, apoi au început să-și găsesască drumul spre laboratoare și întreprinderi industriale, pentru a efectua măsurători și a controla procese în timp real. Rețelele azi, urmăresc același drum.

Încrederea obținută în sistemele de control ale proceselor bazate pe PC-uri se bazează atât pe produsele hardware cât și pe cele software apărute. Utilizatorii din industrie caută soluții de a conecta calculatoarele industriale, utilizând atât tehnica rețelor DISTRIBUITE cât și pe cea a controlului unui proces de la un calculator de proces central GAZDĂ - HOST, chiar și în cazul în care punctele de măsură ce trebuie monitorizate și controlate sînt răspândite pe o arie geografică mare.

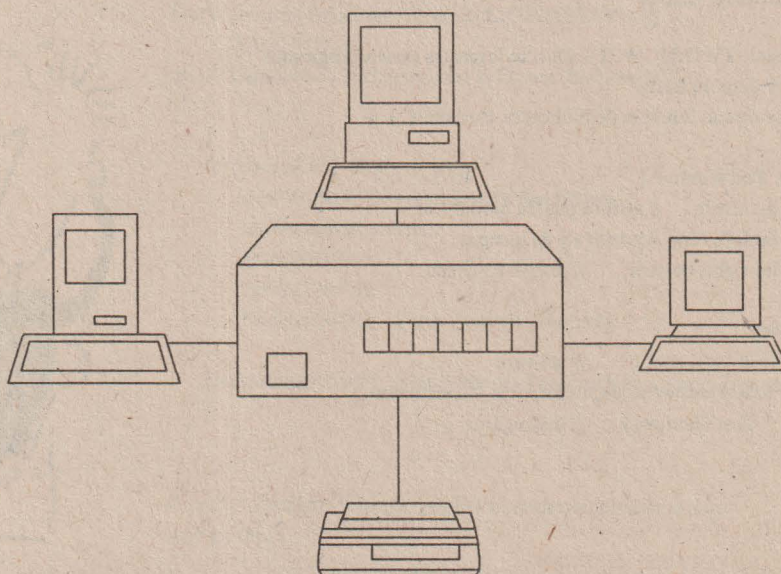
De asemenea, pentru a putea colecta datele de la punctele de măsură aflate la distanță, inginerii de proces studiază posibilitatea de a controla un anumit proces dintr-un punct aflat la o distanță oarecare față de locul unde se află calculatorul de proces. Informația derivată din proces, (calitate, cantitate, etc.) este folosită la un nivel superior,

făcînd parte integrantă din fluxul total informațional al întreprinderii.

În multe aplicații, calculatoarele IBM PC-XT/AT/386 și compatibile ce lucrează sub sistemul de operare MS-DOS au reprezentat focarul tuturor aplicațiilor bazate pe măsură și control, iar calculatoarele microVAX și VAX sînt mai mult utilizate ca SUPERVIZOR și sistem de management informațional în rețea.

În aplicații de control de sine stătătoare (ce nu lucrează în rețea), s-au realizat un număr de aplicații soft excelente pentru achiziții de date și control pentru laborator și industrie. Dintre acestea cele mai populare pachete de programe includ LABTECH NOTEBOOK pentru achiziții de date și LT/CONTROL precum și GENESIS pentru controlul proceselor.

Recent, proiectanții acestor pachete de programe, le-au dezvoltat mai departe pentru a permite lucrul



în rețele distribuite (PEER - TO-PEER) cît și în rețele IERARHICE de tip GAZDĂ-LA-CLIENT - HOST-TO-CLIENT.

Ofertele firmei Laboratory Technologies Corporation, se bazează atît pe pachetul de programe LT/CONTROL ce rulează la nivelul calculatorului GAZDĂ din camera de control, cît și pe LT/CONTROL sau LABTECHNOTEBOOK, ce rulează în nodurile din rețeaua întreprinderii.

Firma ICONICS, producătorul pachetului de programe GENESIS, a anunțat recent programul GEN-NET, care este un sistem distribuit ce rulează în fiecare nod de rețea, oferind posibilitatea de comunicare între noduri, pe baza unei legături ce poate fi Ethernet, Arcnet sau Token Ring, operînd sub NETBIOS.

Ambele opțiuni de rețea, LT/CONTROL și GENESIS permit ca un program ce rulează într-un calculator să folosească date dintr-un alt program ce rulează într-un alt calculator. Rețelele distribuite permit, ca datele achiziționate în timp real, fișierele și mesajele de alarmă să poată fi transmise la toate nodurile rețelei.

Rețele distribuite

Rețelele distribuite, se folosesc în general în aplicații în care există mai multe puncte de control, care centralizează informații dintr-o întreprindere sau dintr-un proces complex și datele generate într-un punct afectează luarea unei decizii într-un alt punct. În aceste aplicații, în fiecare nod se folosesc de obicei calculatoare de aceeași putere de calcul. Un inginer de proces va avea nevoie de posibilitatea afișării vizuale pe un monitor în fiecare nod al rețelei. Această configurație este de asemenea utilizată în aplicații care cer o independență în fiecare nod, astfel ca în cazul defectării unui nod, acest lucru să nu afecteze alte noduri ale rețelei.

Pentru acest tip de aplicații sînt necesare pachete soft complete,

pentru a adresa punctele de intrare/ieșire (I/O) din fiecare nod. Aceste pachete trebuie să dea inginerului, posibilitatea de a-și dezvolta propriile sale strategii de control, folosind algoritmi standard pentru a controla intrările și ieșirile analogice și numerice.

Algoritmii trebuie să includă funcții ca PID, filtrare, Leadleg și Deadtime. Pachetul va trebui de asemenea să poată efectua operații trigonometrice, matematice și logice. În timpul rulării, posibilitatea de a memora date de la diferite puncte de măsură preselectate, este obligatorie. Azi, aceste programe trebuie să fie capabile să lucreze și în rețea.

Tendențele de perspectivă ale acestor pachete software sînt de a include din ce în ce mai multe funcții, ceea ce înșă, de cele mai multe ori, are ca efect creșterea nevoii de memorie din ce în ce mai mare. Adaptoarele grafice cel mai des utilizate sînt EGA și VGA. Necesitățile de stocare se vor concretiza în utilizarea unui harddisk de maximum 40 MBytes. În spiritul unei eventuale dezvoltări ulterioare, este bine ca sistemul ales să fie bazat pe procesorul 80286 sau 80386.

Datorită necesității memoriei adiționale cerute de soft-ul de rețea, sistemele trebuie să fie prevăzute cu memorie expandată sau extinsă. Este de dorit utilizarea unei memorii între 1 și 4 MBytes.

Genesis - în rețele distribuite

Pachetul de programe GEN-NET al firmei ICONICS, oferă posibilitatea lucrului în rețele distribuite, permițînd ca datele culese în timp real de la diferitele puncte de măsură, fișierele și mesajele de alarmă să poată fi transmise în rețea între multiplele noduri GENESIS - compatibile. Fiecare nod GENESIS - compatibil, este capabil să funcționeze INDEPENDENT, în timp real și să asigure în continuare funcționa-

rea rețelei în cazul căderii unui nod.

GEN-NET este compatibil cu rețelele bazate pe NETBIOS sau ARCNET. NETBIOS - un protocol de comunicații pentru rețele distribuite dezvoltat de IBM - este un standard DE FACTO și permite realizarea unor conexiuni virtuale între PC-urile rețelei. NETBIOS este livrabil pentru toate rețelele standard (Ethernet, Arcnet și Token Ring) ca și pentru alte tipuri de rețele industriale.

GEN-NET este integrat în pachetul GENESIS, prin tehnicile sale bazate pe CAD, iconuri și ușurința de folosire. Nodurile noi ale rețelei, se pot adăuga sau modifica cu aceeași ușurință ca și modificarea funcțiilor de I/O. GEN-NET include un soft de rețea și nu necesită alte pachete de soft pentru gestionarea rețelei.

LT/Control

în rețele distribuite

Spre deosebire de GENESIS, LT/CONTROL nu are inclus și soft-ul de rețea și de aceea nu poate opera într-o rețea distribuită decît dacă folosește un pachet de software de rețea standard ca de exemplu IBM PC LAN, 10 NET sau LANTASTIC. Pachetul LANTASTIC, elaborat de firma Artisoft Inc., are față de celelalte avantajul că folosește foarte puțină memorie RAM și are un preț de cost scăzut.

Rețelele distribuite au un preț de cost scăzut, permițînd posibilitatea interschimbării datelor între calculatoarele din nodurile rețelei, ce controlează procesul. Performanțele nu sînt chiar atît de bune ca și în cazul rețelelor construite pe principiul IERARHIEI, dar oricum aceasta se datorează mai mult sistemului de operare MS-DOS, care constituie o frînă pentru transferul datelor la viteze mari. De asemenea, trebuie acordată o mai mare atenție securității datelor, pentru că

acest lucru este mai greu de făcut în rețelele distribuite.

Rețele locale ierarhice

Într-o rețea locală ierarhică, se utilizează un FILE SERVER pentru a pune la dispoziția stațiilor de lucru soft-ul necesar realizării funcțiilor de achiziție de date în rețea. Fiecare stație de lucru funcționează independent, dar este legată la un calculator GAZDĂ - HOST, care rulează un program bazat pe strategia citirii intrărilor de la stațiile din nodurile rețelei. Stațiile de lucru din noduri nu pot comunica între ele în mod direct, decât prin intermediul FILE SERVER-ului.

FILE SERVER-ul și calculatorul GAZDĂ pot fi un același calculator, sau pot fi două calculatoare diferite. Datorită faptului că performanța întregului sistem depinde de caracteristicile FILE SERVER-ului, este foarte important alegerea unui FILE SERVER cu performanțe ridicate, în special dacă el și calculatorul GAZDĂ sînt realizate în același PC.

În configurațiile ierarhice, performanțele pe care trebuie să le aibă calculatoarele din noduri sînt minimale. În majoritatea situațiilor acestea pot fi XT-uri fără harddisk prevăzute cu un monitor, necesar pentru a afișa mesaje unui operator.

LT/Control

în rețele ierarhice

În aplicațiile ierarhice, LT/CONTROL poate opera în calculatorul GAZDĂ, definind strategia folosirii în timp real a datelor provenite de la stațiile din nodurile rețelei. Un monitor ce servește ca interfață utilizator poate fi atașat la calculatorul GAZDĂ, pentru a da posibilitatea inginerului de proces de a avea un control activ asupra întregii rețele.

Soft-ul ce rulează pe stațiile din nodurile rețelei, are cerințe mici. El

trebuie să acceseze punctele de măsură analogice, să le digitizeze și să trimită valorile la calculatorul GAZDĂ, în timp real. Acest program trebuie de asemenea să accepte date de la calculatorul GAZDĂ și să le dirijeze la modulele de ieșiri analogice și numerice. "Închiderea buclei" se realizează la nivelul calculatorului GAZDĂ. Stațiile din nodurile rețelei funcționează ca niște dispozitive de I/O.

Cerințele soft ale calculatoarelor din nodurile rețelei se realizează ușor cu versiunile RUN-TIME ale pachetului de programe LABTECH NOTEBOOK. Acesta este un pachet de programe de I/O ce poate dirija în timp real datele la, sau de la, rețea. Transferul de date către calculatorul GAZDĂ se poate face cu viteze de pînă la 10 MB/sec.

NOTEBOOK poate rula atît în modul HIGH SPEED - viteză ridicată (fără monitor, datele fiind trimise cu viteze mari) cît și în modul NORMAL (datele colectate transmițîndu-se cu viteze mai mici, dar datele în timp real putînd fi monitorizate pe un monitor conectat la o stație dintr-un nod oarecare).

Controlul întregii rețele este realizat de către FILE SERVER, în care se instalează un sistem de operare de rețea cu rolul de a dirija traficul în rețea. Piața oferă astăzi un număr mare de asemenea sisteme de operare în rețea dintre care însă cel mai popular este NOVELL NETWARE, ce reprezintă un standard DE FACTO. Acest sistem de operare are diferite versiuni în funcție de numărul de utilizatori (de la 4 la 100).

Genesis în rețele ierarhice

Pachetul de programe GENESIS, poate de asemenea lucra în configurații ierarhice, utilizînd diverse drivere soft de rețea ca de exemplu, DACNET, dezvoltat de firma ADAC Corporation pentru interfețele ARCNET. În această configurație, GENESIS se lansează în calculatorul GAZDĂ, stabilind o strategie de preluare a datelor

obținute de la stațiile de lucru din noduri (ce pot fi configurații hardware cu preț de cost scăzut), fiecare dintre ele putînd fi bazate pe PC-uri configurate ca sisteme de achiziție de date în care rulează un program relativ simplu de colectare a datelor.

În orice nod, programul ADLIBI/O, scanează continuu toate intrările, atît cele analogice cît și cele numerice și memorează datele rezultate în matrici de memorie de intrare locale. Nodul poate de asemenea accepta date de la calculatorul GAZDĂ, pe care le memorează în matricile de memorie locale de ieșire. La nivelul calculatorului GAZDĂ, dacă GENESIS necesită o dată de intrare de la un nod, transmite o cerere de date de intrare via driver-ul DACNET. Nodul răspunde prin trimiterea datelor cerute prin rețea la GAZDĂ. Calculatorul GAZDĂ operează asupra datelor folosind algoritmi de control din pachetul de programe GENESIS generînd un fișier de ieșire pentru ieșirile analogice sau numerice.

Astfel, sistemul lucrează folosind o strategie bazată pe scanare și cerere. Datele nu sînt puse în rețea decît dacă ele sînt cerute de calculatorul GAZDĂ, eliminînd astfel eventualele coliziuni ale acestora.

Topologia este viabilă doar în cazul în care atît calculatorul GAZDĂ cît și stațiile de lucru din noduri sînt funcționale. În cazul în care calculatorul GAZDĂ "cade", acest lucru este detectat de soft-ul de rețea care pune întreaga rețea sub controlul unui program aflat în memoria sau pe harddisk-ul uneia dintre stațiile de lucru dintr-un nod.

Kallo Tibor

THE

right option

MX®
COMPUTERS

with

ATI Graphic Cards

MAG Monitors



in hardware

 **CalComp**

Panasonic

 **HEWLETT
PACKARD**



A&C
INTERNATIONAL S.A.



Tel 40-0-53.53.15.

Performanță în grafica 3-D:

AutoCAD Release 11

Pentru a desena obiecte care să arate ca în realitate aveți nevoie într-adevăr de un sistem în trei dimensiuni. Desenele de arhitectură par să prindă viață când le construiești în 3-D și adaugi umbriri naturale elementelor constructive.

Construirea unui model tridimensional al unui automobil vă permite să vă rotiți în jurul acestuia având posibilitatea să-l priviți din orice unghi doriți. Este mult mai eficient să priviți dintr-o poziție mai îndepărtată la o imagine 3-D a obiectului, așa cum este el perceput în realitate, decât să priviți trei reprezentări disjuncte în 2-D ale acestuia. Orice proiectant poate beneficia de posibilitatea de a privi imagini multiple ale obiectelor fără să fie nevoit să creeze desene pentru fiecare perspectivă în parte, economisind astfel multe ore în procesul proiectării. Instrumentele pentru generarea de suprafețe reale, precum și "plimbări" animate prin structuri arhitecturale vă pot ajuta să comunicați mai eficient ideile Dvs. și probabil să vă puneți semnătura mult mai rapid pe un proiect finalizat.

În cei zece ani de existență, AutoCAD a stabilit multe standarde industriale pentru proiectarea și desenarea pe calculatoare sub DOS. AutoCAD continuă să aibă o poziție foarte puternică, deoarece este cea mai larg folosită și ușor integrabilă aplicație CAD. Împreună cu cea mai recentă versiune, AutoCAD Release 11, Autodesk furnizează opțiunea Advanced Modeling Extension (AME) pentru a adăuga AutoCAD-ului facilitățile de a modela obiecte solide 3D. AME este integrat pe deplin în AutoCAD și puteți folosi același set de comenzi pentru crearea și manipularea obiectelor solide. Pentru sarcini mai complexe, cum ar fi operații booleene, analiză finită și dinamică, și detecția de interferențe, comenzile AME sînt indispensabile. Indiferent că începeți cu forme solide de bază sau rotiți și extrudați obiecte 2D, puteți realiza foarte repede ascunderea liniilor și generarea suprafețelor. Pentru ca AutoCAD-ul să obțină modele care să arate ca în realitate și în culori naturale, cu parametri de iluminare și texturi, este nevoie de extensia opțională AutoShade; cumpărînd versiunea 2.0 a acesteia beneficiați și de înalta tehnologie inclusă de

pachetul RenderMan. Această opțiune vă permite să controlați caracteristicile suprafețelor, incluzînd gradul de opacitate, textură, reflectivitate, precum și un număr nelimitat de surse de lumină. Se pot importa imagini scanate care pot deveni astfel texturi pentru suprafețe. Modelele astfel realizate pot fi animate, existînd și posibilitatea de a le studia prin deplasarea unei camere de luat vederi prin interiorul structurilor proiectate.

Avînd la dispoziție redarea cu 16,7 milioane nuanțe cromatice într-un sistem organizat pe 24 de biți veți fi limitați doar de structura hardware, capacitatea memoriei și de propria dumneavoastră imaginație.

Arhitectura deschisă a programului AutoCAD vă permite flexibilitatea de a-l adapta pentru aplicații diferite. Această adaptabilitate este una din caracteristicile remarcabile ale pachetului. AutoCAD are un limbaj de programare integrat, AutoLisp, care vă dă posibilitatea să rulați aplicații scrise în AutoLisp realizate de terțe firme.

Cu ajutorul interfeței ADS (AutoCAD Development System) puteți lucra cu limbaje de programare de nivel înalt cum este limbajul C pentru a

CE FACE SOFT-UL DE CAD TRIDIMENSIONAL?

Automatizează procesul de realizare a desenelor complexe și ușurează efectuarea reviziilor. Produsele performante furnizează instrumente pentru generarea suprafețelor reale și animarea parametrică.

AVANTAJE

Desenele finale arată foarte bine și sînt ușor de actualizat. Folosind mai degrabă programe 3-D decât 2-D aveți posibilitatea să generați vederi noi fără a fi necesar să realizați un nou desen.

DEZAVANTAJE

Aceste pachete de soft au o curbă de învățare abruptă. Introducerea informației 3-D cu ajutorul unui dispozitiv de intrare 2-D poate să nu fie la îndemîna oricui.

RECOMANDĂRI

AutoCAD 386 este standardul industrial din motive bine întemeiate:

- este sistemul CAD cel mai răspîndit printre utilizatori (peste 400.000 instalări în lumea întreagă)
- este utilizat de ingineri și proiectanți în aplicații specifice pentru arhitectură, construcții civile, inginerie mecanică, electrică și electronică, chimică, sisteme informatice geografice (GIS), facilități de conducere și planificare, desktop publishing și ilustrare tehnică, ș.a..
- rulează pe o gamă largă de platforme de calcul: PC/MS-DOS, OS/2 și SCO Xenix 386, Apple Macintosh II, Sun Microsystems, DEC și Apollo.
- există suport pentru sute de dispozitive periferice.
- are o arhitectură deschisă - se poate foarte ușor adapta de către utilizator
- există peste 600 pachete de aplicații dedicate
- modelează tridimensional prin muchii (wire-frame) sau prin suprafețe (corp solid).
- are facilități pentru schimbul de fișiere cu alte programe în multiple formate (DXF, IGES, HPGL, WPG, RIB, ASCII, coordonate x,y,z)
- vizualizare dinamică cu perspectivă.
- rețele extinse de centre de instruire și grupuri de utilizatori independente.

face un pas mai departe în adaptarea programului AutoCAD.

Structura flexibilă de comenzi a programului, capabilă să satisfacă orice preferință personală, vă permite să alegeți comenzi din meniurile situate în partea dreaptă sau în partea superioară a ecranului, meniuri ferestre (disponibile cu anumite drivere pentru ecran), meniuri pentru tablete grafice și meniuri de tip buton.

Un model complex poate fi un teren bogat pentru compararea instrumentelor din pachetul de bază AutoCAD cu cele introduse prin extensia sa AME. Să presupunem că într-un desen de arhitectură o coloană a unui templu are 25 de caneluri sau corpuri de rotație. Printr-o abordare cu modelare de corp solid specifică AME putem construi doar 24 de caneluri, dar cu comenzile de bază AutoCAD am putea desena 25.

Folosind însă Coons Patch (o pânză de suprafață complexă reprezentată printr-o serie de noduri sau puncte) se pot ascunde intersecțiile dintre elementele unei construcții 3-D.

Pentru desenarea unei cupole folosindu-se modelarea de corp solid, prin operații booleene de scădere, se poate tăia o sferă la nivelul intersecției cu pereții construcției.

AutoCAD poate rula într-o configurație cu unul sau două monitoare (al doilea monitor afișează linia de comandă și meniurile de comenzi). În modul protejat pe 32 de biți AutoCAD folosește un manager de memorie virtuală pentru a accesa direct toată memoria din sistemul dumneavoastră fără să utilizeze memoria expandată. AutoCAD este compatibil de asemenea cu majoritatea rețelelor bazate pe calculatoare personale.

După revista "PC Magazine"

Vincentiu Turică

Probleme în grafica 3-D

Cel mai popular pachet de programe pe piața de microCAD este AutoCAD al firmei Autodesk. Într-un anumit sens, deși nu tocmai complet, este cel mai performant din zona CAD pe PC. Majoritatea utilizatorilor îl consideră un program dificil de utilizat, dar este un rău necesar și merită să depășești dificultățile de învățare. Chris DeLucchi este președintele AutoCAD User's Group din San Diego și autorul seriei "The AutoCAD Cookbook". Acesta declară: "AutoCAD poate să nu fie cel mai bun, dar este foarte ușor adaptabil după preferințe și este parte a unei infrastructuri foarte largi - mulți producători au dezvoltat pe acest nucleu extensii dedicate". Există literalmente sute de aplicații dezvoltate pe acest nucleu, cu ajutorul cărora AutoCAD poate face orice vă puteți imagina fără a fi necesar un efort de programare în AutoLisp. Alte pachete de programe destinate proiectării asistate au limbaje de programare proprii, dar niciunul nu are aceeași popularitate ca limbajul AutoLisp. AutoCAD este disponibil pe mai multe platforme decât orice altă aplicație CAD: PC, Macintosh, diferite stații grafice sub sistemul de operare Unix. Totuși, în grafica 3-D, "pentru generarea unor modele puțin mai complexe, toate aplicațiile de nivel mediu, incluzând AutoCAD, sînt îngenucheate", declară DeLucchi. De ce, va prefera

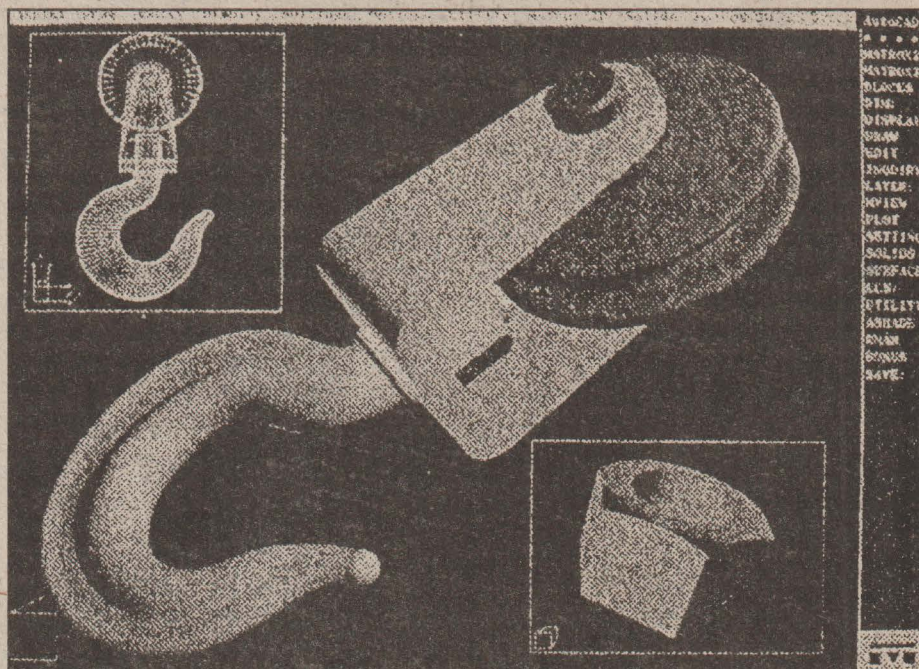
cineva să utilizeze AutoCAD dintre toate pachetele de programe disponibile pe PC?

"Mai întîi, datorită adaptabilității sale. Mulți gîndesc că nu este plăcut și nici ușor de utilizat, dar pentru a elimina această piedică ca să-l faceți mai prietenos, îl puteți modifica foarte ușor. AutoCAD este un program foarte îndrăgit.

Al doilea motiv ar fi infrastructura acestuia - grupuri organizate de utilizatori (User Groups), cărți, reviste, dealeri, școli, precum și operatori experimentați".

Dave Sander, proiectant experimentat, consultant în probleme de CAD, declară: "Partea cea mai costisitoare a realizării unui desen nu este nici calculatorul și nici softul, ci salariul pe care instituția îl plătește operatorului pentru ceea ce face. Alegeți programul care are cea mai adecvată interfață cu activitatea Dvs. Pe de altă parte, asigurați-vă că aveți destulă putere de calcul ca să nu fiți nevoit să așteptați după computer".

Sfatul lui Sander? "Întrebați-vă ce face în prezent departamentul din care faceți parte. Dacă e vorba despre proiecte simple sau care nu implică comunicația cu alte instituții, puteți alege un sistem CAD mai puțin performant. Pe măsură ce activitatea Dvs se dezvoltă, ușurința în utilizare devine mai puțin importantă. Asta înseamnă că aveți mai multe persoane care folosesc softul în mod regulat, tinzînd astfel să devină mai experimentați. Problema nu este că ei trebuie să piardă trei luni învățînd să lucreze într-un sistem. Problema cu programele 3D este că oamenii nu sînt obișnuiți să gîndească tridimensional. Desenatorii au fost antrenați ani de zile cu reprezentările bidimensionale pe hîrtie. Chiar dacă le oferi un sistem 3D, ei gîndesc totuși în 2D.



FOXPRO 2.0

Tehnologia Rushmore

Reacția față de tehnologia Rushmore a lui Fox a generat două păreri opuse. Unii o compară cu faimosul citat al lui Arthur Clarke: "Orice tehnologie suficient de avansată nu poate fi distinsă de magie". Ceilalți răspund: "Orice tehnologie suficient de avansată nu poate fi distinsă de o demonstrație dichisită". Care este adevărul?

Era foarte greu de dat un răspuns după privirea fugară de la prezentare. Președintele lui Fox Software, dr. David Fulton, a deschis o bază de date cu un milion de articole și a tastat comanda "COUNT FOR Street = Main Street AND State = 'CA'". În mai puțin de o secundă interogarea s-a executat, auditoriul a fost buimăcit, iar Fulton a jubilat.

La un studiu mai atent, se descoperă că unele clauze utile, cum ar fi SET ORDER, pot înrăutăți performanțele lui Rushmore.

Deci, locul ocupat de Rushmore în intervalul dintre magia autentică și iluzia deșartă depinde de necesitățile aplicației, ceea ce va încerca să clarifice și articolul de față.

Deși este posibil ca, în viitor, Fox să îmbunătățească această tehnologie, un lucru este sigur: Rushmore este rapid. Este uimitoare viteza cu care, după o interogare inițială se desfășoară interogările ulterioare. Fox a avut dintotdeauna algoritmi de "cache" foarte buni, dar aceasta nu poate fi singura explicație. De exemplu:

```
COUNT FOR Sex = 'M' AND
Age >100
```

```
COUNT FOR Sex = 'M' AND
Age < 10
```

Dacă prima interogare, pe o bază de date cu un milion de articole, se face leneș - în patru secunde, cea de-a doua interogare se termină înainte de a apuca să-ți privești ceasul. Este limpede că Rushmore memorează un fel de rezultate intermediare ale primei interogări. Această tehnică face ca al doilea COUNT să nu o ia de la început. Rushmore poate să memoreze rezultatele unor rutine pentru cazul în care se dorește rularea acelor rutine mai târziu.

Un alt aspect interesant al tehnologiei Rushmore este "incompatibilitatea" cu date ordonate. Alegerea unei ordonări diferită de ordinea fizică a articolelor poate, pentru baze de date mari, să facă un BROWSE FOR încet ca un melc.

Misterul pare să se dezlege dacă reflectăm asupra limitării de 500000 de articole pentru unele utilizări - limitare motivată de Fox prin "motive tehnice întemeiate". Unii utilizatori susțin că această limitare este legată de indecși. S-a emis ipoteza că Rushmore utilizează hărți de biți pentru a memora rezultatele fiecărei expresii ce a corespuns unui câmp indexat, cu câte un bit pentru un articol. Aceasta ar complica considerabil informația necesară găsirii articolelor ce satisfac condițiile unei interogări. De asemenea, explică limitarea la 500000 de articole pentru utilizarea lui Rushmore în versiunile non - 386: limitarea segmentelor la 64K în arhitecturile 8086 și 80286 ar putea explica această limită, deoarece în 64K nu se pot păstra hărți de biți mai mari de 512000 de elemente. În modul protejat 386, mărimea segmentelor este mult superioară, ceea ce ar însemna că Rushmore



ar putea lucra cu baze de date ce se apropie de 32 miliarde de articole.

Dar cum crează și cum utilizează Rushmore hărțile de biți? Să luăm exemplul următor:

```
LOCATE FOR Charges >
100000 AND MONTH(In-
cur_date) = 3
```

Dacă există doi indecși, după expresiile Charges și MONTH (Incur-Date), se vor crea două hărți de biți prin parcurgerea celor doi indecși. Dacă baza de date conține opt înregistrări, vom putea vedea două hărți de biți memorate sub forma 10000010 și 00100011. Cele două hărți de biți sînt comparate logic cu operatorul AND, obținîndu-se 00000010. Bitul "1" are un offset de 7 biți, deci FoxPro extrage a șaptea înregistrare. Dacă apoi executăm:

```
LOCATE FOR Charges >
100000 AND Age >= 18
```

Rushmore va căuta numai în indexul Age pentru a crea harta de biți 11011100 pentru Age >=18 și va face operația AND cu harta de biți pentru Charges - care există deja, obținînd valoarea 10000000 care desemnează prima înregistrare. Este foarte important de reținut că această hartă de biți este păstrată în ordinea fizică a înregistrărilor. Dacă dorim să vedem

rezultatele ordonate după un anumit index, trebuie să revenim la index. Accesul lui Rushmore ordonat după index poate fi semnificativ încetinit în funcție de mărimea fișierului, mărimea memoriei disponibile pentru "cache" și de gradul de fragmentare a datelor.

Totuși, Rushmore folosește câteva "scurtături" iscusite. El poate să preia inversa unei hărți de biți în loc să parcurgă din nou indexul. Dacă există o hartă de biți care garantează că nu se vor găsi înregistrări, Rushmore nu se va mai deranja să creezeze celelalte hărți de biți. Un exemplu ar putea fi:

Interogarea 1:

```
COUNT FOR Age = 300
```

(0 înregistrări)

Interogarea 2:

```
COUNT FOR Age = 300 AND  
(Sex = 'M' OR Diagnosis  
= '650')
```

În acest caz nu se vor crea hărți de biți pentru Sex sau Diagnosis.

Impactul tehnologiei Rushmore asupra stilului

Abia am explicat câteva chestiuni și apar întrebările. Este nevoie să ne schimbăm stilul de programare? Tehnica Rushmore este la fel de utilă în aplicații cu mai mulți utilizatori ca în interogări ad-hoc ale unui singur utilizator?

Între anumite limite, Rushmore permite să se scrie mai puțin riguros și să se obțină, încă, performanțe bune.

Tehnologia este proiectată pentru a extrage seturi de articole. La celălalt pol, comanda FoxPro SEEK este optimizată pentru găsirea unui singur articol. Dacă avem nevoie să găsim un singur articol, un SEEK va fi mai rapid decât LOCATE FOR. Clauza FOR activează Rushmore, care va construi un set de hărți de biți în eventualitatea că dorim să vedem mai mult decât primul articol ce corespunde condiției. Aceasta cere timp. Posibilitatea de a observa diferența de viteză depin-

de de mărimea fișierului raportată la viteza mașinii.

SET ORDER poate pune Rushmore la grea încercare dacă fișierele sînt mari. Tehnologia Rushmore a fost proiectată să creeze hărți de biți neordonate și să returneze răspunsurile rapid, dar SET ORDER forțază FoxPro-ul să revină la indecși și la baza de date pentru a determina ordinea mulțimii rezultat. (Notă: SET ORDER nu afectează operația COUNT sau operația SELECT din SQL).

S-au efectuat teste pe tabele mari (de la 300000 la 2 milioane articole), modificînd numărul de înregistrări din mulțimea rezultat. Pe măsură ce mulțimea rezultat devine mai mică, timpul de execuție crește - uneori chiar considerabil. Această tendință s-a manifestat atît pe sisteme cu memorie multă (24 Mb), cît și pe sisteme cu memorie minimală (640 Kb).

Crearea din mers a unui index condițional temporar pare să dureze mult prea mult ca să merite să fie făcută (Rushmore nu mărește viteza lui INDEX FOR). În unele cazuri pare realizabilă crearea unei copii sortate a mulțimii rezultat cu comanda SELECT din SQL. Cealaltă opțiune ar fi să nu se utilizeze Rushmore în aceste situații, ci să se folosească propriile tehnici anterioare de programare. Totuși, ele ar putea să vi se pară lente după ce ați văzut ce poate face Rushmore în interogări neordonate.

Un ultim avertisment pentru cei neatenți: deschiderea oricărui fișier .IDX va seta implicit ordinea. Pentru a evita o mulțime de apeluri la asistența tehnică în legătură cu o "scamă" în viteza lui Rushmore încercați un USE <numedb> INDEX <numeid> ORDER 0.

O altă plîngere obișnuită survine cînd se încearcă utilizarea lui Rushmore în interogări ce nu sînt optimizabile. Fox definește expresia optimizabilă ca <expresie cu index> <operator relațional> <ex-

presie cu constante>. Un exemplu simplu ar fi:

```
Firstname >= 'Kevin'
```

unde există un fișier index - după cîmpul Firstname - deschis. Dacă avem, totuși, numai un index după Lastname+Firstname, nu se poate optimiza expresia Firstname = 'Kevin'. Este necesar ca Rushmore să găsească un index care corespunde expresiei de interogare. În cazul expresiilor cu caractere, el compară valorile indecșilor caracter cu caracter de la sînga la dreapta. Viteza de căutare în index ar fi puternic redusă dacă ar trebui să caute fiecare octet al indexului pentru a vedea care chei au o anumită valoare "ascunsă" în interiorul lor.

Astfel se explică motivul pentru care nu se poate utiliza curent operatorul de căutare de subșiruri \$. Totuși, FoxPro ar putea să facă o căutare octet cu octet dacă ar folosi indecși după o singură cheie. Aceasta ar reduce volumul de date ce trebuie parcurse într-un fișier de indecși multipli sau într-o bază de date.

S-ar părea că nu mai este necesar să-i atenționăm pe utilizatorii "înraiiți" de FoxPro ca Rushmore nu va crește ca prin farmec viteza desenărilor pe ecran, a manipulării tastaturii, a GET-urilor, a APPEND BLANK-urilor, sau a altor funcții de acest gen, dar aureola ce însoțește tehnologia Rushmore pare să-i fi determinat pe unii să se aștepte la tot felul de însușiri miraculoase. Nu vă amăgiți: Rushmore este mai ales o metodă de căutare și localizare de articole.

Pe ce dăm banii?

Înainte de a trece la interogări mai elaborate, să ne oprim puțin la COUNT. De obicei, demonstrațiile publice cu FoxPro 2.0 includ evidențierea vitezei lui Rushmore executînd COUNT pe baze de date mari. Din nefericire, cei care dezvoltă programe nu utilizează COUNT prea mult. Iată câteva su-

Baze de date

gestii în legătură cu modul în care se poate profita de avantajele celei mai rapide comenzi Rushmore.

Tabelele încrucișate de frecvență constituie o primă estimare foarte bună atunci când ne analizăm datele. Ele ne ajută să "simțim" numerele și să localizăm rapid potențiale nepotriviri în date. Comanda

```
SELECT  Diagnosis,
COUNT  (*)
FROM    Hospital
GROUP  BY  Diagnosis
```

ne poate ajuta să evităm ore de depanare ce ar putea surveni ulterior. Variațiuni pe această temă sînt:

```
SELECT  Diagnosis
DISTINCT FROM Hospital
sau
SELECT  COUNT(DISTINCT
Diagnosis)
FROM    Hospital
```

Dacă aveți o bază de date cu un inventar, poate fi mult mai rapid să

faceți un COUNT pentru toate înregistrările "vîndute" unui client decît un SUM pentru subtotalurile dintr-o bază de date cu comenzi. De fapt, toate celelalte comenzi care utilizează Rushmore impun citirea din baza de date la un anumit moment. Nu vă așteptați ca ele să fie la fel de rapide ca și COUNT, care - de obicei - citește numai indexul.

Unii programatori au sugerat că bătrîna strategie manuală SEEK / WHILE va fi mai rapidă decît o condiție FOR optimizată ad hoc. Aceasta era valabil pentru versiunile anterioare ale programelor de test pentru FoxPro 2.0. Dar vremurile s-au schimbat. S-au testat diferite proceduri de extragere a datelor variind spațiul de memorie, înregistrările extrase, gradul de dispersie a datelor și modul de sortare al datelor de ieșire. Rezultatele pot fi văzute în tabelul 1.

În general, SEEK/WHILE este optim numai atunci cînd se știe că mulțimea obținută ca rezultat va fi mică. În toate celelalte cazuri Rus-

hmore pare să fie la fel de bun, sau chiar mai bun. În unul din cazuri - un rezumat (AVERAGE, SUM, MAX) al unei submulțimi mari în condițiile unui spațiu de memorie redus - a fost cu un ordin de mărime mai rapid. Altfel, viteza a fost numai de două sau trei ori mai mare.

Dacă nu se pune problema performanței, utilizarea construcției FOR de către Rushmore face ca programarea și depanarea să fie mai simple decît pentru SEEK/WHILE, care cere urmărirea stării curente a lui SET ORDER în cod. De asemenea, construcția FOR nu are corespondent SEEK/WHILE atunci cînd există OR în condiție (de exemplu nu există căi de a crea un index concatenat pentru a căuta Field1 = x OR Field2 = y).

Rushmore și SQL vor face ca și programarea relațională să fie mai atrăgătoare. Sistemele complexe ce depind de date redundante și totaluri de rulare pot fi înlocuite cu baze de date normalizate mai simple. SUM-urile executate "din zbor" vor fi, uzual, mult mai ușor de făcut.

Aceasta ne îndeamnă să studiem în continuare o altă noutate Fox: interogări care citesc numai fișierele index, dar nu și bazele de date. În mod obișnuit, operațiunile COUNT și SELECT <fields> DISTINCT sînt singurele operațiuni ce rulează cu viteze mici. O altă interogare uzuală ce ar putea beneficia de citirea exclusivă a fișierelor index este sumarea pe un singur cîmp. De exemplu:

```
SUM Charges For Charges
>1000
```

cere numai citirea unui index al aceluși cîmp, fără să se atingă de baza de date.

RQBE - SQL-ul lui FoxPro 2.0 și Rushmore constituie o combinație excelentă pentru analiza datelor. Se pot crea și executa interogări atît de repede și de simplu, încît ne putem permite să fim creativi. Acolo unde, poate, v-ați decis să nu mai explorați noi zone deoarece ar fi durat prea mult, puteți acum să simulați diverse scenarii. Deoarece FoxPro salvează rezultate interme-

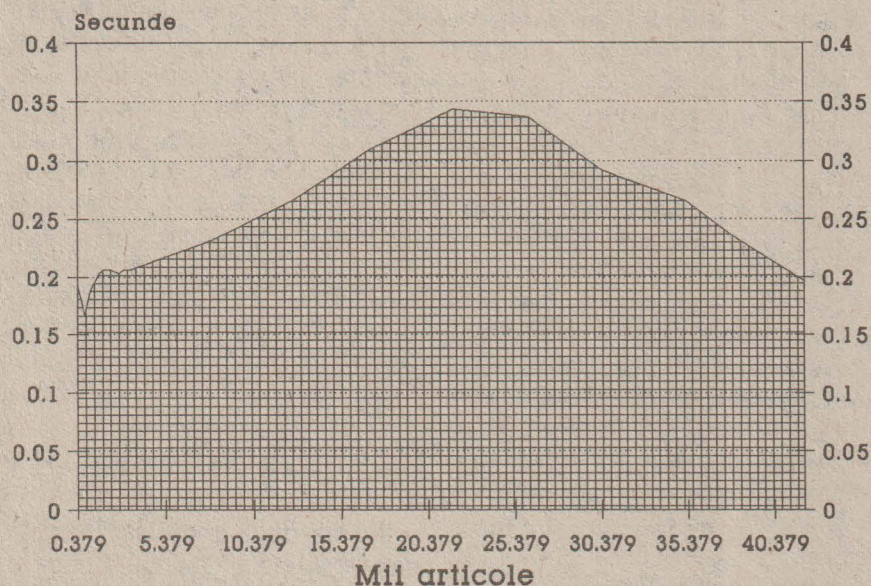


Figura 1. Operațiunea lui FoxPro 2.0 de căutare în index pentru construirea unei hărți de biți este foarte eficientă. După găsirea a aproximativ jumătate din înregistrări, timpul de execuție scade.

diare și are un "cache" excelent, se pot rafina întrebările - pentru un număr dublu de interogări nu este nevoie de un interval dublu de timp.

Cîteva obiecții

Dacă bazele de date sînt utilizate atît pentru interogări ad-hoc cît și pentru tranzacții, este posibil să nu se dorească indexarea fiecărui cîmp, deși aceasta ar permite accelerarea interogărilor de către Rushmore. De fiecare dată cînd se modifică sau se adaugă un articol, FoxPro are de efectuat suplimentar actualizarea acelor indecși. Aceasta este o decizie ce se ia în funcție de situație.

Pentru un sistem de colectare de date, indecșii construiți pe măsură ce se adaugă articole unul cîte unul pot deveni foarte fragmentați. O anume valoare cheie care nu este unică poate fi localizată într-o regiune puternic dispersată a unui fișier index. Valorile distribuite aleator în fișierul bază de date și în fișierul index fac "viața grea" pentru hard disc.

Fragmentarea indexului este o chestiune controversată numai dacă este total ignorată. Timpul necesar pentru ca Rushmore să creeze o hartă de biți crește spectaculos dacă arborele index devine foarte fragmentat. Dacă se folosesc indecși compuși, problema se multiplică. Cel mai simplu leac este indexarea periodică a bazei de date.

Utilizarea intensă a lui Rushmore sau adăugările frecvente de pachete de articole pot face ca indexarea să devină nefolositoare. Dacă nu se poate face reindexarea suficient de des, se poate lua în considerare împărțirea indexului compus în mai multe fișiere compacte de index după o singură cheie. Astfel se pierde avantajul important al indecșilor structurali compuși (noutate în FoxPro) deschiși automat împreună cu baza de date.

De asemenea, Rushmore nu este direct sensibil la SET DELETED

Tabela 1

Rushmore contra SEEK / WHILE

Tabela 1.A Rushmore contra SEEK/WHILE pe date dispersate

	Memorie multă	Memorie minimală
Mulțime rezultat mică	Egalitate	SEEK/WHILE
Mulțime rezultat medie	RUSHMORE	Egalitate
Mulțime rezultat mare	RUSHMORE/Egalitate*	RUSHMORE

Tabela 1.B Rushmore contra SEEK/WHILE pe date ordonate

	Memorie multă	Memorie minimală
Mulțime rezultat mică	SEEK/WHILE	SEEK/WHILE
Mulțime rezultat medie	Egalitate	Egalitate
Mulțime rezultat mare	Egalitate*	RUSHMORE/Egalitate*

* Rushmore are avantajul de a extrage datele în ordine fizică

În general, utilizarea lui SEEK/WHILE este optimă numai atunci cînd se știe că mulțimea rezultat va fi mică. În toate celelalte teste Rushmore pare să fie mai bun sau egal.

ON. Deoarece starea DELETED este memorată în articol, ea este în esență un alt cîmp. Se poate crea un index după acel "cîmp" indexînd după funcția DELETED(). În multe aplicații, o soluție mai bună este să se pună blank fără a mai fi necesar să se adauge NOT DELETED() în toate interogările.

Rushmore în Rețea

Lucrul cu date partajate într-o arhitectură file server - cum ar fi FoxPro LAN, a fost dintotdeauna o decepție după ce ați văzut ce performanțe atinge FoxPro cu ceva memorie pentru "cache". Problema fundamentală este cum poate utilizatorul de la o stație de lucru să actualizeze o copie a articolelor sale care se află în memoria "cache" de pe o altă stație. Pentru toate bazele de date file server de pe piață răspunsul este: nu poate. Pentru toate bazele de date care

utilizează această arhitectură soluția este să nu utilizeze "cache" pentru datele partajate. La prima vedere, s-ar părea că Rushmore trebuie să urmeze aceleași reguli, devenind ineficace în LAN.

Dar nu de data aceasta. Fox folosește "cache" pentru hărțile de biți Rushmore și indecșii folosiți la construirea lor, și, pur și simplu, "simte" că un alt utilizator modifică un index. Dacă se dă ulterior o comandă care folosește o hartă de biți, se parcurg din nou indecșii modificați. De exemplu, dacă o hartă de biți reprezintă toate articolele care satisfac (Age > 10) și cineva modifică valoarea unei anumite înregistrări din 4 în 5, harta de biți nu se va schimba - și totuși se va face o nouă parcurgere. Fenomenul poate fi observat comparînd timpii de execuție.

Această abordare ne poate determina să preferăm să avem mai multe fișiere index în locul unui singur fișier cu indecși multipli pentru

Baze de date

a obține performanțe optime pentru Rushmore în rețea. De ce să permitem ca modificarea unei singure chei să forțeze o nouă parcurgere a numeroșilor indecși memorați într-un fișier de indecși multipli utilizați într-o interogare Rushmore? Este mai bine să limităm noua parcurgere la o singură cheie prin păstrarea fiecărei chei în propriul ei fișier index.

Indecși în general

Într-adevăr, studierea modului în care Rushmore realizează - în general - parcurgerea indecșilor conduce la concluzia că este mai bine să se folosească fișiere index individuale și compacte.

Pentru a se testa modul în care Rushmore parcurge indecșii, s-a plasat un fișier index pe un disc proaspăt formatat și s-a observat afișajul Smartvu care indică pistele la care se face acces. Apoi s-a testat comanda COUNT FOR pe diverse tipuri de indecși pentru a vedea cât se parcurge din index și în ce ordine.

În principiu, există trei tipuri de indecși în FoxPro 2.0. Primul tip testat a fost un fișier .idx de tip FoxPro 1, nefragmentat. Comanda COUNT FOR Fld1 = 'A' a parcurs întregul index deși nodurile asociate acestei valori erau situate la începutul fișierului. Căutarea a fost, în cea mai mare parte, secvențială cu câteva salturi în diferite secțiuni ale fișierului.

Al doilea test a avut în obiectiv un fișier .idx FoxPro 2.0 nefragmentat. Acest test nu a citit fiecare pistă și a făcut acces la piste aproape secvențial.

FoxPro 2.0 are implementate și fișiere de indecși multipli (.cdx). Un fișier .cdx poate conține mai mulți indecși, numiți etichete; etichetele .cdx sînt indecși compactați. Acest ultim test a folosit o bază de date de coduri poștale cu un fișier .cdx indexînd toate cele trei cîmpuri: cod, oraș și stat. COUNT FOR Cod = '3' a parcurs numai porțiunea care

conținea eticheta cod poștal a fișierului .cdx. Din nou, parcurgerea a fost - în majoritate - secvențială, dar a sărit unele piste și a revenit la ele mai tîrziu.

Cînd indecșii sînt fragmentați, procesul de căutare nu mai este secvențial. În cazul fișierelor .cdx, FoxPro nu a mai căutat numai în prima parte a fișierului. Deoarece eticheta era dispersată de-a lungul întregului fișier, Rushmore a trebuit să facă salturi mari. Uneori a trebuit să sară înainte și înapoi între piste aflate la mare distanță una de alta. Din acest motiv - și din altele, este bine să se utilizeze fișiere separate de indecși compacti.

Din motive de performanță, Fox ar trebui să adauge facilitatea de deschidere automată a fișierelor index .cdx și la indecșii compacti separați. Într-un singur antet s-ar putea memora toate numele indecșilor compacti separați care ar trebui deschiși automat.

Concurență

Rushmore are cîteva neajunsuri mai deranjante decît eșecul accidental în furnizarea performanțelor maxime. Dacă a fost construită o hartă de biți Rushmore pentru o comandă ca BROWSE FOR, este posibil ca ea să nu mai fie evaluată dacă alți utilizatori modifică datele care intră sub incidența criteriului de selecție. O soluție ar putea fi utilizarea unei comenzi SHOW WINDOW <nume fereastră browse> REFRESH pentru a actualiza fereastra BROWSE. Dar acest lucru nu se face automat. Din acest motiv unii vor prefera să folosească, pentru fișierele partajate, comanda BROWSE KEY care este mai puțin flexibilă (și care nu utilizează Rushmore).

O problemă similară se manifestă și pentru SCAN FOR / ENDSCAN. Mulțimea rezultat Rushmore se crează atunci cînd se întîlnește primul SCAN FOR în program și ea nu va mai fi modificată,

indiferent de ce se petrece în interiorul buclei SCAN. În orice situație în care mulțimea rezultat obținută cu Rushmore poate fi afectată în timpul executării comenzii, fie se va utiliza clauza NOOPTIMIZE în comandă pentru a dezactiva Rushmore, fie se vor utiliza metode tradiționale - cum ar fi SEEK/WHILE - pentru a rezolva problema.

Concluzii

Operațiunea de explorare a indexului în vederea construirii unei hărți de biți este extrem de eficientă. După cum se vede din Figura 1, după ce s-au găsit circa jumătate din articole, timpul de execuție scade. Aceasta sugerează că Fox determină, la începutul explorării arborelui index, care cale (condiția căutată sau inversa ei) va conduce mai repede la răspuns. Mai mult, timpul necesar pentru a efectua un COUNT FOR pentru un număr dublu de articole nu este de două ori mai lung. Noile tipuri de indecși scurtează acest timp mai mult decît vechile fișiere .idx.

Dacă nu vă lăsați ademeniți să faceți mai multe cu Rushmore, veți putea observa un trafic mai redus în rețele. Și totuși, este de presupus că mulți vor fi seduși de performanțele unor proceduri pe care nu le-au mai încercat pînă acum și vor mări traficul. Rămîne de văzut în ce măsură se poate implementa un FoxPro 2.0 pe un sistem mare cu mai mulți utilizatori care să poată trata atît procesarea de tranzacții, cît și interogările ad hoc.

Utilizat corect, Rushmore poate rezolva probleme pe care puțini le-au crezut realizabile pe un PC. Mai trebuie încă să se lucreze la finisarea programelor în domeniul performanțelor de viteză, dar mult mai puțin ca înainte. Probabil că nu există un alt produs mai rapid pînă cînd ne hotărîm să dăm mult mai mulți bani pe sisteme de gestiune a bazelor de date pe mini și/sau mainframe.

(C. N.)

BAZE DE DATE DISTRIBUITE

Distribuirea datelor constituie o idee atrăgătoare din mai multe motive. Ea oferă posibilitatea de a exploata MIP-urile ieftine de pe PC-uri, stații de lucru și borduri UNIX, de a putea configura operațiunile pe computer ale organizațiilor care au birouri răspândite în țară. În pofida acestor avantaje, cel mai bun sfat care poate fi dat utilizatorilor de calculatoare care au în vedere o abordare distribuită a bazei de date este - de obicei - "Nu faceți așa ceva".

Totuși, temeinicia acestui sfat depinde de ce se intenționează să se facă. "Baza de date distribuite" are înțelesuri diferite pentru diferiți oameni.

Pe de o parte, cei care vînd sisteme de gestiune a bazelor de date pot să pretindă în ofertă posibilități de distribuire pur și simplu pentru că sînt permise citirea și accesul pentru actualizare de la distanță într-o rețea, în timp ce, pe de altă parte, teoreticienii insistă asupra faptului că adevărata distribuție este asigurată numai dacă aplicațiile și utilizatorii pot trata o bază de date ca un singur depozit logic de date, indiferent de gradul de împrăștiere în rețea.

Între aceste două poziții, se găsesc diverse nivele de satisfacere, prezentînd opțiuni utile și diverși coeficienți de risc.

Citiri și actualizări la distanță

Aproape toți cei care comercializează sisteme de gestiune a bazelor de date oferă facilități de citire și de acces pentru actualizare de la distanță ca un prim pas către distribuția bazelor de date. Această facilitate permite unui utilizator să manipuleze de la distanță tabele într-o rețea, dar accesul poate fi limitat la o bază de date la un anumit moment de timp în cadrul aceleiași tranzacții. Trebuie să remarcăm că există două moduri de gestionare a accesului la date într-o rețea. Primul mod constă în controlul local, cu fiecare acces fizic efectuat prin rețea. Acest mod este ineficient și poate duce la timpuri de răspuns inacceptabili, din cauza consumului de timp al rețelei care se adaugă celui necesar fiecărui acces fizic.

O altă soluție este controlul accesului la baza de date către un proces "server" de pe mașina situată la distanță, care procesează un acces logic - cum ar fi o interogare SQL, și întoarce rezultatul prin rețea. Acest mod de operare este esențial pentru bazele de date

distribuite și cele mai multe produse funcționează astfel.

Accesul la distanță face posibile configurațiile "client-server", deși aceste configurații nu implică în mod necesar distribuția datelor. Avantajul evident este că serverele de date pot fi ajustate pentru asigurarea unei deserviri rapide a datelor. Dacă un singur server de date se dovedește insuficient, se pot adăuga altele, și astfel se face primul pas către baze de date distribuite.

Baze de date distribuite

Imediat de există mai mult de o bază de date, viața începe să devină mai complicată, chiar dacă ele se găsesc pe același calculator. Este foarte posibil ca, de exemplu, o tabelă cu clienți să fie ținută într-o bază de date și o tabelă cu comenzi într-o alta. De aceea ar fi rațional să se unească cele două tabele pentru a lista clienții și comenzile pe care le-au depus. Surprinzător, nu toate produsele de baze de date suportă aceasta, deși majoritatea o permit inclusiv produsele cele mai utilizate ca Ingres, Sybase și Oracle.

Totuși, dacă se amestecă diferite produse de baze de date, este din ce în ce mai puțin posibil să se poată realiza astfel de "uniri". În această situație devenim dependenți de 4GL pentru a putea trata problema via "porțile" pe care le oferă către diverse baze de date. În orice caz, dacă intenționați să implementați o bază de date distribuită, este bine să nu amestecați produsele de baze de date deoarece fiecare oferă posibilități diferite și concilierea lor poate deveni un coșmar.

Prima regulă a distribuirii datelor este că datele se vor distribui numai pentru motive de viteză. Este mult mai bine să existe o singură bază de date care servește toate aplicațiile rulate de o organizație. Totuși, aceasta nu este totdeauna practic, deoarece chiar și mașinile foarte puternice, cum ar fi Model 204 de la Computer Corp of America, sînt limitate ca volum de date și număr de tranzacții pe care le pot manipula. De asemenea, mai există și o limită a puterii calculatorului.

Primul pas este, de obicei, să se divizeze resursa masivă de date în cîteva baze de date astfel încît nici o aplicație să nu aibă nevoie să apeleze mai mult de o bază de date, cu excepția unor accesări ocazionale de citire. Aceasta este strategia utilizată de multe amplasări mari și ea produce rareori probleme.

Cu toate acestea, unele instalări au nevoie să distribuie datele într-un mod mai sofisticat. De exemplu, o companie de distribuire poate avea câteva depozite în locuri diferite, fiecare fiind servit de un computer local. Într-un asemenea sistem, tranzacțiile cu mișcările stocurilor pot cuprinde mai multe calculatoare. În acest caz, poate să existe o necesitate autentică pentru o bază de date distribuită.

Idealul pentru bazele de date distribuite este să se lucreze logic cu o singură bază de date, eventual răspândită peste mai multe noduri ale rețelei. Implementarea va consta în câteva baze de date fizice, dar din punctul de vedere al utilizatorilor și al aplicațiilor există o singură bază de date. Atingerea acestui ideal nu este deloc simplă, iar interogările și actualizările pot ridica probleme semnificative.

Interogări distribuite

Atunci când există două baze de date în noduri diferite ale unei rețele și când o interogare unește tabele din fiecare din ele, softul de gestionare trebuie să decidă unde să "gospodărească" tabela rezultat. Unirea a două tabele va genera o tabelă virtuală, și întrebarea care se pune este unde să se depună această tabelă virtuală. Este posibil, de exemplu, să se selecteze două articole dintr-o tabelă și cinci mii din cealaltă, astfel încât prin rețea vor călători fie două articole, fie cinci mii. În mod evident, este mai bine să se transmită numai două, dar decizia trebuie să o ia softul de gestiune.

Aici se pune problema unei optimizări distribuite și foarte puține produse oferă vreo sofisticare în această zonă. Cele câteva care o fac includ Ingres, Informix și NonStop SQL al lui Tandem, dar chiar utilizarea unuia din aceste produse nu rezolvă în mod necesar problema. Dacă există mai multe baze de date și mai multe noduri implicate, problema se compune. Dacă bazele de date sînt distribuite fără a ține cont de interogările ce ar putea fi executate, atunci este foarte posibil ca performanța să devină inacceptabilă indiferent de flexibilitatea optimizării interogărilor. O posibilitate ar fi duplicarea unor baze de date sau a unor submulțimi ale bazelor de date în diferite noduri ale rețelei, astfel încât interogările să fie nevoite să cuprindă mai multe noduri cît mai rar. Atît timp cît softul de control știe de existența acestor "server-e de interogare", problema se reduce.

Totuși, duplicarea datelor trebuie planificată adecvat, astfel încât datele duplicate să poată fi actualizate ori de cîte ori este necesar. În funcție de sistem, este necesar ca datele duplicate să fie la zi, sau poate fi acceptabil ca ele să fie cu o zi "în urmă". În ambele cazuri duplicarea datelor trebuie gestionată adecvat

și, în mod evident, este de dorit să existe posibilitatea duplicării automate. Din nou, există numai cîteva produse care asigură această necesitate, printre care Adabas, Datacom:DB, Rdb și NonStop SQL.

Posibilitățile de duplicare ar trebui să includă planificarea momentului efectuării duplicării și capacitatea de a transmite numai datele modificate pentru a reduce la minimum traficul în rețea. Pe lîngă utilitatea în realizarea interogărilor, utilizarea acestei metode mai prezintă și alte avantaje. De exemplu, într-un sistem de vînzare cu amănuntul ar putea folosi la transmiterea săptămînală sau zilnică a listelor de prețuri curente. De asemenea, ar putea fi util la extragerea de date pentru spreadsheet-uri, programe de grafică, și altele.

Actualizări distribuite

Dacă, dintr-un motiv sau altul, o interogare distribuită eșuează, aceasta nu va afecta integritatea datelor. Dar dacă eșuează o actualizare distribuită, afirmația anterioară nu mai este adevărată. Pentru a asigura integritatea datelor în cadrul actualizărilor distribuite, este necesar să se asigure executarea în două faze. Acest proces se asigură că toate bazele de date implicate într-o tranzacție sînt gata să recepționeze datele, efectuează tranzacția, apoi verifică dacă actualizarea s-a efectuat corect în toate bazele de date implicate în proces. Dacă, dintr-un motiv oarecare, actualizarea a eșuat în unul dintre noduri, se va șterge tranzacția, revenind la situația anterioară. Există puține produse care oferă o execuție în două faze, printre care Datacom: DB, Rdb, Ingres, Empress, Sybase, iar mulți alți producători sînt hotărîți să ofere și ei această facilitare.

Totuși, faptul că este disponibilă o astfel de posibilitate, nu ar trebui să fie o încurajare în implementarea de tranzacții distribuite. Sistemele care se întind peste mai multe noduri sînt vulnerabile la căderea oricăreia dintre ele și sînt, deci, mai puțin sigure. De asemenea, tranzacțiile care implică mai mult de un nod al unei rețele sînt mai lente decît cele care accesează date locale și există posibilitatea - cea mai rea dintre toate - de apariție a unor blocaje globale.

Un blocaj global este un conflict de blocare ("zăvoîre") a înregistrărilor care cuprinde mai mult de un nod al rețelei. Aceasta este o situație asemănătoare unei îmbrățișări mortale, în-care două tranzacții nu se pot termina deoarece s-au "zăvoîrit" una pe alta. Fenomenul poate avea un efect vast, paralizînd cîteva din nodurile rețelei. Din aceste motive, tranzacțiile distribuite ar trebui să fie folosite numai acolo unde nu există o altă opțiune satisfăcătoare.

Transparență

Din ceea ce am spus pînă acum, s-ar părea că implementarea unei baze de date distribuite este realizabilă dacă se dispune de optimizare distribuită, duplicare automată și execuție în două faze a actualizărilor. Acest lucru este aproape adevărat, dar există o altă caracteristică care este utilă - alții ar spune chiar imperios necesară - transparența.

Transparență înseamnă că aplicația nu este nevoită să cunoască locul unde sînt plasate fizic datele. Transparența completă a localizării datelor implică un S.G.B.D. capabil să gestioneze traseele actualizărilor în cadrul rețelei și capabil să recupereze totul în urma eșecului în oricare din noduri. Nu există pe piață produse capabile să satisfacă aceste cerințe, în special din cauza condițiilor de comportare în caz de eșec.

Oricum, structura rețelei și amplasarea fizică a bazelor de date pot fi definite ușor, iar cererile de date pot fi "cartografiate" la definirea lor, ceea ce asigură un grad rezonabil de transparență a amplasării. Această posibilitate este prezentată la cîteva produse din care amintim Ingres, Sybase, Informix, Unify și DB2.

Avantajul transparenței amplasării constă în posibilitatea de a reorganiza datele în rețea sau de a modifica chiar rețeaua fără a fi nevoie să se adapteze aplicația. Probabil că astfel de manevre nu se fac frecvent, dar dacă ori de cîte ori este nevoie de astfel de modificări se ajunge la schimbări în codul programului, devin evidente virtuțile unui anumit nivel de transparență.

Aspectul - probabil - cel mai curios al bazelor de date distribuite în constituie competiția vînzătorilor de a le oferi. Este ca și cînd ar exista mii de organizații care își doresc cu disperare să își distribuie datele și care de-abia așteaptă apariția produsului potrivit. Și totuși, nu este cazul.

Cele mai multe firme care trec dincolo de o simplă configurație client-server, către o bază de date complet distribuită, sîrșesc prin a regreta acest paas și a reveni în scurt timp. De aceea o recomandăm rareori. Este mai ușor să nu înveți pe calea cea mai grea.

N. C.

Cele douăsprezece porunci pentru baze de date distribuite

Cele douăsprezece "porunci" pe care trebuie să le îndeplinească o bază de date distribuită transparent - enunțate de C. J. Date - amintesc de versurile celebrului cîntec "Crazy He Calls Me" al lui Billy Holiday:

Ce-i greu o să fac chiar acum;

Ce-i imposibil o să dureze puțin mai mult.

Vă prezentăm în continuare enunțul celor douăsprezece legi așa cum au fost ele enunțate de Date. Care este părerea dumneavoastră ?

- ❑ Regula 1. Autonomia locală: Datele locale sînt deținute și gestionate local, inclusiv responsabilitatea și securitatea: nici un post nu depinde de altele pentru a funcționa.
- ❑ Regula 2. Toate posturile sînt egale: Nici un post nu se bazează pe un post central.
- ❑ Regula 3. Funcționare continuă: O oprire planificată nu trebuie să fie niciodată necesară, deci instalările efectuate la un post nu afectează funcționarea celorlalte; bazele de date pot fi create sau distruse fără a opri vreo componentă.
- ❑ Regula 4. Transparența amplasării: Utilizatorii nu au nevoie să știe unde sînt amplasate datele pentru a le extrage.
- ❑ Regula 5. Transparența fragmentării: Relațiile între elementele datelor pot fi fragmentate pentru stocare, dar acest lucru este transparent pentru utilizator.
- ❑ Regula 6. Transparența duplicării: Relațiile și fragmentele sînt reprezentate fizic prin copii multiple, stocate separat, dar transparent pentru utilizator.
- ❑ Regula 7. Procesarea interogărilor distribuite: Activitatea de intrare/ieșire se poate desfășura la mai multe posturi, permițînd optimizarea interogărilor atît local cît și global.
- ❑ Regula 8. Actualizări distribuite: Tranzacțiile singulare pot executa codul la mai multe posturi.
- ❑ Regula 9. Independența hardware: Toate mașinile participă ca parteneri egali.
- ❑ Regula 10. Independența față de sistemul de operare: Sînt suportate mai multe sisteme de operare.
- ❑ Regula 11. Independența față de rețea: Sînt suportate mai multe rețele.
- ❑ Regula 12. Independența față de bazele de date: Se asigură suportul pentru diverse baze de date cu interfețe de același fel.

O privire asupra graficii animate

O privire rapidă și atentă asupra spoturilor publicitare și videoclipurilor actuale ne arată că animație pe calculator și-a făcut deja intrarea în afacerile zilei. Oricît de ușor ar părea rezultatul final, construcția fiecărei imagini în parte într-un sistem grafic necesită mult timp, este foarte complicată, și de aceea foarte scumpă.

Odată cu interfața Renderman se naște un nou standard. Articolul ce urmează oferă o privire de ansamblu asupra tehnicilor fundamentale și tendințelor actuale din domeniu.

Apariția unei noi tehnici pentru grafica computerizată, Rendering-ul, va influența designul viitoarelor calculatoare, sisteme economice și aplicații. Rendering-ul constă în diverse tehnologii: programe de tip C.A.D. și efectele speciale din industria de film. Rendering-ul este cu adevărat pretențios: chiar și facilitățile de excepție ale stațiilor de lucru cu unități centrale rapide; cu procesoare de virgulă mobilă, cu accelatoare de grafică și alte extensii par a fi atunci cînd pe ele se rulează un program de Rendering, extrem de lente. Programele actuale de Redering creează imagini grafice deosebite, în timp ce software-ul mai are un drum lung de parcurs.

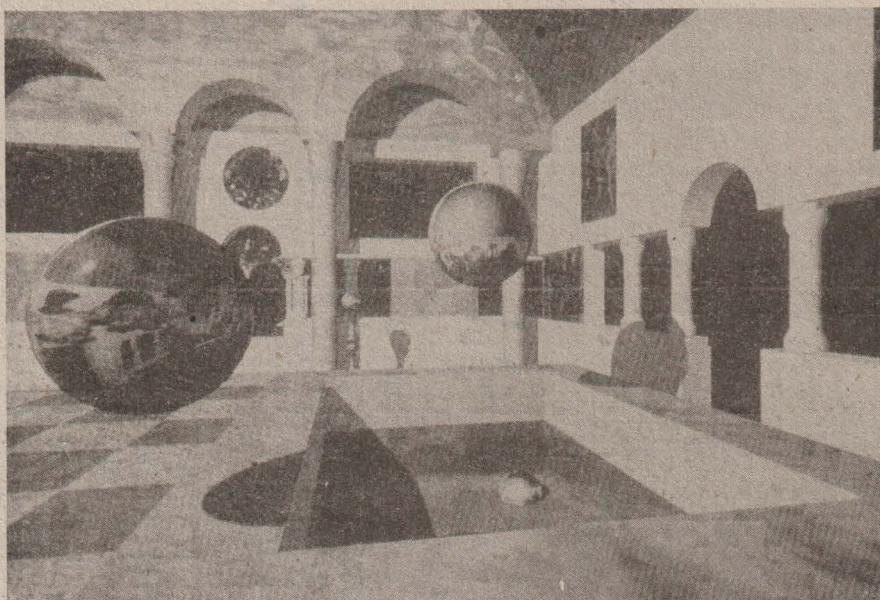
Aplicațiile grafice de astăzi vor fi folosite în programe de CAD a căror structură de bază va fi constituită din imagini. Programele de Rendering fac posibilă utilizatorului controlarea amănunțită a compoziției unei imagini: umbre, saturarea culorilor, efecte de lumină, culori, reflexii și alte elemente.

Rendering-ul se diferențiază de "modelarea solidului", dar ambele metode sînt strîns legate între ele și se dezvoltă în paralel. Scopul original al CAD-ului era desprinderea procesului de calcul de procesul de construcție al unei clădiri sau de finisare industrială a unui obiect. De-a lungul timpului a ieșit în evidență că introducerea formațiilor complete, tridimensionale despre un obiect mărește flexibilitatea algoritmilor de calcul și astfel face posibilă folosirea calculatorului pentru a vedea cum obiectele interacționează între ele.

Pentru a construi o imagine realistă calculatorul are nevoie de informații foarte detaliate asupra pro-

prietăților suprafețelor exterioare ale obiectelor și de o descriere suficientă a luminilor și a informațiilor de de fundal, informații de care "modelarea solidă" nu ține cont.

Cererea din ce în ce mai mare de fidelitate a imaginii va conduce la folosirea calculatoarelor în artă și în industria de film. Diferența față de "modelarea solidă" este că obiectele sînt înconjurată de lumini, umbre și alte elemente de compoziție. Prin colaborarea cu oamenii de știință ce studiază grafica pe calculator, constructorii au făcut deja pași importanți în crearea unor imagini cît mai naturale. Programele CAD



posedă tehnologia fundamentală de care au nevoie cercetătorii și ilustratorii, dar le lipsește facilitatea de a poziționa și dimensiona corect obiectele.

Imagini fotorealiste

Există o serie de motive care fac Redering-ul interesant. Rendering-ul este pentru grafica pe calculator ceea ce a fost suprealismul pentru arta abstractă. Tehnologia a ajuns la un punct de convergență; oamenii de știință și artiștii lucrează împreună la experimente ce se pot concentra în mai multe domenii pentru a construi algoritmi grafici puternici și eficienți: ca răspuns la cererea anterioară de sisteme de tip Rendering, industria de calculatoare a început să construiască sisteme asemănătoare cu sistemele Mainframe și introducerea puterii grafice în server-e și

a le da în mâinile utilizatorilor evoluți. Luate împreună algoritmi, hardware-ul și presiunea pieței de CAD au dus la dezvoltarea explozivă din domeniul Rendering-ului. Prin Rendering o imagine devine într-un mod simplu o imagine plină de conținut. Principala muncă în domeniul "modelării solide" constă în dezvoltarea modelelor simple, schematice, și apoi atașarea oricărui obiect component a unei culori unice. Rendering-ul transformă imaginea CAD în imagini grafice fotorealiste. Utilizatorii programelor de Rendering au în principal două probleme: hardware-ul și modul de descriere al graficii. Renderingul funcționează cel mai bine cu un hardware de 24 biți la care, pentru obținerea celor trei culori fundamentale, roșu, verde, albastru, se rezervă câte 8 biți (la o reprezentare color pe 32 biți am mai avea disponibili încă 8 biți pentru informații de control). Pentru rulări simple sînt necesari cel puțin 4 MB de memorie iar pentru rulări complexe necesarul devine de 8 MB. Unitățile centrale rapide (cel puțin 25 MHz) și coprocesoarele grafice ajută la accelerarea procesului.

Dar chiar și pe cele mai puternice sisteme timpul rămîne un factor restrictiv. E nevoie de cîteva minute pentru un obiect ce ocupă doar a opta parte din ecran. Anexarea poligoanelor sau întrebuițarea Anti-Aliasing-ului mărește timpul necesar construcției unei imagini complete. Poate dura 20 sau 30 min construcția unei imagini care este "zidită" de la bază și pînă în momentul în care utilizatorul poate determina dacă lumina sau culoarea sînt bine plasate. Modificări minime în poziția unei imagini pot duce la modificarea proporțiilor, ceea ce mărește tentația și necesitatea de a experimenta din nou. Timpul necesar construirii unei imagini poate îngredi foarte mult și chiar îndepărta utilizatorul. Doar cei ce folosesc sisteme foarte puternice au posibilitatea reală de a experimenta. Toți ceilalți pot face acest lucru doar în cadrul unor sesiuni limitate în timp.

Principalele programe CAD pot da imaginilor prin această metodă un efect tridimensional, prin aceea că suprafețelor mari li se pot atașa texturi diferite sau culori diferite. Prin Rendering suprafețele exterioare sînt împărțite în părți foarte mici, ce pot fi umbrite individual.

Deși poate fi important ca un singur obiect, de exemplu un automobil, să poată fi considerat ca fiind compus din mai multe obiecte luate împreună, la fel ca și în cazul scenelor interesante (un automobil rulînd pe o stradă, un elefant fugind printr-un ținut de porțelan) reflexele de lumină ale obiectelor singulare în astfel de scene sînt complicate și foarte greu de construit corect. Transpunerea unui scenariu tridimensional într-o imagine bidimensională ridică și alte probleme de genul:

- care părți ale obiectelor se văd în imaginea construită?
- ce culoare au părțile vizibile ale fiecărui obiect?

La rezolvarea acestor probleme se folosesc două tehnici principale: tehnica Z-buffering și tehnica Ray-tracing. Metoda cea mai simplă de a lucra cu probleme de genul "ce se va vedea?" este metoda Z-buffering cunoscută și ca metoda "scan-line". În tehnica Z-buffer obiectele sînt prelucrate unul după celălalt. Pentru fiecare pixel separat algoritmul Z-buffer "privește" înapoi în scenariu pentru a vedea dacă obiectul studiat poate fi văzut din acest punct. Dacă poate fi văzut apare "verificarea Z-buffer". Z-buffer-ul este de fapt o zonă de memorie în care pentru fiecare pixel al construcției se păstrează imaginea următoare pînă cînd obiectul este complet construit. Cînd obiectul curent se află în apropiere se ia în considerare că toate imaginile construite anterior vor fi umbrite de acesta. Umbrirea este de fapt procesul de alegere a culorii pe care o va avea punctul dat. Există trei metode principale de umbrire care se folosesc împreună cu Z-buffering-ul:

- ☒ umbrirea Lambert - cea mai simplă și mai rapidă tehnică. Obiectul care va trebui umbrît este împărțit în poligoane și fiecărui poligon i se atribuie o culoare în funcție de proporțiile lui și de poziția față de sursa de lumină. De aici rezultă o reprezentare cu fațete:
- ☒ umbrirea Gourand - lucrează tot cu poligoane. Culoarea va fi împărțiată pe toată suprafața unui poligon, pornind din colțuri. Procesorul de mediere a culorii îmbunătățește rezultatul dar prelungește semnificativ timpul de lucru.
- ☒ umbrirea Phong - se calculează lungimea suprafeței exterioare a obiectului tridimensional relativă la poligon și se umbrește cu această valoare. Cu această metodă se obțin rezultate și mai bune decît cu metoda Gourand dar se înrăutățește și mai tare timpul de răspuns.

Umbrirea costă timp

Fiecare tehnică are slăbiciunile ei. Umbrirea Lambert se vede întotdeauna că e generată pe calculator. La umbrirea Gourand acest lucru nu mai e atît de vizibil dar se mai pot încă recunoaște urmele unora dintre poligoane, mai ales în zonele cu un grad mare de curbare. Chiar și la umbrirea Phong se mai cunosc marginile poligoanelor dar acest lucru poate fi evitat prin specificarea unui număr mai mare de poligoane în perimetrele curbate puternic, ceea ce va duce la o construcție de lungă durată. Z-buffering-ul este o tehnică de forță brută care conduce la imagini artificiale. Cînd fiecare obiect este generat separat umbrele, reflexiile unui obiect asupra celorlalte transparența,

răămîn nefolosite. Prin urmare Anti-Aliasiny-ul este greu de obținut. Această artificialitate a reprezentării științifice nu reprezintă un impediment pentru că foarte des este nevoie de descrierea împreună a unor obiecte separate și interacțiunea acestora și mai puțin de obținerea unei imagini realiste.

Pe de altă parte Z-buffering-ul oferă facilitatea de a avea un control puternic asupra diverselor efecte Z-buffering-ul este astăzi sprijinit în diverse moduri direct de hardware. Un Z-buffer este în esență o zonă de memorie rapidă analog cu un Frame-buffer, conceput special pentru asta, pentru a memora informațiile intermediare în timpul construcției unei imagini. Z-buffer-ul îmbunătățește Rendering-ul în mod esențial la prețuri acceptabile.

Ray-tracing-ul nu este un leac universal

Ray-tracing-ul este un algoritm complicat și conduce la cele mai bune rezultate. El este unul dintre cele mai vechi și cele mai bune metode de rendering. Obiectele pentru Ray-tracing-ul unui scenariu trebuie să fie toate încărcate în memorie. După aceasta o rază va fi trasată dintr-un punct fix înspre ecran. Dacă raza întâlnește un obiect sînt memorate informații despre iluminarea în punctul de întâlnire și se construiește un arbore de raze. Raza se reflectă apoi din obiect și întâlnește apoi un alt obiect. Cînd raza părăsește scena sau după un anumit număr de reflectări se pornește o nouă rază dintr-un alt pixel. Acest proces continuă pînă cînd toți pixelii din ecran au fost adresați și arborele este complet. Arborele va fi apoi prelucrat pînă cînd pentru fiecare punct al ecranului va fi calculată o anumită umbră. Ray-tracing-ul poate lucra și cu lumină foarte puternic reflectată, umbră se face corect și poate fi continuată pînă cînd se obține Anti-Aliasing-ul. Cu toate acestea interacțiunea umbrelor cu obiectele transparente poate fi tratată fals și interacțiuni de lumină subtile de exemplu, în condițiile de iluminare în spații închise, pot să nu fie multumitoare. Ray-tracing-ul permite rezultate de bună calitate dar este de zece ori mai încet decît tehnicile de scan-line. El se încetinește exponențial cînd numărul obiectelor din scenariu se mărește și asta cu toate că această tehnică pare a fi cea mai puternică. Nici una din tehnicile de Rendering descrise pînă acum nu tratează prea bine lumina slabă. Cînd vă puteți găsi în interiorul unei clădiri priviți odată înspre un colț. Suprafețele luminoase și întunecoase dau nenumărate reflexii ale luminii între suprafețele exterioare, care sînt foarte greu de imitat. O tehnică de preprocesare, care prelucrează aceste reflexe luminoase slabe, se numește "radiozitate". Radiozitatea împarte obiectele nu numai în poligoane cărora le ține culoarea, dar ține cont și de efectele de lumină care se vor reflecta de la un poligon la altul. Prin

aceasta se vor putea prelucra efecte de culoare fină cum ar fi reflexia roz care vine de la un perete roșu. Cu Radiozitatea vor fi calculate doar efectele pe care le au obiectele singulare asupra altor obiecte. După aceea vor fi folosite acele tehnici de rendering pentru a obține imaginea completă. Deși se obține rezultatul final ca la un Rendering, tehnica radiozității are nevoie de o mulțime de timp. Radiozitatea este foarte importantă pentru obținerea unor imagini de înaltă calitate astfel încît producătorii de stații de lucru fac eforturi mari pentru a sprijini această metodă.

Chiar și cu cele mai bune tehnici și cele mai economice, Rendering-ul nu e suficient pentru ochiul uman. Ochiul uman este foarte sensibil la lumini fine, uneori contează o singură dungă întunecoasă sau luminoasă. Rezolvarea acestei probleme constă în introducerea mai multor poligoane. Aceasta ar putea ajuta sau nu în funcție de de detaliile fine ale luminii și de poziționare. Un Rendering efectiv necesită o grămadă de cunoștințe și de prelucrări ale luminii, și mult timp liber. Producerea unei imagini realiste generată pe calculator este determinată de trei factori: calitatea obiectului geometric de bază dintr-un pachet CAD, calitatea suprafețelor exterioare care au fost alese pentru fiecare obiect, și de exactitatea fizică a Rendering-ului. De aceea, fotorealismul va fi mereu îmbunătățibil și va fi în puterea acestor factori să se obțină rezultate din ce în ce mai bune.

"Imaginile atrăgătoare" sînt o răspaltă meritorie, dar dacă se află și în mișcare, ele devin minunate. Luați de exemplu, mai multe imagini care diferă între ele foarte puțin într-o secvență oarecare și veți obține o animație. O asemenea animație nu poate fi generată în timp real - amintiți-vă că poate dura pînă la 30 min. producerea unei singure imagini pe un calculator de birou cu performanțe slabe. Pe de altă parte putem genera animația construind imaginile una după alta și memorîndu-le pentru a ale afișa ulterior.

Este necesară răbdarea

Animația și Rendering-ul au dus spoturile publicitare și industria filmului la cea mai înaltă calitate. Aceste filme au depășit domeniul experimentalului și au intrat în domeniul științei, în aplicațiile militare sau în alte domenii importante.

Cu toate că animația prin Rendering consumă foarte mult timp există totuși cîteva trucuri cu care se poate micșora numărul de ore necesar. De exemplu, animarea doar a unor mici părți dintr-o scenă micșorînd astfel zona ce va trebui reconstituită. Calculatorul permite astfel de trucuri, ca de exemplu "intermedierea" metodă în care animatorul generează figura de început și de sfîrșit iar calculatorul generează pozițiile intermediare. Dar "intermedierea" generează ex-

clusiv metodele schematice și rendering-ul este cel care generează mai departe umplerea fiecărei imagini.

O altă problemă a animației este spațiul de memorie. Un ecran întreg de 640x480 pixeli și cu o culoare reprezentată pe 8 biți are nevoie de aproximativ 300KB. La 30 de imagini (cadre) pe secundă avem nevoie pentru un minut de animația de 540MB de memorie și aceasta este mai mult decât pot avea majoritatea, utilizatorilor. Deci, utilizatorii ce vor să folosească animația și Rendering-ul trebuie să genereze mai întâi câteva cadre, să le memoreze, apoi să genereze următoarele cadre. Toate acestea trebuie salvate pe bandă. Din acest motiv filmul nu mai este la fel de ușor de editat și pe diskett-ă.

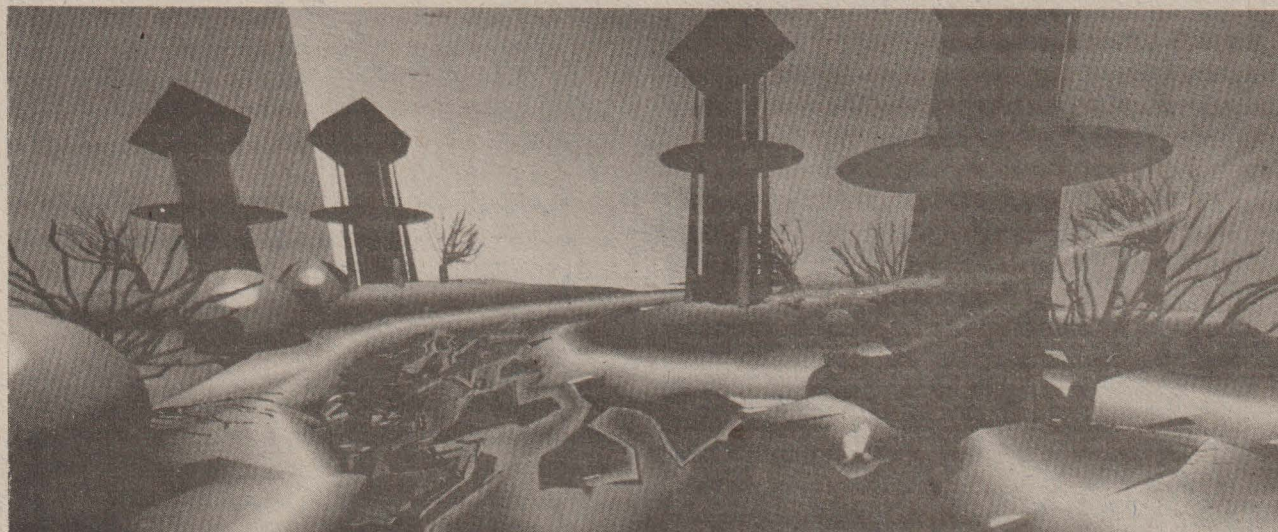
Firma Pixar a început ca un compartiment al firmei Lucas Film, avînd sarcina să creeze efecte speciale pentru filme cu ajutorul graficii pe calculator. Acum cîțiva ani firma a fost cumpărată de Steve Jobs, ca întemeietorul firmei Apple și întemeietorul NeXT. Pixar lucrează independent față de NeXT în unele domenii chiar cu rezultate mai bune. Pînă nu demult Pixar a fost cunoscută prin efectele speciale din film, Recent, filmul său de scurt metraj "Tin Toy" a cîștigat premiul Oscar. Pixar a mai produs cunoscutul efect de geneză pentru filmul "Star Trek II": "mînia lui Khan". În acest timp Pixar a descoperit cel puțin două lucruri importante: un extraordinar algoritm de Rendering și o interfață cu ajutorul căreia alte programe pot comunica cu algoritmul. Algoritmul este cunoscut sub numele de "REYES" prescurtare de la "Renders Everything You Ever Saw". Interfața se numește "Interfața Renderman". REYES a fost dezvoltată de Loren Carpenter cu ajutorul unor specialiști de la SGI. Algoritmul atinge calitățile Ray-tracing-ului. Pixar susține că viteza de generare a lui REYES la imaginile cele mai simple este mai slabă decât a Ray-tracing-ului aproape la fel de rapidă pentru

imagini mai complicate, cu multe obiecte. Astfel spus Renderman este o interfață între un sistem de Rendering, ca de exemplu REYES. Ea este independentă de algoritm, hardware, sau viteze de execuție.

Tot ce e necesar pentru generarea unei imagini cade în sarcina comenzilor RIB (Renderman Interface Biffe stream). Restul este făcut de sistemul Rendering. Există astăzi Kit-uri de produse de dezvoltare pentru PC, cu Macintosh și pe alte stații de lucru. Producătorii de soft au cumpărat licențele de la Pixar, și folosesc interfața Renderman.

Pe această cale Pixar a devenit standard universal pentru Rendering. Pachetele software care conțin în ele metoda Rendering oferă împreună cu ele și interfața Renderman. Pentru PC interfața a fost folosită deja de "Cadkey", FastCAD și mai tîrziu de AutoCAD. Pentru Macintosh este utilizat de Statavision 3D și de swivel 3D. NeXT a promis că interfața Renderman va fi disponibilă și în noul său sistem color. Formatul RIB a devenit noul standard de comunicare date, date care trebuie eventual folosite la Rendering așa cum EPS este un standard pentru date care trebuie eventual tipărite la imprimantă. Astfel s-a născut o metodă simplă de comunicație între PC, Macintosh și marii giganți precum Cray. Transmiterea rapidă în rețea a informației de imagine a devenit foarte ușor de practicat.

Stadiul actual al Rendering-ului amintește de stadiul fotografiei la începutul acestui secol. Experții au acces la această tehnologie și în curînd, și mare masă de oameni. Imaginile în mișcare sînt posibile azi doar pe calculatoarele cele mai performante, dar mîine vor fi accesibile pentru toți utilizatorii. Ne bucurăm deja cînd hardware-ul are putere din ce în ce mai mare și odată cu acesta mult și posibilitățile Rendering-ului.



Programare VGA - Animație

În primul articol despre programare VGA în 256 culori am arătat cum se obține o rezoluție de 320 x 240 pixeli. Am amintit și avantajele pe care le prezintă acest mod față de celelalte moduri. În acest articol vom prezenta tehnica de paginare a memoriei ecran, precum și un program de animație care va folosi această tehnică.

Știm că în modul 320x240x256 memoria ecran este organizată pe patru plane, fiecare plan de memorie putând fi referit ca o matrice de 240 linii și 80 coloane. Rezultă că o imagine de 320x240 pixeli ocupă în total 76800 octeți, câte 19200 octeți în fiecare plan de memorie. Dar capacitatea unui plan de memorie este de 65536 octeți, deci această imagine ocupă mai puțin de o treime din memoria ecran.

```
Port[$3D4]:=12;
Port[$3D5]:=Hi(Adr);
Port[$3D4]:=13;
Port[$3D5]:=Lo(Adr);
```

Este important de reținut faptul că această adresă este aceeași pentru toate cele patru plane de memorie. Inițial (adică după setarea modului grafic), se afișează conținutul zonei care se află la începutul memoriei ecran.

Iată un exemplu de paginare pentru modul 320 x 240 x 256:

- pagina 0: începe la adresa relativă 0;
- pagina 1: începe la adresa relativă 19200;
- pagina 2: începe la adresa relativă 38400;

Rămân în această situație 7936 octeți nefolosiți pentru fiecare din cele patru plane de memorie. Ajunși în acest punct observăm că procedurile scrise pînă acum nu ne mai sînt de folos, confirmînd parcă o concluzie extrasă dintr-un tratat de morfologie: "Dacă constructorii ar fi construit clădiri așa cum scriu programatorii programe, atunci prima ciocănitore care ar fi venit ar fi distrus civilizația". Să mai facem încă o observație: procedurile XWritePixel și XReadPixel sînt apelate foarte des, și de aceea ar fi de dorit ca ele să fie scrise în limbaj de asamblare, pentru ca viteza de desenare a imaginii să fie cît mai mare.

Iată de ce în acest articol reluăm în întregime unit-ul Xgrafica, îmbogățit cu două proceduri noi: XSetActive și XSetPVideo (ele corespund funcțional proce-

durilor SetActivePage și SetVideoPage din unit-ul Graph). De asemenea, procedurile XWritePixel și XReadPixel sînt scrise în limbaj de asamblare. De data aceasta ne-am luat cîteva măsuri de precauție, deoarece în articolul următor dorim să utilizăm mouse-ul în modul grafic 320x240x256. Atunci vom vedea la ce folosesc cele cîteva instrucțiuni care în acest moment par inutile.

Vom explica în continuare cîteva principii folosite la scrierea programului Xanim. Acesta prezintă un pătrat care se mișcă în sens trigonometric pe o traiectorie circulară, pînă ce se apasă o tastă. Pentru început se desenează fundalul în primele două pagini. Pentru a ascunde fazele pregătitoare, și deci pentru a arăta privitorului doar rezultatul final (adică animația), pentru toate culorile utilizate în program am apelat procedura XSetColor, astfel că inițial este afișat pe display un fond albastru. După ce am pregătit fundalul în ambele pagini se setează în mod corespunzător culorile utilizate și se trece la animație. Totul se petrece ca la cinematograful: inițial se fac toate pregătirile în cabina de proiecție, după care se stinge lumina și se derulează filmul. Animația este realizată după regulile următoare:

- în tabelul tridimensional Pozitii avem memorate coordonatele succesive ale pătratului;
- în fiecare moment avem o pagină de lucru (setată de procedura SetPActive), și o pagină de afișare (setată de procedura SetPVideo);
- pagina de lucru nu coincide cu pagina de afișare;
- în pagina de lucru ștergem pătratul de pe poziția dinainte, și desenăm pătratul cu două poziții în față (nu cu una!); pătratul va fi desenat cu o poziție în față în cealaltă pagină;
- desenarea pătratului într-o poziție este precedată de salvarea porțiunii respective a fundalului; ștergerea pătratului se face prin refacerea fundalului în poziția respectivă;
- după ce am desenat pătratul în noua poziție, pagina de lucru devine pagină de afișare, iar pagina de afișare devine pagină de lucru.

De ce am folosit această tehnică de programare, care la prima vedere pare cam sofisticată? De ce nu am folosit o tehnică mai simplă, folosind o singură pagină de memorie video? În cazul în care animația

este ceva mai complexă, operațiile respective (refacerea fundalului și desenarea zonei mobile în noua poziție) pot duce la efecte mai puțin plăcute (suprapunere treptată a imaginii noi peste cea veche etc). În cazul folosirii mai multor pagini de memorie video astfel de efecte nu mai apar, deoarece privitorul vede doar pagina de afișare, în care în acel moment nu se întâmplă nimic. În momentul în care în cealaltă pagină de memorie s-a construit complet noua imagine, cele două pagini de memorie își schimbă în același timp rolurile (pagina de afișare devine pagină de lucru, iar pagina de lucru devine pagină de afișare).

Am văzut la început că în modul 320x240x256 pot fi folosite trei pagini de memorie ecran. În general o animație se realizează cu rezultate bune folosind două pagini de memorie (cum am văzut mai sus). La ce putem folosi a treia pagina de memorie în acest mod grafic? Să presupunem că avem diferite secvențe de animație care se derulează succesiv (fiecare secvență terminându-se la apariția unui eveniment extern: apăsarea unei taste, a unui buton mouse, etc.). Evident, pregătirea unei secvențe cere oarecare timp. Pentru a nu oferi privitorului o imagine monocromă (cum este cazul programului prezentat), se poate desena un cadru în pagina a treia, care va fi afișat pe durata pregătirii acestei secvențe. Acest cadru va fi static, dar se poate rezolva și acest neajuns folosind întreruperea de timp (în revista "If" nr. 6/1990 este prezentat un model de program pe această temă). Procedura conectată la întreruperea de timp poate afișa în poziții aleatoare pixeli de diferite culori. Atenție! pentru a realiza acest lucru se apelează procedura XSetActive cu argumentul 2, după care trebuie restaurată pagina de lucru dinaintea apelului. O metodă similară se folosește la televiziune în pauzele dintre două emisiuni.

Să facem câteva observații în legătură cu procedurile XGetImg și XPutImg. Apelând procedurile XReadPixel și XWritePixel se face de fiecare dată comutarea de pe un plan de memorie pe altul, operație care este consumatoare de timp. Dacă se structurează corespunzător tabloul Zons, astfel încât să se memoreze separat fiecare plan de memorie, procedurile XGetImg și XPutImg pot fi scrise astfel încât să opereze mai întâi numai în planul 0 de memorie, apoi numai în planul 1, s.a.m.d. Lăsăm cititorului plăcerea de a rezolva acest exercițiu de optimizare.

mat. Nelu Cozac

XGRAFICA.ASM

```
data segment word public
    extrn adrpagina:word
data ends
```

```
code segment byte public
    assume cs:code,ds:data
```

```
calcul proc near
    cmp cx,0
    js calcer
    cmp cx,320
    jae calcer
    cmp dx,0
    js calcer
    cmp dx,240
    jae calcer
    mov ax,0a000h
    mov es,ax
    mov ax,dx
    mov bx,80
    imul x
    add ax,adrpagina
    mov bx,cx
    shr bx,1
    shr bx,1
    add bx,ax
    and cl,3
    ret
calcer:
    stc
    ret
calcul endp

    public xwritepixel
xwritepixel proc far
    push bp
    mov bp,sp
    mov cx,[bp+10]
    mov dx,[bp+8]
    call calcul
    jc xwriter
    mov ax,0102h
    shl ah,cl
    mov dx,3c4h
    pushf
    cli
    out dx,ax
    mov al,byte ptr [bp+6]
    mov es:[bx],al
    popf
xwriter:
    pop bp
    ret 6
xwritepixel endp

    public xreadpixel
```

Laborator

```
xreadpixel      proc far
    push        bp
    mov         bp,sp
    mov         cx,[bp+8]
    mov         dx,[bp+6]
    call        calcule
    mov         al,0
    jc         xreader
    mov         ah,cl
    mov         al,4
    mov         dx,3ceh
    pushf
    cli
    out         dx,ax
    mov         al,es:[bx]
    popf
xreader:
    xor         ah,ah
    pop         bp
    ret         4
xreadpixel      endp
code            ends

end
```

XGRAFICA.PAS

```
unit xgrafica; {$D-,L-,S-}
```

```
interface uses dos;
var adrpagina:word;
```

```
procedure xsetgraph;
procedure xwritepixel(x,y:word; p:byte);
function xreadpixel(x,y:word):byte;
procedure xsetcolor(c,r,g,b:byte);
procedure xsetpactive(nrp:byte);
procedure xsetpvideo(nrp:byte);
procedure xterminate;
```

```
implementation
```

```
var regs:registers; i,j,k:word;
    me:array[0..239,0..79] of byte absolute $a000:0;
const valori:array[1..10] of word =
    ($0d06,$3e07,$4109,$ea10,$ac11,
    $df12,$0014,$e715,$0616,$e317);
```

```
procedure xsetgraph;
begin
    regs.ax = 19; intr(16,regs);
    portw[$3c4] = $604;
```

```
    portw[$3c4] = $100;
    port[$3c2] = $e3;
    portw[$3c4] = $300;
    port[$3d4] = $11;
    port[$3d5] = port[$3d5] and $7f;
    for i = 1 to 10 do portw[$3d4] = valori[i];
    portw[$3c4] = $f02;
    fillchar(me,240*80*3,#0);
    adrpagina = 0;
end;
```

```
{$! xgrafica}
```

```
procedure xwritepixel(x,y:word; p:byte); external;
function xreadpixel(x,y:word):byte; external;
```

```
procedure xsetcolor(c,r,g,b:byte);
begin
    with regs do
        begin
            ah = 16; al = 16; bh = 0; bl = c;
            ch = g; cl = b; dh = r;
        end;
        intr(16,regs);
end;
```

```
procedure xsetpactive(nrp:byte);
begin
    adrpagina = nrp*240*80;
end;
```

```
procedure xsetpvideo(nrp:byte);
var k:word;
begin
    k = nrp*240*80;
    port[$3d4] = 12; port[$3d5] = hi(k);
    port[$3d4] = 13; port[$3d5] = lo(k);
end;
```

```
procedure xterminate;
begin
    regs.ax = 3; intr(16,regs);
end;

end.
```

XANIM.PAS

```
uses xgrafica,crt;
const lgecran = 240*80; nrpozitii = 90;
var i,j,k,l,p,q,r:integer; x,y,z:real;
    pozitii:array[0..nrpozitii-1,1..2] of integer;
```

```

zons:array[0..1,0..15,0..15] of byte;

procedure xgetimg(t:integer);
begin
  k:=pozitii[t,1];
  l:=pozitii[t,2];
  for i:=0 to 15 do for j:=0 to 15 do
    zons[p,i,j]:=xreadpixel(k+i,l+j);
end;

procedure xputimg(t:integer);
begin
  k:=pozitii[t,1];
  l:=pozitii[t,2];
  for i:=0 to 15 do for j:=0 to 15 do
    xwritepixel(k+i,l+j,zons[p,i,j]);
end;

begin
  for i:=0 to nrpozitii-1 do
    begin
      z:=pi*i*2/nrpozitii;
      x:=100*cos(z); y:=100*sin(z);
      pozitii[i,1]:=round(x+152.0);
      pozitii[i,2]:=round(112.0-y);
    end;
  xsetgraph;
  xsetcolor(0,0,0,63);
  xsetcolor(1,0,0,63);
  xsetcolor(2,0,0,63);
  xsetcolor(4,0,0,63);
  xsetcolor(5,0,0,63);
  xsetcolor(6,0,0,63);

  for p:=0 to 1 do
    begin
      xsetpactive(p);
      for i:=0 to 319 do for j:=0 to 239 do
        xwritepixel(i,j,1);
      for i:=0 to 319 do
        begin
          for j:=0 to 1 do
            xwritepixel(i,j,2);
          for j:=119 to 120 do
            xwritepixel(i,j,2);
          for j:=238 to 239 do
            xwritepixel(i,j,2);
          end;
        for j:=0 to 239 do
          begin
            for i:=0 to 1 do
              xwritepixel(i,j,2);
            for i:=159 to 160 do
              xwritepixel(i,j,2);
            for i:=318 to 319 do
              writepixel(i,j,2);
            end;
            for i:=2 to 118 do
              for j:=0 to 2 do
                begin
                  xwritepixel(i+j,i,4);
                  xwritepixel(319-i-j,i,5);
                end;
            for i:=237 downto 121 do
              for j:=0 to 2 do
                begin
                  xwritepixel(239-i+j,i,5);
                  xwritepixel(i+j+78,i,4);
                end;
            end;
          end;
          xsetcolor(1,0,0,63);
          xsetcolor(2,0,0,63);
          xsetcolor(4,63,0,0);
          xsetcolor(5,63,0,63);
          xsetcolor(6,63,63,0);
          p:=0; xsetpactive(p); xgetimg(0);
          p:=1; xsetpactive(p); xgetimg(1);
          p:=0; q:=0;

          repeat
            xsetpactive(p); xsetpvideo(1-p);
            r:=(q+2) mod nrpozitii;
            xputimg(q); xgetimg(r);
            k:=pozitii[r,1];
            l:=pozitii[r,2];
            for i:=0 to 15 do for j:=0 to 15 do
              xwritepixel(k+i,l+j,6);
            q:=(q+1) mod nrpozitii; p:=1-p;
            delay(25);
          until keypressed;

          xterminate;
        end.
    end;
  end;

```

BANNER

Programul BANNER a fost des utilizat în ultima vreme pentru realizarea unor anunțuri publicitare de dimensiuni mari, în forme estetice implicând efecte grafice deosebite ca: evidențierea conturilor semnelor folosite, suprapuneri prin dublare sau chiar triplare însoțite de "colorări" cu diferite nunațe de gri, aspect de perspectivă sau spațial, tensionări sau chiar torsionări ale înscrisurilor, încadrarea semnelor în alte forme prestabilite (ca de exemplu în balonașe) etc.

datorită dimensiunilor reduse (nucleul său este de aproape 400 K și nu necesită drivere speciale), portabilității deosebit de extinse, precum și perspectivele pe care le deschide.

Pentru afișarea pe monitor și imprimarea pe hîrtie programul folosește aceleași fișiere-font, pe care le recunoaște după extensia FNT. În momentul lansării în execuție se face o recunoaștere a acestor fișiere-font ce se află în același directory ca și programul executabil BAN-

ner. facem cîteva considerații preliminare.

1) În fiecare fișier-font original se găsesc descrierile a 123 de semne, care corespund:

- tastelor normale (95 de semne, pentru codurile ASCII între 32 și 126 inclusiv);

- combinațiilor `ctrl + tasta`, care dau coduri între 0 (`ctrl + @`) și 27 (`ctrl + j`); acestora le corespund deci 28 de semne speciale, care în fonturile originale sînt triunghiuri (în diverse poziții), buline, stele, săgeți diverse, semne-cărți de joc etc.

2) Se presupune că toate semnele descrise în font au aceeași înălțime. Pentru a avea posibilitatea de a face comparații care să implice lanțurile semnelor, vom considera că înălțimea comună este 256 (împărțită în `ascender=192` și `descender=64`).

3) Semnele sînt descrise prin contururi, formate din segmente și curbe Bezier. "Umplerea" zonelor plane este controlată de către program, prin opțiunile alese. Informații despre interpolarea de tip Bezier pot fi găsite în orice carte recentă de proiectare asistată de calculator (Computer Aided Design). Programul

BANNER folosește doar curbe Bezier bazate pe 4 noduri.

4) Transformările imaginii grafice nu constituie o surpriză pentru cei familiarizați cu funcțiile complexe (și transformările conforme); re-



Dat fiind că programul permite prelucrarea a doar (cel mult) două rînduri de semne - pe fiecare rînd putîndu-se afla cel mult 64 - el nu poate fi considerat sistem de editare propriu-zis. Totuși îl apreciem ca exemplu de program bine realizat,

NER.EXE. Așadar utilizatorul poate crea noi fișiere-font sau își poate utiliza propriile fișiere-font după dorință.

Înainte de a trece la descrierea structurii unui astfel de fișier-font, să

comandarea celor interesați studierea cursurilor elementare de teoria funcțiilor complexe.

Structura unui fișier-font nu - me . FNT este următoarea:

I. Zona header, conținând denumirea de afișare a fontului. Această denumire, și nu cea a fișierului-font, este preluată în program și folosită pentru selectarea fontului.

bytes	conținut
0 - 3	AB CD 00 05
4	lungimea L a denumirii de afișare (ca string)
5 - L + 4	denumire de afișare (string)

II. Zona de adrese relative. Această zonă conține 250 bytes, organizați ca 125 de întregi (high-low).

Primii 124 ne dau adresele relative ale semnelor în zona descrieri rectoriale (începînd cu semnul 32 - blancul - a cărui adresă relativă este 00 00; menționăm că după adresa semnului 126 urmează adresa primului semn special, corespunzător lui `ctrl + @`); cel de-al 125-lea ne indică lungimea zonei de descrieri vectoriale.

III. Zona de descrieri vectoriale. Aici, fiecărui semn descris în font i se precizează lățimea și conturul, într-o succesiune de bytes al cărei început și sfârșit pot fi aflate ușor consultînd adresele relative succesive din zona anterioară.

Descrierea semnului începe printr-un byte (unsigned), în care i se indică lățimea, urmat de secvențe de comenzi. Aceste secvențe pot fi de patru tipuri (Tabelul 1).

Pentru interpretarea corectă a secvențelor de comenzi, trebuie să ținem seamă de următoarele. Inițial, înainte de a prelua descrierea semnului, programul BANNER.EXE setează un flag și rezervă loc unui

"punct" special Q. În orice caz, prima comandă este de forma 60 P0 (salt la punctul inițial P0). Înainte de executarea unei comenzi de tip polisegment

$2n P_1 P_2 \dots P_n$, (n - cifră hex) se testează dacă P_1 coincide cu P_0 , iar în caz afirmativ se desetează flagul și se atribuie lui Q valoarea P_0 . Prin executarea comenzii se trasează segmentele P_0-P_1 , P_1-P_2 , ... $P_{n-1} - P_n$, apoi se modifică $P_0=P_n$.

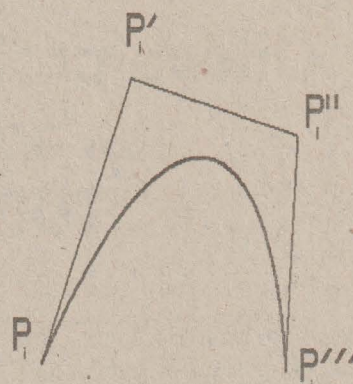
Prin executarea unei comenzi $4m P'_1 P''_1 P'''_1 \dots P'_m P''_m P'''_m$, (m - cifră hex)

se trasează m curbe Bezier cu nodurile P_i, P'_i, P''_i, P'''_i ($1 < i < M$), unde $P_1=P_0$ iar $P_{i+1}=P''_i$ (pentru $i < m$). În final se modifică $P_0=P''_m$.

Înainte de executarea unei comenzi E0 sau 60 P0, se verifică dacă flagul este desetat; în caz afirmativ se setează și se trasează segmentul Q-P0, apoi se execută comanda.

Sperăm că datele prezentate mai sus vor permite doritorilor să creeze după dorință fonturi tip "BANNER".

Romulus Maier



Tabelul 1

Tip	Conținut	Interpretare
1	$2n x_1 y_1 \dots x_n y_n$	polisegment, format din segmente implicînd punctul anterior și punctele P_1, \dots, P_n de coordonate $P_i = (x_i, y_i)$. Aici x_i și y_i sînt considerați octeți fără semn, iar n este cifră hex
2	$4m x'_1 y'_1 x''_1 y''_1 x'''_1 y'''_1 \dots x'_m y'_m x''_m y''_m$	polilinie formată din curbe Bezier (al căror număr este dat de cifra hex m)
3	60 x y	salt la punctul P de coordonate (x,y). Aici x și y sînt considerați, ca și mai sus, octeți fără semn.
4	E0	terminarea descrierii semnului

Cîteva probleme de logică rezolvate și comentate în limbajul TURBO PROLOG V2.0

Să enunțăm alte probleme care pot fi rezolvate în mod facil cu ajutorul limbajului susmenționat.

PROBLEMA 1:

Într-un bloc cu 4 etaje locuiesc familiile K, L, M, N la etaje diferite. Știm că:

- (a) familiile K și L nu locuiesc la etaje alăturate.
- (b) familia M locuiește la unul din etajele aflate sub familia K.
- (c) familia N locuiește la unul din etajele dintre familiile L și M.

Întrebări:

- (I) Dacă familia M locuiește la etajul 3, care este ordinea familiilor de jos în sus?
- (II) Care dintre afirmațiile de mai jos poate fi adevărată în condițiile (a), (b) și (c)?

- (A) M locuiește la etajul 1
- (B) M locuiește la etajul 2
- (C) M locuiește la etajul 4
- (D) K locuiește la etajul 1
- (E) N locuiește la etajul 1

Există mai multe moduri în care putem rezolva problema. Vom aborda pentru început calea cea mai simplă: vom scrie trei predicate care să conțină condițiile din enunț. Predicatele vor avea drept argumente patru variabile de tip "caracter", care vor simboliza cele patru familii. Poziția unui caracter în lista de argumente determină etajul la care locuiește familia respectivă. Predicatele vor avea numele "a", "b" și "c", conform condițiilor din enunț.

PREDICATES

```
a(CHAR, CHAR, CHAR, CHAR)
b(CHAR, CHAR, CHAR, CHAR)
c(CHAR, CHAR, CHAR, CHAR)
```

rezultat

CLAUSES

```
a('K',_,_'L',_) . %1
a('K',_,_,_'L') . %2
a(,'K',_,_'L') . %3
a('L',_,_'K',_) . %4
a('L',_,_,_'K') . %5
a(,'L',_,_'K') . %6

b('M',_'K',_,_) . %7
b('M',_,_'K',_) . %8
b('M',_,_,_'K') . %9
b(,'M',_'K',_) . %10
b(,'M',_,_'K') . %11
b(,,_'M',_'K') . %12

c('L',_'N',_'M',_) . %13
c('L',_'N',_,_'M') . %14
c('L',_,_'N',_'M') . %15
c(,'L',_'N',_'M') . %16
c('M',_'N',_'L',_) . %17
c('M',_'N',_,_'L') . %18
c('M',_,_'N',_'L') . %19
c(,'M',_'N',_'L') . %20
```

rezultat:-

```
a(A,B,C,D), %21
b(A,B,C,D), %22
c(A,B,C,D), %23
write(A,B,C,D),nl, %24
fail. %25
```

GOAL rezultat.

Clauzele %1..6 ale predicatului "a", au fost scrise cu argumentele 'K' și 'L' așezate în așa fel încît să satisfacă condiția pusă la punctul (a). Pentru aceasta am scris toate variantele în care 'K' și 'L' nu sînt alăturate.

Clauzele predicatelor "b" și "c" vor fi scrise la fel, în

scop

LIDERUL REZELELOR DE CALCULATOARE
ÎN ROMÂNIA VĂ PROPUNE SOLUȚII
COMPLETE CU CELE MAI AVANSATE
ECHIPAMENTE DE TEHNICĂ DE CALCUL

- **EVEREX** - întreaga gamă de calculatoare personale și periferice (imprimante laser, modemi, unități de benzi magnetice, plăci video, scanners)
- **IBM** - calculatoare personale - PS/1 și PS/2
- **NOVELL** - software de rețea
- **3 COM** - echipamente pentru comunicații de date
- **SUMMAGRAPHICS / HOUSTON INSTRUMENTS** - plotters; scanners; digitizers; graphic tablets
- **AMERICAN POWER CONVERSION** - surse neîntreruptibile
- **AMERICAN SMALL BUSINESS CORPORATION** - software pentru proiectare asistată pe calculator - design CAD
- **EPSON** - imprimante matriciale și laser; consumabile
- **WESTERN DIGITAL** - hard discuri
- **DIGIBOARD** - plăci multiplexoare
- **STANDARD MICROSYSTEMS CORPORATION** - componente de rețea

BUCUREȘTI, STR. POLONĂ NR. 86, SECTOR 1

TEL. 117421; 119248; FAX : 117374

Noi sistem soluția !

mod corespunzător enunțului.

În final am definit predicatul "rezultat", care folosește clauzele anterioare pentru a găsi răspunsul. Observăm că predicatul %7, %8 și %9 au argumentele cu același nume așezate în aceeași poziție. Această aranjare a argumentelor va folosi instanțierea, adică legarea variabilelor, inițial libere, de valori care să fie acceptate concomitent de predicatul "a", "b" și "c". Instanțierea se va face prin testare, pînă cînd va rezulta o soluție. Mai precis, se ia prima clauză și se atribuie valoarea 'K' pentru primul argument și valoarea 'L' pentru al treilea argument. Argumentele care nu ne interesează în acest moment au fost reprezentate prin variabila anonimă "_". Deoarece clauzele sînt declarații ale unei baze de date, compilatorul va genera un mesaj benign de avertizare, prin care ne spune că toate argumentele anonime vor fi legate în mod automat la o valoare de referință. Apoi se va încerca instanțierea argumentelor din prima clauză a predicatului "b". Acest lucru nu va reuși deoarece argumentul 'K' nu figurează în aceeași poziție în cele două clauze. În această situație vor fi testate următoarele clauze pînă cînd nu se va găsi un conflict între argumente. Observăm că prima clauză nu poate realiza o instanțiere cu nici o clauză a predicatului "b". Ca urmare prima clauză a predicatului "a" va fi abandonată și va fi abordată cea de-a doua clauză. Acest mecanism continuă pînă cînd se identifică acele clauze ale predicatelor "a", "b" și "c", care, în mod simultan, nu intră în contradicție între ele. Aceasta va fi prima soluție, respectiv

MKNL

rezultată din clauzele %3, %7 și %19. Predicatul %24 va scrie cele patru variabile pe ecran și va trece la rînd nou.

Predicatul %25, "fail", va sili reluarea instanțierii, pentru a găsi și alte soluții. (Reamintim faptul că predicatul "fail" raportează "eșec" în orice situație.) Într-adevăr va mai fi găsită și soluția :

LNMK

rezultată din clauzele %5, %12 și %13.

Din cele două soluții, răspunsurile vor fi

(I) LNMK

(II) (A)

Acest program rezolvă în mod satisfăcător problema noastră. Observăm că este destul de ușor să greșim la scrierea clauzelor sau să ometem anumite variante oferite de aceeași condiție a problemei. Înainte de a găsi o rezolvare mai elegantă a aceleiași probleme, să mai exersăm cu o problemă asemănătoare, pe care nu o mai comentăm:

PROBLEMA 2:

Obiectelor K, L, M, N, O, P trebuie să li se atribuie notele 1, 2, 3, 4, 5, 6, 7 conform unei liste de instrucțiuni.

Din această listă s-a pierdut o parte, rămînînd condițiile:

- (a) K are nota 2
- (b) N are notă mai mare decît K
- (c) R are notă mai mică cu 1 decît N
- (d) L și O au note mai mari decît R
- (e) P are o notă cu 6 mai mare sau mai mică decît

M

(f) toate obiectele au note diferite între ele

(I) Care din afirmațiile de mai jos este cu certitudine adevărată?

- (A) P are nota 1
- (B) L are nota 6
- (C) R are nota 6
- (D) O are nota 6
- (E) K are notă mai mică decît P

(II) Dacă știm în plus că O are nota 6, care este nota lui L?

- (A) 2
- (B) 3
- (C) 5
- (D) 4
- (E) nu se poate determina

(III) Dacă știm în plus față de condițiile (a)...(f) că R are notă mai mare decît P, cîte moduri de atribuire a notelor mai rămîn posibile?

- (A) 0
- (B) 1
- (C) 2
- (D) 4
- (E) nu se poate determina

(IV) Care din afirmațiile de mai jos este cu certitudine adevărată în condițiile (a)...(f)?

- (A) M are nota 2
- (B) M are nota 6
- (C) M are nota 7
- (D) O are nota 1
- (E) O are o notă cu 1 mai mare sau mai mică decît

L

Și rezolvarea:

code=1200

PREDICATES

b (STRING, STRING, STRING, STRING)

c (STRING, STRING, STRING, STRING, STRING)

d (STRING, STRING, STRING, STRING,

```
STRING)
```

```
    e (STRING, STRING)
```

```
raspuns1
```

```
CLAUSES
```

```
% conditiile a si f vor fi continute in toate predicaterile b,c,d si e.
```

```
b("N",_,_,_).
```

```
b(,"N",_,_).
```

```
b(,,"N",_).
```

```
b(,_,,"N").
```

```
c("R","N",_,_,_).
```

```
c(,"R","N",_,_).
```

```
c(,,"R","N",_).
```

```
c(,_,,"R","N").
```

```
d("R","N","L","O",_).
```

```
d("R","N","L",_,,"O").
```

```
d("R","N",_,,"L","O").
```

```
d("R","N","O","L",_).
```

```
d("R","N","O",_,,"L").
```

```
d("R","N",_,,"O","L").
```

```
d(,"R","N","L","O").
```

```
d(,,"R","N","O","L").
```

```
e("P","M").
```

```
e("M","P").
```

```
raspuns1:-
```

```
b(D,E,F,G),
```

```
c(C,D,E,F,G),
```

```
d(C,D,E,F,G)
```

```
,e(A,G),
```

```
write(A,"K",C,D,E,F,G),nl,fail.
```

```
GOAL raspuns1.
```

Pentru variație am folosit variabile de tip STRING în loc de CHAR.

Soluțiile furnizate de program vor fi următoarele:

```
PKRNLOM
```

```
MKRNLOP
```

```
PKRNOLM
```

```
MKRNOLP
```

Din aceste soluții putem vedea că răspunsurile sînt următoarele:

```
(I) (c)
```

```
(II) (c)
```

```
(III) (c)
```

```
(IV) (e)
```

După cum am menționat anterior, acest mod de rezolvare, deși este simplu, nu este și cel mai elegant deoarece trebuie să găsim noi toate combinațiile posibile în funcție de condițiile din enunț. Pentru a evita această sursă posibilă de erori, vom încerca să găsim un algoritm care caută singur toate combinațiile posibile, din care se vor selecta ulterior doar soluțiile care satisfac condițiile din enunț. Să enunțăm a treia problemă pe care să o rezolvăm în acest mod și după aceea să vedem cum se poate adapta această metodă pentru rezolvarea problemelor enunțate anterior.

PROBLEMA 3:

Un inspector trebuie să viziteze patru clase K,L,M,N în cursul unei zile. El dorește ca:

- 1) să inspecteze clasa K înaintea claselor L și M
- 2) să inspecteze clasa L înaintea clasei N
- 3) a treia clasă inspectată să fie M

În cîte moduri pot fi vizitate clasele și care sînt aceste moduri?

Rezolvare:

```
DOMAINS
```

```
LISTA=STRING*
```

```
DATABASE
```

```
variante_posibile(LISTA)
```

```
variante_corecte(LISTA)
```

```
PREDICATES
```

```
generez_variantele
```

```
aranjamente(LISTA,LISTA)
```

```
aranjamentel (STRING,LISTA,
```

```
LISTA)
```

```
aflu_ponderea(LISTA,STRING,
```

```
INTEGER)
```

```
append(LISTA,LISTA,LISTA)
```

```
lungimea(LISTA,INTEGER)
```

```
conditii
```

Cursuri

CLAUSES

```
aranjamente([], []).

aranjamente([X|L],M) :-
aranjamente(L,N),
aranjamentel(X,N,M).

aranjamentel(X,[A|L],[A|M]):-
    A<>X,
    aranjamentel(X,L,M).

    aranjamentel(X,L,[X|L]).

generez_variantele:-
L=["K","L","M","N"],
aranjamente(L,L),
    assertz(variante_posibile(L)),
fail.

generez_variantele.

aflu_ponderea([],_,0).

aflu_ponderea(Lista,El,Nr):-
append(X,[El|_],Lista),
lungimea([El|X],Nr).

append([],X,X).

append([H|Tail],L,[H|Tail2]):-
-append(Tail,L,Tail2).

lungimea([],0).
lungimea([_|Tail],Len):-
lungimea(Tail,T_len),Len=T_len+1.

conditii:-
variante_posibile(Lista),
    aflu_ponderea(Lista,"K",NrK),
    aflu_ponderea(Lista,"L",NrL),
    aflu_ponderea(Lista,"M",NrM),
    aflu_ponderea(Lista,"N",NrN),
    NrK<NrL, NrK<NrM, %conditia 1
    NrL<NrN, %conditia 2
Nrm=3, %conditia 3
retract
(variante_posibile(Lista)),
```

```
assertz
(variante_corecte(Lista)),fail.

conditii:-variante_corecte(X),
write(X),nl,
fail.

conditii.

GOAL generez_variantele,
conditii.
```

Pentru a obține toate variantele posibile de aranjare a elementelor unei liste am definit predicatul "aranjamente". Acesta va prelua lista și va returna o listă cu același număr de elemente, aranjate în alt mod. Observăm că are o structură aproape identică cu predicatul de sortare a unei liste (vezi articolul "Folosirea listelor în limbajul PROLOG"), diferența constând din modificarea condiției $A > X$, în $A < > X$. Condiția $A < > X$ poate lipsi atunci când lista nu are duplicate.

Predicatul "generez_variantele" definește lista pe care o prelucrăm, apelează predicatul "aranjamente" și reține în bază de date o soluție cu ajutorul predicatului "assertz". Constatăm din nou prezența predicatului "fail" la sfârșitul definiției. Acesta, după cum ne este cunoscut deja, va forța găsirea tuturor soluțiilor, respectiv va determina clauza să reia căutarea soluțiilor pînă la epuizarea tuturor variantelor posibile. După reținerea în baza de date din memorie a tuturor combinațiilor, predicatul "generez_variantele" va abandona prima clauză și o va executa pe a doua, care nu face nici o acțiune, dar asigură ieșirea din predicat.

Pentru a putea pune condițiile din enunț, va trebui în prealabil să definim predicatul "aflu_ponderea", care permite să aflăm poziția oricărui element dintr-o listă. Prima clauză a acestui predicat va returna zero pentru listele vide. A doua clauză va folosi predicatul "append", care, apelat în modul

```
"append(X,[Element|Y],Lista)"
```

cu argumentele "Lista" și "Element" cunoscute, va rupe "Lista" în punctul în care întîlnește "Element" și va returna lista X care conține toate elementele de la începutul listei pînă la "Element" și lista Y care conține restul de elemente (adică cele de după "Element", pînă în capăt). Acum nu ne rămîne decît să numărăm elementele listei X, la care adăugăm și "Element". Pentru aceasta am definit predicatul "lungimea" care returnează numărul de elemente dintr-o listă. Și acest predicat a fost discutat într-un articol anterior.

Reținem faptul că predicatul

```
aflu_ponderea(Lista,Element,Nr)
```

cu argumentele Lista și Element cunoscute, va returna Nr, care arată al cîtelea element este "Element"

în "Lista".

Din acest moment putem defini predicatul "condiții" pentru a selecta doar acele răspunsuri care satisfac enunțul problemei.

Din baza de date "variante_posibile", se extrag înregistrările care îndeplinesc condițiile din enunț și se rețin în baza de date "variante_corecte". Astfel condiția 1) impune ca elementul K să fie așezat în fața elementelor L și M, adică :

$Nrk < Nrl$ și $Nrk < Nrm$.

În mod asemănător sînt puse și celelalte condiții. Din nou predicatul "fail" forțează parcurgerea întregii baze de date.

A doua clauză a predicatului va afișa conținutul bazei de date "variante_corecte" adică:

["K", "L", "M", "N"]

Acesta este răspunsul dorit.

Cu mici modificări se pot rezolva și problemele anterioare.

□ Pentru problema nr.1 se fac modificările:

```
Nrk<>Nrl+1,Nrk<>Nrl-1,      %cond   a
Nrm<Nrk,                      %cond   b
sau(Nrl,Nrn,Nrm)              %cond   c
```

În prima clauză a predicatului "condiții". În plus se va defini un predicat numit "sau" pentru condiția "c" astfel:

PREDICATES

```
sau(INTEGER,INTEGER,INTEGER)
```

CLAUSES

```
sau(L,N,M):-N<L,N>M.
```

```
sau(L,N,M):-N>L,N<M.
```

□ Pentru problema nr. 2 se fac modificările:

prima clauză a predicatului "generez_variante" va fi rescris astfel:

generez_variante:-

```
L=["K","L","M","N","O","P","R"],
```

```
aranjamente(L,L1),
```

```
L1=[_,"K",_,_,_,_,_],
```

```
%conditia (a)
```

```
assertz(variante_posibile(L1)),
```

fail.

În primul rînd a fost modificată lista L conform enunțului și în al doilea rînd a fost introdusă condiția (a) în acest loc deoarece aceasta evită de la început reținerea soluțiilor care cu certitudine nu duc la rezultate corecte. Un alt motiv pentru care condiția (a) a fost introdusă aici și nu în predicatul "condiții" este acela că se preîntîmpină generarea unei baze de date de 7! elemente (adică 5040 de elemente) și se vor genera doar 6! elemente (adică 720).

În prima clauză a predicatului "condiții" se fac modificările:

```
aflu_ponderea(Lista,"K",Nrk),
aflu_ponderea(Lista,"L",Nrl),
aflu_ponderea(Lista,"N",Nrn),
aflu_ponderea(Lista,"O",Nro),
aflu_ponderea(Lista,"R",Nrr),
Nrn>Nrk,                % cond b
Nrr=Nrn-1                % cond c
Nrl>Nrr,Nro>Nrr        % cond d
sau(Lista),              % cond e
retract
(variante_posibile(Lista)),
assertz
(variante_posibile(Lista)),
fail.
```

Și se va defini predicatul "sau" în felul următor:

PREDICATES

```
sau(LISTA)
```

CLAUSES

```
sau(["P",_,_,_,_,_,_,"M"]).
```

```
sau(["M",_,_,_,_,_,_,"P"]).
```

Modificările sînt relativ minore și nu afectează în mod esențial logica generală a programului inițial. Folosind acest program ca bază de plecare, putem rezolva o gamă largă de probleme de logică. Mai mult, anumite predicate definite aici pot fi folosite fără modificări (doar eventual la declararea tipurilor de date) în alte programe.

ing. Mircea Pantea

Metode de regăsire cu multiatribut:

Metoda semnăturii.

Metodele de regăsire cu multiatribut sînt cunoscute în literatura de specialitate sub denumirea de metode ale semnăturii datorită modului de organizare și extracție a informației. Aceste metode au fost folosite separat sau în combinație cu metodele de inversie pentru obținerea de chei secundare de regăsire (Knuth 1973, Knott 1975, Nievergeld 1984). Cele mai multe dintre ele se bazează pe suprapunerea codurilor.

Fiecare cuvînt cu semnificație pentru regăsire este codificat printr-o anumită metodă, pentru ca apoi să fie stocat într-un fișier separat, numit fișier semnătură. Stocarea codurilor unul după altul va da performanțe maxime la regăsire dar, titimpul de acces, în cazul fișierelor semnătură mari, va fi deasemenea mare. De aceea, aceste coduri pot fi suprapuse cu SAU logic (BOOLEAN), pentru a obține un cod (o semnătură) pentru mai multe atribute. Metoda suprapunerii codurilor a fost propusă pentru prima dată de C.N.Mooers în 1949. El a inventat un ingenios echipament mecanic care era capabil să rezolve foarte rapid cereri de regăsiri pentru o bază de date bibliografică. Extracția cuvintelor cheie se realiza manual și pentru hash-funcție era utilizată o tabelă. De la început s-a pus problema minimizării probabilității de a avea rezultate false (false drops). Stiassny (1960) a ajuns la concluzia că probabilitatea de a avea rezultate false este minimă dacă într-o semnătură numărul de bits 1 este egal cu numărul de bits 0. În 1964 Kautz și Singleton au încercat găsirea unei metode de proiectare a unui sistem de semnături care nu dau rezultate false. Acest lucru este posibil prin crearea unei tabele de control, dar metoda, teoretic interesantă, practic nu este realizabilă.

Bourne (1963) a ilustrat pentru prima dată această metodă. Să presupunem o înregistrare care conține trei cuvinte cheie: "text", "metodă", "codificare". Fiecare cuvînt cheie este codificat printr-o hash-funcție rezultînd un șir de F bits dintre care m sînt 1. Aceste șiruri sînt suprapuse cu funcția SAU logic obținîndu-se "semnătura" înregistrării. După cum arătam mai sus, aceste semnături pot fi stocate în fișiere semnătură sau, astfel-formate, pot fi utilizate în cazul metodei de inversiune a termenilor, pentru determinarea locației înregistrării.

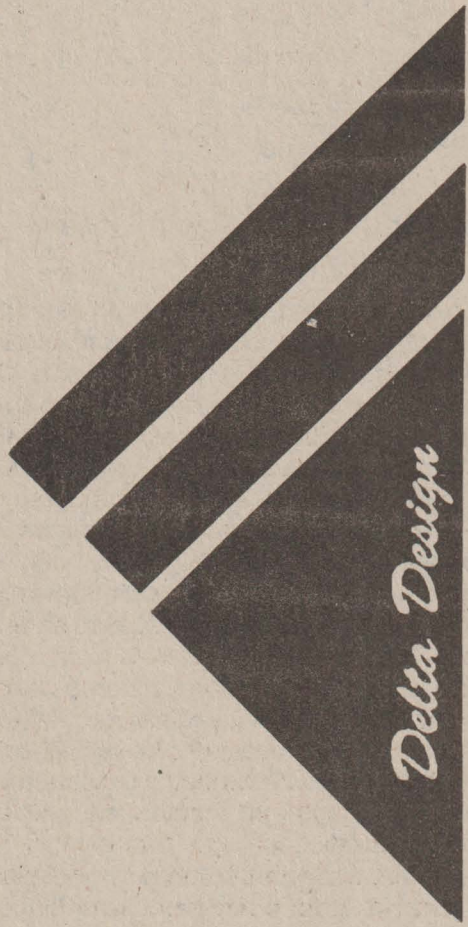
Fig. 1. Exemplu de suprapunere a codurilor ($F=16, m=4$):

text	001	100	001	010
metodă	000	101	011	000
codificare	001	000	001	110
semnătură	001	101	011	110

1. Modalități de folosire a metodei semnăturii

Pentru folosirea semnăturilor, primul care a dat o soluție interesantă pentru determinarea locației înregistrării, a fost Gustafson în 1971. El a presupus că atributele sînt cuvintele cheie care descriu un document și o înregistrare corespunde unui document. Bineînțeles, numărul de atribute per document trebuie să fie fix (Gustafson a propus 6). Să presupunem o hash-funcție H care face să corespundă unui cuvînt w un număr întreg $H(w)$ în intervalul $L0..15D$. Semnătura unui cuvînt este un șir de 16 bits dintre care toți sînt 0 cu excepția bit-ului de pe poziția $H(w)$. Semnătura înregistrării, obținută printr-o operație SAU logic (OR) va conține $k < 6$ bits setați de valoarea 1. În concluzie, vor fi $\text{comb}(16, 6) = 8008$ semnături distincte, deci tot atîtea înregistrări. Putem reprezenta aceste 8008 posibilități într-o hash-tabelă cu 8008 noduri. Interesant la această metodă este faptul că numărul de căutări descrește foarte repede (aproape exponențial) în funcție de numărul de termeni dintr-o cerere de regăsire (query) conjunctivă. Astfel, o cerere pentru un singur cuvînt necesită $\text{comb}(15, 5) = 3003$ încercări (se parcurg 3003 noduri ale hash-tabelei), una pentru două cuvinte $\text{comb}(14, 4) = 1001$ încercări, etc.

Mai tîrziu, au fost studiate și propuse metode pornind de la metoda lui Gustafson (Rothnie și Lozano în 1974, Rivest în 1976, Aho și Ullman în 1979, Lloyd în 1980). Aceștia au propus ca semnătura unei înregistrări să fie creată prin concatenarea semnăturilor atributelor. Această idee crează probleme în special la regăsiri pentru că și o cerere simplă necesită a fi expandată într-una disjunctivă, ceea ce înseamnă timp mare de răspuns.



Delta Design S.A., Hotel București, 63-81 Victoria Street,
Wing D - Agencies, Stair E 2, 4th Floor, Sect. 1, Bucharest,
Phones: 15.42.26 / 40 015.45.80 Ext.2404 / Fax: 40 13.60.40.

99.94% of the users fully satisfied of the delivered products

Authorised distributors for the following companies:

Tulip Computers, Apple Computer Inc., Microsoft, Autodesk,
Ashton Tate, Claris, Corel, Delta Design International etc.

Fundamente

Presupunând că avem 6 cuvinte cheie/document, și căutăm documentul ce conține cuvântul "informație", cererea pentru regăsirea acestui cuvânt va trebui formulată:

cuv - cheie1 = "informație" sau

cuv - cheie2 = "informație" sau

....

cuv - cheie6 = "informație".

Mai elegantă, metoda lui Gustafson ridică următoarele probleme:

- Performanțele descresc cu creșterea fișierului;
- Numărul de cuvinte cheie dintr-un document este mare și, în acest caz, hash-tabela trebuie să fie foarte mare; în situația în care este formulată o cerere cu 3-4 cuvinte, va fi parcursă o mare porțiune a bazei de date. De asemenea, cererile care nu sînt conjunctive, crează dificultăți.

Ideea lui Gustafson a fost preluată și de alți cercetători care au încercat să o facă aplicabilă, dar nici unul nu a găsit o soluție optimă. Knott (1971) a propus utilizarea unei hash-funcții adaptive, care se schimbă în funcție de mulțimea și distribuția termenilor inserați. Aceste funcții utilizează o structură de arbore suplimentară, nodurile acestei structuri fiind pointeri către grupuri de informație aflată pe un mediu de stocare secundar. Codul unui atribut se obține prin parcurgerea acestui arbore. Larson (1978) a dezvoltat această idee folosind o pădure de arbori ca structură auxiliară de date. Martin (1979) sugerează o codificare în "spirală" bazată pe o funcție exponențială (neuniformă). Lloyd și Ramamohanarao (1982) utilizează o cheie secundară, etc.

Aceste metode nu au dat pînă acum rezultate practice, convenabile. Cea de-a doua idee, cea a stocării semnăturilor într-un fișier separat, fișierul semnătură, este mult mai interesantă.

În acest caz nu apare restricția legată de numărul fix de cuvinte cheie dintr-un document, pentru că documentul poate fi divizat în blocuri logice care vor conține un număr constant de cuvinte cheie. În 1969, Files și Huskey au aplicat această metodă pentru o bază de date bibliografică. Ei au folosit o listă de cuvinte ce se exclud de la codificare și o procedură de evitare a acestora. Pentru codificare, au utilizat o procedură numerică drept hash-funcție.

Tsichritzis și Christodoulakis (1983), urmași de Larson (1983) au propus utilizarea fișierului semnătură fără a întrebuița suprapunerea codurilor. În felul acesta, poziția informației este păstrată, pentru că sînt stocate semnăturile fiecărui cuvânt, una după cealaltă (concatenate). În 1984, Rabitti și Zizka au atras atenția că metoda prin concatenare implică legături mai greoaie cu CPU (unitatea centrală) decît cea a suprapunerii codurilor.

Pflatz (1980), a sugerat utilizarea unui descriptor indexat de fișiere (în terminologia acestuia), de fapt un sistem de mai multe nivele de fișiere semnătură. Semnătura din nivelul i este obținută prin suprapunerea semnăturilor de pe nivelele i - 1. Semnătura înregistrărilor de pe primul nivel este obținută prin concatenarea semnăturilor corespunzătoare cuvintelor cheie din înregistrare.

Fig. 2. Fișiere semnătură pe mai multe nivele:

Fișier de date			descriptor nivel 1	descriptor nivel 2
Nume	Sex	Salariu		
Smith, J.	M	40000	010 0 01	010 1 11
Smith, M.	M	50000	010 0 10	100 0 11
Smith, P.	F	20000	010 1 00	
Wong, B.	M	15000	100 0 00	
Wong, M.	M	60000	100 0 11	
Wong, T.	M	50000	100 0 10	

Roberts (1979) a folosit o metodă cu un singur nivel de fișier semnătură pentru o aplicație tip agendă telefonică. El a propus următoarele idei interesante:

- Stocarea semnăturilor în fișierul semnătură se face în maniera "bit-slice", adică primul bit al tuturor semnăturilor este stocat consecutiv, apoi al doilea, și așa mai departe. Dacă îngreunează actualizările, această metodă reduce foarte mult timpul de regăsire.
- Crearea semnăturilor în așa fel încît termenii care apar frecvent în cererile de regăsire să fie tratați într-un mod special.

În 1985, Faloutsos a făcut o analiză matematică a modului de organizare a fișierului semnătură pentru evitarea rezultatelor false la regăsire, și a stabilit un sistem de reguli ce trebuiesc aplicate la proiectarea unei unui fișier semnătură pentru a putea reduce cu 50% rezultatele false (false drops).

Principalele dezavantaje sînt legate de timpul de răspuns mare în cazul atunci cînd fișierul semnătură este mare și bineînțeles, probabilitatea răspunsurilor false.

2. Metode de proiectare a fișierului semnătură și de extracție a semnăturilor

Semnătura fiecărui cuvânt (Word Signature)

Vom nota în continuare această metodă cu WS. Fiecare cuvânt al unui document (cu excepția cuvintelor comune: și, sau, etc.) este codificat într-un șir de F

bits (nu intervine un număr fix de bits 1) și aceste coduri sînt concatenate pentru a forma semnătura documentului. Pentru regăsire, este codificat cuvîntul cerut și apoi, este parcurs întreg fișierul semnătură, pas cu pas.

Fig. 3. Ilustrare a metodei WS

	text	metodă	codificare
Semnătura cuvîntului	0000	0100	0111
Semnătura documentului	0000	0100	0111

Suprapunerea codurilor (Superimposed Coding)

Ne vom referi la această metodă, în continuare, prin SC. Ea a fost propusă de Christodoulakis și Faloutsos în 1984 și se bazează pe metoda propusă de Moores în 1949. Fiecare document este divizat în blocuri logice, conținînd fiecare D cuvinte distincte și necomune. Fiecare cuvînt este codificat într-un șir de F bits, dintre care m sînt 1, apoi aceste coduri sînt suprapuse cu SAU logic pentru a obține semnătura blocului (vezi Figura 1). Alegerea valorilor pentru D, F și m se face în așa fel încît numărul de bits 1 din semnătura blocului să reprezinte jumătate din numărul total de bits, în felul acesta realizîndu-se optimizarea performanțelor. Pentru căutarea unui cuvînt, se formează mai întîi semnătura aceluia cuvînt. Să presupunem că va avea bits 1 în pozițiile 1, 5, 7, 9. Vor fi examinate pe rînd semnăturile blocurilor și, dacă în pozițiile 1, 5, 7, 9 va exista 1, acele blocuri vor fi considerate ca avînd inclus acest cuvînt.

În ideea de a avea posibilitatea de regăsire a termenilor trunchiați, pentru a fi codificare și stocare, cuvintele sînt divizate în triplete, avînd marcat începutul și sfîrșitul.

Fig. 4. Exemplu pentru codificare dacă se dorește regăsire de termeni trunchiați.

I = "Început", E = "Sfîrșit"

$Sigw(data) = Sig(I,d,a) \text{ OR } Sig(d,a,t) \text{ OR } Sig(a,t,a) \text{ OR } Sig(t,a,E)$

Compresia blocurilor de bits (Bit-block compression)

Această metodă va fi referită prin BC. Propusă pentru prima dată de Faloutsos (1985), metoda BC presupune utilizarea pentru codificare metoda SC, ca apoi vectorul de bits corespunzător unui bloc să fie comprimat separat. Comprimarea se poate face în mai multe feluri, Faloutsos a propus memorarea poziției bits 1 în cadrul blocului.

Codificarea pe întreaga lungime (Run - length encoding)

Mclroy, în 1982, a aplicat pentru prima dată această metodă, la care ne vor referi cu inițialele RL. Este o metodă de compresie, Mclroy comprimînd cu ajutorul ei un dicționar de 30000 de cuvinte. Fiecare cuvînt este codificat într-un vector de bits conținînd un număr fix de bits 1. Ceea ce se memorează este numărul de 0 dintre două apariții succesive de 1. Avantajul acestei metode este legat de compresia excelentă pe care o realizează, dar, timpul de răspuns la interogare este foarte încet. Deasemenea, vectorul trebuie să conțină zone largi de bits 0, pentru ca compresia să fie eficientă.

Codificarea în ordinea poziției (Bit - Slice)

Metoda BS, este, după cum se va vedea foarte eficientă la regăsiri, dar pune probleme serioase operațiilor de actualizare sau ștergere. Codificarea cuvintelor se face cu o hash-funcție, și memorarea lor se realizează în felul următor: prima dată sînt memorăți bits de pe prima poziție, apoi cei de pe a doua, ș.a.m.d. Se poate aplica această metodă în combinație cu metoda SC, obținîndu-se semnături ale blocurilor care sînt stocate așa cum a fost arătat mai sus.

Fig. 5. Exemplu pentru metoda BS.

Să presupunem că avem semnăturile a 4 blocuri, $F=5$:

(01001), (01101), (11101), (00110)

Memorarea acestora în fișierul semnătură se va face:

(0010), (1110), (0111), (0001), (111)

Să presupunem că se dorește regăsirea cuvîntului a cărui semnătură este $Sig(q) = (00101)$. Algoritmul de verificare va cerceta doar pozițiile 3 și 5 (acolo unde este 1) și rezultă pentru poziția 3 blocurile 2,3,4 și pentru poziția 5 blocurile 1,2,3. Rezultatul se obține prin intersecția celor două răspunsuri, deci cuvîntul căutat se va afla în blocurile 2 și 3.

3. Comparație între metode.

Toate variantele discutate ale metodei semnăturii pot da răspunsuri false ("fale drops"), pentru că este posibil ca o semnătură să corespundă unei întrebări, dar de fapt cuvîntul respectiv să nu existe în textul respectiv. Notăm probabilitatea de a avea un răspuns fals cu F_d .

F_d = probabilitatea ca o semnătură să rezulte în urma aplicării metodei că aparține unui text și de fapt să nu aparțină.

Faloutsos și Christodoulakis au studiat în 1984 F_d pentru metodele WS și SC.

Fundamente

În cazul WS, F_d este:

$$F_{d,ws} = 1 - (1 - 1/S_{max})^D \quad (1)$$

Ecuția (1) poate fi justificată într-un mod intuitiv. Ea indică faptul că F_d este independentă de dimensiunea vocabularului, a bazei de date sau de frecvența aparițiilor unor cuvinte.

În cazul SC, F_d este:

$$F_{d,sc} = (1/2)^{m_{opt}} \quad (2)$$

$$m_{opt} = F \cdot \ln 2 / D \quad (3)$$

Aceste valori au fost determinate statistic pentru o bază de date cu intrări bibliografice de 3,3 MegaByte. Ecuția (3) implică concluzia că F_d este minim atunci când jumătate din numărul de bits din blocul semnătură este format din bits 1.

În 1985, Faloutsos a stabilit formule aproximative din care poate rezulta variația F_d în funcție de creșterea dimensiunii F a semnăturii.

$$\log 2F_{d,ws} = \log 2D - F_{ws}/D \quad (4)$$

$$\log 2F_{d,sc} = -F_{sc} / (D \cdot \log 2e) = \\ = -F_{sc} / D \cdot 0,693 \quad (5)$$

$$\log 2F_{d,rl} = n \cdot (1 + \log 2 \log 2e) - F_{rl}/D = \\ = 1,528 \cdot n - F_{rl}/D \quad (6)$$

$$\log 2F_{d,bc} = n \cdot (1 + \log 2e - \log 2 \log 2e) - \\ - F_{bc}/D = 1,913 \cdot n - F_{bc}/D \quad (7)$$

(Nu am găsit o formulă pentru metoda BS)

În aceste ecuații:

F_{d,xx} = F_d pentru metoda XX

F_{xx} = Dimensiunea blocului semnătură pentru metoda XX

D = Numărul de cuvinte noncomune din bloc

S_{max} = Numărul maxim de semnături distincte

(WS)

f = Dimensiunea semnăturii cuvântului în bits (WS)

m = Numărul bits 1 din semnătura blocului (SC)

m_{opt} = Valoarea optimă pentru m (SC)

B = Dimensiunea vectorului (BC, RL)

n = Numărul bits 1 corespunzători unui cuvânt (BC,

RL)

b = dimensiunea blocului de bits (BC)

b_{opt} = valoarea optimă pt. b (BC)

Concluziile ce rezultă din aceste formule și din studii experimentale și statistice sînt:

- Pentru metode bazate pe compresie, cele mai bune rezultate se obțin pentru n=1.
- Pentru toate metodele, curbele devin linii drepte pentru dimensiuni mari ale semnăturilor.
- Cel mai bun timp la regăsire îl oferă BS și apoi SC.

- RL este cea mai economicoasă metodă în privința spațiului folosit.
- BC reprezintă un compromis între SC și RL.
- WS păstrează ordinea aparițiilor cuvintelor în texte.
- Metoda BS ridică probleme la actualizare.
- Dimensiunea fișierului semnătură este, pentru metodele SC, BS, WS aproximativ 10% din documentele originare (baza de date), în comparație cu un fișier invers ce ocupă aprox. 300%.

În momentul cînd, în urma studiului materialelor din bibliografie, am construit structura de date abstracte Signature pentru arhivare și regăsire de date documentare prin metoda semnăturii, am ales metodele SC și BC pentru implementare ca avînd cele mai bune performanțe și fiind ușor de implementat. În cazul unei aplicații ce folosește această structură de date abstracte, în funcție de tipul acesteia (se cere un timp de răspuns foarte bun sau nu, se fac deseori actualizări sau nu, etc.) se alege una din soluții.

Pentru metoda SC am ales:

$$F = 256;$$

$$D = 40;$$

atunci

$$m_{opt} = F \cdot \ln 2 / D = 4,58$$

De aici, am ales

$$m = 5.$$

Problema metodei semnăturii rămîne încă o problemă de cercetare, dezvoltare și optimizare viitoare.

aprilie, 1992

ing. Mirela Guțică

Bibliografie:

- Access Methods for Text - C. Faloutsos - 1985;
- Signaturverfahren fur Sekundarschlüssel-Zugriff - H. Eirund - Oldenburg - W. Deutschland, 1991

MODULELE MULTICIP

UN NOU CONCEPT PENTRU ÎNCAPSULAREA DE MARE DENSITATE

Introducere

În ultima decadă se vorbește tot mai mult despre circuitele integrate specifice pe aplicații, deoarece aceste ASIC (fie ele rețele de porți sau circuite complet la comandă) au evoluat rapid dintr-o alternativă de proiectare neobișnuită și costisitoare către metoda preferată (cea mai eficientă) de proiectare a sistemelor.

Datorită progreselor tehnologiilor semiconductoare și perfecționărilor majore în domeniul proiectării și producției asistate de calculator, problemele de proiectare și realizare structuri au fost rezolvate, dar se ivesc altele: interconectare, încapsulare, montare în echipament. Tehnologia clasică, ce a fost utilizată cu succes pentru încapsularea dispozitivelor LSI, nu mai poate satisface cerințele VLSI. De fapt, multe din beneficiile câștigate de VLSI pe cip se pierd acum la încapsulare. Plăcile de cablaj multistrat și circuitele hibride cu straturi groase [1], cu dimensiunea minim absolută a traseelor de $125/\mu\text{m}$ (pas de $250/\mu\text{m}$), sînt inadecvate pentru interconectarea corespunzătoare a acestor dispozitive complexe, cu număr foarte mare de intrări și ieșiri. Este necesară o tehnologie care să accepte o densitate de încapsulare sporită și viteze mai mari ale semnalelor.

La vitezele de lucru ridicate la care s-a ajuns în prezent apar certe limitări în privința capsulelor. Astfel, PQFP (capsulele plate de plastic) actuale, cu montare pe suprafață, merg foarte bine la frecvența de 50

Mhz și poate chiar 80 MHz (necesitînd terminale suplimentare de masă și de alimentare pentru ecranarea laterală împotriva diafoniei), dar nu și la peste 100 MHz.

Mai importantă este căldura disipată generată de circuitele ultrarapide actuale, chiar și în tehnologia CMOS. Unele tipuri depășesc 1 W, apropiindu-se de limita de 1,2 W a capsulei PQFP. Înseamnă că sînt necesare noi materiale de încapsulare, cu conductivitate termică mare, sau noi soluții constructive de PQFP, care să încorporeze radiatoare sau "vias" termice.

Încapsularea în sistemul TAB (Tape Automated Bonding), extensia logică pentru QFP, permite densități mai mari de interconectare (cercetarea de laborator arată că se pot chiar depăși 1000 de terminale, dar în viitorul apropiat, aplicațiile uzuale, eficiente ale TAB vor fi pentru circuite cu 300 - 400) însă ridică probleme la "implementare".

Procesul clasic de lipire prin retopire (solder reflow) permite, la majoritatea fabricanților, montarea capsulelor cu pas de 0,63 ... 0,5 mm. Dar, pentru mai mult de 400 de terminale, este necesar, conform tabelului de mai jos, un pas mai mic de 0,4 mm, care pune în pericol montarea subsistemelor, deoarece probabilitatea formării scurtcircuitelor (punți de aliaj de lipit) și a golurilor (circuite deschise) devine foarte mare. Deasemenea, la un pas de 0,4 mm, trebuie utilizat un sistem de recunoaștere automată a formelor, pentru a plasa cu precizie componentele ASIC pe placă.

Adoptarea tehnologiei TAB cu pas fin înseamnă ca producătorii de sisteme trebuie să treacă la o combinație de retopire convențională și sudură a terminalelor exterioare (OLB), așa cum a fost pusă la punct pentru TAB.

Tabelul 1. Relația între numărul de terminale și pasul necesar:

Nr. pini	200	300	400	500	600	700	1000
Pas(mm)	0.6	0.5	0.4	0.3	0.3	0.1	0.04
	clasică		AB OLB				

Adoptarea deciziei de trecere la TAB duce la unele consecințe inevitabile:

- un echipament de producție pentru sudura conexiunilor exterioare pentru TAB poate costa peste 100.000\$. Sudura OLB este un proces secvențial, relativ lent comparat cu abordarea clasică pentru circuitele ASIC cu montarea pe suprafață: robot de plasare tip pick-and-place și cuptor de retopire, deci o prelucrare în paralel, pe loturi. În loc de mai multe componente pe secundă, durează mai multe secunde interconectarea unui dispozitiv;
- inginerii producători de sisteme trebuie să învețe să decupeze și să plieze terminalele într-o formă adecvata locului de amplasare. Pînă acum, operația era efec-

tuată pentru ei de fabricantul de dispozitive la cerere. Se obține astfel un avantaj prin faptul ca se minimizează manevrarea și transportul dispozitivelor cu terminalele formate, care sînt fragile. Se reduce astfel deplasarea terminalelor și apariția de necoplanități;

□ apar considerente de proiectare termică. În tehnologia TAB cu pas fin, inginerii de sistem vor avea de făcut față unor densități de putere disipată de 4 - 10 ori mai mare. Volumul mare al circuitului integrat, atunci cînd era încapsulat de către producătorii de ASIC, putea disipa mult mai multă căldură decît o face structura neîncapsulată cu care lucrează proiectanții în tehnologia TAB. Dacă proiectanții de sisteme vor utiliza circuite ASIC cu pas fin direct pe placă, ei trebuie să înțeleagă și să rezolve problemele termice suplimentare ce apar, atît pe placă cît și la nivel de sistem;

□ în fine, costurile inițiale asociate tehnologiei TAB sînt ridicate. Producătorii de echipamente mai ambițioși ce doresc să treacă la tehnologia de mare densitate, să facă tranziția de la tehnica tradițională, de montare a componentelor în găuri la varianta de montare pe suprafață cu pas fin, trebuie să investească între 3 și 10 milioane de dolari, sau chiar mai mult, în funcție de volumul de producție și dezvoltările implicate.

Noua soluție - MCM

Mai mult ca oricînd, pentru îmbunătățirea performanțelor ASIC sînt esențiale metodologii de încapsulare noi. Limitările de interconectare (rutare) și fabricație ale cablajelor imprimate convenționale creează o cerere mare pentru plăci cu densitate de linii mai mare fără un preț considerabil sporit. La fel, menținerea densității actuale a plăcilor și dezvoltarea unor noi

scheme de încapsulare a componentelor ar putea satisface cerințele noilor produse. Între timp, o tehnologie evoluționară de încapsulare, de departe asemănătoare cu circuitele hibride, prezintă soluții clare pentru aceste cerințe de ASIC cu pas fin care tocmai apar. Modulele multicip (MCM), ce conțin o putere de calcul considerabilă sub forma de VLSI în cadrul a numai cîteva componente, apar acum în capsule obișnuite, precum DIL sau QFP. Ca sistem selfconsistent, MCM-urile prezintă proiectanților o capsulă cu un număr mai redus de terminale/pini, ce pot fi manipulate în mod obișnuit de echipamentele de producție deja existente.

Principiul constă în utilizarea interconectărilor cu straturi subțiri pe siliciu. În structurile de siliciu este inerentă o interconectare cu densitate foarte mare de trasee, mult superioară celei ale cablajelor imprimate clasice sau modulelor multistrat pe ceramică (MCA). Această capacitate se aplică la MCM-uri pentru a produce substraturi pe baza de siliciu, în fabrici de plachete depășite moral. Uzura morală a fabricilor de VLSI fiind foarte ridicată actualmente, reprofilarea lor pe efectuarea de interconexiuni pentru modulele multicip constituie o "valorificare superioară" ce nu-i de neglijat. Se pot realiza curent interconexiuni pe substraturi de siliciu, care sînt cu cel puțin un ordin de mărime mai avansate decît structurile clasice (cablajele sau circuitele hibride pe ceramică).

Un substrat de Si activ pentru MCM reduce deasemenea puterea totală disipată, ceea ce înseamnă o reducere majoră de preț. Cînd se atașează mai multe cipuri la un substrat de modul, liniile / căile de semnal sînt mai scurte, ceea ce reduce puterea de comandă necesară pe cip. Ori, se știe că la CMOS disipația statică este foarte redusă, iar cea dinamică depinde de frecvența de lucru, tensiunea de alimentare și capacitățile de sarcină. Liniile mai scurte înseamnă capacități mai reduse. Cînd substratul de siliciu este

populat cu elemente active și pasive precum tranzistori de comandă, rezistoare de sarcină (adaptare, terminatoare de linie, tragere "pull-up" la alimentarea pozitivă), condensatoare, puterea degajată în modulul respectiv se reduce cu circa 20%. Acest nivel este semnificativ pentru minimizarea costului sistemelor de înaltă performanță bazate pe circuite ASIC.

La nivel elementar, MCM de astăzi combină 4 - 10 elemente diferite pe un substrat, fie cu straturi subțiri, fie cu straturi groase. O derivată a MCM, denumită placă funcțională, apare acum ca o tehnologie mai avansată ce adăpostește tehnologii LSI complexe. Cele mai multe aplicații pentru plăcile funcționale au fost fixate să utilizeze un conector clasic cu 60 de pini la capăt sau un conector cu 120 de pini pe laterală.

Inițial, acest MCM particular a debutat ca o extensie de memorie (pe placă) în sistemele de calcul (add-in). Pe placa de bază a calculatorului pot fi incluși 2, 4 sau 16 Mb de memorie. Acum, datorită cipurilor pe bandă TAB, plăcile funcționale pot include modemuri, procesore grafice, coprocesoare matematice sau de comunicații, controllere pentru interconectarea în rețeaua locală (LAN).

Un astfel de modul multicip avansat poate conține, la dimensiuni de numai 5 x 5 cm, pînă la 45 de cipuri de integrare LSI. De exemplu, memoria tridimensională produsă de Texas Instruments conține 8 EEPROM-uri de 256 K pe 9 biți și 32 de cipuri de RAM statice de 1 M a 4 biți, plus 2 circuite specifice (ASIC) și 3 de prelucrare digitală a semnalelor (DSP). Folosind organizarea tridimensională, substraturile de Si și interconectarea flip-TAB în interiorul modulului, exteriorul se prezintă ca o capsulă "clasică" pentru montarea pe suprafață cu pas fin. Într-adevar, Quad - Flatpack-ul respectiv are 308 terminale cu pas de 0,635mm și poate fi montat cu echipamentele actuale, în vreme ce în

THE

choice

 **AUTODESK**

Microsoft

 **NOVELL**

Lotus

 **Fox**

BORLAND

Aldus

COREL

Symantec



A&C
INTERNATIONAL S.A.

in software



Tel 40-0-53.53.15.

interior sînt 800 de conexiuni, interconectate cu un pas foarte fin.

Producătorii de sisteme trebuie să constate care tehnică de încapsulare corespunde cel mai bine nevoilor unor sisteme particulare. Trebuie să evalueze cu grijă mai mulți parametri de selecție, ce includ costul, înălțimea, disipația termică, nivelul de performanțe, materialele și factorul de formă. Deși flip - TAB - ul merge bine, pot exista și alte abordări, precum urmează:

1. În metoda devenită clasică, cipuri pe placă interconectate TAB (TAB COB), structurile sînt pe cablaj cu bumbii în sus, iar terminalele exterioare ale grilei TAB sînt pliate astfel încît să emuleze o capsulă convențională pentru montarea pe suprafață, cu terminale în aripa de pescăruș. Această variantă este aplicată astăzi în multe sisteme comerciale, fiind relativ ieftin de implementat, iar factorii de formă sînt similari celorlalte capsule.

2. Dacă minimizarea spațiului pe verticală este de o importanță deosebită, cum ar fi aplicațiile pentru "module inteligente" ce necesită o grosime minimă (cărți de credit inteligente), atunci este mai bine de utilizat metoda cu cipuri în placa cu TAB cu "buzunar" (P-TAB CIB). În această variantă (Chip In Board, deci cipuri în interiorul plăcii) cipurile sînt plasate în cavități ale substratului (buzunare) astfel încît înălțimea totală se reduce.

3. În fine, dacă evacuarea puterii disipate este principala problemă a sistemului (dispozitive de putere, circuite ultrarapide), se utilizează cipuri răsturnate pe placă, interconectate TAB. Cipurile sînt cu fața cu bumbii de contact în jos, iar spatele este disponibil pentru atașarea la un sistem de disipare termică adecvat, care să asigure performanțele termice dorite.

Perspective

În viitorul apropiat, MCM pe substrat de siliciu vor permite pro-

iectarea și realizarea unor funcții foarte puternice în module de numai 5 x 7,6 cm. Exemplul reluat de mai multe reviste occidentale prezintă un cablaj special (cu miez de Invar) cu componente în tehnologia montării pe suprafață, suporturi de structuri cu și fără terminale, cu pași fini, de 0,65 și 0,5 mm. Placa aceasta, "variante veche", are o suprafață de 15,5 x 15,5 cm², iar echivalentul funcțional "nou" în MCM (cu componente flip - cip TAB) are dimensiuni 6 ori mai reduse. Această abordare nu este încă în producția de serie, dar în laborator se ivește deja următoarea generație de MCM-uri. Folosind pe substrat structuri flip-cip, se pot reduce în continuare dimensiunile, cu prețul unor complicații la montaj. Substratul multicip poate fi apoi înglobat într-o capsulă clasică sau atașat direct pe placa de cablaj. În următorii doi ani (91 - 93), tehnologia MCM se va infiltra în aplicațiile de sistem de înaltă performanță, atît în domeniul comercial, aerospațial cît și militar. Desigur, în special armata va exploata reducerile de dimensiuni și greutate pe care le asigură MCM-urile. Sistemele existente vor fi îmbunătățite (upgraded) amplasînd funcții și posibilități suplimentare într-o cutie de aceleași dimensiuni.

MCM avansate

Etapa ulterioară (1993 - 95) va consta din module multicip care încep să-și adauge blocuri constructive tridimensionale (spațiul de deasupra plăcii de interconectare adăpostește circuitele integrate. Într-o încapsulare 3D, dispozitivele active sînt stivuite și montate perpendicular pe planul de interconectare ce distribuie semnalele și tensiunile de alimentare între dispozitive. Este cea mai densă dintre tehnicile de împachetare cunoscute - dacă puterea disipată nu ridică probleme, încapsularea tridimensională promite o îmbunătățire a densității de memorie de pînă la 250

- 300 de ori față de DIP-uri sau SOIC-uri. Spre jumătatea anilor '90 modulele de sistem vor fi bazate pe conceptul de placă funcțională. Vor folosi substraturi de siliciu și vor beneficia de avantajele celei de-a treia dimensiuni. Aceasta se realizează în prezent în laborator cu sistemele de montaj ce folosesc tehnologiile TAB și flip-cip.

În plus, topologia de substraturi adaptată aplicației rezolvă problemele de interconectare între numeroasele cipuri.

Substraturile speciale de siliciu și MCM-urile stivuite disipă suficientă căldură ca să accepte densitățile mari de putere ce rezultă din cipurile strîns împachetate. Siliciul are o bună conductivitate termică, iar acolo unde este necesar se utilizează disipatoare de beriliu. Un avantaj suplimentar al substraturilor active de siliciu este posibilitatea de a încorpora circuite funcționale complexe și/sau componente pasive.

Cristian Malide - ICCE București

Bibliografie

1. Brevetul US 4.551.788 din 5 nov. 1985, pe numele lui Daniel, Larry Yglesia intitulat "Multi-Chip Carrier Array".

2. Brevetul US 4.764.846 din 16 aug. 1988, atribuit lui Go, T.G. de la Irvine Sensors, SUA, "High Density Electronic Package Comprising Stacked Sub-Modules",

3. Rossi, R.D. (National Starch Chemical), "High Density Interconnection Packaging of I.C.'s", HCT sept.'89, pag. 35.

4. Barbou, R. (Sextant Avionics), "Hybrids + ASIC's = A High Density Packaging Concept", HCT nov.'89, pag. 11.

5. Buschbom, M.L. (Texas Instruments), "ASIC Designs Demand Broader Perspective", Electronic Design, 11 Jan.'90, pag. 111.

**MIFIX
GESTOC
REPARATII
METROLOGIE**

**P micro
ATCI S.R.L.**

**ISCIR
CASA
BANCA
SALARII**

FACTURARI

PLAY.ASM

Vă propun un program care crează un fond sonor interesant. Acest program se instalează rezident în memorie și lucrează cu trei întreruperi: cea de disc (int 13H), cea de tastatură (int 09H) și cea de timp (1CH).

Funcționare: orice acces la disc va trece printr-o procedură care, dacă accesul reprezintă o scriere/citire/formatare, va emite un sunet a cărui frecvență depinde de cilindrul și sectorul accesat. (Pentru asta există vectorul "tabela", care asociază '1' unei asemenea funcții și '0' unei funcții de resetare/interogare etc. - codul funcției este primit în registrul AH). Procedura care lucrează pe întreruperea de timp va opri semnalul sonor după circa 0.33 secunde, pentru a nu deveni supărător. Procedura pentru tastatură va comuta (sunet sau nu) după cum este sau nu activă tasta Scroll Lock.

Programul este absolut original.

ing. fiz. Cristian Paris

PLAYASM

```

;
;RADIX 16
;   Constante:
PORT_A equ 60h ; adresa port A 8255
PORT_B equ 61h ; adresa port B 8255
TIMER equ 40h ; adresa port timer 8253
;
;   Macro def:
;-----
get_set_vect MACRO n ; citeste un vector de intrerupere
mov ax,35&n ; si scrie in loc un nou vector
int 21
mov cs:oldo&n,bx
mov cs:olds&n,es
mov ax,25&n
mov dx,offset newint&n
int 21
ENDM
;-----
cseg SEGMENT
ASSUME cs:cseg

ORG 2c
env_seg label word ; adresa context program

ORG 100
start: jmp entry ; punct de intrare in cod
;
;   Sectiune de date
;-----
oldvec1c label dword ; adresa originala procedura timer
oldo1c dw ?
olds1c dw ?
oldvec13 label dword ; adresa originala procedura disc
oldo13 dw ?
olds13 dw ?
oldvec09 label dword ; adresa originala procedura keyboard
oldo09 dw ?
olds09 dw ?

```

```

tabela db 0,0,1,1,1,1,0,0,0,0,0,1,1,0,0,1,1
db 10 dup(0)
contor db 0 ; daca < 6,are voie sa cinte
steag db 1 ; comandat de la tastatura:
; steag == 1,are voie sa cinte
; steag == 0,nu are voie
;-----
; Procedura apelata de 18.2 ori pe secunda ( INT 1Ch )
;
newint1c PROC far
pushf
call cs:oldvec1c ; executa procedura originala
cmp byte ptr cs:steag,1 ; are voie sa cinte ?
jne short gata ; nu
inc cs:contor ; contor + +
cmp byte ptr cs:contor,4 ; contor == 6 ?
jne short gata ; nu,lasa sa cinte
push ax ; da,opreste sunet...
in al,PORT_B
and al,0fch
out PORT_B,al
pop ax
mov byte ptr cs:contor,0 ; ...si reseteaza contor
gata: irt
newint1c ENDP
;-----
; Procedura de tratare intrerupere disc (INT 13h)
;
newint13 PROC far
cmp ax,0ff00 ; pentru autorecunoastere
jne short ok
xchg ah,al
irt
ok:
cmp byte ptr cs:steag,1 ; are voie sa cinte ?
jne short gata2 ;nu
push ax
push bx
lea bx,tabela
mov al,ah ; codul functiei pus in AL
xlat tabela
cmp al,1 ; AL == 1 ? da,sa sune
jne short gata1 ; nu,sa nu sune
;
; Construiesc frecventa sunetului dupa formula
; (empirica): (NrSector*16) OR ((NrCap+1)*128)
;
push cx
push dx
and cl,3f ; mascheaza bitii de pista
xor ch,ch
mov bx,cx ; NrSector in BX
mov cl,4
shl bx,cl ; NrSector*16
xor dl,dl ; DX va contine NrCap*256...
inc dh ; ... +256
shr dx,1 ; ... /2
or bx,dx ; BX contine frecventa sunetului
;
; Cinta pe frecventa din AX
mov ax,34ddh ; contor timp = 1234ddh
mov dx,0012
div bx ; imparte (DX:AX) la BX
mov bx,ax
in al,PORT_B
test al,3 ; difuzor pornit ?
jne short go ; da,continua
or al,3 ; nu,porneste difuzor
out PORT_B,al

```

```

mov     al,0b6      ; selecteaza TIMER 2
out     TIMER + 3,al ; scrie TIMER MODE REG

go:
mov     al,b1
out     TIMER + 2,al      ; scrie LSB
mov     al,bh
out     TIMER + 2,al      ; scrie MSB
mov     byte ptr cs:contor,0 ; reseteaza contor
pop     dx
pop     cx

gata1:
pop     bx
pop     ax

gata2:
jmp     cs:[oldvec13] ; executa procedura originala
newint13 ENDP
;-----
; Procedura de tratare intrerupere keyboard
;
newint09 PROC far
push    ax
in      al,PORT_A
cmp     al,46      ; a fost ScrollLock ?
jne     short iese ; nu,continua
xor     byte ptr cs:steag,1 ; da,schimb steag
in      al,PORT_B ; opreste sunet
and     al,0fc
out     PORT_B,al

iese:
pop     ax
jmp     cs:[oldvec09] ; executa procedura originala
newint09 ENDP
;-----
; Zona de cod tranzient:
;-----
last_adr label byte ; sfirsit cod rezident
mess     db "Play installed,press ScrollLock to (com)mute."
         db 0d,0a,"$"
mess2    db "Play already installed.",0d,0a,"$"
;
entry:
mov     ax,0ff00   ; autorecunoastere
int     13
cmp     ax,0ff    ; deja instalat ?
jne     short inst ; nu,instaleaza
mov     ah,09     ; da,gata
lea     dx,mess2
int     21
mov     ax,4c00   ; termina
int     21

inst:
IRP     int_no,c,13,09
get_set_vect int_no
ENDM

lea     dx,cs:mess
mov     ah,09
int     21
mov     es,cs:env_seg ; elibereaza blocul de context
mov     ah,49
int     21
mov     dx,offset cs:last_adr ; marime cod rezident...
mov     cl,4      ; ...transformata in paragrafe
shr     dx,cl
inc     dx        ; ... + 1 pentru siguranta
mov     ax,3100  ; termina rezident
int     21

cseg    ENDS

END     start

```

Program autoreproducător

Referitor la articolul "Cine ridică mănusa?" apărut în "if" nr. 8/91, în care ați tratat problema conceperii unui program autoreproducător, vă trimit un program scris în TURBO PASCAL 5.0 care afișează pe ecran în urma compilării și execuției copia exactă a sursei sale.

Ideea programului este de a afișa fiecare rînd din partea de declarații a programului și în același timp instrucțiunea prin care se realizează afișarea este concatenată la un șir ce va conține în final toate aceste instrucțiuni, separate prin perechea de caractere CR/LF. Această dublă acțiune este realizată de procedura q. Ulterior, șirul S este tipărit prin procedura r, care afișează și sfîrșitul programului.

Am folosit în locul caracterelor codurile ASCII ale acestora pentru a evita dublarea semnelor apostrof ce delimitează un șir de caractere.

mat. Cristian Nemeș

```

program p;
var S:string;
procedure q(T:string);
begin
    writeln(T);
    S := S + #113#40#39 + T + #39#41#59#10#13
end;

procedure r;
begin
    write(S, #114#10#13#101#110#100#46)
end;

begin
    q('program p;');
    q('var S:string;');
    q('procedure q(T:string);');
    q('begin');
    q('writeln(T);');
    q('S := S + #113#40#39 + T + #39#41#59#10#13');
    q('end;');
    q('procedure r;');
    q('begin');
    q('write(S,#114#10#13#101#110#100#46)');
    q('end;');
    q('begin');
    r
end.

```

Meniuri verticale în PARADOX

Înainte de a vedea cum se rezolvă problema meniurilor verticale în sistemul de gestiune a bazelor de date PARADOX, trebuie să precizăm câteva aspecte.

Astfel, în primul rând trebuie subliniat faptul că acest S.G.B.D. încurajează foarte mult lucrul în propriul său "stil". Aceasta implică - de fapt - că programele trebuie să fie ușor de scris, scurte și rapide. Bineînțeles că, folosind comenzile și funcțiile PAL (Programming Application Language - limbajul de programare propriu PARADOX), se poate devia de la stilul impus; în acest caz trebuie să facem un compromis și să acceptăm faptul că este nevoie de un cod mai complex care, implicit, poate să ruleze mai încet.

Creerea de meniuri verticale este un astfel de caz în care se deviază de la "stilul PARADOX". Interfața cu utilizatorul a PARADOX-ului este proiectată în jurul meniurilor orizontale. În prezent, interfețele grafice cu utilizatorul (GUI = Grafic User Interface) utilizează meniurile verticale, modă ce a fost introdusă de lansarea pe piață a Windows-ului, și ca exemple ar putea fi citate majoritatea IDE (Integrated Development Environment), majoritatea utilităților - PC Tools, Norton Commander, etc.

Meniurile orizontale se pot genera simplu în PARADOX utilizând comanda `SHOWMENU` cu următoarea sintaxă:

Showmenu

Optiunea1: Descrierea1;

.

OptiuneaN: DescriereaN
default optiuneaX

to numevariabila

unde `optiunea1, ..., optiuneaN` sînt șiruri ce reprezintă opțiunile meniului care va apărea pe prima linie a ecranului, iar `Descrierea1, ..., DescriereaN` sînt explicațiile corespunzătoare opțiunilor meniului ce apar pe a doua linie a ecranului în momentul în care "bara de rulare" (scrolling-bar) este poziționată pe opțiunea corespunzătoare.

Programul prezentat în continuare a fost conceput în speranța că își va găsi o utilitate pentru cititorii noștri utilizatori ai acestui S.G.B.D., dat fiind faptul că PARADOX-ul nu oferă nici o primitivă pentru generarea, respectiv utilizarea meniurilor verticale.

Descrierea programului

Programul conține o procedură numită `Menu_Vertical()`, proiectată pentru a fi utilizată ca o procedură standard. Ea dimensionează automat mărimea meniului pe ecran ținînd cont de numărul de

opțiuni pe care le-ați specificat, opțiuni care pot fi în număr de cel mult 20. Script-ul continuă cu un mic program de aplicație care demonstrează modul în care funcționează procedura. Rutina a fost scrisă folosind comenzile PAL de poziționare a cursorului, de colorare a unor porțiuni pe ecran cu o anumită culoare, de scriere a unui text, etc. Dintre aceste comenzi amintim:

@ x,y - poziționarea cursorului

?? text - afișarea unui text la poziția curentă a cursorului

Text - EndText - afișarea unui text pe mai multe rînduri

Paintcanvas - schimbarea culorii pentru ceea ce este afișat pe ecran

style - precizează culoarea ce se va utiliza pentru afișare

Procedura `Menu_Vertical` este declarată cu șapte parametri formali ce controlează poziția și tipul chenarului precum și culoarea meniului. Această procedură utilizează comanda `Paintcanvas`, una dintre cele mai puternice comenzi PAL. `Paintcanvas` permite afișarea textului și modificarea culorii pe ecran, cu un control explicit al locației, mărimii, culorii, umplerii și chenarului zonei ce va fi desenată.

"Umbra" meniului se desenează prin simpla schimbare a culorii, astfel încît ceea ce exista înainte pe ecran nu va fi șters sau suprascris. Dar atunci cînd avem nevoie să desnăm "cutia" meniului, `Paintcanvas` ne permite să umplem aria cu spații (cod ASCII 32). Această manevră conduce efectiv la suprascrierea imaginii precedente, creînd o zonă opacă.

Opțiunea `Border` a comenzii `Paintcanvas` este folosită pentru desenarea chenarului în jurul meniului. PARADOX va utiliza caracterele din șirul opțiunii `Fill` așa cum trebuie fără a mai fi nevoie să facă deplasarea cursorului în jurul meniului pentru desenarea chenarului și "pictarea" fiecărui element al său în parte.

În prima parte a procedurii, prompt-ul (textul din partea superioară a ecranului) și opțiunile meniului sînt desenate folosind simple scrieri pe ecran. A doua parte este constituită din ciclul de procesare a tastelor apăsate. Funcția `getchar()` așteaptă un caracter (apăsarea unei taste) de la utilizator și îl memorează într-o variabilă.

Mișcarea barei de evidențiere a opțiunilor este însoțită de modificarea unei variabile locale, `case_alegere`, care conține opțiunea curentă a meniului (opțiunea scoasă în evidență în momentul respectiv). Înaintea cererii caracterului de la utilizator, se scoate în evidență opțiunea printr-o culoare specială utilizînd comanda `Paintcanvas`. După ce s-a tastat un caracter, se readuce la culoarea normală opțiunea curentă.

Tastările se procesează în mod tradițional, verificând fiecare tastă validă și ajustând bara de scoatere în evidență în mod corespunzător. Tastele valide pentru ieșire sînt `Esc` și `Enter`.

Deoarece opțiunile meniului sînt conținute într-un tablou, variabila `care.alegere` indexează tabloul; de asemenea ea conține opțiunea selectată cînd utilizatorul apasă `Enter`. În exemplul nostru, odată ce opțiunea a fost selectată cu `Enter`, se afișează un mesaj ce arată opțiunea aleasă. Într-o aplicație reală, se vor apela diferite proceduri - în funcție de opțiunea selectată din meniu.

Extinderea exemplului

Pentru cei interesați în extinderea exemplului de meniu vertical, prezentăm în continuare cîteva idei care merită să fie încercate:

- În funcție de deplasarea barei între opțiuni, să se afișeze o descriere a acestora pe a doua linie a ecranului. (Utilizați un tablou suplimentar în care se memorează descrierile, indexarea lui făcîndu-se tot cu ajutorul variabilei de selecție).
- Schimbați meniul într-o listă de opțiuni (oricît de multe) care să poată defila în fereastra de pe ecran:
 - Folosiți o tabelă pentru a memora elementele listei. Declarați un tablou cu un număr de elemente egal cu numărul articolelor din tabelă și citiți elementele în tablou la intrarea în procedură.
 - Cînd se apasă o tastă, decideți dacă mișcarea implică schimbarea poziției barei sau dacă mișcarea implică și defilarea listei de opțiuni. Implementați o procedură separată pentru fiecare tip de mișcare.
 - Urmăriți elementul din listă aflat în partea de sus a ferestrei și opțiunea curentă relativ la opțiunea din partea de sus. Folosind aceste variabile, opțiunea curentă este egală cu opțiunea de sus + opțiunea curentă - 1.

ing. Ioan Iacob

MENIU.SC

```
proc meniu_vertical (rind.sus, ; rindul coltului stnga sus
coloana.sus, ; coloana coltului stnga sus
lungime.max, ; marimea celei mai lungi optiuni
culoare.meniu, ; culoarea meniului si optiunilor
culoare.optiune, ; culoarea pentru evidentiere
tip.chenar ; afisarea chenarului cu linie dubla sau simpla
)
```

```
private cite, x, care.alegere
```

```
cite = arraysize (alegeri)
care.alegere = 1
cursor off
```

```
paintcanvas attribute 7
```

```
rind.sus + 1, coloana.sus + 2, rind.sus + cite + 2, coloana.sus +
lungime.max + 5
```

```
paintcanvas fill " "
attribute culoare.meniu
rind.sus, coloana.sus, rind.sus + cite + 1, coloana.sus +
lungime.max + 3
```

```
if tip.chenar = "s" then ; desenarea chenarului cu linie simpla
paintcanvas border
fill "" + fill ("", lungime.max + 2) + "" + fill ("", cite * 2) +
"" + fill ("", lungime.max + 2) + ""
attribute culoare.meniu
rind.sus, coloana.sus, rind.sus + cite + 1, coloana.sus +
lungime.max + 3
```

```
endif
if tip.chenar = "d" then ; desenarea chenarului cu linie dubla
paintcanvas border
fill "" + fill ("", lungime.max + 2) + "" + fill ("", cite * 2) +
"" + fill ("", lungime.max + 2) + ""
attribute culoare.meniu
rind.sus, coloana.sus, rind.sus + cite + 1, coloana.sus +
lungime.max + 3
```

```
endif
```

```
style attribute culoare.meniu
```

```
for x from 1 to cite ; ciclul pentru scrierea optiunilor meniului pe ecran
@ rind.sus + x, coloana.sus + 2 ?? alegeri[x]
endfor
```

```
while true ; Ciclul de procesare a tastelor apasate de utilizator
paintcanvas attribute culoare.optiune
rind.sus + care.alegere, coloana.sus + 2, rind.sus + care.alegere,
coloana.sus + lungime.max + 1
; Prin aceasta comanda se "picteaza" optiunea cu culoarea
de evidentiere (diferitade culoarea meniului)
retval = getch() ; se citeste tasta apasata
paintcanvas attribute culoare.meniu
rind.sus + care.alegere, coloana.sus + 2,
rind.sus + care.alegere, coloana.sus + lungime.max + 1
; Prin aceasta comanda se readuce optiunea anterioara
a culoarea meniului
```

```
switch
```

```
case retval = -72 : ; UP
if care.alegere = 1 then
care.alegere = cite
else
care.alegere = care.alegere - 1
endif
case retval = -80 : ; DOWN
if care.alegere = cite then
care.alegere = 1
else
care.alegere = care.alegere + 1
endif
case retval = -75 : return ; LEFT
case retval = -77 : return ; RIGHT
case retval = -71 : ; HOME
care.alegere = 1
case retval = -79 : ; END
care.alegere = cite
```

```

case retval = 27 : ; ESC
    ;Daca se apasa ESC programul stocheaza in variabila
    ;de sistem retval un numar egal cu numarul optiunilor
    ;meniului plus unu
    return cite + 1
case retval = 13 : ; ENTER
;Daca se apasa ENTER programul intoarce numarul optiunii
;curente din meniu
    return care.alegere
    otherwise :
        beep
endswitch
endwhile
endproc

```

;Program pentru demonstrarea modului de lucru al procedurii clear

```

clear
array alegeri[3]
alegeri[1] = "Introducere date"
alegeri[2] = "Actualizare date"
alegeri[3] = "Consultare"
while true
    menu_vertical(3,30,16,78,30,"d")
    switch
        case retval = 1 :
            message "Ati ales optiunea ",retval
        case retval = 2 :
            message "Ati ales optiunea ",retval
        case retval = 3 :
            message "Ati ales optiunea ",retval
        case retval = 4 :
            message "Ati apasat ESC" sleep 1500 quitloop
    endswitch
    sleep 1500
endwhile

```

Mai multe comenzi într-o linie DOS

Vă propunem un scurt program care permite să se dea mai multe comenzi într-o singură linie de comandă MS-DOS.

Programul se numește "E". Pentru executarea lui se tastează E urmat de comenzi. Comenzile se termină cu caracterul "#" și sînt separate prin spații. Caracterele de redirectare și "piping" se vor substitui astfel:

```

! în loc de |
{ în loc de <
} în loc de >

```

Programul a fost testat sub MS-DOS 5.0 cu Turbo C Versiunea 2.0.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include .h

```

```

void stringcat( char x[],char y[] ), stringchg( char x[],char c,char d );
char *delblank( char *x );

```

```

void stringcat( char x[],char y[] )
{
    int i,j;

    i = j = 0;
    while( x[i] != '\0' )
        i ++;
    if( i >= 255 )
        return;
    while( (x[i ++] = y[j ++]) != '\0' )
        ;
}

```

```

void stringchg( char x[],char c, char d )
{
    int i;

    i = 0;
    while( x[i] != '\0' )
    {
        if( x[i] == c )
            x[i] = d;
        i ++;
    }
}

```

```

char *delblank( char *x )
{
    char *ptr;

    if( x[0] != '' )
        return( x );
    do
    {
        ptr = strpbrk( x, " " );
        if( ptr != NULL )
            x ++;
    } while( (ptr != NULL) && (x[0] == ' ') );
    return( x );
}

```

```

void main( int argc,char *argv[] )
{
    char *ptr, *tok;
    char string[255];
    int i, numbags;

    string[0] = '\0';
    numbags = argc - 1;
    for( i = 1; i < numbags; i ++ )
    {
        stringcat( string,argv[i] );
        if( i < numbags )
            stringcat( string, " " );
    }
    stringchg( string, '!', '|' );
    stringchg( string, '{', '<' );
    stringchg( string, '}', '>' );
    ptr = strpbrk( string, "#" );
}

```

```

if( ptr != NULL )
    tok = "#";
else
    tok = ",";
ptr = strtok( string,tok );
printf( "\n" );
puts( ptr );
if( system( ptr ) != 0 )
    perror( strerror );
while( ptr != NULL )
    {
    ptr = strtok( NULL,tok );
    if( ptr == NULL )
        break;
    printf( "\n" );
    ptr = delblank( ptr );
    puts( ptr );
    i = system( ptr );
    }
}

```

Mulțime Mandelbrot

lată o nouă variantă de program de desenare a unei mulțimi Mandelbrot.

Programul QMANDEL.ASM este scris pentru PC-uri pe 32 de biți (80386 sau 80486) cu interfață grafică VGA.

Se assemblează cu TASM și se link-editează cu TLINK dându-se comanda:

```
TLINK /t qmandel.obj
```

pentru a se obține un fișier de tip .COM.

Pe un calculator cu 80386 SX la 16MHz programul a desenat un ecran complet în circa 62 secunde.

QMANDEL.ASM

```
; Program: qmandel.asm
```

```
; Funcție: desenare pe PC386 cu VGA
```

```
; Limbaj: TASM, TLINK /t
```

```

cseg SEGMENT PARA 'CODE'
    ASSUME CS:cseg, DS:cseg, SS:cseg, ES:cseg
    ORG 100h
start PROC FAR
    .386
    mov ax,0012h
    int 10h
    mov esi,xl
_13:
    mov edi,yo
_11:
    mov eax,esi
    mov ebx,edi
    mov bp,iter

```

```

_12:
    dec bp
    je SHORT nopt
    mov ecx,eax
    imul ebx
    xchg eax,ecx
    imul eax
    xchg eax,ebx
    imul eax
    mov edx,ebx
    add edx,eax
    cmp edx,max
    jg SHORT pixel
    sub ebx,eax
    mov eax,ecx
    sar eax,7
    add eax,edi
    sar ebx,8
    add ebx,esi
    xchg eax,ebx
    jmp SHORT _12

```

```

pixel:
    mov cx,si
    mov dx,WORD PTR cs:yo
    sub cx,WORD PTR cs:xl
    sub dx,di
    xor bh,bh
    mov ax,bp
    and al,0fh
    mov ah,0ch
    int 10h

```

```

nopt:
    dec edi
    cmp edi,yu
    jne _11
    inc esi
    cmp esi,xr
    jne _13
    mov ah,8
    int 21h
    mov ax,0003h
    int 10h
    mov ax,4c00h
    int 21h

```

```

start ENDP
xl DD -480
xr DD 160
yo DD 240
yu DD -240
max DD 500000
iter DW 32

```

```

cseg ENDS
END start

```

THE deal

in p.c. field

H A R D W A R E
S O F T W A R E
C A D - C A M
D E S K T O P
P U B L I S H I N G
T e c h n i c a l
s u p p o r t
T R A I N I N G
C o n s u l t i n g

雅仕 A&C
INTERNATIONAL S.A.

Tel 40-0-53.53.15.



Un nou serviciu pentru cititorii noștri:

SHARE - if

Pentru o mai bună circulație a informației între ofertanți și solicitanți de programe (și nu numai ...), vă oferim posibilitatea să:

- economisiți timpul necesar introducerii programelor mai lungi, al căror listing a apărut în "if".

- preluați pe dischetă, pentru eventuale multiplicări, textul articolele din "if" pe care ați vrea să le aveți disponibile și în această formă

- preluați acele programe și texte de articole care ne-au fost trimise spre publicare, dar pe care, din variate motive (cel mai adesea, lipsa de spațiu) - nu ajungem să le publicăm. (Desigur, aceste materiale le difuzăm numai atunci când există acordul autorului - încă nu am găsit o variantă de recompensare a autorului pentru această variantă de difuzare) O listă incompletă: criptare fișiere, completare formulare, o parolă solidă, nucleu de multitasking, "catalog" de dischete, etc.

- faceți cunoștință cu acele programe din care ni s-au trimis demo-uri, respectiv să difuzați programele dumneavoastră demonstrative prin intermedierea noastră. Acest din urmă punct sperăm să intereseze în mod deosebit pe toți autorii noștri de programe, căroră le este destul de greu să-și facă publicitatea de care au, evident, nevoie.

Cei dornici să includem demo-urile lor în oferta noastră sînt rugați să achite, odată cu expedierea dischetei sursă, suma de 500 lei în contul nostru: Cont nr. 40729-96010402, Banca Română pt. Dezvoltare Mureș.

(Cei care trimit materiale informativ/didactice, și nu publicitare, nu au de achitat nimic.) Prețul pe care îl practicăm pentru o dischetă este, din păcate, greu de definit în avans, din motive cunoscute de toată lumea. De aceea vă rugăm să vă interesați - telefonic sau în scris - în momentul lansării comenzii. În principiu, prețurile sînt identice cu prețurile cu care, la rîndul nostru, am achiziționat dischetele, la care se adaugă 50 lei pentru munca noastră și, suplimentar, taxa de expediere. Desigur, cei care ne trimit dischetele lor, au de achitat doar taxa de copiere și expediere, iar cei care trec sau trimit pe cineva pe la redacție, vor avea de achitat doar cei 50 lei, taxa de copiere.

Vă rugăm ca pe comanda pe care ne-o trimiteți să specificați pe lîngă numele pachetului și capacitatea și formatul dischetelor pe care doriți să vă furnizăm produsul.

Cod	Nume pachet	Nr. dischete	Format	Capacitate	Limba
Sisteme de operare					
29	DR - DOS 6.0	1	3,5"	1,4M	E
Proiectare circuite electronice					
41	Addi-Data: Signal Tools	1	5,25"	1,2M	E
42	MicroSim Corporation: Design Center Demo	1	5,25"	1,2M	E
43	ORCAD	1	5,25"	1,2M	E
44	ORCAD: PC Board Layout Tools	1	5,25"	360K	E
Aplicații					
12	CAPP	1	5,25"	1,2M	D
2	Buch, Trio	2	5,25"	360K	D
39	BALPRO - balanță contabilă	1	5,25"	360K	R
40	UNO - completare formulare	1	5,25"	360	R
45	Hamor Soft: hMISS - secretariat, hCONT - contabilitate, hGEN&hUTIL - dezvolt. aplicații	1	5,25"	360	R
Utilitare, biblioteci					
14	Borland Windows Tools	1	5,25"	360K	D
17	KAO BTX-Dekoder + Demo	1	5,25"	360K	D
36	QuickStep Tools	2	5,25"	1,2M	D
6	Spindrif Libraries Demo	1	5,25"	360K	E
7	BKS Demo	1	5,25"	360K	D
47	Verificator - Complex S.R.L.	1	5,25"	1,2M	R
Baze de date					
10	Paradox 3.0	1	5,25"	360K	D
46	Personal Database Demo - AvantGarde Soft.	1	5,25"	1,2M	R
31	Foxpro 2.0	2	3,5"	720K	D
18	SOS - Software Tools Datenbank	1	5,25"	360K	D
24	Advanced Revelation	1	3,5"	720K	E
25	Rbase 3.1	1	3,5"	720K	D
32	WindowBase	1	5,25"	1,2M	D
33	Address One	1	5,25"	360K	D
37	Superbase 4	2	3,5"	720K	D
Grafică, DTP, CAD					
11	Time Works Publisher	2	5,25"	360K	D
15	MEGA CAD	1	5,25"	360K	D
27	Aldus Page Maker 4.0	1	3,5"	720K	E
30	Draw Perfect 1.1 si Office 3.0	1	3,5"	1,4M	D
9	Micrografx	1	5,25"	360K	D
Cursuri					
5	DOS - Curs IBM	1	5,25"	360K	D
Procesoare de texte, OCR					
28	Word Perfect for Windows	1	3,5"	1,4M	D
35	Scout OCR	1	5,25"	360K	D
40	Ekta Editor V1.5	1	5,25"	360K	E
Măsură și control					
3	Labtech Notebook Demo	1	5,25"	1,2M	E
4	Labtech Control	1	5,25"	1,2M	E
34	Scope	1	5,25"	360K	D
Jocuri, diverse					
8	Pearl Agency	1	5,25"	360K	D

Pentru fanii Fox

În urma apariției reclamei pentru produsul Fox Pro 2.0 au sosit pe adresa redacției noastre câteva cereri pentru dischete demonstrative.

Îi rugăm pe toți cei interesați să trimită talonul respectiv pe adresa:

DARIAN ROM SUISSE SRL

3400 CLUJ-NAPOCA

str. Observatorului nr 146/15

tel: 95-123611

fax: 95-124567

Noi am transmis taloanele primite destinatarului de drept.

eurobit



**Editura EUROBIT
TIMIȘOARA**

tel. 96/116629, 96/115702

C.P. 639, Of.p. 5

Timișoara 1900

Vă oferă lucrarea:

"Programarea orientată pe obiecte în limbajul C + +"

autor: prof. univ. dr. Ioan Jurca

134 pagini format A5, preț 225 lei.

În curînd va apare

"if" nr. 5 / 1992

Important

În curînd va apare cartea

**Programarea în limbajul
Pascal**

editată de Micro ATCI Tîrgu
Mureș

Manualul este o traducere din limba germană și este destinat atît celor care vin în contact pentru prima oară cu acest limbaj de programare, cît și celor care doresc să se perfecționeze învățînd noi metode și tehnici de programare. Volumul este bogat în exemple, toate programele fiind testate cu compilatorul Turbo Pascal 6.0 al firmei BORLAND International.

Tirajul fiind limitat, asigurați-vă că nu veți pierde lucrarea expediînd o comandă fermă pe adresa:

Micro ATCI, C.P. 64, O.P. 1
4300 Tîrgu Mureș

Talon de comandă-abonament

Subsemnat(a/ui)
doresc să-mi trimiteți | | exemplare din fiecare număr al revistei **if**,
începînd cu numărul | | , pe adresa:

.....
.....
.....
Telefon: acasă la servici
.....

Pentru aceasta am expediat prin mandat poștal/virament/cec lei în contul nr. 40729-96010402 deschis la Banca Română pt. Dezvoltare Mureș. Abonamentul funcționează pînă la epuizarea sumei; în cazul că banii pe care i-am trimis nu mai ajung pentru un alt număr, voi fi anunțat de acest lucru în timp util, astfel încît nu risc să pierd nici un număr. Dacă mă răzgîndesc și renunț la abonament, mi se returnează toată suma rămasă.

if 4 / 92

Data:

Semnătura

Five minutes and you're in business



În viață, majoritatea lucrurilor sînt dificile la început, iar primele cinci minute sînt cele mai grele. De aceea, IBM a realizat un calculator cu care vă puteți înțelege din primele cinci minute.

IBM Personal System PRO

În cinci minute, "sînteți în business".

În cîteva zile îl stăpîniți. După o săptămînă, vă gîndiți deja la viitor.

The new **IBM** PS PRO 386 SX

386 SX, 20 MHz, AT BUS, 2 MB RAM, 80 MB HDD, 1,44 MB FDD, monitor VGA

color 14", mouse. Oferta standard include și pachetele de programe: DOS 5.0, Windows 3.0, Works 2.0.

Modelul PS PRO 386 SX poate fi obținut cu ușurință, la un preț foarte accesibil de la

Micro ATCI Tîrgu Mureș

IBM authorized remarketer

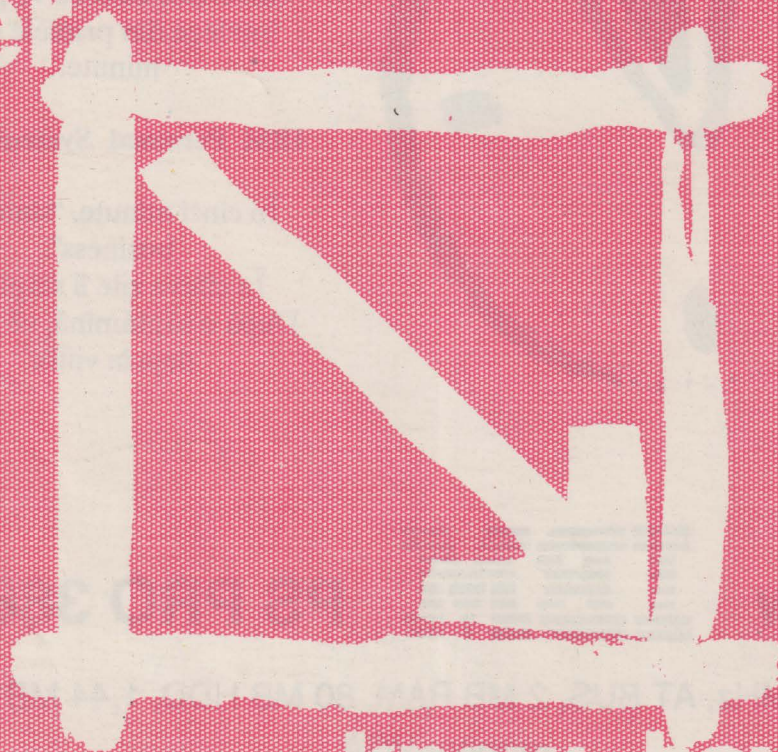
str. Gheorghe Doja nr.36

Tel / Fax: 954 - 31660

*The shortest
way between*

CAD

user



**the
know-how in
AUTOCAD**

is passing through



A & C
INTERNATIONAL S.A.