ROMEO CHELARIU

SISTEME DE OPERARE ŞI LIMBAJE DE PROGRAMARE

(ÎNDRUMAR DE LABORATOR)

ROMEO CHELARIU

SISTEME DE OPERARE ŞI LIMBAJE DE PROGRAMARE

(ÎNDRUMAR DE LABORATOR)

IAŞI-2004

CUPRINS

LUCRAREA 1

CALCULATOARE PERSONALE. ARHITECTURĂ. ECHIPAMENTE FIZICE

1. NOȚIUNI INTRODUCTIVE

Calculatorul numeric a fost conceput inițial ca instrument de lucru pentru matematicieni şi fizicieni cu scopul de a uşura efectuarea de calcule complexe. *Charles Babbage* (1792-1871), proiectantul primei maşini de calcul, a justificat necesitatea acesteia prin volumul mare de calcul necesar contruirii unor tabele matematice. Din păcate, ideile lui Charles Babbage, inclusiv cele privind calculul paralel, au trebuit să aştepte un secol, până când tehnologia electronică a permis realizarea lor practică. După anul 1945, considerat ca an de naştere a calculatoarelor electronice, evoluția calculatoarelor a fost marcată, pe de o parte, de lărgirea domeniului de aplicații, iar pe de altă parte, de o evoluție tehnologică impresionantă asociată cu definirea unei noi ştiințe, **ştiința calculatoarelor** (*computer science*).

Calculatorul, cunoscut şi sub denumirea de sistem de calcul, este perceput de marea majoritate a utilizatorilor ca fiind o ***maşină*** ce execută un ***set de instrucțiuni***, reunite într-un ***program***, cu scopul de a prelucra datele furnizate de către aceştia (***date de intrare***) pentru a obține rezultate (***date de ieşire***). Utilizarea unui sistem de calcul poate contribui în mod substanțial la rezolvarea unei anumite probleme ce apare în activitatea unei persoane. Acest lucru este posibil prin respectarea unei anumite etapizări ce trebuie parcursă pentru a rezolva problemele cu ajutorul sistemelor de calcul (Figura 1.1).

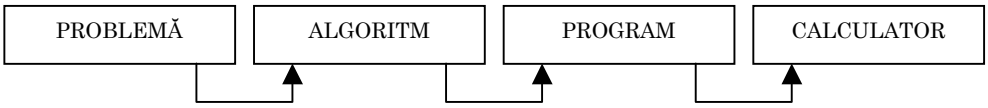


Figura 1.1. Etapele parcurse în rezolvarea unei probleme când se utilizează calculatorul.

Utilizarea calculatorului şi rezolvarea problemelor cu ajutorul acestuia necesită cunoaşterea şi definirea unor noțiuni şi termeni specifici.

Un ***calculator numeric*** (*computer*) este un sistem (ansamblu) definit prin componentele sale (echipamente fizice, programe) conectate astfel încât să formeze o entitate coerentă cu o funcție bine definită. Funcția sistemului de calcul este de a transforma informațiile transmise sistemului de către utilizator sub formă de instrucțiuni şi date de intrare, în informații necesare utilizatorului transmise acestuia sub formă de date de ieşire (Figura 1.2).

Orice calculator cuprinde două componente principale: (1) *echipamentele fizice*; (2) *programe*.

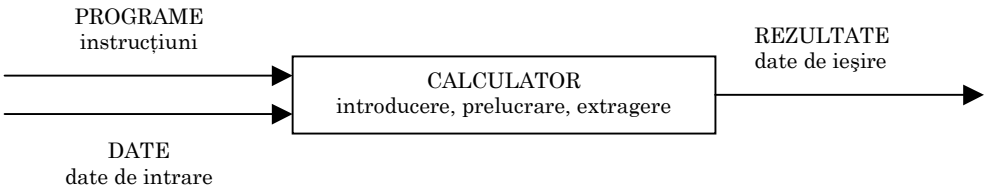


Figura 1.2. Modelul ***black-box*** (cutie neagră) al unui calculator.

Echipamentele fizice sunt reprezentate de totalitatea componentelor electronice, electrice, electromecanice care realizează împreună funcțiile sistemului de calcul, fiind cunoscute sub denumirea de ***hardware***. Acesta este nivelul fizic, perceput imediat de către utilizator.

Totalitatea programelor ce permit utilizatorului accesul la toate funcțiile sistemului de calcul sunt cunoscute sub denumirea de ***software.***

Persoanele ce utilizează un computer sunt fie ***programatori***, fie ***utilizatori.*** Prin programator se înțelege o persoană ce crează un program sau corectează unul creat anterior. Persoana ce foloseşte, pentru un anumit scop, un program creat anterior se numeşte utilizator.

2. ARHITECTURA CLASICĂ A COMPUTERELOR

Arhitectura unui computer poate fi definită ca o schemă *schemă funcțională generală* utilizată pentru realizarea constructivă a oricărui calculator.

Structura funcțională clasică a unui computer este cea ***serială***, numită şi *arhitectură von Neumann*.

Principiile funcționale fundamentale ale unui calculator von Neumann sunt:

- Calculatorul are un dispozitiv numit ***memorie*** unde sunt păstrate, atât timp cât este necesar, instrucțiunile, datele supuse prelucrării şi rezultatele obținute;
- Calculatorul posedă o ***unitate de calcul aritmetic şi logic***, care execută operațiile codificate în instrucțiunile programului;
- Calculatorul posedă o ***unitate de comandă şi control***, care determină execuția instrucțiunilor în mod secvențial, câte una la un moment dat, în ordinea stabilită prin program; de asemenea, unitatea de comandă şi control determină transferul datelor ce urmează a fi prelucrate, între memorie şi unitatea aritmetică şi logică;
- Calculatorul are în componență o ***unitate de intrare*** şi o ***unitate de ieşire***, prin intermediul cărora se realizează comunicarea cu mediul extern (utilizator, proces industrial, calculator);

Unitatea aritmetică şi logică împreună cu unitatea de comandă şi control alcătuiesc ***unitatea centrală de prelucrare*** (*CPU-Central Processing Unit*), cunoscută şi sub denumirea de ***procesor***. Procesorul şi memoria alcătuiesc *unitatea centrală*.

Structura funcțională a unui calculator von Neumann se poate reprezenta grafic ca în Figura 1.3.

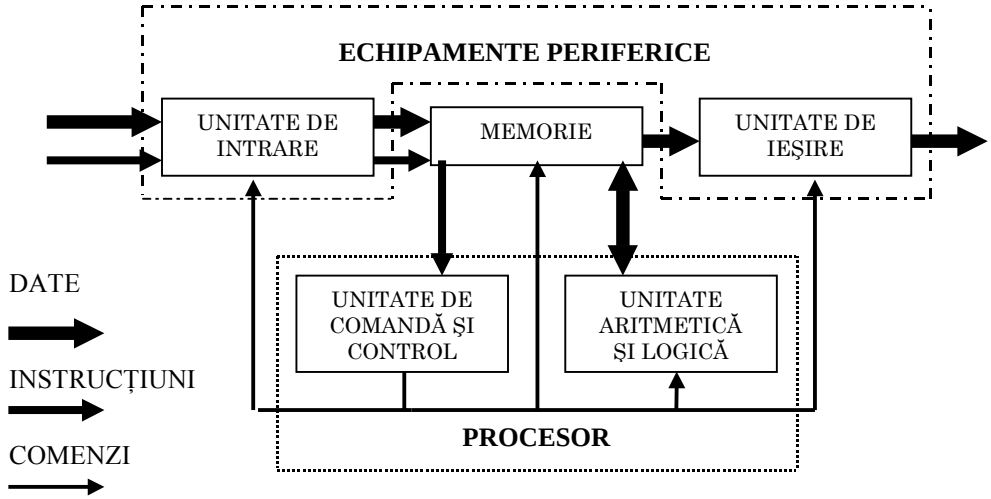


Figura 1.3. Structura funcțională a unui calculator von Neumann.

3. ECHIPAMENTE FIZICE

În interiorul calculatoarelor circulă semnale electrice transmise sub formă de impulsuri. Pentru a memora şi transmite informația (instrucțiuni, date) prin intermediul acestor semnale, calculatoarele folosesc un sistem de reprezentare a informației pe două nivele. Practic, în calculator există semnale electrice de tensiune ce are o valoare mai mare, interpretată ca având valoarea *unu* (*1*) şi semnale electrice de tensiune cu o valoare mai mică, distinctă de prima, interpretată ca având valoarea *zero* (*0*) (Figura 1.4). Valorea *0* , respectiv, valoarea *1* se numeşte ***bit*** (***bi****nary digit*), reprezentând cea mai mică unitate de măsură pentru informație.

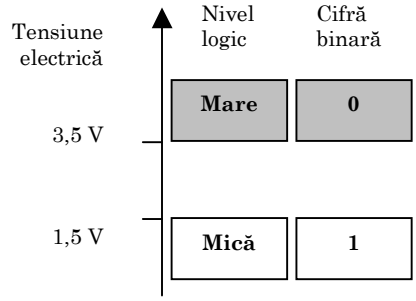


Figura 1.4. Echivalența tensiune electrică – bit.

Operațiile aplicate informației necesită gruparea biților într-o succesiune. O succesiune de *8* biți se numeşte ***byte*** (***octet***). Un octet poate reprezenta în calculator un caracter, cum ar fi *A* sau *4*.

3.1. MICROPROCESORUL

În cazul calculatoarelor personale (*personal computer-PC*) sau microcalculatoarelor, procesorul (CPU) este materializat fizic de un dispozitiv electronic numit ***microprocesor*** (*μP*).

Unitatea centrală de prelucrare (CPU) îndeplineşte două funcţii esenţiale:

- Transferul şi prelucrarea datelor;
- Controlul activităţii întregului sistem de calcul (PC).

O unitate centrală de prelucrare a informaţiei, având funcţiile enunţate anterior, care coordonează un sistem de calcul structurat ca în Figura 1.3 şi care, fizic, se prezintă sub forma unui *cip* se numeşte microprocesor.

Există mai multe clasificări ale noţiunii de microprocesor. După tipul de sarcini ce pot fi realizate eficient, se disting:

- Microprocesoare de uz general, nespecializate;
- Microprocesoare specializate, ca de exemplu procesoare de intrare/ieşire, utilizate la transferul datelor dintre calculator şi mediul extern, coprocesoare matematice, specializate în operaţii aritmetice de utilitate generală, coprocesoare grafice, destinate îmbunătăţirii vitezei şi calităţii afişării imaginilor etc;

În continuare, se vor face referiri doar asupra microprocesoarelor standard, de uz general. Caracteristicile microprocesorului îşi pun amprenta asupra performanţelor şi întregii organizări ale calculatorului din componenţa căruia face parte.

Microprocesorul este un cip format dintr-un circuit integrat complex, care conţine un număr de tranzistoare echivalente de ordinul milioanelor, ce prelucrează date prin executarea, în mod secvenţial, a unor operaţii logice şi/sau aritmetice diverse în conformitate cu instrucţiunile furnizate de utilizator. Conectarea microprocesorului la celelalte componente fizice ale calculatorului se face prin intermediul unor picioruşe numite *pini*. Numărul acestora depinde de tipul microprocesorului. În interiorul CPU am putea regăsi subunităţi precum: *magistrale interne* (conductori pentru transmiterea semnalelor electrice), *registre* (memorii de capacitate mică folosite pentru memorarea unor stări sau valori intermediare), *unitatea aritmetică şi logică* (*ALU-Arithmetic and Logic Unit*), *unitate de calcul aritmetic cu numere reale*, unităţi specializate (*buffer* de instrucţiuni etc). Indiferent de tipul de microprocesor există două unităţi comune şi anume:

- *Unitatea de execuţie*, care realizează în mod efectiv operaţiile folosind zonele de memorie incluse microprocesorului, cunoscute sub denumirea de *registre*; scopul registrelor este de a memora date, adrese de memorie,

adresa următoarei instrucțiuni ce trebuie executată, precum şi indicatori de stare care arată cum s-au terminat instrucțiunile anterior executate;

- *Unitatea de interfață cu magistrala externă*, care realizează transferul datelor şi instrucțiunilor de la şi spre microprocesor.

Diferențierea microprocesoarelor , din punctul de vedere al performanțelor, se face în funcție de *cantitatea de memorie ce poate fi adresată*, *setul de instrucțiuni executabile* (numărul şi tipul), *viteza de lucru.* Este de reținut faptul că un microprocesor nu poate adresa o cantitate infinită de memorie, ci numai cantitatea maximă impusă de procesul său constructiv. Valoarea maximă a memoriei adresabile este importantă deoarece un program nu poate fi executat decât dacă el se află în memorie, astfel încât, o memorie adresabilă de dimensiune mai mare facilitează o execuție mai rapidă a programului. Cu cât setul de instrucțiuni specific unui microprocesor este mai mare cu atât se pot rezolva probleme din ce în ce mai complexe. Viteza de lucru a unui microprocesor este determinată, în principal, de următorii parametri: *frecvența ceasului intern*, *mărimea şi numărul registrelor interni*, *mărimea magistralei de date.* Ceasul intern este un oscilator ce trimite pulsuri la intervale egale de timp, bine determinate. Executarea unei instrucțiuni de către microprocesor se face într-un număr de etape realizate în intervalul delimitat de două semnale succesive ale ceasului intern. Frecvența cu care se generează aceste semnale se numeşte frecvența ceasului intern. Prin creşterea numărului registrelor şi a capacității de memorare a acestora se micşorează numărul de operații de transfer de date cu memoria internă, îmbunătățind astfel viteza de lucru a microprocesorului. Volumul de date ce este transferat la un singur puls al ceasului intern este mai mare cu cât mărimea magistralei de date este mai mare, prin aceasta micşorându-se numărul de operații de transfer de date cu memoria.

În continuare sunt prezentate câteva din reperele importante ale evoluției microprocesoarelor:

- Microprocesorul Intel (80)286 face trecerea la magistrala de 16 biți şi aduce primele mecanisme de multitasking (utilizarea partajată a resurselor sistemului de calcul între mai multe programe); setul de instrucțiuni este extins prin utilizarea unui coprocesor matematic;
- Microprocesorul Intel (80)386 impune magistrala pe 32 biți şi include mecanisme de gestiune evoluată a memoriei;
- Microprocesorul Intel 486 utilizează noi tehnici de multitasking şi include un nivel suplimentar de memorie *cache* (memorie intermediară de performanță ridicată între microprocesor şi memoria internă); coprocesorul matematic este inclus în acelaşi cip;
- Microprocesorul Intel Pentium (586) are o magistrală internă de 64 biți şi capacități sporite de multitasking;

- ❖ Microprocesorul Intel Pentium II/III (686) are un set lărgit de instrucțiuni; instrucțiunile suplimentare sunt dedicate prelucrării multimedia şi sunt codificate *MMX* şi *SSE*;
- ❖ Microprocesorul Intel Pentium IV (786) are noi instrucțiuni dedicate datelor multimedia (*SSE2*) ; organizarea instrucțiunilor înaintea execuției.

Principalele tipuri de microprocesoare sunt prezentate în Tabelul 1.1, iar caracteristicile de bază ale ultimilor tipuri de procesoare în Tabelul 1.2.

Tabelul 1.1

Principalele tipuri de microprocesoare

Generația	Intel		AMD		Cyrix	Frecvența (MHz)
(80)-86	I8086	I8088 I80186	-		-	4
(80)286	I80286		Am286		-	8÷16
(80)286	I386	I386SX I386DX	Am386		-	20÷40
486	I486	I486SX I486DX I486DX-2 I486DX-4	AMD486	486DX 486DX-2 486DX-4 5x86(DX5)	Cyrix486	20÷120
586	Pentium	Pentium (I) Pentium Pro	AMD K5		Cyrix5x86	75÷166
686	Pentium II/III	Klamath Katmai Celeron	AMD K6	K6 K6-2	Cyrix6x86 IBM 6x86	166÷450 450÷650
786	Pentium III Pentium IV	Celeron Coopermine Tualatin Willamette Northwood P4EE Precott	AMD K7	Duron Athlon	VIA Cyrix III	650÷1200 1200÷3200

Tabelul 1.2

Carcateristici de bază ale microprocesoarelor actuale

Caracteristici	Willamette	Northwood	Pentium 4 Extreme Edition (P4EE)	Prescott	Athlon 64 (FX51, 3400+)
Proces de fabricație	180 nm	130 nm	130 nm	90 nm	130 nm
Număr de tranzistori (milioane)	42	55	178	125	106
Dimensiune pastilă (mm^2)	170	131	237	112	193
Cache L1 (KB)	8	8	8	16	128
Cache L2 (KB)	256	512	512	1024	1024
Cache L3 (KB)	NA	NA	2048	NA	NA
Frecvența maximă (GHz)	2	3,4	3,4	3,4	2,2

Pentru ultimile generații de microprocesoare, pentru a mări viteza de lucru, memoria cache de nivel 2 (L2) este inclusă pe suportul acestora şi nu pe placa de bază având capacități de până la 1024 KB. Aceste ultime tipuri de procesoare Intel au posibilitatea reală de a executa simultan mai multe activități, facilitate pe care Intel a denumit-o *Hyper Threating*. Microprocesorul *Prescott* a primit şi un număr de intrucțiuni *SSE* suplimentare, codificate *SSE3*.

3.2. MEMORIA INTERNĂ

Memoria unui calculator se numeşte *memorie internă* deoarece în componența sa mai există şi alte dispozitive de memorare, dar cu alte caracteristici şi funcții, care alcătuiesc împreună aşa-numita *memorie externă*.

În memoria unui calculator sunt stocate (memorate) *obiecte*, adică informație codificată într-un anumit mod, folosind sistemul binar. Modelul de reprezentare a informației, în formă scrisă, reprezintă o ***dată***. Deosebirea dintre informație şi dată este echivalentă cu cea dintre un obiect şi modelul său. Informația stocată în memorie poartă denumirea generică de *date*. În memorie sunt stocate atât instrucțiunile programelor ce se execută, cât şi datele pe care acestea le prelucrează, folosind acelaşi mod de codificare. În acest mod prin denumirea de date vom înțelege atât datele ce se prelucrează (date de intrare, intermediare, finale, de ieşire etc), cât şi instrucțiunile ce se execută.

Din punct de vedere fizic, memoria internă este formată tot din cipuri ce au proprietatea de a memora secvențe de biți. În cazul memoriei, un bit este materializat printr-un ***element fizic cu două stări distincte*** ce pot fi codificate cu:

- ***Fals*** sau ***0***;
- ***Adevărat*** sau ***1***.

Un astfel de element poate fi un obiect magnetizabil cu două stări magnetice *N-S* sau *S-N* fără a-şi schimba poziția (Figura 1.5).

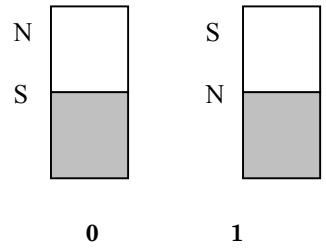


Figura 1.5. Bitul fizic al memoriei interne

Având în vedere cele prezentate, memoria internă poate fi considerată ca fiind formată dintr-o mulțime finită de elemente, cu două stări distincte, aşezate succesiv.

Unitățile de măsură a memoriei sunt:

Bitul (***0***, ***1***), notat ***b***;

Byte (***octet***), format din *8* biți succesivi, notat ***B***;

Kilobyte, notat ***KB***, 1 KB = 2^{10} B = 1024 B;

Megabyte, notat ***MB***, 1 MB = 2^{10} KB = 1 048 576 B;

Gigabyte,notat ***GB***, 1 GB =2^{10} MB = 2^{20} KB = 2^{30} B.

Deoarece reprezentarea informației în calculator se face folosind sistemul binar (baza 2), multiplii byte-ului vor fi puteri ale lui 2, după cum se poate observa mai sus. Byte-ul şi multiplii săi sunt unități de măsură a capacității de memorare atât pentru memoria internă, cât şi pentru alte dispozitive de stocare a informației.

Memoria internă este formată din mai multe părți de dimensiune egală, fiecare fiind numită ***locație de memorie***. Acestea sunt identificabile printr-o ***adresă de memorie***. Adresele de memorie sunt numere naturale ce arată poziția în memorie a fiecărei locații (0, 1, 2, …). Dacă se grupează *n* elemente binare de memorie, se obține o locație de memorie ce poate avea 2^n stări distincte. De exemplu, dacă $n = 4$, numărul de stări diferite este $2^4 = 16$, şi anume: *0000, 0001, 0010, 0100, 1000, 0011, 0110, 1100, 1010, 0101, 1001, 0111, 1011, 1101, 1110*. Aceasta înseamnă că o *dată* ce trebuie memorată poate avea doar una din valorile de *4* biți anterioare. În cazul calculatoarelor, locațiile de memorie sunt de *8* biți consecutivi, formând astfel un *byte*.

Locațiile de memorie se grupează în ***celule de memorie***. Identificarea unei celule de memorie se face prin ***adresa*** şi ***lungimea*** ei. *Adresa* celulei de memorie este dată de *adresa primului octet*, iar *lungimea* de numărul de elemente binare grupate într-o celulă de memorie.

Cuvântul (*word*) este o unitate logică de informație, formată de obicei din 16 biți (2 octeți). În cazul în care este format din 4 octeți se numeşte ***cuvânt dublu***. Lungimea cuvântului depinde de tipul constructiv al calculatorului.

Memoria internă este caracterizată de doi parametri: (1) ***capacitatea de memorare***, (2) ***timpul mediu de acces*** sau ***frecvența de lucru***.

Capacitatea memoriei interne, caracteristică importantă pentru viteza şi eficiența cu care va lucra calculatorul, depinde de tipul microprocesorului şi se măsoară în KB sau MB. Practic, memoria internă este formată din plachete de memorie cu capacități de 4, 8, 16, 32, 64, 128, 256, 512 MB. *Timpul mediu de acces* se măsoară în *ns* şi se referă la intervalul de timp care este necesar memoriei interne pentru a fi citită sau scrisă datele, iar *frecvența* în *MHz*.

Memoria internă este formată din două tipuri de cipuri de memorie cu caracteristici diferite şi anume:

- ***Memorie RAM*** (***R****andom* ***A****ccess* ***M****emory* – memorie cu acces aleatoriu), care are următoarele caracteristici: poate fi *citită* sau *scrisă*, este *volatilă*, adică datele memorate se păstrează atât timp cât calculatorul este conectat la rețeaua electrică, iar fiecare locație de memorie poate fi accesată imediat;
- ***Memorie ROM*** (***R****ead* ***O****nly* ***M****emory* – memorie doar citită), care are următoarele caracteristici: poate fi numai *citită*, *nu poate fi scrisă* sau modificată, este *nevolatilă* (*permanentă*).

Memoria RAM este folosită pentru a memora date sau instrucțiuni ale programelor care se execută la cererea utilizatorului, iar memoria ROM conține

date necesare la pornirea şi în timpul funcționării calculatorului.

3.3. MAGISTRALE (BUS-URI)

Deşi denumirea lor este pretențioasă, aceste *magistrale* nu sunt altceva, din punct de vedere fizic, decât un număr de conductori electrici, prin intermediul cărora se realizează transmiterea semnalelor electrice ce realizează transferul de date şi comenzi între componentele principale ale calculatorului.

Magistrala de date, bidirecțională, permite circulația datelor (operanzi/ rezultate), a instrucțiunilor şi chiar a adreselor.

Magistrala de adrese, unidirecțională, permite microprocesorului să localizeze datele din memorie sau dispozitivele de intrare/ieşire, aşa încât pe această magistrală circulă numai adrese.

Magistrala de control, permite circulația, bidirecțională, a semnalelor de comandă şi control de la/către microprocesor.

3.4. INTERFEȚE

Pentru a putea face legătura între unitatea centrală şi echipamentele periferice, se folosesc dispozitive numite ***interfețe***. Interfața unui echipament periferic primeşte/transmite datele de la/către unitatea centrală, după care le transmite/primeşte la/de la echipamentul periferic. Interfața realizează conversia datelor din codul specific calculatorul într-o formă accesibilă realizării comunicării calculator-mediu extern (utilizator, proces, calculator),. Comunicarea dintre interfață şi echipamentul periferic se poate face în două moduri şi anume: (1) *secvențial*, (2) *paralel*.

Unele interfețe au denumiri specifice: adaptor video, controler de disc, adaptor pentru fax/modem, placă de sunet, placă de rețea etc.

3.5. ECHIPAMENTE PERIFERICE

Echipamentele fizice ce intră în componența unui calculator personal şi care materializează fizic unitățile de intrare/ieşire se numesc ***echipamente periferice***.

3.5.1. Memoria externă

Deoarece memoria internă a unui calculator este volatilă şi de capacitate relativ redusă, posibilitățile de păstrare a datelor, atât în timpul funcționării, cât mai ales după oprirea acestuia, sunt mult reduse. Astfel, a apărut necesitatea adăugării unor echipamente de memorare diferite de memoria internă. Caracteristicile memoriei externe sunt: este *nevolatilă*, poate fi atât *citită* cât şi *scrisă*. O astfel de memorie,

nevolatilă, cu rol de stocare timp îndelungat a datelor, este cunoscută sub denumirea de ***memorie externă***. Memoria externă a unui calculator este formată din discurile magnetice.

De menționat că trebuie făcută distincție între unitatea de memorie externă şi suportul fizic de memorare.

Unitățile de discuri magnetice sunt nelipsite din componența oricărui sistem de calcul, fiind utilizate şi ca extensie a memoriei interne. Memorarea datelor se face pe suprafețele magnetice ale unui disc sau pachet de discuri. Un disc magnetic este format dintr-un suport, ce poate fi flexibil (material plastic)sau rigid (aluminiu sau aliaje ale acestuia), pe a cărui suprafețe se depune un film subțire de substanță magnetică, care formează partea activă a acestuia. În cazul pachetului de discuri, acestea se montează paralel şi echidistant pe un ax comun.

Pentru transferul datelor pe/de pe un astfel de disc sau pachet de discuri, acestea trebuie introduse într-un dispozitiv numit ***unitate (drive) de disc***. Unitatea de disc este prevăzută cu un dispozitiv special, numit *dispozitiv de acces*, format dintr-un număr de *capete de citire/scriere* egal cu numărul de suprafețe active ale suportului de memorare. Capetele de citire/scriere au, în funcție de tipul echipamentului periferic, un anumit număr de poziții distincte față de axul pachetului. Fiind montate rigid pe acelaşi dispozitiv de antrenare, capetele de citire/scriere se deplasează simultan, găsindu-se la aceeaşi distanță față de axul suportului magnetic. Fiecare cap de citire/scriere, pe suprafața activă asociată, descrie o circumferință numită *pistă*, ce are drept centru, centrul axului pachetului de discuri (Figura 1.6a). Totalitatea pistelor de acelaşi diametru, în cazul pachetelor de discuri, formează un *cilindru* (Figura 1.6b). O pistă la rândul ei este divizată în mai multe zone distincte, numite *sectoare*, cu o anumită mărime. Fiecare sector are aceeaşi mărime, care de obicei este 512 B. Unitatea de adresare a discului magnetic este sectorul de disc, care are asociată o valoare numită *adresă*.

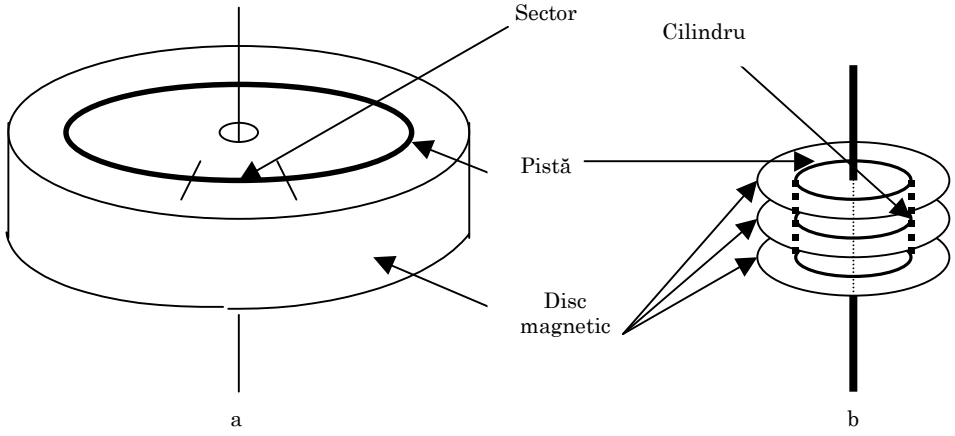


Figura 1.6. Împărțirea fizică a discurilor magnetice.

Caracteristicile de bază ale unităților de discuri magnetice sunt, în principal, următoarele:

- Caracteristici fizice ale discurilor (rigid, flexibil, număr de discuri, diametrul etc);
- Capacitatea de memorare (MB, GB);
- Viteza de transfer (KB/s);
- Timp de acces (ms);
- Norma de conectare, care poate fi PIO Mode 1÷4, ATA, UltraDMA 33, 66, 100 etc;
- Densitatea de înregistrare.

Unitățile cu *discuri fixe* (montate permanent în unitatea de disc) şi *suport rigid* sunt cunoscute sub denumirea de ***hard-disk***.

Hard-disk-urile moderne au o serie de îmbunătățiri constructive care permit creşterea performanțelor, dintre care amintim:

- Includerea unei memorii *cache* (*buffer*) cu capacități între 256 KB÷8 MB;
- Mecanisme de detectare şi corecție a erorilor;
- Viteze mai mari de rotație a discurilor (5400, 7200, 15000 rot/min);
- Suspensii pentru protejarea împotriva şocurilor mecanice etc.

Unitățile cu *disc amovibil* (poate fi montat sau extras din unitatea de disc ori de câte ori este necesar) şi *suport flexibil* sunt cunoscute sub denumirea de ***floppy-disk***. Discul propriu-zis este cunoscut şi sub denumirea de *dischetă*. Dischetele sunt alcătuite dintr-un singur disc magnetic, cu suport flexibil, protejat de o carcasă de material plastic, în interiorul căreia se mişcă liber. Deşi au existat mai multe variante constructive, astăzi se utilizează doar unitățile floppy-disk pentru dischete cu diametrul de 3,5 " (inches) cu o capacitate de memorare de 1,44 MB.

3.5.2. Tastatura

Tastatura este principalul dispozitiv periferic al unui calculator prin intermediul căruia se transmit informații (comenzi, date) către unitatea centrală. De la tastatură datele circulă unidirecțional către unitatea centrală, fiind un periferic de intrare. Cuplarea tastaurii la calculator se face prin intermediul unui cablu de conectare şi a unui conector special ce intră în legătură cu interfața corespunzătoare şi magistralele sistemului.

Versiunile clasice ale tastaturilor calculatoarelor personale au un număr de 101/102 taste. Din punctul de vedere al funcționalității, o tastatură standard cu 101 taste cuprinde următoarele grupe:

- *Taste caracter*, ce cuprinde tastele corespunzătoare literelor alfabetului latin, mari şi mici, cifrele arabe şi caracterele speciale (!,@,",#,$; %,^,&,*,(,),_, -, =,+,/,\, | ,;,:,',~,`,);
- *Taste numerice*, care cuprind taste numerice propriu-zise, cu două caractere înscrise şi taste cu operatorii aritmetici;
- *Taste de alegere a modului de lucru* pentru tastele caracter şi tastele numerice, **Caps Lock**, care prin activare determină obținerea literelor mari, **Num Lock**, care prin activare determină obținerea tastelor numerice, iar prin dezactivare tastele corespunzătoare celui de-al doilea caracter înscris; tasta **Insert** (**Ins**-în zona tastelor numerice), schimbă modul de lucru al tastelor caracter din inserare în suprapunere şi invers;
- *Taste de modificare a funcției unor taste sau grupe de taste*, **Shift** (⇧), care prin menținerea în stare apăsată permite tastarea caracterului superior de pe tastele cu două caractere sau inversează acțiunea tastei Caps Lock; tastele **Ctrl** şi **Alt**, separat sau împreună, sunt utilizate în combinații cu alte taste pentru generarea de comenzi, direct de la tastatură; tipul de comenzi generate şi combinațiile de taste depind de programul ce se execută în momentul respectiv pe calculator;
- *Taste de deplasare/poziționare*, prin apăsare determină deplasarea cursorului în direcția indicată de fiecare tastă; *cursorul* este o reprezentare grafică distinctă a poziției curente în cadrul unui document şi care focalizează atenția utilizatoruluiasupra poziției în care au loc diverse operații; tastele de deplasare sunt tastele *săgeți* (← ↑ ↓ →), **Page Up** (determină deplasarea cursorului cu un număr mai mare de rânduri spre început), **Page Down** (determină deplasarea cursorului cu un număr mai mare de rânduri spre şfârşit), iar tastele de poziționare sunt **Home** (determină poziționarea cursorului la început de rând) şi **End** (determină poziționarea cursorului la sfărşit de rând); prin apăsarea simultană a tastei Ctrl şi a tastei Page Up, respectiv Page Down, cursorul se deplasează la început, respectiv sfârşit de pagină; prin apăsarea simultană a tastei Ctrl şi a tastei Home, respectiv End, cursorul se deplasează la început, respectiv sfârşit de document;
- *Taste funcționale*, care determină lansarea imediată a unei comenzi predefinite, în funcție de programul ce se execută de către calculator; aceste taste sunt codificate **F1, F2, …, F12**;
- *Taste cu acțiune bine definită*, pentru care gradul de standardizare a acțiunii lor este foarte ridicat; aceste taste şi acțiunile lor specifice sunt:
  - **Enter** - execuție comandă sau trecere la un rând nou; când sunt active tastele numerice îndeplineşte funcția de **=** ; se află în zona tastelor numerice;

- **RETURN** ()- execuție comandă sau trecere la un rând nou;
- **Esc** – anulează comanda, respectiv programul în curs de execuție;
- **Back Space** ()- şterge un caracter în stânga cursorului;
- **Delete** (**Del**-în zona tastelor numerice)- şterge un caracter de pe poziția curentă a cursorului;
- **Tab** ()- determină trecerea la rubrica următoarea (pentru anumite programe) sau afişează un spațiu liber format de mai multe caractere blanc (spațiu liber);
- **Print Screen**-determină tipărirea (copierea) ecranului curent;
- **Scroll Lock**-opreşte defilarea pe ecran a datelor determinate de execuția unei comenzi;
- **Pause/Break**-determină oprirea (temporară sau definitivă) a anumitor procese (programe).

Apăsarea unei taste (combinații de taste) determină transmiterea către unitatea centrală a unor date, care pot fi de tip text, funcții sau comenzi. Lucrul cu tastele presupune trei regimuri: (1) apăsarea scurtă a tastei (modul cel mai frecvent de lucru); (2) apăsarea continuă a unei taste, pentru menținerea acțiunii acesteia; (3) apăsarea simultană a două sau trei taste pentru obținerea de combinații de taste ce determină acțiuni specifice.

Tastaturile recente sunt fabricate în diverse variante numite fie *tastaturi ergonomice* (asigurarea de poziții relaxante pentru mâini), fie *tastaturi multimedia* ce au integrate taste specializate în funcții multimedia.

3.5.3. Sistemul video

Sistemul video este format din două componente:

- ***Monitorul*** (*display*), dispozitiv electronic ce permite vizualizarea datelor introduse de la tastatură sau rezultate în urma execuției unor comenzi sau programe;
- ***Placa video*** (*adaptor video*), care realizează legătura între unitatea centrală şi monitor, permițând conversia datelor ce sunt afişate pe ecranul monitorului în codul binar, specific calculatorului.

Modurile de operare ale sistemului video sunt:

- Modul video *grafic*, când suprafața de afişare este organizată ca o matrice cu *m* linii şi *n* linii, fiecare element putând fi reprezentat ca un punct adresabil în mod unic, caracterizat prin *culoare*, *intensitate luminoasă, dimensiune*; un astfel de element grafic se numeşte *pixel*; în prezent, acesta este modul cel mai frecvent de afişare a datelor indiferent de tipul acestora;

- Modul video *text*, când suprafața afişabilă este formată din celule caracter delimitate de 25 linii şi 80 coloane, controlul datelor afişate pe ecran făcându-se la nivel de caracter alfanumeric; în acest caz, pixelii nu mai sunt adresabili individual, în fiecare celulă putând fi afişat un caracter alfanumeric.

Placa video preia datele ce trebuie afişate pe ecranul monitorului (texte, imagini) şi le adaptează pentru a fi *înțelese* de către monitor. Astfel sunt comandați individual (poziție, culoare, intensitate) pixelii care formează imaginea afişată pe ecran.

Placa video este caracterizată prin:

- *Rezoluție*, care reprezintă numărul maxim de elemente ce pot fi afişate pe ecranul monitorului; această caracteristică este similară cu cea a monitorului;
- *Numărul maxim de culori*;
- *Caracteristicile memoriei proprii, etc*;

Plăcile video au în componență pe lângă memoria proprie, un microprocesor propriu (*chipset*), convertorul digital-analogic al datelor (RAMDAC), magistrală internă (32, 64, 128 biți). Semnalul de ieşire al plăcilor video este, de obicei, analogic, cu excepția celor destinate pentru a conecta monitoare speciale, când ieşirea poate fi de tip digital. Cei mai cunoscuți producători de *chipset-uri grafice* sunt firmele ATI şi NVIDIA.

Cele mai importante caracteristici ale monitoarelor sunt:

- *Mărimea imaginii*, exprimată prin *dimensiunea diagonalei* ecranului măsurată în *inch*; mărimile tipice sunt de 14", 15", 17", 19", 21";
- *Numărul de culori* ce pot fi afişate, exprimat prin puteri ale lui 2 (2^4, 2^8, 2^{16}, 2^{24}, 2^{32});
- *Rezoluția de afişare a ecranului*, definită atât pe verticală cât şi pe orizontală; cele două rezoluții sunt diferite; conform standardului *VGA* (*Video Graphics Adapter*), considerat împreună cu extensiile *SVGA* (*Super VGA*), matricea de afişare a modului video grafic are rezoluțiile de 640x480, 800x600, 1024x768
- *Definiția*, care reprezintă dimensiunea unui pixel în mm; valorile cele mai utilizate sunt cuprinse în intervalul *0,24÷0,28* mm.

Din punctul de vedere al reglajelor referitoare la luminozitate, contrast, dimensiuni, poziționare, geometrie etc, în timp s-au manifestat două tipuri de monitoare: cu *reglaj analogic*, când reglajele se făceau individual prin potențiometre, şi cu *reglaj digital*, când reglajele se fac prin intermediul unui meniu (imagini grafice) ce se afişează pe ecran.

Din punct de vedere constructiv, cele mai răspândite sunt monitoarele ce folosesc pentru formarea imaginii un tub catodic (*CRT*). Un rol important la acest tip de monitoare îl are şi *rata de împrospătare a imaginii* (*refresh rate*) numită şi *frecvența verticală*, datorită impactului asupra calității imaginii şi sănătății utilizatorului. Conform normelor actuale (protecția muncii, ergonomie), imaginea trebuie să se refacă de 75 ori/secundă. În ultimii ani au apărut şi monitoare ce folosesc alte metode de formare a imaginii, cum ar fi: *LCD* (*Liquid Crystal Display*), *PDP* (*Plasma Display Panel*), *OELD* (*Organic ElectroLuminiscence Display*). Aceste noi tipuri de monitoare tind să înlocuiască treptat monitoarele cu afişare CRT.

Numărul de culori ce se pot afişa se mai numeşte şi *adâncime de culoare* şi se exprimă, de obicei, prin numărul de biți utilizați. Astfel, cu 16 biți se pot reprezenta maxim 2^{16} = 65536 culori (*High Color*), iar cu 32 biți s-ar putea afişa 2^{32} = 4 294 967 296 culori (*True Color*).

Deşi monitorul este perifericul de ieşire tradițional, fiind deci unidirecțional, există şi monitoare bidirecționale (intrare/ieşire), tip *touch-screen*, care permit şi introducerea de comenzi printr-un sistem de senzori de suprafață.

Producerea de monitoare este reglementată printr-o serie de norme, cum ar fi:

- Norme de siguranță-TŰV/GS, CE, Nemko, Fimko etc;
- Norme ergonomice-ISO 9241-3, ISO 9241-8;
- Norme pentru radiații-MPRII, TCO'95, TCO'99 etc.

Fiind constituit din monitor şi placă video, parametrii de funcționare ai sistemului video (rezoluție, număr de culori, rată de împrospătare) sunt determinați de componenta cu valorile mai mici, aşa încât performanțele sale sunt date de componenta mai puțin performantă.

3.5.4. Imprimanta

Imprimanta este un echipament periferic de ieşire, unidirecțional, ce poate fi conectat la unitatea centrală şi care are rolul de a tipări rezultatele obținute cu ajutorul calculatorului într-o problemă oarecare.

Până în prezent s-au realizat şi utilizat tipuri diferite de imprimante pentru calculatoare personale, ce se deosebesc prin performanțe şi metoda de tipărire.

Principalele caracteristici ale unei imprimante, ce caracterizează performanțele acesteia, sunt:

- *Rezoluția*-reprezintă numărul de puncte pe care le poate afişa imprimanta într-un inch; rezoluția pe verticală nu este obligatoriu egală cu cea pe orizontală; cu cât rezoluția este mai mare, cu atât calitatea imaginii imprimate va fi mai bună;

- *Viteza de imprimare*, a cărei unitate de măsură depinde de metoda de imprimare folosită;
- *Dimensiunile maxime ale hârtiei* pe care fiecare imprimantă poate tipări; dimensiunile hârtiei sunt cele specifice formatelor standardizate (A3, A4, A5, B5 etc).

După metoda de imprimare se deosebesc următoarele tipuri principale de imprimante:

- *Imprimante matriceale*, la care imprimarea se realizează prin intermediul unui *cap de imprimare*, prevăzut cu ace acționate electromagnetic ce percutează o bandă tuşată pe suprafața hârtiei, care se deplasează orizontal de-a lungul hârtiei; calitatea imprimării este determinată de numărul de ace, care poate fi *9*, *18* sau *24*; rezoluția acestor imprimante este scăzută, de circa*180 dpi* (***d****ots* ***p****er* ***i****nch*), ceea ce permite tipărirea de texte sau imagini fără pretenții calitative deosebite; viteza de imprimare se măsoară prin numărul de caractere imprimate pe secundă (*cps*) şi are valori cuprinse în intervalul 100÷400 cps sau chiar mai mult în cazul imprimantelor matriceale rapide; acest tip de imprimante lucrează obişnuit în mod text;
- *Imprimante cu jet de cerneală*, la care imprimarea se face prin pulverizarea fină pe hârtie a unor picături de cerneală, folosind duze microscopice controlate riguros spațial şi temporal, duze ce intră în componența unui *cap de imprimare* care poate fi deplasat orizontal; pentru aceste imprimante există posibilitatea realizării imaginilor color; viteza de imprimare se măsoară în *pages per minute* (*ppm*);
- *Imprimante laser*, care foloseşte o rază laser sau mici diode luminiscente pentru a încărca electrostatic un cilindru de imprimare, corespunzător caracterului ce urmează a fi imprimat; cilindrul vine în contact cu o pulbere specială, numită *toner*, care aderă pe suprafețele încărcate electrostatic, după care trece peste suprafața hârtiei de imprimat unde este ars tonerul; viteza de imprimare se măsoară în *pages per minute* (*ppm*).

În ceea ce priveşte utilizarea unui anumit tip de imprimantă în funcție de tipul şi calitatea documentului ce trebuie imprimat, se pot face următoarele recomandări, prezentate în Tabelul 1.3.

Pentru a îmbunătăți performanțele imprimantelor cu jet de cerneală, sau a celor cu laser , acestea sunt dotate cu memorie RAM proprie, putând astfel prelua corespunzător sau chiar integral datele. Imprimatele evoluate şi destinate lucrului în medii complexe, unde se prevăd 10 000÷300 000 pg/lună sunt prevăzute, pe lângă memorie internă proprie, cu microprocesoare, respectiv hard-disk-uri.

Principalele caracteristici de performanță ale imprimantelor uzuale sunt indicate în Tabelul 1.4.

Tabelul 1.3

Recomandări de utilizare a imprimantelor în funcție de tipul şi calitatea documentului

Imprimantă matricelaă	• Prețul relativ mic al consumabilelor şi posibilitatea de a folosi copierea prin indigo sau hârtie autocopiativă o recomandă pentru a fi utilizată pentru tipărirea/completarea de formulare tipizate, cu largă aplicabilitate în domenii precum contabilitatea; • Constituie prima alternativă când se tipăresc tiraje de mărime medie-mare pe format A3 transversal, frecvent întâlnit în contabilitate.
Imprimantă cu jet de cerneală	• Pentru că oferă o calitate bună a documentului imprimat, este folosită pentru documente uzuale, ce conțin atât text cât şi imagini grafice; • Este prima alegere în tipărirea documentelor color; • Consumabilele având un preț relativ ridicat în raport cu volumul de date ce poate fi imprimat, nu este indicată pentru situațiile de imprimare în tiraje mari;
Imprimantă cu laser	• Calitatea deosebit de bună, viteza mare de tipărire, raport preț consumabile/volum mare de imprimare avantajos, recomandă acest tip de imprimantă pentru documente ce necesită precizie/calitate mare şi volum mare şi foarte mare de imprimare.

Tabelul 1.4

Caracteristici de performanță ale imprimantelor uzuale

Tip imprimantă	Rezoluție [dpi]	Format hârtie	Viteză de imprimare	Culoare
Matriceală	~ 180	A3, A4, A5, B5	100÷400 cps	Monocrom
Cu jet de cerneală	300÷2800	A4, A5, B5	3÷15 ppm	Monocrom Color
Cu laser	300÷1200	A4, A5, B5	5÷50 ppm	Monocrom Color

3.5.5. Mouse-ul

Mouse-ul este un periferic de intrare, unidirecțional, care face parte din categoria dispozitivelor de *digitizare* (introducere digitală a datelor). Ca funcționalitate, mouse-ul este asemănător cu tastatura.

Modelul clasic este un dispozitiv de dimensiuni nu prea mari, uşor manevrabil, având la partea superioară 2 sau 3 butoane, iar la partea inferioară o bilă ce se roteşte liber într-un locaş special, cu sensibilitate şi viteză reglabile. Prin deplasarea mouse-ului pe o suprafață orizontală are loc rotirea bilei. Rotirea bilei este descompusă de componentele mouse-ului în deplasare pe două direcții *sus-jos* şi *stânga-dreapta*, deplasare care determină pe ecranul monitorului o mişcare echivalentă a unui *cursor*. Cursorul specific mouse-ului este o reprezentare grafică a poziției curente de pe ecran, de obicei, o săgeată. Cursorul focalizează atenția asupra locului în care vor avea loc acțiunile determinate de apăsarea butoanelor mouse-ului (validare/confirmare de opțiuni, indicare de poziții, deplasări etc). Cursorul de mouse se mai numeşte şi *pointer* sau *indicator*. Declanşarea unei acțiuni (transmiterea unei comenzi, selectarea unei

zone dintr-un text etc) se face prin poziționarea cursorului pe obiectul, imaginea, zona de pe ecran care interesează şi apăsarea unuia dintre butoane.

Exploatarea mouse-ului este determinată de următorii parametri:

- *Rezoluție* (*densitate de puncte transmise*, *sensibilitate*), care depinde de proiectarea hardware;
- *Viteza de apăsare* a unui click dublu;
- *Timp de reacție* la apăsarea butoanelor.

Recent, au apărut diverse variante constructive cum ar fi: mouse-ul cu rotiță, mouse-ul *fără fir*, mouse-ul optic etc.

3.5.6. Alte echipamente periferice

Scanner-ul este un dispozitiv periferic unidirecțional, prin intermediul căruia se pot transfera către o unitate centrală diverse date în format grafic. Datele transferate pot fi prelucrate (mutare, copiere, mărire, micşorare, modificare, conversie etc) cu programe specializate (editor de imagine, OCR-Optical Character Recognition). Un scanner este caracterizat prin *rezoluție optică* (*mxn* dpi), *rezoluție interpolată* (*uxv* dpi), *adâncimea de culoare* (*x* biți), *format* (A4, A3 etc), *tehnologie* (senzori *CCD* sau *CIS*) etc.

Unitățile CD-ROM sunt unități periferice care permit citirea datelor memorate permanent (timp îndelungat) pe discuri compacte (*CD – Compact Disk*). Compact disk-ul este un disc flexibil, dar care stochează datele folosind o metodă optică. Există şi unități destinate citirii/scrierii unor CD-uri care pot fi scrise/rescrise. Capacitatea de stocare este de 650 sau 700 MB. Principala caracteristică a unităților CD-ROM este viteza de transfer, indicată prin factorul de multiplicare a vitezei de transfer a primului periferic de acest tip (150 KB/s). În prezent o dezvoltare deosebită o au compact disk-urile cu densitate mare de înregistare, numite *DVD* şi care necesită unități dedicate numite *unități DVD*.

Calculatoarelor personale actuale le pot fi ataşate o gamă largă de periferice (legătură rețea, sisteme audio, camere digitale, dispozitive video, televizor etc). Legătura se face fie folosind porturile (loc de conectare) interfețelor de tip serial (COM, USB) sau paralel (LPT), fie folosind echipamente fizice dedicate (placă de rețea, placă audio, placă video cu TV-out, tuner TV etc) cu rol de interfață şi care folosesc conexiuni interne standardizate ale unității centrale (ISA, PCI, AGP).

3.5.7. Configurația calculatoarelor personale

Configurația unui calculator personal se referă la echipamentele fizice ce intră în componența sa, fiind o descriere sumară şi strictă a componentelor, ce are scopul de a

furniza date minime asupra firmelor producătoare şi parametrii principali ce caracterizează performanţele şi domeniul de utilizare a sistemului.

Un exemplu de configuraţie a unui calculator personal poate fi următoarea:

- ***Placă de bază***: *ASUS P4V8X-X, SATA, RAID, P4X533*
- ***Procesor***: *PENTIUM 4 2,6 GHz/533*
- ***Memorie***: *256 MB DDR333 PC2700 Syncron*
- ***Unitate disk fix***: *HDD 120 GB 7200 RPM SAMSUNG*
- ***Unitate floppy disk***: *FDD 1,44 MB 3,5" ALPS*
- ***Unitate optică***: *CD-RW ASUS 52x32x52x*
- ***Interfaţă video***: *FX 5200 128 MB DDR ASUS, tv-out, DVI*
- ***Interfaţă audio***: *on board, 6 ch*
- ***Placă reţea***: *Realtek 10/100 Mbps UTP*
- ***Carcasă***: *Miditower ATX Millenium 400 W*
- ***Tastatură***: *PS2 win*
- ***Mouse***: PS2
- ***Multimedia***: *Boxe active LS26A*
- ***Monitor***: *VIEWSTAR 17" CRT OSD TCO99*

În Figura 1.7 este reprezentat schematic un calculator personal cu o configuraţie minimă de echipamente fizice.

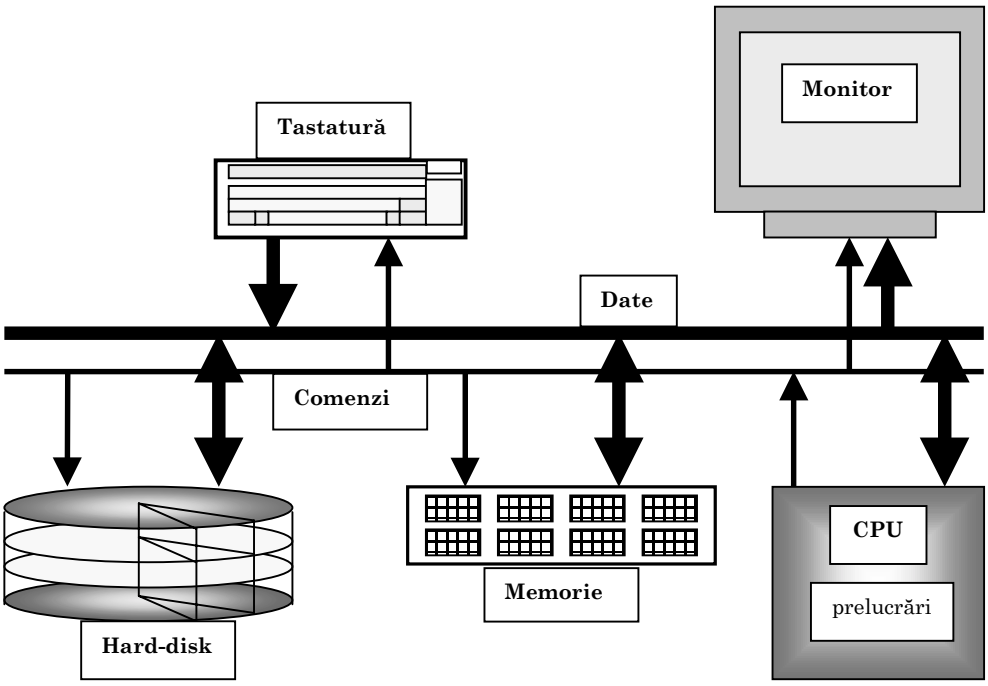


Figura 1.7. Configuraţie minimă a unui calculator personal

LUCRAREA 2

CALCULATOARE PERSONALE. SISTEME DE OPERARE

1. NOȚIUNI INTRODUCTIVE

A doua componentă importantă a unui sistem de calcul este reprezentată de ansamblul de programe disponibile la un moment dat, cunoscut sub denumirea generică de *software*. În lipsa acestei componente, calculatoarele nu ar putea transfera, memora sau prelucra date, nu ar putea realiza multitudinea de aplicații, de la simple jocuri până la simulări complexe utilizate în diverse domenii de activitate (cercetare, industrie, medicină etc).

Programele sunt *colecții de instrucțiuni* care determină execuția unor operații, activități, comenzi de către echipamentele fizice din componența unui calculator pentru a răspunde unei cerințe concrete a utilizatorului.

Execuția unui program, numită şi *rulare*, presupune existența acestuia în memoria internă, executarea secvențială a instrucțiunilor prin transferul acestora în microprocesor, transformarea în instrucțiuni proprii microprocesorului, execuția acestora folosind datele transferate din memoria internă, transferul rezultatelor în memoria internă şi generarea de comenzi către echipamentele periferice pentru transferul rezultatelor. Execuția secvențială a instrucțiunilor de către procesor, în ordinea în care ele sunt scrise în program, poate fi alterată de o instrucțiune, fie necondiționat, fie print-o condiționare locală dată de anumite stări logice.

Programele ce aparțin unui sistem de calcul la un moment dat, pot fi împărțite în:

- ***Programe sistem***, grupate în *programe de inițializare* necesare pornirii calculatorului, ***sistem de operare*** (**SO**), care fac posibilă funcționarea normală şi realizarea sarcinilor primite de calculator de la utilizatori prin intermediul programelor dedicate, *programe utilitare* necesare funcționării sistemului cu performanțe maxime, diagnosticări şi depanări etc;
- ***Programe de aplicații***, care rezolvă problemele specifice ale utilizatorilor apelând la programele sistem.

Un *sistem de operare* se poate considera ca fiind o colecție de programe cu funcții de coordonare şi control asupra resurselor calculatorului.

Prin *resursele* unui calculator se înțelege totalitatea echipamentelor fizice şi a programelor ce pot fi utilizate la un moment dat, în timpul funcționării acestuia.

Una din funcțiile de bază ale sistemului de operare este de *a ascunde* complexitatea gestionării hardware-ului şi de a degreva utilizatorul de detaliile referitoare la comanda echipamentelor fizice.

2. TIPURI DE SISTEME DE OPERARE

În prezent nu există un criteriu unitar de clasificare a sistemelor de operare, motiv pentru care se folosesc mai multe criterii pentru a le putea compara şi caracteriza. Dintre aceste criterii cele mai utilizate sunt:

- După configurațiile hardware ale calculatoarelor ;
- După modul de alocare a resurselor solicitate la un moment dat;
- După organizarea internă a programelor ce intră în componența SO.

După configurațiile hardware pe care le deservesc, se disting:

- Sisteme de operare pentru microcalculatoare (PC); aceste sisteme de operare au ca principale sarcini activarea sistemului la pornirea acestuia, schimbul de date între CPU şi memorie sau dintre memorie şi echipamentele periferice, organizarea datelor într-un sistem structurat care să permită atât menținerea corectitudinii şi integrității, cât şi manevrarea şi prelucrarea acestora etc;
- Sisteme de operare pentru minicalculatoare; aceste sisteme de operare au sarcini mai complexe, cum ar fi *partajarea* (împărțirea) resurselor sistemului de calcul între diferiți utilizatori, planificarea CPU pentru a răspunde cererilor tuturor utilizatorilor etc;
- Sisteme de operare pentru calculatoare medii, mari, care nu se deosebesc prea mult, ca funcții, de cele ale minicalculatoarelor.

După modul de alocare a resurselor, se disting:

- Sisteme de operare *monotasking*; sistemele de calcul cu un astfel de sistem de operare execută, la un moment dat, un singur program, care rămâne activ din momentul lansării (activării) până la terminarea execuției;
- Sisteme de operare *multitasking*, care permite utilizarea partajată a resurselor sistemului de calcul de către mai multe programe.

După organizarea internă a programelor componente, se disting:

- Sisteme de operare *monolitice*, formate dintr-o colecție de *proceduri* (programe) care pot fi apelate după necesități; execuția unei astfel de proceduri nu poate fi întreruptă, aceasta trebuind să execute sarcina în totalitate; acest tip de sistem de operare nu se mai utilizează.
- Sisteme de operare cu *nucleu*, care concentrează sarcinile importante într-o colecție de *rutine* (proceduri) care împreună alcătuiesc nucleul sistemului de operare; toate cererile către echipamentele fizice necesită apeluri ale acestor rutine; acest sistem de operare este specific microcalculatoarelor;
- Sisteme de operare cu *structură stratificată*.

3. SISTEMUL DE FIŞIERE

3.1. Conceptul de fişier

Calculatoarele dispun pentru memorarea şi stocarea permanentă (timp îndelungat) a datelor diferite suporturi fizice corepunzătoare unui anumit echipament periferic. Fiecare periferic cu rol de memorare şi stocare a datelor are propriile caracteristici şi mod de organizare fizică. Pentru a face posibilă utilizarea calculatoarelor, sistemele de operare îşi exercită funcţiile folosind o modalitate uniformă de organizare a datelor, din punct de vedere logic.Astfel, a apărut noţiunea de *fişier* şi *organizarea fişierilor* într-un sistem recunoscut de SO.

Prin ***fişier*** se înţelege o colecţie de date elementare (*articole*) grupate şi organizate logic într-o unitate de memorare, cu scopul de a permite controlul accesului, regăsirea şi modificarea elementelor componente.

Pentru ca un utilizator să poată realiza diferite operaţii asupra fişierelor şi a conţinutului acestora, sistemul de operare are în componenţa sa un set de rutine cunoscut sub denumirea de *sistem de gestiune a fişierelor*. În acest mod, SO asigură corespondenţa fiecărui fişier cu un anumit periferic.

Din punct de vedere a utilizatorului ce efectuează operaţii asupra unui fişier prin intermediul unui program, sistemul de operare face ca acesta să perceapă fişierul ca un întreg, întreg ce rezultă din organizarea sa logică, fără nici o referire reală la implementarea acestuia pe suportul fizic. Prin sistemul de gestiune a fişierelor, sistemul de operare are sarcina de a regăsi la nivel fizic toate datele organizate logic într-un fişier. Pentru a putea îndeplini această sarcină a fost necesar ca suporturile de memorare şi stocare să aibă propria organizare fizică.

3.2. Organizarea fizică a fişierelor

Echipamentele periferice sunt, din punct de vedere a funcţiei pe care o au, de două tipuri:

- *Periferice de schimb de date* cu mediul extern calculatorului (tastatură, monitor, imprimantă, mouse, etc);
- *Periferice de stocare a datelor* pe timp îndelungat, ce folosesc suporturi magnetice.

Din punct de vedere a *unităţii de schimb* de date între memoria internă şi echipamentele periferice, acestea din urmă se împart în:

- *Periferice bloc* (unităţi de disc magnetic);
- *Periferice caracter* (imprimanta, tastatură, mouse, monitor).

Dintre suporturile fizice de memorare şi stocare a datelor cel mai utilizat este suportul de tip disc magnetic. Aşa cum se observă din clasificarea de mai sus, unitatea de schimb de date între memoria internă şi unităţile de disc magnetic este *blocul*. În acest caz, *operaţiile de I/O* (*Input/Output*-Intrare/Ieşire) între memorie şi discul magnetic sunt efectuate în grupuri de unu sau mai multe sectoare de disc. Adresarea unui anumit sector implică precizarea cilindrului, pistei, feţei şi sectorului. Ca urmare, un disc poate fi tratat de către SO ca un tablou unidimensional de *blocuri disc*. Astfel, la nivel fizic, SO consideră fişierul ca o succesiune de blocuri. În fapt, la nivelul fizic al discului magnetic, datele ce alcătuiesc fişierul respectiv pot să ocupe un spaţiu contiguu sau nu.

Utilizatorul, prin programul folosit, are la un moment dat acces numai la o anumită entitate din cadrul unui fişier, cunoscută sub numele de *articol* sau *înregistrare*. Putem spune că un fişier este o mulţime ordonată de înregistrări. Înregistrarea reprezintă mulţimea datelor care fac obiectul unei singure operaţii I/O, în condiţiile specifice fiecărui suport fizic.

O unitate de discuri magnetice (hard-disk, dischetă) se numeşte, generic, *volum*. În cadrul unui volum datele sunt grupate de către SO, la cererile utilizatorului, în fişiere.

3.3. Tipuri de acces la articole

Accesul la un articol reprezintă metoda utilizată pentru localizarea la nivel fizic a articolelor ce alcătuiesc un fişier.

Se cunosc două tipuri de acces:

- *Accesul secvenţial*, care presupune efectuarea , în prealabil, a unui număr de accese egal cu numărul de articole anterioare articolului considerat;
- *Accesul direct*, ce presupune existenţa unei metode de căutare şi identificare a articolului considerat fără parcurgerea secvenţială a articolelor care îl preced; mecanismele prin care este posibil accesul direct sunt: (1) *acces direct prin număr de poziţie*, numit şi *acces relativ*, (2*) acces direct prin conţinut*, când este necesar ca SO să utilizeze o *cheie de acces*.

În practică, de multe ori, se combină accesul direct cu cel secvenţial.

3.4. Principalele tipuri de fişiere

Criteriile de clasificare a fişierelor sunt destul de numeroase, în continuare fiind prezentate cele mai uzuale din punct de vedere practic:

- După lungimea (mărimea) unui articol:
  - Fişiere cu articole de format fix, în care articolele au aceeaşi lungime;

- Fişiere cu articole de format variabil, în care fiecare articol are propria lungime de reprezentare.

❖ După posibilitatea de afişare sau tipărire a conținutului se disting:
- Fişiere text, al căror conținut poate fi afişat pe ecranul unui monitor sau poate fi tipărit la imprimantă;
- Fişiere binare, formate din şiruri de octeți consecutivi, fără nici o semnificație externă, la afişare sau tipărire.

❖ După suportul fizic pe care este memorat sau stocat fişierul:
- Fişiere pe disc magnetic;
- Fişiere tastatură;
- Fişiere imprimantă;
- Etc.

❖ După modurile de acces permise:
- Fişiere secvențiale, care permit numai accesul secvențial; fişierele care au alt suport fizic decât discul magnetic sunt fişiere secvențiale;
- Fişiere cu acces direct, care permit accesul direct cel puțin la operația de citire.

3.5. Specificarea fişierelor

Un utilizator pentru a putea specifica un fişier trebuie să respecte anumite reguli sintactice impuse de SO. În general, referirea la un fişier se face printr-un şir de caractere, diferite de caracterul *spațiu liber*. Şirul de referință conține cinci zone, separate prin caractere speciale:

PERIFERIC	**:**	**CALE**	****	**NUME FISIER**	**.**	**EXTENSIE**	**;**	**VERSIUNE**

Sintaxa specifică fiecărui SO foloseşte, eventual, numai unele zone. Caracterele **:** , \ , **.** , **;** fac parte din sintaxa specificatorului de fişier.

PERIFERIC indică echipamentul periferic unde se află suportul fizic al fişierului. Aceste nume sunt impuse de SO. *Perifericul implicit* este perifericul de tip unitate de disc magnetic, pe care se află fişierele cu care lucrează utilizatorul la un moment dat.

CALE este un şir de caractere, specific fiecărui SO, prin care se localizează poziția fişierului în structura ansamblului de fişiere. În absența specificării explicite a căii, sistemul de operare consideră o *cale implicită*.

NUME FIŞIER este o succesiune de caractere alfanumerice (litere, cifre, caractere speciale), atribuite unui fişier de către utilizatorul care l-a creat.

EXTENSIE (*TIP*) este o succesiune de caractere atribuite fişierului în momentul creerii acestuia, având drept scop reflectarea conținutului fişierului. În general, fiecare SO are extensii standard pentru anumite tipuri de fişiere.

VERSIUNE este un număr care diferențiază mai multe versiuni ale aceluiaşi fişier. Nu este specifică tuturor SO.

În anumite situații, utilizatorul doreşte ca printr-o singură specificare să desemneze nu un singur fişier, ci o *familie* (grup) de fişiere. În acest scop se utilizează caracterul ***** (asterisc), care înlocuieşte orice şir de caractere. Această modalitate de specificare se numeşte *specificare generică* a unei familii de fişiere. Există şi alte modalități de specificare generică, caracteristice numai anumitor SO. Specificarea generică poate apărea doar în zonele NUME FISIER, EXTENSIE, VERSIUNE.

3.6. Operații cu fişiere

3.6.1. Operații cu fişiere la nivel de articol

Principalele operații ce pot fi efectuate asupra unui fişier la nivel de articol sunt:

- *Citirea* (*READ*), citirea unui articol din fişier;
- *Scrierea* (*WRITE*), scrierea unui articol nou în fişier;
- *Inserarea* (*INSERT*), inserarea unui articol nou între două articole existente;
- *Ştergerea* (*DELETE*), ştergerea unui articol dintr-un fişier;
- *Modificarea* (*MODIFY*),modificarea unui articol cu un altul.

3.6.2. Operații cu fişiere la nivel de fişier

În continuare sunt prezentate principalele operații ce se pot efectua asupra fişierelor considerate ca entități de sine stătătoare, indiferent de conținutul lor. Deoarece suportul de tip disc magnetic oferă cele mai multe posibilități de lucru, se va face referire doar la operațiile cu acest tip de fişiere, care sunt:

- *Deschidere* (*OPEN*), informează SO că un anumit fişier va fi activat de către un program ce se execută;
- *Închidere* (*CLOSE*), anunță SO că fişierul specificat încetează a mai fi activ în timpul execuției programului activ;
- *Creare* (*FILE CREATION, NEW*), anunță şi determină SO să activeze un nou fişier;
- *Salvare* (*SAVE AS. SAVE*), informează şi determină SO să includă şi să memoreze fişierul specificat în sistemul de fişiere;

- *Ştergere* (*DELETE*), informează şi determină SO să elimine fişierul specificat din sistemul de fişiere;
- *Copiere* (*COPY*), realizează o copie a fişierului specificat;
- *Schimbarea numelui* (*RENAME*), schimbă numele unui fişier existent în sistemul de fişiere;
- *Listarea* (*PRINT*), este un caz particular de copiere a unui fişier, într-un fişier la imprimantă sau ecranul monitorului;
- *Concatenarea* (*CONCATENATION*), mai multor fişiere într-un singur fişier;
- *Compararea,* prin care se verifică dacă două fişiere sunt identice, prin parcurgerea lor secvențială;
- *Compactarea* (*COMPRESSION*), stocarea unuia sau mai multor fişiere în formă compactă, folosind un spațiu de stocare mai redus decât spațiul necesar pentru fişierele necompactate.

4. SISTEMUL DE OPERARE DOS

Sistemul de operare ***DOS*** (***D****isk* ***O****perating* ***S****ystem*) este unul din sistemele de operare destinat calculatoarelor personale. Sistemul de operare DOS este un SO cu nucleu, monoutilizator şi monotasking. Realizatorii acestui SO nu au acordat o atenție deosebită modului de comunicare calculator-utilizator, astfel încât, comenzile sunt transmise în mod text folosind şiruri de caractere ce pot fi interpretate de rutinele SO.

4.1. Structura sistemului de operare DOS

Sistemul de operare DOS se compune din:

- ***BIOS*** (***B****asic* ***I****nput* ***O****utput* ***S****ystem*), ce cuprinde programele pentru testarea componentelor hardware ale sistemului (la inițializarea acestuia), proceduri pentru comanda şi controlul operațiilor I/O la nivel fizic, proceduri pentru tratarea întreruperilor externe (de la tastatură, de la unitatea de disc flexibil etc); această parte a sistemului de operare este stocată în memoria ROM a calculatorului;
- ***IBMBIO.COM*** (***IO.SYS***), care conține unele extensii ale BIOS-ului, permițând extinderea configurației standard prin adăugarea de noi echipamente periferice fără a fi necesară modificarea BIOS-ului;
- ***IBMDOS.COM*** (***MSDOS.SYS***), care conține procedurile de tratare a operațiilor I/O la nivel logic cu discul magnetic;
- ***COMMAND.COM***, care este interpretorul de comenzi, având ca sarcini asigurarea interfeței cu utilizatorul, făcând astfel posibilă comunicarea utilizator-calculator.

4.2. Sistemul de fişiere DOS

Pentru a funcționa sub sistemul de operare DOS, un calculator trebuie să aibă în configurație cel puțin o unitate de disc magnetic (hard-disk sau floppy-disk), pe care să fie memorate componentele SO. Discul magnetic pe care sunt memorate componentele sistemului de operare se numeşte *disc sistem*. Procesul prin care se obține un disc sistem se numeşte *instalare* (*generare*). Instalarea presupune crearea, folosind un calculator funcțional, a imaginii discului sistem ce trebuie să cuprindă toate programele şi rutinele SO necesare şi suficiente îndeplinirii cererilor utilizatorilor folosind configurația hardware existentă.

Înaintea instalării SO, orice disc magnetic trebuie supus unor operații pregătitoare: ***partiționare***, urmată de ***formatare***. *Partiționarea* se foloseşte pentru hard-disk cu scopul de a crea secțiuni cu destinații diferite. Cel mai adesea un hard-disk poate conține două partiții: *DOS primară*, pe care se vor regăsi componentele SO, şi *DOS extinsă*, care poate cuprinde una sau mai multe unități logice. O *unitate logică* este o parte a discului fix (hard-disk) care, din punct de vedere logic, este considerată ca fiind o unitate separată de disc. *Formatarea* trebuie aplicată tuturor unităților logice de disc după partiționarea discului. Prin formatare se crează structura de organizare a fişierelor recunoscută de DOS. În timpul formatării se verifică integritatea fizică şi execuția corectă a operațiilor de citire/scriere pentru fiecare sector, se izolează sectoarele cu defecte, se creează *tabela de alocare a spațiului FAT* (*Fat Allocation Table*) şi *directorul rădăcină*, iar opțional se pot copia şi componentele sistemului de operare (pentru discurile sistem). Tabela de alocare a spațiului conține date despre spațiul liber şi cel ocupat, fiind actualizată la fiecare operație de creare, respectiv ştergere de fişiere de pe disc. Directorul rădăcină, identificat prin caracterul \, reprezintă primul nivel al structurii distemului de fişiere DOS.

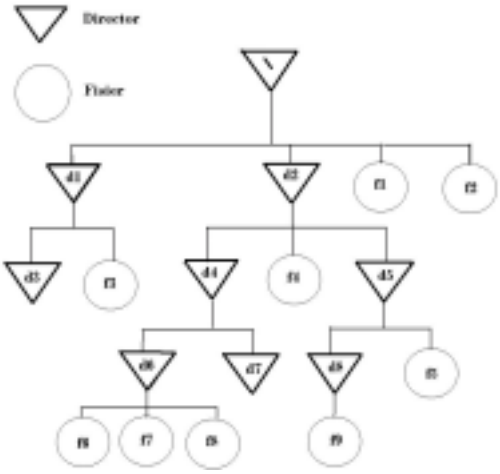


Figura 2.1. Structura arborescentă a sistemului de fişiere DOS

Sistemul de operare DOS organizează fişierele, pe fiecare suport de tip disc magnetic, într-o structură arborescentă. Acest lucru este posibil datorită faptului că DOS grupează fişierele, după dorința utilizatorului, în *directoare* (Figura 2.1), începând cu directorul rădăcină. Acesta poate conține numele unor fişiere, dar şi al altor directoare, care la rândul lor pot conține nume de fişiere şi/sau directoare ş.a.m.d. Spre deosebire de directorul rădăcină, toate celelalte directoare se numesc *subdirectoare*.

Directorul este un fişier special creat de sistemul de operare DOS, ce conține numele fişierelor şi subdirectoarelor ce îi urmează în structura arborescentă, precum şi adresa de început a acestora. Cu excepția directorului rădăcină, creat de sistemul de operare prin formatare, celelalte directoare sunt create de SO, de obicei, la cererea utilizatorului. Pentru a putea fi deosebite fiecărui director i se atribuie un *nume* în momentul creerii. Directorul în care SO identifică un fişier numai prin specificarea numelui său şi, eventual, a extensiei (dacă există sau este necesară) este cunoscut sub denumirea de *director curent*.

Pentru a specifica în mod complet un fişier, în cazul sistemului de operare DOS, trebuie indicate perifericul pe care se află, directorul în care se crează sau există, numele fişierului şi extensia acestuia, dacă este necesar. Sintaxa unui specificator de fişier DOS este următoarea:

[d:][cale]\numefisier[.extensie]

Semnificațiile câmpurilor din specificatorul de fişier sunt următoarele:

- **d:** , reprezintă numele logic al unității de disc, specificat printr-o literă;
- **cale**, reprezintă o succesiune de nume de directoare, începând cu directorul rădăcină, separate prin caracterul \ , care permite localizarea în mod unic a fişierului în structura sistemului de fişiere; la specificarea câmpului *cale* se poate folosi şi numele **.** (punct), pentru directorul curent, respectiv **..** (punct, punct), pentru directorul imediat ascendent directorului curent;
- **numefisier**, reprezintă numele simbolic al fişierului alcătuit dintr-o succesiune de 1÷8 caractere, care pot fi litere, cifre sau caractere speciale; prin utilizarea caracterelor ***** (înlocuieşte un şir de caractere) şi **?** (înlocuieşte un singur caracter) se poate specifica generic o familie de fişiere;
- **extensia**, reprezintă numele extensiei fişierului, formată din 1÷3 caractere (de obicei, trei).

Există câteva nume rezervate, pe care sistemul de operare DOS le foloseşte pentru desemnarea unor periferice din configurația hardware. Ca urmare, aceste nume nu pot fi folosite pentru denumirea unor fişiere:

- **aux, com1**, care este primul port al adaptorului de comunicație serială;
- **con**, care semnifică tastatura (la intrare) şi monitorul consolei (la ieşire) (echipamentele periferice pentru comunicarea calculator-utilizator);
- **prn, lpt1**, primul port al adaptorului de comunicație paralelă;
- **com2,** (**com3**), alte porturi de comunicație paralelă;
- **lpt2,** (**lpt3**), alte porturi de comunicație paralelă;
- **null**, periferic virtual, utilizat pentru testarea unor programe.

Specificarea extensiei este opțională la crearea fişierului, fiind însă obligatorie în cazul referirii unui fişier care a fost creat anterior, iar specificatorul cuprinde acest câmp, precum şi în cazul fişierelor care vor fi prelucrate ulterior cu alte programe.

Cele mai frecvente extensii şi tipul de fişier indicat sunt prezentate în Tabelul 2.1.

Tabelul 2.1

Extensii şi tipurile corespunzătoare de fişiere

Extensia	Tipul fişierului	Extensia	Tipul fişierului
EXE	Fişier executabil	DAT	Fişier de date
COM	Fişier executabil (comenzi)	TXT	Fişier text (ASCII)
BAT	Fişier de comenzi	DOC	Fişier document
HLP	Fişiere de asistență	DBF	Fişier bază de date
INF	Fişier cu informații pentru SO	XLS	Fişier calcul tabelar
SYS	Fişier sistem	FOR	Fişier cod sursă
INI	Fişier de inițializare	OBJ	Fişier cod obiect

4.3. Interfața utilizator

Comenzile sistemului de operare permit utilizatorului să comunice cu sistemul de calcul. În general, prin intermediul acestora pot fi făcute copieri, comparări, ştergeri de fişiere, se poate afişa conținutul unor directoare, se pot obține informații despre calculator şi sistemul de operare etc.

Există două tipuri de comenzi DOS: (1) *interne*, rezidende în memoria internă (stocate pe întrega durată de funcționare a sistemului), (2) *externe*, rezidente pe discul sistem, pentru executarea lor fiind necesară încărcarea în memoria internă. Sistemul de operare DOS tratează orice fişier cu extensiile COM, EXE, BAT ca şi o comandă externă.

Trebuie reținute câteva considerații generale, valabile pentru toate comenzile DOS:

- comenzile sunt urmate, de obicei, de una sau mai multe opțiuni (parametri); unii parametri sunt obligatorii, alții opționali; dacă cei opționali nu sunt specificați, sistemul de operare le atribuie valoarea lor implicită (predefinită);
- comenzile şi parametrii pot fi scrise atât cu litere mari cât şi litere mici;
- comenzile şi parametrii trebuie separate prin spații libere, virgulă, punct, punct şi virgulă, backslash (/), considerate ca separatori;
- comenzile devin active la apăsarea tastei RETURN (ENTER).

În funcție de rolul pe care îl au în gestiunea resurselor sistemului, comenzile DOS se împart: (1) comenzi cu privire la volume; (2) comenzi cu privire la directoare; (3) comenzi cu privire la fişiere; (4) comenzi filtru; (5) comenzi pentru configurarea sistemului; (6) comenzi informaționale.

5. SISTEMELE DE OPERARE WINDOWS

Succesul familiei de sisteme de operare ***Windows*** a fost asigurat de interfața grafică (***GUI***-*Graphical User Interface*), odată cu apariția primului său membru.

Nivelul de îndemânare necesar în realizarea unei acțiuni (lansarea unei comenzi, activarea unui program etc) constituie, din perspectiva utilizatorului, una din principalele deosebiri între o interfață grafică (*GUI*) şi o interfață text. Pentru a putea activa un program pentru editarea unui text, în cazul lucrului sub DOS, utilizatorul trebuia să cunoască directorul unde era memorat programul, apoi să tasteze numele acestuia şi să determine execuția lui prin apăsarea tastei RETURN. Doar utilizatorii experimentați, printr-o configurare corespunzătoare a sistemului, puteau activa doar prin nume un program. Însă, mai ales în cazul comenzilor, se foloseau o serie de parametri care trebuiau să fie corect folosiți, necesitând un efort de memorare. Apariția interfețelor GUI, a micşorat sensibil nevoia de a memora şiruri de caractere, activarea programelor fiind posibilă prin intermediul unor imagini grafice afişate pe ecranul monitorului. Pentru interfețele text aplicațiile sunt accesate prin tastatură, însă în cazul sistemelor cu interfață grafică se utilizează mouse-ul sau alt dispozitiv de tip *pointer*.

Mai important decât avantajele oferite de utilizarea interfeței grafice pentru dialogul utilizator-calculator, este faptul că Windows sunt sisteme de operare *multitasking*. *Multitasking*-ul presupune încărcarea în memorie a cel puțin două programe în acelaşi timp, fiecare având rezervată propria zonă de memoria şi timp procesor în mod alternativ, astfel încât utilizatorul are impresia că cele două programe se execută simultan. Alte avantaje ale sistemelor Windows sunt comunicația şi transferul de date între aplicații prin mecanismul *DDE* (*Dynamic Data Exchange*), precum şi protocolul *OLE* (*Object Linking and Embedding*), care face posibilă combinarea diferitelor aplicații şi integrarea în acelaşi fişier a datelor obținute cu programe diferite. Aceste caracteristice împreună cu *DLL*-urile (*Dynamic Link Library*), care ajută la dezvoltarea de aplicații prin furnizarea de subrutine (programe) standard , precum şi alte caracteristici fac din fiecare Windows un sistem de operare ce poate fi accesibil şi util tuturor categoriilor de utilizatori.

Organizarea sistemului de fişiere este similară cu cea descrisă în cazul sistemului de operare DOS. Sistemele de operare Windows grupează fişierele în *foldere* (dosare) şi fişiere, iar DOS în directoare, subdirectoare şi fişiere. Folderul este asemănător unui dosar, care poate conține alte dosare, pagini document sau pagini-trimiteri. Paginile-trimiteri conțin date asupra locului unde se află alte dosare sau documente, fiind cunoscute sub denumirea de *shortcuts* (scurtături). Shortcut-ul asigură *maparea*, adică trimiterea la locația originală a fişierului sau folderului țintă. Specificațiile de fişier şi folder sunt asemănătoare celor din DOS, însă Windows acceptă nume cu maximum 255 caractere, inclusiv caracterul *spațiu liber*.

Numele de *Windows* (ferestre) provine de la *ferestrele* definite pe ecranul monitorului ce conțin imagini (*pictograme, icon-uri, butoane etc*) folosite fie pentru activarea aplicațiilor pe care le reprezintă fie transmiterea de comenzi către sistem.

5.1. Interfața grafică Windows

Oricare sistem de operare Windows utilizează ecranul monitorului ca fiind suprafața de lucru a unui birou real. Această suprafață se numeşte *desktop*. Desktop-ul constă din toată suprafața afişabilă a ecranului şi cuprinde toate celelalte elemente de interfață (ferestre, pictograme, casete de dialog etc). Aici se pot aranja diferitele obiecte în funcție de dorința şi necesitățile de lucru ale utilizatorului. Lucrul cu un anumit obiect necesită *deschiderea* acestuia, ceea ce determină apariția unei *fereastre de aplicație* (*application window*) pe desktop. Când s-au încheiat activitățile legate de respectivul obiect, el poate fi *închis* sub forma unei imagini grafice (pictograme) şi aşezat pe desktop în aşa fel încât să poată fi redeschis la nevoie. Alături de ferestrele de aplicație există şi alte tipuri de ferestre necesare desfăşurării dialogului dintre utilizator şi sistem (casete de dialog, ferestre de atenționare, meniuri contextuale etc), aşa cum se poate observa în Figura 2.2.

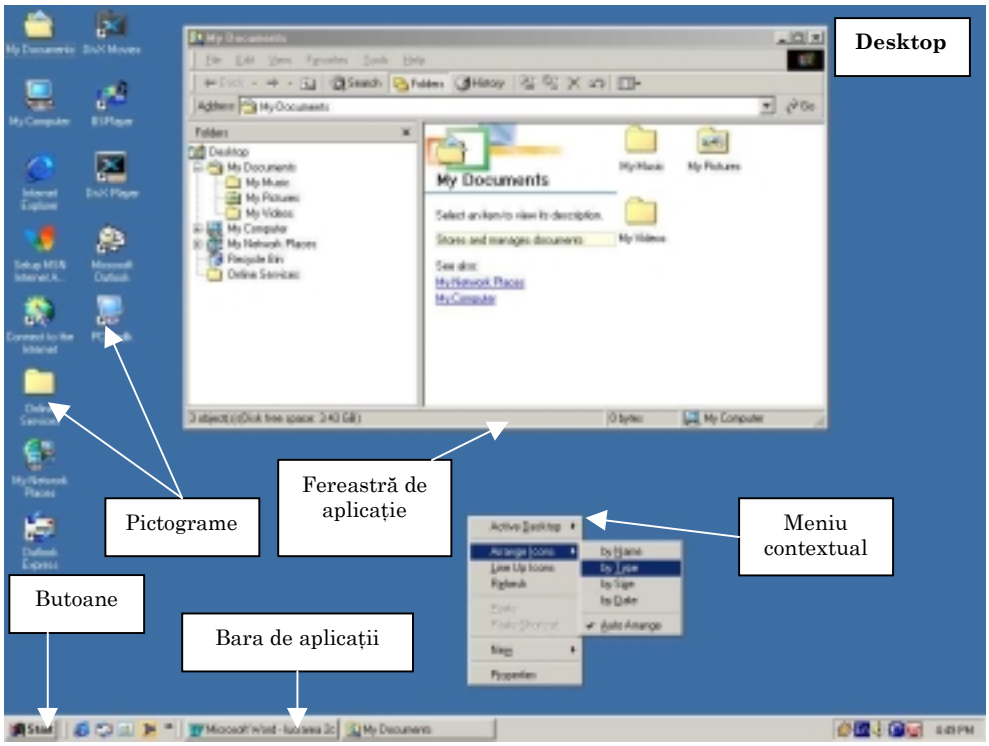


Figura 2.2. Elemente ale interfeței grafice Windows.

Pentru a putea fi utilizate ferestrele, indiferent de tipul lor, conțin o serie de obiecte grafice, cum ar fi: meniul, bara de unelte, bara de stare, bara de defilare (orizontală sau verticală), butoane, comutatoare, declanşatoare etc (Figura 2.3, 2.4, 2.5).

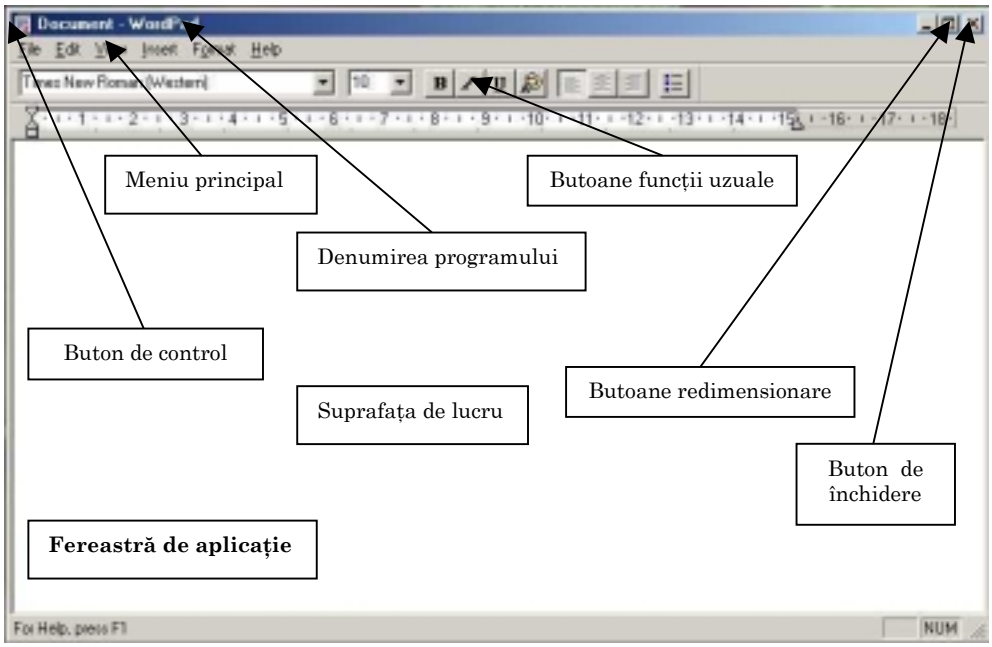


Figura 2.3. Elemente grafice ale unei ferestre de aplicație

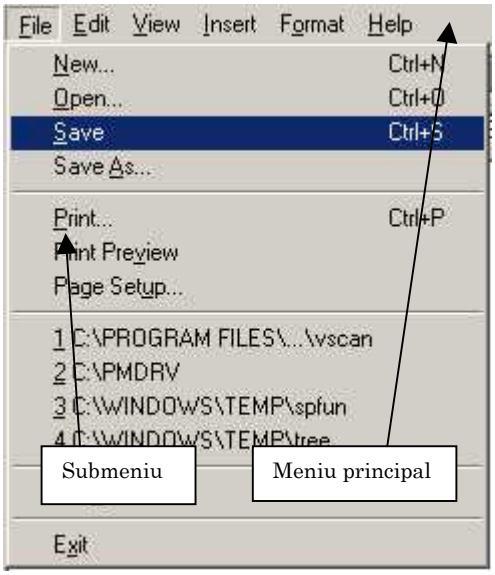


Figura 2.4. Meniuri atașate unei ferestre de aplicație

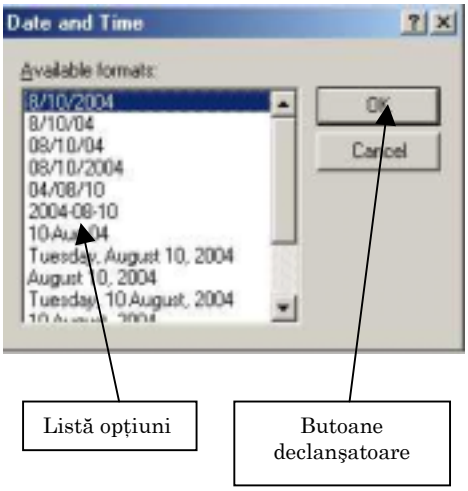


Figura 2.5. Casetă de dialog

5.2. Programe utilitare

Sistemele de operare Windows au integrate şi programe utilitare cu care pot fi realizate diferite tipuri de lucrări. Cele mai cunoscute şi utile sunt *My computer*, *Windows Explorer*, *Control Panel*.

5.2.1. Programul utilitar *My Computer*

Programul *My Computer* este un utilitar de gestiune a resurselor sistemului, care în fereastra de aplicație specifică apar sub formă de pictograme (Figura 2.6).

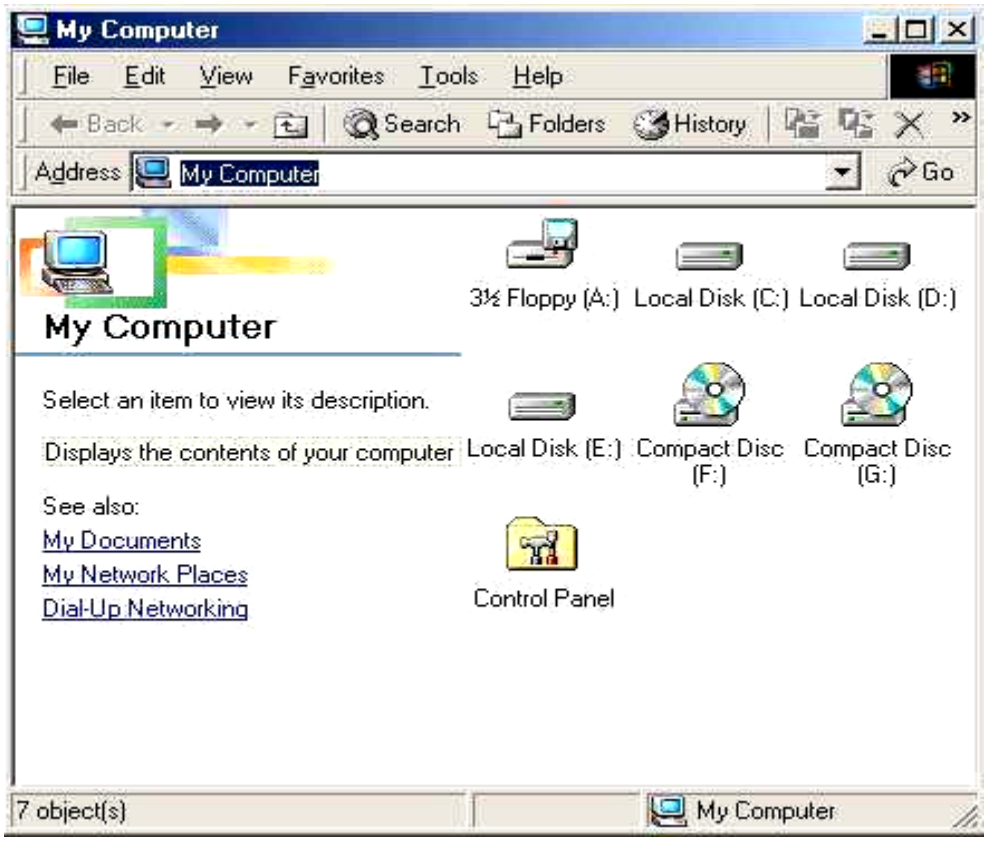


Figura 2.6. *Fereastra programului utilitar My Computer.*

Efectul provocat de un *dublu-clic* (apăsarea dublă a butonului stâng al mouse-ului) după poziționarea cursorului grafic al mouse-ului pe o astfel de pictogramă depinde de tipul acesteia:

- *folder*, deschiderea folderului determină afişarea în fereastră a folderelor şi fişierelor (sub formă de pictograme) pe care le conține;

- *unitate de disc*, afişarea în fereastră (sub formă de pictograme) a fişierelor şi folderelor pe care le conține;
- *fişier executabil*, pornirea programului respectiv;
- *fişier document*, deschiderea documentului prin activarea programului cu care a fost creat sau unul de acelaşi tip.

Deschiderea unei unități de disc, activând opțiunea *New* a meniul *File*, se pot crea noi foldere, fişiere de diferite tipuri etc. Celelalte opțiuni ale meniului principal şi opțiunile submeniurilor corespunzătoare permit gestiunea sistemului de fişiere, configurarea modului de lucru cu acest program, modul de afişare a datelor pe care le afişează.

5.2.2. Programul utilitar *Windows Explorer*

Foarte înrudit şi asemănător cu utilitarul *My Computer*, programul *Windows Explorer* este utilizat pentru gestionarea sistemului de fişiere, însă prin configurația ferestrei este mult mai adecvat *explorării* acestuia (Figura 2.7).

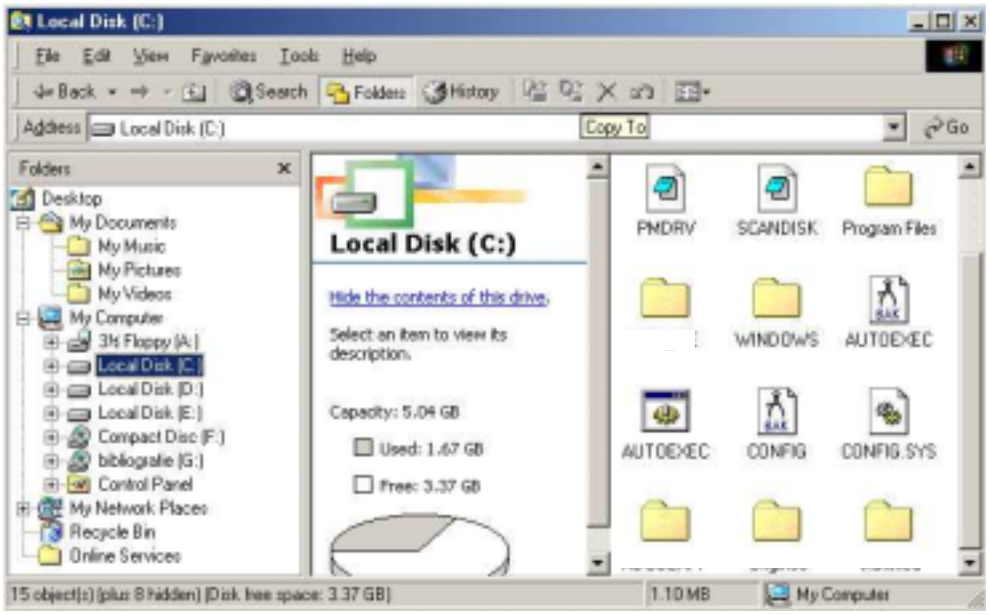


Figura 2.7. Fereastra corespunzătoare programului utilitar *Windows Explorer*.

Se observă că fereastra principală este împărțită în mai multe ferestre. Cea din stânga cuprinde structura arborescentă a sistemului de fişiere, nivelele de subordonare fiind de la stânga la dreapta. Un dublu-clic pe fiecare din nume va determina afişarea conținutului acestuia în fereastra din dreapta, care este asemănătoare cu fereastra

My Computer. Se observă că pentru unele nume din fereastra stângă există o casetă cu simbolul **+** , ceea ce indică faptul că respectivele foldere conțin alte foldere şi/sau fişiere. Execuția unui clic pe acest simbol determină afişarea structurii folderului şi apariția simbolului – în caseta corespunzătoare numelui folderului.

Utilitarul *Windows Explorer* are posibilitatea de a face operații de copiere/mutare de foldere/fişiere prin metoda *drag-and-drop* între cele două ferestre. Această operație se realizează prin poziționarea cursorului grafic pe numele folderului/fişierului (sau grupului selectat de astfel de obiecte), se apasă şi se menține apăsat butonul stâng al mouse-ului, până când prin deplasarea cursorului împreună cu numele selectate se ajunge în noua locație din a doua fereastră, după care se eliberează butonul mouse-ului şi se aşteaptă efectuarea operației.

Prin poziționarea cursorului pe numele obiectului şi apăsarea butonului drept al mouse-ului (clic) se deschid *meniuri contextuale* (listă verticală de opțiuni ce se referă la operații specifice obiectului pe care s-a efectuat clic) care permit efectuarea de operații asupra acelui obiect.

5.2.3. Programul utilitar *Control Panel*

Acest utilitar are rolul de a personaliza configurarea sistemului de calcul în funcție de necesitățile utilizatorului. Prin acest program utilizatorul are acces la date privind resursele sistemului de calcul şi poate modifica modul lor de funcționare (Figura 2.8).

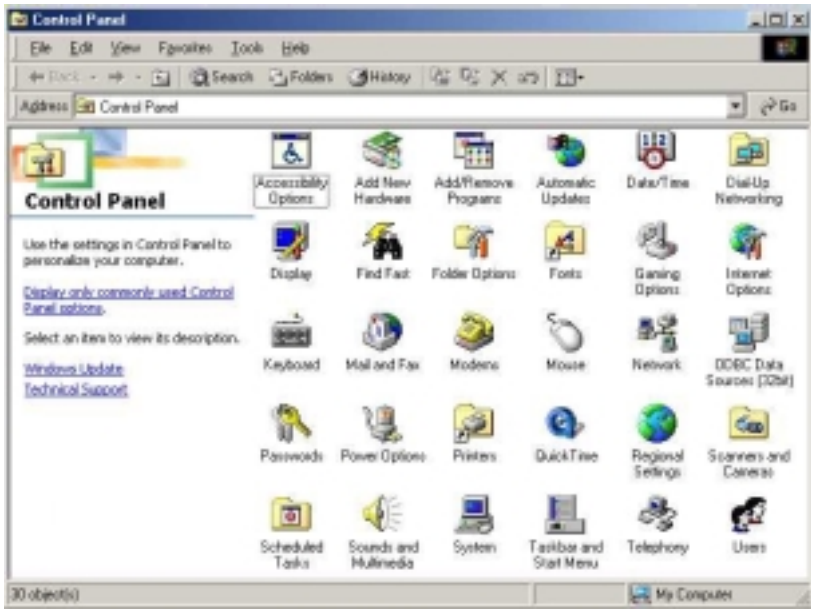


Figura 2.8. Fereastra de lucru a utilitarului Control Panel

LUCRAREA 3

CALCULATOARE PERSONALE. SOFTWARE DE APLICAȚII

1. NOȚIUNI INTRODUCTIVE

Pentru a putea rezolva probleme specifice (creare şi modificare de documente de tip text, imagini grafice, calcule ştiinţifice etc), pe lângă sistemul de operare, resursele software ale unui calculator trebuie completate cu *programe de aplicaţii*. Dacă sistemul de operare asigură buna funcţionare şi întreţinerea sistemului de calcul, programele de aplicaţii oferă utilizatorului mijloacele de a crea fişiere de un anumit tip şi de a modifica datele pe care acesta le conţine, ele fiind dedicate unor activităţi cum ar fi: (1) editarea şi procesarea de fişiere tip text sau document, (2) gestiunea bazelor de date, (3) calcul tabelar, (4) crearea de imagini grafice, (4) proiectarea asistată de calculator, (5) crearea de publicaţii, (6) prelucrarea de imagini grafice, (7) comunicarea între calculatoare, (8) dezvoltarea de software, (9) procesare multimedia, (10) gestiune financiar-contabilă, (10) poştă electronică etc.

Pe lângă programele de aplicaţii, există şi aşa-numitele *programe utilitare*, utilizate pentru gestiunea resurselor sistemului, compresia datelor, detectarea şi combaterea viruşilor etc.

2. EDITAREA ŞI PROCESAREA DE TEXTE ŞI DOCUMENTE

Crearea unui fişier de tip text sau document este asemănătoarea cu activitatea de redactare a unui text folosind maşina de scris. Dacă în cazul utilizării maşinii de scris suportul pe care se stochează textul este hârtia, atunci când folosim calculatorul tipul de suport este divers: suport magnetic, suport optic, suport de hârtie. Este evident că această activitate necesită şi programe de aplicaţii adecvate. Ele sunt capabile, atunci când utilizatorul doreşte, să îndeplinească pe lângă funcţia de creare a textului şi funcţiile de *salvare* (stocarea pe un suport magnetic), *deschidere*, *închidere*, *imprimare*, *modificare*.

Trebuie menţionat faptul că este indicat să se facă distincţie între fişiere text şi fişiere document. Fişierele text nu conţin decât litere, cifre şi caractere speciale (setul de caractere ASCII), iar fişierele document vor conţine atât caracterele ASCII cât şi obiecte create cu alte tipuri de programe de aplicaţii (imagini grafice, tabele, sunet etc). pentru crearea şi prelucrarea fişierelor text se folosesc *editoarele de texte* (*Notepad*, *Edit* etc), iar pentru fişierele document, *procesoarele de documente* (*WordPad*, *Microsoft Word*, *Microsoft Works*, *Lotus WordPro*, *Corel Wordperfect* etc).

Procesoarele de documente dispun de comenzi prin care se controlează caracteristici referitoare la:

- Încadrarea documentului în pagini cu dimensiuni standardizate, prin indicarea distanțelor dintre text şi marginile formatului de pagină;
- Evidențierea sau estomparea unor porțiuni din document;
- Realizarea unor tipuri de aliniere diferite;
- Aspectul caracterelor;
- Organizarea obiectelor în pagină;
- Inserarea de obiecte create cu alte programe de aplicații (imagini grafice, fotografii, sunet, animație etc).

Cu ajutorul programelor de procesare documente moderne se pot obține documente foarte complexe: cu mărimi şi aspect ale fonturilor într-un număr mare de combinații, text organizat pe coloane multiple cu dimensiuni diferite, cu inserări de obiecte de diverse tipuri, cu zone de diferite culori etc. Însă atunci când gradul de complexitate cerut pentru organizarea datelor în document este ridicat, cum este cazul la crearea de publicații, trebuie să se recurgă la programe de aplicații specializate în *DeskTop Publishing* (*DTP*).

Există două motive principale care determină utilizarea din ce în ce mai frecventă a calculatorului în editarea şi procesarea de texte şi documente:

- *Aspectul grafic*; folosind resursele unui calculator se obține relativ uşor un nivel ridicat al calității grafice a textelor şi documentelor;
- *Valoarea de întrebuințare*; folosind pentru stocare suportul magnetic, datele conținute în fişierul corespunzător textului sau documentului vor putea fi refolosite, fie pentru imprimare, transmitere prin mijloace electronice, includerea în alte documente etc.

2.1. Caracteristici de lucru comune ale programelor pentru editare şi procesare de text

Cele mai frecvente operații care apar în timpul creerii sau modificării unui text sau document, cu excepția scrierii textului folosind tastatura, sunt: (1) poziționarea sau repoziționarea cursorului în interiorul spațiului alocat pe ecran, care se poate efectua fie cu tastele direcționale fie prin poziționarea cursorului grafic al mouse-ului şi executarea unui clic; (2) selectarea unei zone de text cu scopul de a o şterge, copia, muta sau modifica; selecția se poate realiza în două moduri, fie folosind combinația dintre tasta Shift şi una din tastele direcționale, fie prin menținerea apăsată a butonului drept al mouse-ului şi deplasarea acestuia până se selectează întreaga zonă; (3) renunțarea la ultimile operații efectuate, folosind opțiunea *Undo* din meniul *Edit*.

Indiferent de programul utilizat există o serie de comenzi de editare care pot fi obținute folosind aceleaşi taste sau combinații de taste (Tabelul 3.1).

Tabelul 3.1

Comenzi de editare obținute prin acționarea unor taste singure sau în combinații

Tasta sau combinația	Acțiunea produsă
Deplasarea şi poziționarea cursorului în interiorului textului sau documentului	
Săgeată stânga/dreapta	Deplasare stânga/dreapta a cursorului cu un caracter
Săgeată sus/jos	Deplasare sus/jos a cursorului cu un rând
Home	Salt la începutul rândului curent
End	Salt la sfârțitul rândului curent
Page Up	Salt la pagina anterioară de afişare pe ecran
Page Down	Salt la pagina următoare de afişare pe ecran
Ctrl + săgeată stânga	Salt la începutul cuvântului din stânga poziției curente
Ctrl + săgeată dreapta	Salt la sfârşitul cuvântului din dreapta poziției curente
Ctrl + săgeată sus	Salt la începutul paragrafului anterior
Ctrl + săgeată jos	Salt la începutul paragrafului următor
Ctrl + Page Up	Salt la începutul paginii anterioare
Ctrl + Page Down	Salt la începutul paginii următoare
Ctrl + Home	Salt la începutul documentului
Ctrl + End	Salt la sfârşitul documentului
Ştergerea/inserarea de caractere sau şiruri de caractere	
Delete/Del	Şterge un caracterdin dreapta poziției curente a cursorului
BackSpace	Şterge un caracter din stânga poziției curente a cursorului
Insert	Activează/dezactivează modul de scriere cu inserare şi dezactivează/activează modul de scriere prin suprapunere
Comenzi de copiere, mutare, inserare	
Ctrl + C	Copie caracterul sau şirul de caractere sau obiecte în *clipboard* (zonă de memorie alocată pentru stocarea temporară a datelor în vederea transferului în altă poziție în acelaşi document sau în alte documente sau fişiere)
Ctrl + X	Mutarea datelor selectate în clipboard şi ştergerea lor din document
Ctrl + V	Inserarea datelor stocate în clipboard la poziția indicată
Comenzi de anulare, căutare şi înlocuire	
Ctrl + Z	Anulează ultima operație de editare şi reface datele existente înaintea ultimei operații de editare
Ctrl + F	Permite căutarea în cadrul documentului a unui caracter sau şir de caractere şi, eventual, înlocuirea cu un alt caracter sau şir de caractere

2.2. Editorul de texte *WordPad*

WordPad este un editor de texte ce poate fi folosit pentru crearea de documente obişnuite, cu posibilități de lucru relativ reduse. *Wordpad* asigură utilizatorului următoarele facilități de editare: (1) operații la nivel de fişier (creare, salvare, deschidere, închidere, imprimare), (2) operații propriu-zise de editare (creare text, selectare, copiere, decupare, inserare, căutare, înlocuire), (3) operații de personalizare a documentului prin alegerea dimensiunilor paginii, a dimensiunilor zonei cu text, a atributelor (caracteristicilor) caracterelor, definirea stilului paragrafului.

Fereastra de lucru, prezentată în Figura 3.1 cuprinde următoarele elemente: (1) linia de titlu, (2) linia meniului principal, (3) linia pictogramelor pentru operații cu fişiere şi operații de editare, (4) linia pictogramelor pentru stiluri, fonturi, alinieri, (5) rigla de poziționare, (6) linia de stare, (7) suprafața de lucru.

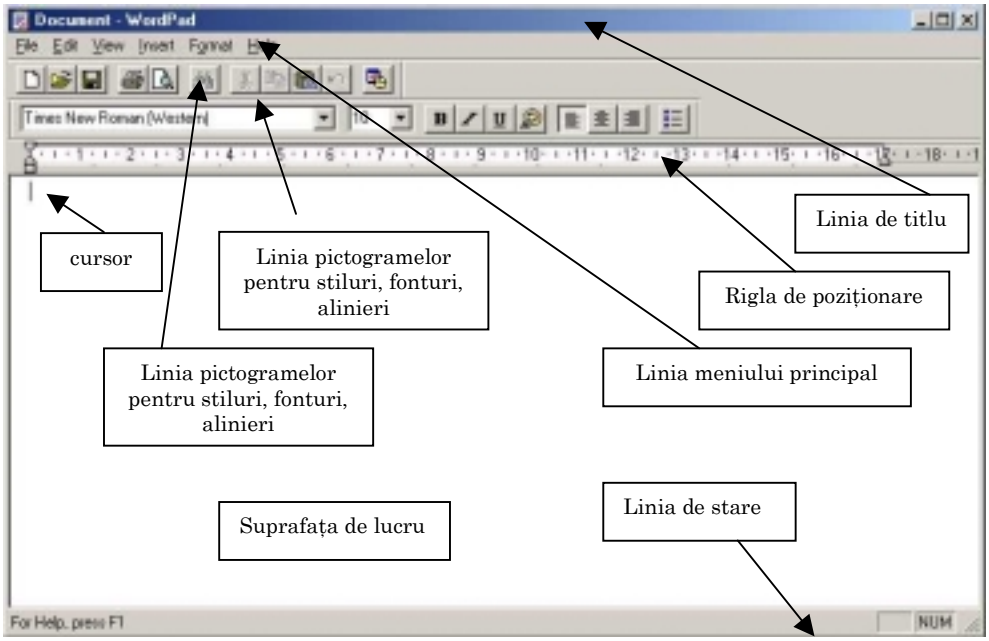


Figura 3.1. Fereastra de lucru a editorului *WordPad*

Linia de titlu cuprinde pictograma editorului, denumirea documentului urmată de a editorului, butoanele de minimizare, maximizare şi de închidere a ferestrei de lucru (Figura 3.2). Prin poziționarea cursorului grafic pe pictograma editorului şi apăsarea butonului drept al mouse-ului se deschide un meniu contextual cu opțiuni pentru ferestre. Acelaşi lucru se obține prin poziționarea cursorului grafic pe linia titlului şi apăsarea butonului drept al mouse-ului.

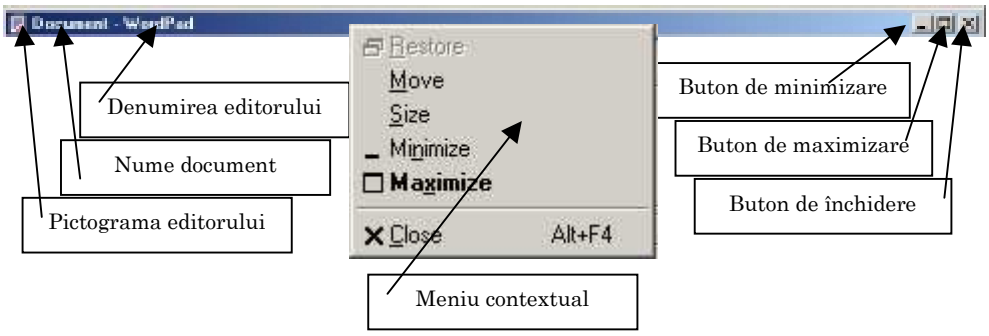


Figura 3.2. Linia de titlu a editorului *WordPad*

Linia meniului principal cuprinde opțiunile principalelor facilități de editare. La activarea fiecărei opțiuni se deschide un submeniu cu opțiuni referitoare la opțiunea meniului principal (Figura 3.3).

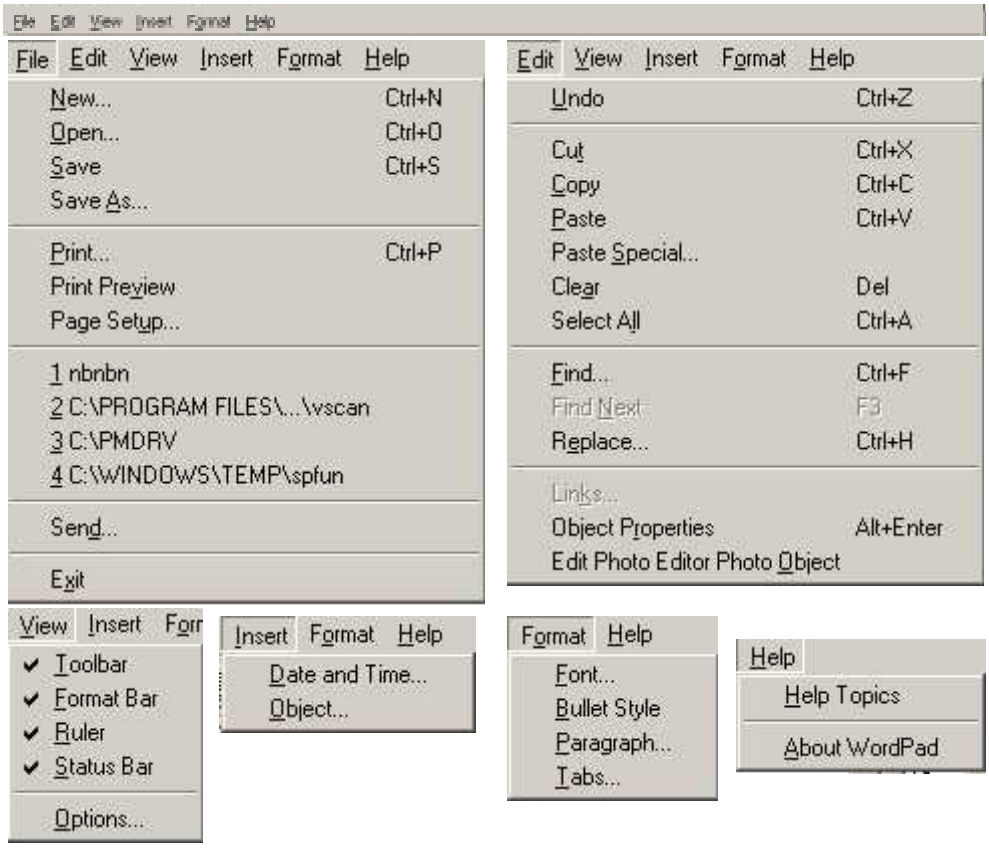


Figura 3.3. Linia meniului principal şi submeniurile corespunzătoare.

Activarea opțiunilor din fiecare submeniu determină îndeplinirea următoarelor funcții:

- Meniul *File*:
  - *New*…, creează un nou document;
  - *Open*…, deschide un document existent;
  - *Save*, salvează fişierul creat sub numele curent;
  - *Save As*…, salvează o nouă copie a fişierului într-un fişier nou cu acelaşi nume (alt folder) sau cu nume diferit;
  - *Print*…, imprimă fişierul activ;
  - *Print Preview*, afişează paginile aşa cum vor fi imprimate;
  - *Page Setup*, permite setarea (alegerea dimensiunilor, modul de aşezare şi formatul standardizat al paginii de imprimare)
  - *Send*…, permite trimiterea fişierului către poşta electronică;
  - *Exit*,dezactivează editorul şi transmite controlul sistemului de operare.

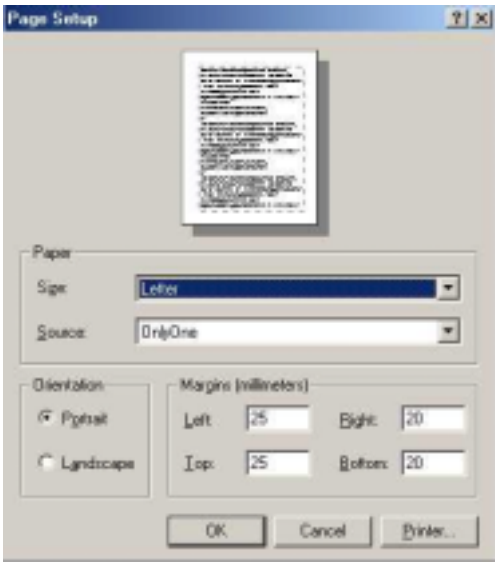


Figura 3.4. Căsuță de dialog pentru opțiunea *Page Setup*

Opțiunile urmate de caracterele **...** prin activare determină deschiderea unor casete de dialog, care permit furnizarea de date suplimentare pentru execuția funcției specifice (Figura 3.4).

De exemplu, prin activarea opțiunii *Page Setup* se deschide caseta de dialog din Figura 3.4. Caseta este divizată în mai multe zone ce au câmpuri proprii, prin deschiderea cărora utilizatorul își alege valorile dorite. Orice casetă de dialog cuprinde cel puțin două butoane declanşatoare: [*OK*] şi [*Cancel*]. Activarea butonului *OK* determină închiderea ferestrei de dialog şi atribuirea valorilor indicate de utilizator pentru câmpurile din casetă. Activarea butonului *Cancel* anulează toate modificările făcute de utilizator asupra câmpurilor şi închide caseta de dialog.

Pentru anumite opțiuni sunt indicate şi combinațiile de taste care determină realizarea aceleaşi funcții.

Activarea opțiunilor se poate face fie prin poziționarea cursorului grafic pe numele acestora şi executarea unui clic cu butonul stâng, fie prin apăsarea succesivă a unor taste. Activarea meniului principal se face prin apăsarea tastei *Alt*, iar poziționarea pe unul din meniuri cu ajutorul tastelor săgeți sau apăsarea literei subliniate. Deschiderea submeniului se face prin apăsarea tastei *Return*, iar poziționarea pe una din opțiuni cu tastele săgeți sau tasta corespunzătoare caracterului subliniat. Accesul la diferite câmpuri şi butoane ale unei casete de dialog se poate face, de asemenea, fie cu mouse-ul prin executarea unui clic în zona câmpului sau butonului fie prin apăsarea tastei *Tab* pentru deplasarea în zona fiecărui câmp, apăsarea tastei *Return* pentru activarea câmpului sau butonului.

- Meniul *Edit*
  - *Undo*, anulează ultima acțiune;
  - *Cut*, decupează din document zona selectată şi o plasează în *Clipboard*;
  - *Copy*, copie zona selectată în *Clipboard;*
  - *Paste,* inserează conținutul *Clipboard*-ului la punctul indicat prin poziția cursorului;
  - *Paste Special…*, înserează în document la punctul indicat obiecte create cu alte programe;
  - *Clear*, şterge zona selectată din document;

- *Select All*, selectează întregul document;
- *Find…*, caută caracterul sau şirul de caractere indicate;
- *Find Next*, caută o nouă locaţie în document a caracterului sau şirului de caractere indicate;
- *Replace…*, caută şi înlocuieşte un carcater sau un şir de caractere cu alt caracter sau şir de caractere;
- *Links*, afişează sau modifică legăturile dintre obiecte şi document;
- *Object Properties*, deschide o casetă de dialog pentru modificarea proprietăţilor obiectului activ.

❖ Meniul *View*:

- *Toolbar*,activarea opţiunii determină afişarea liniei pictogramelor pentru submeniurile File şi Edit;
- *Format bar*, activarea opţiunii determină afişarea liniei pictogramelor pentru stiluri, fonturi, alinieri;
- *Ruler*, activarea acestei opţiuni determină afişarea riglei de poziţionare;
- *Status bar*, activarea acestei opţiuni determină afişarea liniei de stare;
- *Options…*, permite setarea (alegerea) diferitelor tipuri de opţiuni referitoare la configurarea ferestrei aplicaţiei (modul de afişare) (Figura 3.5).

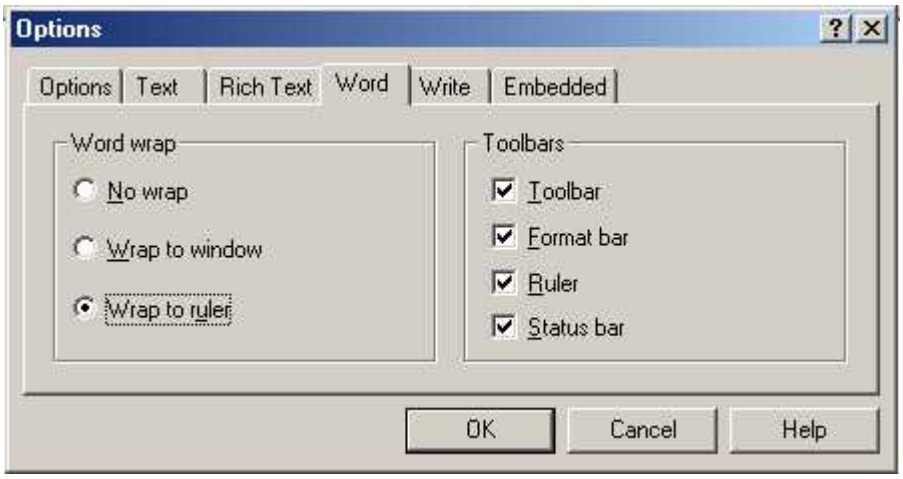


Figura 3.5. Caseta de dialog a opţiunii *Options*.

Această casetă de dialog permite configurarea ferestrei aplicaţiei folosind *butoane radio* (Word wrap) şi *comutatoare* (Toolbars). La un moment dat numai unul din butoanele radio poate fi *activ* (marcat în caseta din stânga). Comutatoarele au două

poziții corespunzătoare stărilor *activat* (marcată în caseta din stânga) şi *neactivat*. În cazul comutatoarelor pot fi activate mai multe la un moment, ele având funcții independente.

Opțiunile meniului *View* se comportă ca un set de comutatoare. Prezența caracterului √ înaintea opțiunii indică faptul că opțiunea este activă. Selectarea ei din nou determină trecerea dintr-o stare în alta.

- Meniul *Insert*:
  - *Date and Time…*, inserează la poziția indicată data şi ora în diverse moduri de afişare;
  - *Object…*, inserează la poziția indicată obiecte create cu alte programe existente în resursele sistemuluisau alte tipuri de obiecte;

- Meniul *Format*:
  - *Font…*, schimbă stilul şi dimensiunea caracterelor;
  - *Bullet Style*, crează o listă în document a căror elemente sunt indicate printr-un punct aşezat în partea stângă;
  - *Paragraph…*schimbă caracteristicile de editare a paragrafelor (mărimea spațiului liber de la începutul paragrafului, alinierea rândurilor);
  - *Tabs…*, setează şi şterge mărimea spațiului liber de tip *Tab* pentru paragrafele selectate.

- Meniul *Help*:
  - *Help Topics*, afişează conținutul sistemului de asistență;
  - *About WordPad*, afişează date despre program, numărul versiunii şi notița de copyright.

Linia pictogramelor pentru operații cu fişiere şi operații de editare este formată din pictograme, fiecare având o semnificație precisă identică cu a unei opțiuni din submeniul *File* sau *Edit* (Figura 3.6). Activarea unei pictograme se face asemănător cu activarea unei taste sau a unui buton prin apăsare. Poziționând cursorul grafic pe pictograma dorită şi executând un clic cu butonul drept al mouse-ului are loc activarea opțiunii corespunzătoare pictogramei şi declanşarea funcției acesteia.

Figura 3.6. Linia pictogramelor pentru operații cu fişiere şi de editare.

Aceste pictograme au un grad ridicat de standardizare ele având aceeaşi semnificație în majoritatea programelor ce rulează pe calculatoare sub Windows.

Corespondența dintre pictograme şi opțiuni este indicată în continuare.

 - *New*,  - *Open*,  - *Save*,  - *Print*,  - *Print Preview*,

 - *Find*,  - *Cut*,  - *Copy*,  - *Paste*,  - *Undo*,  -Date and Time.

Linia pictogramelor pentru stiluri, fonturi, alinieri cuprinde pictograme corespunzătoare unor opțiuni ale submeniului *Format* precum şi unor câmpuri din casetele de dialog corespunzătoare acestora (Figura 3.7).

Figura 3.7. Linia pictogramelor pentru stiluri, fonturi, alinieri

Semnificația pictogramelor este cea indicată în continuare.

 - (*Font)* prezintă *fontul* (forma caracterelor) curent, folosit în documentul activ; executarea unui clic cu butonul drept al mouse-ului pe butonul stâng al acestei pictograme se deschide o fereastră unde sunt afişate fonturile disponibile; executarea unui clic pe un alt font şi închiderea ferestrei determină schimbarea fontului curent; schimbarea fontului se poate face şi pentru caracterele selectate;

 - (*Font Size*) afişează mărimea curentă a fontului utilizat; executarea unui clic cu butonul stâng al mouse-ului pe butonul drept al pictogramei deschide o fereastră unde sunt afişate toate mărimile predefinite pentru fonturi; executarea unui clic pe o altă mărime şi închiderea ferestrei determină schimbarea mărimii fontului; schimbarea mărimii fontului se poate face şi pentru caracterele selectate; schimbarea mărimii fontului la valoarea dorită (exprimată în *pt*, punctul egal cu *1/72* inches) se poate face şi prin tastarea noii mărimi în zona de indicare a valorii curente şi apăsarea tastei *Return*;

 - (*Bold*) determină afişarea şi imprimarea caracterelor în stilul *îngroşat;*

 - (*Italic*) determină afişarea şi imprimarea caracterelor în stilul *înclinat*;

 - (*Underline*) determină afişarea şi imprimarea caracterelor în stilul *subliniat*;

 - (*Align Left*) fixează alinierea textului în raport cu marginea stângă a textului;

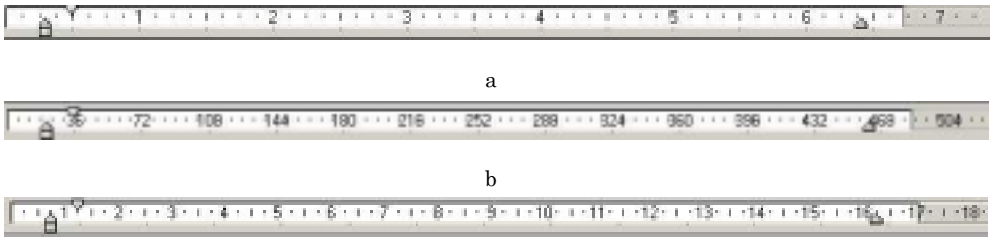
 - (*Center*) fixează centrarea textului în raport cu marginile spațiului de scriere;

 - (*Align Right*) fixează centrarea textului în raport cu marginile spațiului de scriere;

 - (*Bullets*) inserează un marcator pentru fiecare paragraf indicat de utilizator şi aliniază textul în raport cu marcatorul la aceeaşi distanță.

Ultimile pictograme (*Bold÷Bullets*) au rol de comutator, având două stări activat şi neactivat. În starea activat caracterele se afişează şi imprimă conform cu funcţia comutatorului. Funcţia comutatorului acţionează şi asupra caracterelor selectate, prin activarea acestuia după operaţia de selectare.

Rigla de poziţionare indică pe ecran poziţiile marginilor pentru text, coloane, pagină, raportate la dimensiunile reale ale formatului paginii (Figura 3.8).



a

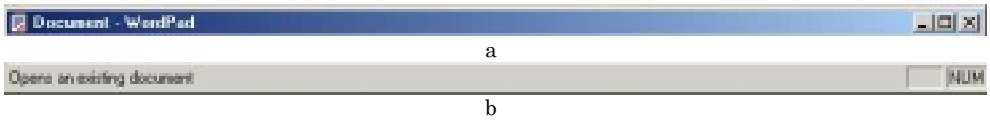
b

c

Figura 3.7. Rigla de poziţionare: a- inches; b- points, c- centimetri.

Pe imaginile riglei se observă existenţa butoanelor cu care se pot alege alinierile pentru paragrafe şi marginile textului în pagină, conform celor două scale posibil de vizualizat (aliniere paragraf, aliniere margini).

Linia de stare conţine date despre documentul cu care se lucrează (Figura 3.9). Linia de stare a editorului *WordPad* conţine doar două câmpuri, unul pentru indicarea tastei ce activează sistemul de asistenţă *Help*, iar al doilea indică dacă tasta *Num Lock* este activă sau nu (Figura 3.9a).



a

b

Figura 3.9. Linia de stare a editorului *WordPad*: a-starea 1, b-starea 2.

În câmpul din stânga sunt afişate şi scurte indicaţii despre funcţiile pictogramelor. Pentru a putea fi afişate trebuie doar poziţionat cursorul ecran pe imaginea pictogramei (Figura 3.9b).

2.3. Editarea unui document cu *WordPad*

Scrierea textului se face conform regulilor obişnuite de utilizare a *tastelor alfanumerice*, *Caps Lock* şi *Shift*.

Ştergerea unui caracter sau şir de caractere se poate face în mai multe moduri: (1) folosind tastele *Backspace* sau *Delete* (*Del*); (2) selectând mai întâi caracterul sau şirul de caracteer (indicat de utilizat) şi apăsarea tastelor arătate mai înainte;

(3) selectarea caracterului sau şirului de caractere (indicat de utilizat) activarea meniului *Edit* şi activată opțiunea *Clear*.

Copierea unui caracter, şir de caractere sau obiect şi memorarea în *Clipboard* se face prin selectarea caracterului, şirului de caractere sau obiectului, urmată de: (1) folosirea combinației de taste *Ctrl+C* (prezentată anterior), (2) activarea meniului *Edit* şi opțiunii *Copy* din submeniul corepunzător, (3) activarea pictogramei corespunzătoare opțiunii *Copy*, (4) utilizarea *meniului contextual*.

Decuparea unui caracter, şir de caractere sau obiect şi memorarea în *Clipboard* se face prin selectarea caracterului, şirului de caractere sau obiectului, urmată de: (1) folosirea combinației de taste *Ctrl+X* (prezentată anterior), (2) activarea meniului *Edit* şi opțiunii *Cut* din submeniul corepunzător, (3) activarea pictogramei corespunzătoare opțiunii *Cut*, (4) utilizarea *meniului contextual*.

Inserarea unui caracter sau şir de caractere se face prin poziționarea cursorului la locul dorit (cu tastatura sau mouse-ul) şi se execută operația de inserare. Aceasta poate fi făcută în următoarele moduri: (1) utilizarea tastaturii pentru scriere; (2) inserarea unui text sau obiect existent în *Clipboard* folosind combinațiile cunosute de taste, meniul *Edit* şi opțiunea *Paste* din submeniul corespunzător, activarea pictogramei *Paste* sau cu ajutorul *meniurilor contextuale* prin poziționare cursor grafic în punctul de inserare, apăsare buton dreapta mouse ce determină deschiderea meniului contextual, activarea opțiunii *Paste*; (3) inserarea prin mecanismul *drag-and-drop* (selectare, poziționare cursor grafic pe zona selectată, apăsare buton stânga + menținere apăsare, deplasare mouse pentru deplasarea selecției la punctul de inserare, eliberare buton).

Crearea unui document nou se face prin: (1) combinația de taste *Ctrl+N*; (2) activarea opțiunii *New* din submeniul corespunzător meniului *File*; (3) activarea pictogramei corespunzătoare opțiunii *New*.

Salvarea unui document apare în două situații: (1) salvarea unui fişier nou creat, căruia trebuie să i se atribuie un nume, sau salvarea unui fişier sub un nume nou; (2) salvarea modificărilor efectuate asupra conținutului unui fişier folosind acelaşi nume.

În prima situație se utilizează opțiunea *Save As* din submeniul corespunzător meniul *File*. Această opțiune poate fi activată doar prin deschiderea submeniului *File*. După activare se deschide o casetă de dialog (Figura 3.10) care cuprinde câmpuri pentru indicarea folderului în care va fi inclus fişierul, pentru indicarea numelui, pentru alegerea tipului fişierului. După efectuarea setărilor se activează butonul [*Save*]. Dacă se doreşte ca salvarea să nu aibă loc cu setările realizate se activează butonul [*Cancel*].

Cea de a două situație se întâlneşte doar în cazul salvării unui fişier ce are deja atribuit un nume. În acest caz se activează opțiunea *Save* din submeniul corespunzător meniului *File*. Se poate folosi şi combinația de taste *Ctrl+S*.

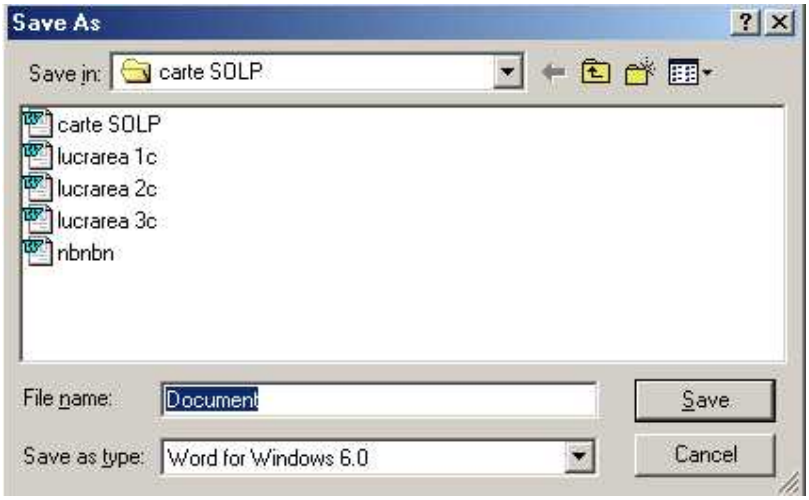


Figura 3.10. Caseta de dialog corespunzătoare opțiunii *Save As*.

Deschiderea unui fişier este posibilă doar pentru un fişier creat anterior, atunci când se doreşte modificarea, imprimarea sau cunoaşterea conținutului său. Operația de deschidere se poate face prin utilizarea combinației de taste *Ctrl+O* sau prin activarea opțiunii *Open* din submeniul corespunzător meniului *File*. Activarea opțiunii determină deschiderea unei casete de dialog (Figura 3.11) care este foarte asemănătoare cu cea a opțiunii *Save As*.

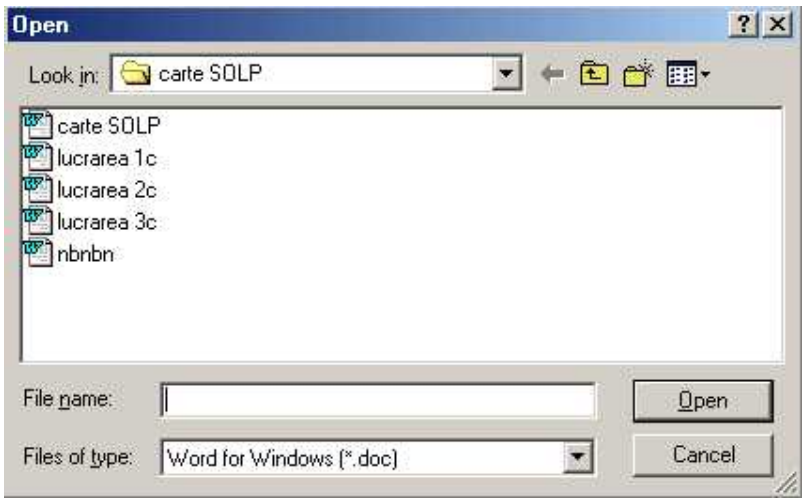


Figura 3.11. Caseta de dialog a opțiunii *Open*

Închiderea unui document se face prin activarea opțiunii *Exit* din submeniul corespunzător meniului *File*. Înainte de închidere se deschide o fereastră de interogare prin care utilizatorul este întrebat dacă salvează sau nu modificările efectuate sau dacă renunță la operația de închidere.

LUCRAREA 4

ETAPELE REZOLVĂRII PROBLEMELOR CU AJUTORUL CALCULATORULUI. DESCRIEREA ALGORITMILOR

1. NOȚIUNI INTRODUCTIVE

Transpunerea pe calculator a modurilor de rezolvare a unor tipuri de probleme presupune, pe lângă cunoaşterea problemei (problemelor) şi cunoştințe privind întocmirea şi reprezentarea algoritmilor, precum şi cel puțin un limbaj de programare.

Limbajul este principalul mijloc de comunicare. Cele mai cunoscute sunt *limbajele naturale*.

Odată cu dezvoltarea ştiinței şi tehnicii au luat naştere *limbajele artificiale* folosite la realizarea comunicării în diferite domenii ale activității umane, fiind utilizate de grupuri restrânse de persoane. Astfel, există limbajul matematic sau al formulelor chimice.

Pentru a putea fi utilizat, orice limbaj trebuie învățat. Învățarea unui limbaj necesită cunoaşterea regulilor de formare a propozițiilor corecte (*sintaxa*) şi a regulilor ce descriu formarea semnificației acestor propoziții (*semantica*).

Propoziția reprezintă o construcție completă, de sine stătătoare, ce are un înțeles. Propozițiile sunt construite folosind *simboluri* (*semne*, *caractere*), în număr finit, care împreună alcătuiesc *alfabetul limbajului* (*setul de caractere*).

Limbajele de programare sunt mijloace de comunicare între programator şi calculator. Programatorul transmite calculatorului *prelucrările* pe care trebuie să le execute acesta din urmă pentru ca pornind de la *datele de intrare* să obțină *datele de ieşire* (*rezultate*).

Prelucrările necesare sunt exprimate printr-o succesiune de comenzi sau instrucțiuni care alcătuiesc ceea ce se numeşte ***program***. Exprimând într-un program ceea ce trebuie să facă un calculator, programatorul îi transmite acestuia din urmă un ***algoritm***. Împreună cu algoritmul, programatorul trebuie să transmită calculatorului şi informații despre ***tipul şi organizarea datelor*** (de intrare, de ieşire şi uneori intermediare), precum şi despre ***relațiile*** dintre ele. Limbajul de programare oferă posibilitatea transmiterii algoritmului şi informațiilor referitoare la date de la programator la calculator.

2. ETAPELE REZOLVĂRII UNEI PROBLEME

Principalele etape pe care un programator trebuie să le parcurgă pentru a putea rezolva o problemă cu ajutorul calculatorului sunt: (1) specificarea problemei; (2) proiectarea algoritmului de rezolvare a problemei; (3) programarea propriu-zisă; (4) testarea şi depanarea programului; (5) exploatarea şi întreținerea programului.

Specificarea problemei. În prima fază se face o analiză a problemei, care are drept rezultat definirea acesteia sub forma unui enunț complet şi precis prin care să se stabilească ce trebuie să facă *programul*, adică *funcțiile sale*. În acest scop este necesar să se identifice datele ce se vor prelucra (datele de intrare) şi rezultatele cerute (datele de ieşire). *Datele de intrare* (*input*), respectiv *datele de ieşire* (*output*) sunt reprezentate ca *variabile*, care furnizează *notații simbolice* pentru date.

În această etapă se stabilesc, de asemenea, reprezentările şi organizarea variabilelor pe suporturile externe de informație (hârtia, discul magnetic, ecranul monitorului).

Variabilele sunt identificate prin nume simbolice şi sunt caracterizate prin *tip* şi *structură*. Acestea determină modul de reprezentare internă şi modul de prelucrare în timpul execuției programului.

Proiectarea algoritmului de rezolvare a problemei. Scopul acestei etape este de a elabora un *algoritm* prin care să fie îndeplinite funcțiile programului, precum şi *structurile de date* folosite.

Noțiunea de algoritm nu are o definiție riguroasă. Se poate spune că un algoritm reprezintă o secvență finită de operații, ordonată şi complet definită prin care datele inițiale (input) sunt prelucrate şi transformate în rezultate (output). Înțelesul mai larg al noțiunii de algoritm este cel de metodă sau procedeu.

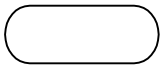
Fiecare propoziție care face parte din descrierea unui algoritm este o instrucțiune ce trebuie executată de un procesor (uman sau artificial). Fiecare instrucțiune implică executarea unor operații prin care datele sunt prelucrate în sensul modificării şi transformării în alte date până la obținerea rezultatelor. În ansamblu, algoritmul specifică posibile succesiuni de operații ce determină transformări ale datelor inițiale, obținându-se în final rezultate.

Principalele proprietăți necesare unui procedeu sau unei metode de prelucrare pentru a fi algoritm sunt: (1) să fie bine definit; (2) să fie descris exact; (3) să fie finit; (4) să fie universal.

În general, o problemă se poate rezolva prin mai multe metode, care de fapt înseamnă mai mulți algoritmi. Acceptarea unui algoritm ca soluție a unei probleme date necesită realizarea următoarelor activități: (1) *elaborarea* propriu-zisă a algoritmului; (2) *exprimarea* algoritmului într-un stil clar şi concis, folosind metode de descriere specifice; (3) *validarea* algoritmului prin verificarea logicii sale; (4) *analiza* algoritmului pe baza unor criterii de apreciere a valorii acestuia; rezultatul analizei permite realizarea unui studiu comparativ pe baza căruia se va alege algoritmul cel mai bine cotat; de regulă, sunt acceptați algoritmii care au un consum minim de resurse (timp de execuție, memorie alocată).

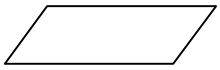
Modalitățile de reprezentare a algoritmilor sunt diverse (reprezentarea în limbaj natural, tabele de decizie, pseudocod, scheme logice etc), însă, cel mai adesea, *descrierea* algoritmilor se face prin: (1) *scheme logice*; (2) *limbaj pseudocod*.

Schemele logice sunt reprezentări grafice ale algoritmilor, folosind simboluri care indică tipul acțiunii. *Simbolurile* utilizate pentru alcătuirea schemelor logice sunt:

- *Simbol terminal* 

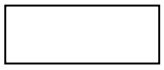
, utilizat pentru crearea *blocului START*, care marchează începutul unui algoritm;nu poate avea decât o singură ieşire;

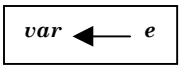
, utilizat pentru crearea *blocului STOP*, care marchează oprirea logică a unui algoritm;nu are decât o singură intrare;

- *Simbol intrare-ieşire* 

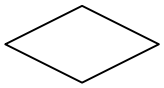
, utilizat pentru crearea *blocului de intrare*, care indică transferul datelor de intrare în ordinea şi tipul stabilite prin listă (*list var*); are o singură intrare şi o singură ieşire;

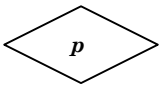
, utilizat pentru crearea *blocului de ieşire*, care indică transferul datelor de ieşire în ordinea şi tipul stabilite prin listă (*list var*););are o singură intrare şi o singură ieşire;

- *Simbol calcul (proces)* 



, utilizat pentru crearea *blocului de calcul* (*proces*), care indică evaluarea unei expresii *e* şi atribuirea rezultatului variabilei cu numele *var*; are o singură intrare şi o singură ieşire;

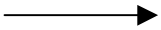
- *Simbol decizie* 



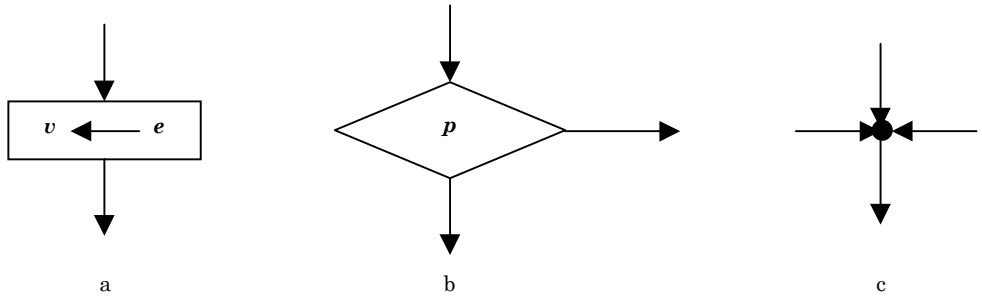
, utilizat pentru crearea *blocului de decizie*, care indică luarea unei decizii în funcție de predicatul *p*; are o singură intrare şi două ieşiri, dar numai una posibilă la parcurgerea algoritmului;

- *Simbol conector* 

,indică legătura între punctele *n* ale unei scheme logice.

- *Simbol nod*  , indică punctele de reunire din shema logică; are mai multe intrări şi o singură ieşire;
- *Simbol săgeată*  , arată modul de parcurgere a algoritmului; indică un singur sens şi face legătura între celelalte simboluri.

Schema logică este un *graf orientat* ce poate conține trei tipuri de noduri: (1) *nod funcțional*; (2) *nod predicativ*; (3) *nod de reunire* (Figura 4.1).



a b c

Figura 4.1. Nodurile unei scheme logice: a-nod funcțional; b-nod decizional; c-nod de reunire.

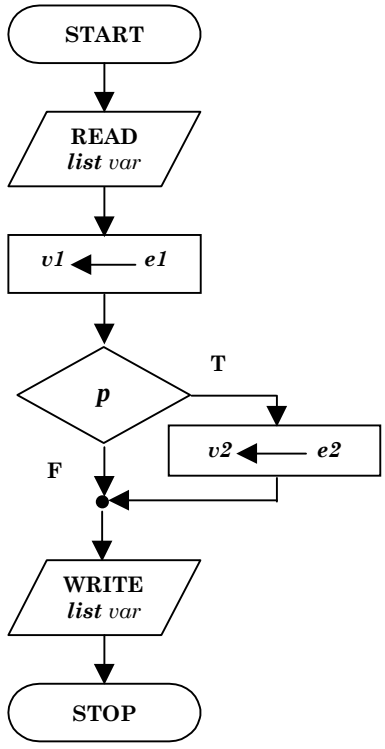


Figura 4.2. Exemplu de schemă logică

În Figura 4.2 se prezintă un exemplu de schemă logică pentru un algoritm oarecare.

Limbajul pseudocod poate fi definit ca un *limbaj simbolic* situat între limbajul natural şi limbajul de programare, obținut prin introducerea unor reguli specifice limbajelor de programare în limbajul natural.

Pseudocodul utilizează setul de caractere al limbajului natural cu ajutorul căruia alcătuieşte un *set de cuvinte simbolice* cu semnificații stricte utilizate în construcții sintactice care simulează structurile necesare descrierii algoritmilor. Aceste cuvinte se mai numesc şi *cuvinte cheie*.

În limbajul pseudocod se pot construi două tipuri de propoziții: *propoziții standard*, ce respectă regulile limbajelor de programare şi *propoziții nestandard*, apropiate de propozițiile limbajului natural. Propozițiile nestandard sunt texte care descriu părți ale algoritmului încă complet elaborate. Pe lângă aceste două tipuri de propoziții, în textul algoritmului pot să apară şi propoziții

explicative, care pot fi considerate nestandard, numite comentarii.

Descrierea (reprezentarea) algoritmilor în pseudocod se caracterizează printr-o mare flexibilitate, posibilitatea exprimării algoritmilor într-un limbaj apropiat de cel natural, o conversie facilă a acestora într-un limbaj de programare.

Găsirea algoritmului constituie, de cele mai multe ori, etapa cea mai dificilă a rezolvării unei probleme cu ajutorul calculatorului. Programatorul trebuie să conceapă o listă de instrucțiuni care să descrie secvența de operații ce vor fi executate de către procesor pentru prelucrarea datelor în scopul obținerii rezultatelor cerute.

Instrucțiunile utilizate pentru descrierea algoritmilor sub formă de pseudocod se împart în două categprii: (1) *declarații de date*; (2) *instrucțiuni executabile* şi *neexecutabile*. Structura formală a unui algoritm descris în pseudocod poate fi cea prezentată mai jos.

```
   program nume simbolic;
!  comentariu;
   declarații de date;
   instrucțiune;
   instrucțiune;
   …
   …
   …
   stop;
   end
```

În exemplul alăturat termenii utilizați au următoarele semnificații:

❖ *program* este cuvântul cheie al primei instrucțiuni prin care algoritmul capătă un *nume simbolic*;

❖ *comentariu* reprezintă una sau mai multe linii prin care se fac precizări asupra destinației algoritmului sau explicații referitoare la logica algoritmului şi rolul instrucțiunilor ; comentariile sunt utile pe parcursul elaborării, validării, analizei şi refacerii;

❖ *declarațiile de date* sunt instrucțiuni neexecutabile prin care se stabilesc structura şi tipul datelor ce sunt transformate în urma execuției programului;

❖ *grupul* de instrucțiuni ce urmează descrie operațiile prin care datele de intrare sunt transformate în rezultate;

❖ instrucțiunea *stop* indică sfârşitul logic al algoritmului şi este o instrucțiune executabilă;

❖ instrucțiunea *end* indică sfârşitul algoritmului şi este *obligatorie*.

În urma descrierii unui algoritm în pseudocod, ales ca soluție a problemei, se obține un *program* ce poate fi executat de un sistem de calcul.

În prezent pentru proiectarea algoritmilor se utilizează frecvent: (1) proiectarea ascendentă sau descendentă, (2) proiectarea modulară, (3) proiectarea structurată.

Proiectarea ascendentă sau descendentă se bazează pe următoarele metode, derivate din modul de abordare a rezolvării problemelor: (1) metoda descendentă (top-down, sus-jos), (2) metoda ascendentă (bottom-up, jos-sus).

Metoda *descendentă* (*top-down*) presupune descompunerea problemei inițiale în subprobleme independente, care vor fi descompuse mai departe în subprobleme de complexitate mai mică, obținându-se o *atomizare* a problemei inițiale. Pentru fiecare subproblemă este elaborat un algoritm de rezolvare propriu pentru care sunt specificate şi interacțiunile cu ceilalți algoritmi. În final, prin rafinarea (detalierea) algoritmului de

rezolvare a problemei inițiale se obține o *structură cvasiliniară* sau *arborescentă*. Metoda *top-down* presupune proiectarea şi testarea algoritmului problemei inițiale, urmate de aceleaşi activități pentru fiecare subproblemă ş.a.m.d. până se ajunge la subproblemele ultimului nivel de descompunere.

Metoda *ascendentă* (*bottom-up*) utilizează algoritmii unor probleme de complexitate mică, pe care îi asamblează obținând algoritmi pentru probleme mai complexe ş.a.m.d. până se ajunge la algoritmul problemei ce trebuie rezolvată. Această metodă are marele dezavantaj că nu permite detectarea erorilor de integrare decât după ce sunt parcurse toate etapele de integrare a algoritmilor subproblemelor.

În practică, de cele mai multe ori se utilizează o metodă de *proiectare mixtă*, ce integrează cele două metode.

Proiectarea modulară presupune descompunerea problemei în subprobleme ce sunt rezolvate individual, obținându-se pentru fiecare un modul. *Modulul* este considerat o unitate structurală de sine stătătoare, fie program, fie subprogram, fie o unitste de program.Un modul poate conține un alt modul sau poate fi conținut într-un alt modul. Un modul poate fi format din mai multe submodule. Proiectarea modulară presupune identificarea modulelor şi a relațiilor dintre acestea, a ordinii de legătură şi de ierarhizare. Rezultă că această metodă poate fi integrată cu cele două metode prezentate mai sus, ambele având drept rezultat obținerea unor algoritmi pentru subprobleme, a căror rezolvare poate duce la obținerea de module. Proiectarea modulară are numeroase avantaje legate, în principal, de descompunerea de la complex la simplu prin aplicarea principiului *Divide et Impera* (împarte şi domină). Problemele simple pot fi mai uşor rezolvate, testate, validate şi analizate. În acest fel identificarea posibilelor erori ce pot să existe este mult mai probabilă.

Proiectarea structurată este o metodă de proiectare a algoritmilor dar mai ales un *stil de programare*. Această metodă este strâns legată de *teorema de structură*, potrivit căreia *orice program executabil este echivalent cu un program alcătuit doar cu trei instrucțiuni simple, numite citire, scriere, atribuire (read, write, ⟵) şi trei tipuri instrucțiuni compuse, numite secvență, selecție, iterație.*

În sens mai larg, proiectarea structurată a algoritmilor constă din totalitatea restricțiilor şi regulilor de elaborare ce definesc un stil de programare, care se bazează pe proiectarea top-down, proiectarea modulară şi celelalte metode de proiectare, având drept rezultat scrierea unui *program structurat*.

Eficiența unui algoritm depinde printre altele, de structura şi modul de descriere a datelor de prelucrat. După complexitatea lor, datele pot fi *date scalare* (*atomice, elementare*) şi *date structurate* (*grupate*).

Datele sunt caracterizate prin *notație* (*identificator*), *atribute* şi *valoare*.

Notația (*identificatorul*) este un nume care se asociază unei date cu scopul de a o distinge de alte date şi de a putea fi referite în mod unic în procesele de prelucrare.

Atributele precizează proprietăți ale datei şi determină modul în care aceasta

va fi tratată în procesul de prelucrare. Ca exemple de atribute putem enumera: *tipul* datei, *domeniul valorilor*, *modul de reprezentare*, *precizia reprezentării* etc.

Valoarea unei date scalare nu poate fi descompusă, însă, valoarea unei date structurate este formată de un grup de valori (*n-uplu*, $n \geq 2$). Datele structurate au o *structură* proprie care defineşte modul de aranjare (dispunere) sau relaţiile ce se pot stabili între elementele ce le compun. *Elementele* unei date structurate sunt la rândul lor date simple sau date structurate.

Conceptual, un *tip de date* este considerat ca fiind o mulţime de valori ce formează *domeniul* tipului de date. Pe acest domeniu se definesc o serie de *operaţii*, *funcţii* şi *relaţii* care operează cu aceste valori. În cazul în care domeniul este format din date simple, se spune că tipul de date este *simplu*, iar când datele sunt structurate tipul de date este *structurat*. O dată structurată are două atribute: (1) elementele componente; (2) structura. Structura defineşte regulile ce stau la baza grupării elementelor.

Din punct de vedere al domeniului de valori (natura) asociat unei date elementare, care conduce la definirea operaţiilor intrinseci, se disting: (1) date întregi (*integer*), (2) date reale (*real*), (3) date booleene (*boolean, logical*), date caracter (*char, string*), date pointer (*pointer*).

Folosind datele elementare de mai sus se pot forma, de exemplu, date structurate omogene (*array*) sau eterogene (*record*).

De modul cum sunt organizate datele supuse prelucrării, depinde eficienţa algoritmului de prelucrare.

Programarea propriu-zisă. Programarea propriu-zisă cuprinde:

- *Editarea fişierului sursă*, într-un limbaj de programare (în cazul nostru FORTRAN);
- *Compilarea fişierului sursă*, operaţie prin care se crează fişierul obiect, în cod maşină;
- *Generarea fişierului executabil* prin operaţia de editare a legăturilor, fişier ce reprezintă programul propriu-zis, ce poate fi executat de calculator;
- *Executarea programului* pe calculator.

Testarea şi depanarea programului. Scopul testării programelor este depistarea şi eliminarea erorilor. Metoda tradiţională de testare este aceea a execuţiei programului pentru seturi de date de intrare şi ieşire a căror corectitudine se stabileşte pe altă cale. Testarea programelor poate fi un mijloc eficient de a indica prezenţa erorilor, dar nu şi un mijloc de a demonstra prezenţa lor.

Exploatarea şi întreţinerea programului. Fiecărui program trebuie să i se alcătuiască o documentaţie prin care să se precizeze toate informaţiile utile despre programul realizat. Documentaţia facilitează înţelegerea programului de către utilizatori, precum şi modificarea ulterioară a acestuia de către specialiştii implicaţi în elaborarea şi întreţinerea sa.

3. ALGORITMUL DE REZOLVARE A ECUAȚIEI DE GRADUL AL DOILEA

Problemă: Fiind dați coeficienții reali ai unei ecuații de gradul al doilea, $a,b \in R$, $a \neq 0$, să se descrie un algoritm prin care să se calculeze, dacă există, rădăcinile reale ale acesteia.

Specificarea problemei. Fiind cunoscuți coeficienții reali ai unei ecuații algebrice de gradul al doilea , să se calculeze rădăcinile reale ale acesteia prin metoda cunoscută; dacă aceste rădăcini există să fie transferate la o unitate de ieşire; dacă nu există, să fie transferat la o unitate de ieşire un mesaj de atenționare.

Stabilirea funcției (funcțiilor) algoritmului.

- Calculează rădăcinile reale ale ecuației:
  $$a \cdot x^2 + b \cdot x + c = 0, \quad a,b,c \in R, \quad a \neq 0$$
- transferă rezultatele la o unitate de ieşire, dacă rădăcinile reale există;
- transferă la o unitate de ieşire un mesaj de atenționare, dacă rădăcinile reale nu există.

Stabilirea datelor, organizarea şi reprezentarea acestora. Cu acest algoritm vor fi prelucrate şi se vor obține următoarele tipuri de date

- *date de intrare* (*input*): *a,b,c* – variabile scalare de tip real; variabila *a* trebuie să fie nenulă; notațiile simbolice folosite sunt *a,b,c*;
- *date de ieşire* (*output*): x_1, x_2 – variabile scalare de tip real; notațiile simbolice folosite sunt *x1, x2*; dacă rădăcinile reale nu există, datele de ieşire vor fi reprezentate de un mesaj de atenționare; notația folosită va fi constanta de tip caracter, *"Ecuatia nu are radacini reale"*; dacă valoarea variabilei *a* este *0* , datele de ieşire vor fi reprezentate de un mesaj de atenționare; notația folosită va fi constanta de tip caracter *"Date de intrare incorecte! a este nenul!"*
- *date intermediare*: Δ- variabilă de tip real; notația simbolică folosită este *delta*; a_1 – variabilă utilizată pentru noua valoare a variabilei *a*, dacă valoarea inițială a fost *0*.

Stabilirea algoritmului de rezolvare. În continuare se prezintă descrierea algoritmului folosind limbajul natural:

instrucțiune	răspuns	acțiune
0	1	2
a. *definește* variabilele: real a, a1, b, c, delta, x1, x2;	-	-
b. *citește* datele de intrare: a, b, c;	-	transfer date de intrare
c. *verifică* dacă a este nul: (a = 0) ?;	adevărat	continuă cu **d**
	fals	salt la **g**
d. *scrie* "Date de intrare incorecte! A este nenul";	-	transfer date de ieşire

0	1	2
e. *citeşte* valoarea variabilei a1;	-	-
f. *atribuie* valoarea variabilei a1 variabilei a;	-	-
g. *atribuie* rezultatul evaluării expresiei $b^2 - 4 \cdot a \cdot c$ variabilei delta;		-
h. *verifică* dacă delta este pozitiv: (delta ≤ 0)?;	adevărat	continuă cu **i**
	fals	salt la **l**
i.*atribuie* rezultatul evaluării expresiei $(-b + \sqrt{delta})/(2 \cdot a)$ variabilei x1;	-	-
j.*atribuie* rezultatul evaluării expresiei $(-b - \sqrt{delta})/(2 \cdot a)$ variabilei x2;	-	-
k. *scrie* x1, x2	-	transfer date de ieşire
l. *scrie "Ecuaţia nu are radacini reale"*;	-	transfer date de ieşire
m. sfârşit	-	-

Descrierea algoritmului folosind schema logică este arătată în Figura 4.3.

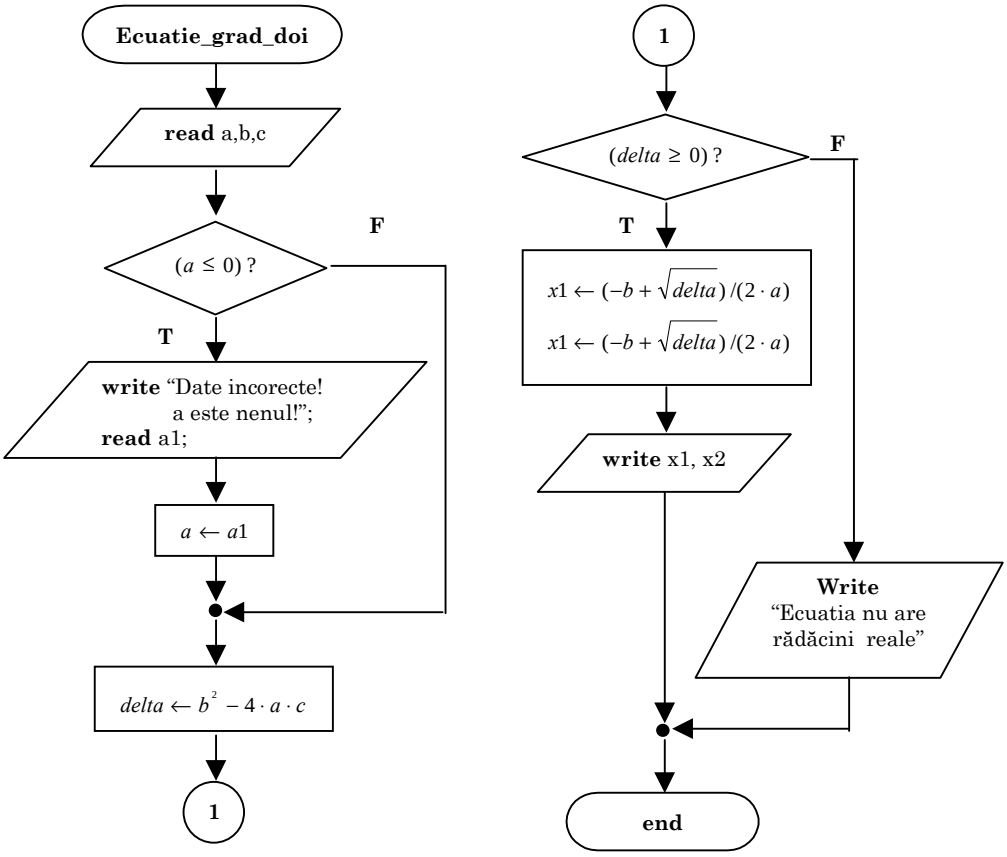


Figura 4.3. Schema logică a algoritmului de calcul a soluțiilor reale ale ecuației de gradul al doilea

LUCRAREA 5

PROGRAMAREA STRUCTURATĂ. INSTRUCȚIUNI FUNDAMENTALE

1. NOȚIUNI INTRODUCTIVE

Programarea structurată reprezintă o manieră de concepere a programelor respectând reguli bine stabilite. Scopul programării structurate este elaborarea unor programe uşor de scris, de depanat şi modificat (actualizat) atunci când se impune acest lucru. Programele obținute sunt clare, ordonate, inteligibile, fără salturi şi reveniri. Programarea structurată permite ca programele să poată fi scrise în *limbaj pseudocod*, limbaj independent de maşină (sistem de calcul), convertibil în orice limbaj de programare. Programarea structurată are ca fundament *teorema de structură*, care se aplică pentru programele (algoritmii) cu un singur punct de început (o singură intrare) şi un singur punct de sfârşit (o singură ieşire). În programarea structurată se utilizează trei instrucțiuni simple (read, write, ←) şi trei instrucțiuni compuse (secvența, selecția, iterația). Instrucțiunile compuse sunt exemple de *structuri de control*: (1) structura secvențială (secvența), (2) structura alternativă (selecția, decizia), (3) structura repetitivă (iterația).

2. INSTRUCȚIUNI FUNDAMENTALE

2.1. INSTRUCȚIUNI SIMPLE

Cele mai simple instrucțiuni de prelucrare ale unui algoritm descris într-un limbaj de programare sunt: instrucțiunea de citire, instrucțiunea de scriere şi instrucțiunea de atribuire.

Instrucțiunea de citire are următoarea formă scrisă în limbaj pseudocod:

read *vname* [,*vname*] ...

în care *read* înseamnă *citeşte*, iar *vname* reprezintă un *nume de variabilă*.

Modul de execuție este următorul: din următoarea înregistrare a fişierului de intrare procesorul citeşte valori pe care le stochează în memorie în celulele cu numele din lista de intrare; ordinea de stocare este ordinea de la stânga la dreapta a variabilelor din lista de intrare.

Instrucțiunea de scriere are următoarea formă scrisă în limbaj pseudocod:

write *entitate*[,*entitate*]...

în care *write* înseamnă *scrie*, iar *entitate* un *nume de variabila* sau o *expresie*.

Modul de execuţie este următorul: în fişierul de ieşire procesorul crează o nouă înregistrare cu valorile entităţilor din listă în ordinea de la stânga la dreapta.

Prin *expresie* se înţelege o serie de prelucrări, indicate prin *operatori*, efectuate asupra unor date, care au rol de *operanzi*.

Instrucţiunea de atribuire are următoarea formă scrisă în limbaj pseudocod :

vname ⟵ *expr*

în care *vname* este un *nume de variabilă*, *expr* este o *expresie*, iar ⟵ reprezintă semnul de *atribuire*.

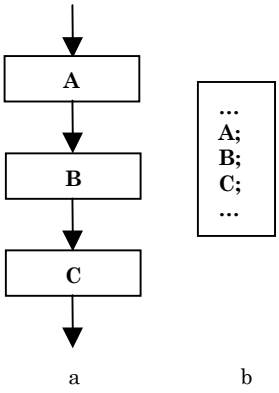
Dacă *expr* este o expresie numerică, *vname* este o variabilă numerică, dacă este logică, *vname* este o variabilă logică, iar daca este o expresie caracter, *vname* este o variabilă caracter.

Instrucţiunea se execută astfel: cu datele existente în memorie procesorul evaluează *expr* şi apoi rezultatul este stocat în memorie la adresa variabilei *vname*.

2.2. INSTRUCŢIUNI COMPUSE

2.2.1. Secvenţa

Secvenţa este o structură formată din două sau mai multe părţi ce se execută fiecare o singură dată. Prin *parte* înţelegem fie o instrucţiune simplă, fie o instrucţiune compusă. Este important de subliniat că secvenţa are o singură intrare şi o singura ieşire. Schema logica a unei secvenţe şi descrierea în pseudocod sunt aratate în Figura 5.1.




```
…
A;
B;
C;
…
```

b

Figura 5.1. Schema logică şi descrierea pseudocod a unei secvenţe.

Modul de execuţie este următorul:

- se execută *A*;
- se execută *B*;
- se execută *C*;

şi secvenţa ia sfârşit.

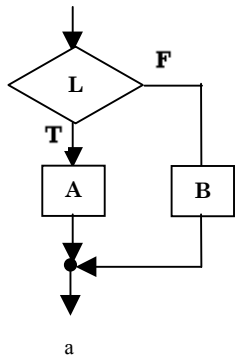
Dacă *A, B, C* sunt instrucţiuni simple, în pseudocod secvenţa poate fi cea scrisă ca mai jos:

Exemplu:

```
…
read term1, term2;
suma ⟵ term1+term2;
write suma;
…
```

2.2.2. Selecția

Selecția este o structură de control cu *două părți A* şi *B*, din care se execută numai una în funcție de rezultatul unui *test logic L*. Această structură compusă are o singură intrare şi o singură ieşire. În Figura 5.2 este reprezentată schema logică şi descrierea în pseudocod a unei selecții.



a

```
if L then
   A;
else
   B;
endif
```

b

Figura 6.2. Schema logică şi descrierea pseudocod a unei selecții.

Modul de execuție a selecției este următorul: cu datele din memorie se evaluează expresia logică *L*; dacă rezultatul este *TRUE* se execută *A* şi selecția ia sfârşit; dacă rezultatul este *FALSE* se execută partea *B* şi selecția ia sfârşit.

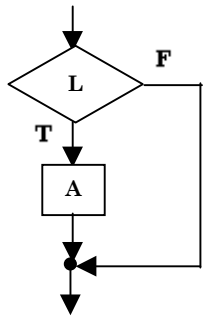
Dacă *A* şi *B* sunt instrucțiuni simple, în pseudocod selecția poate fi cea arătată mai jos:

Exemplu:

```
...
if ( a > 0 ) then
    suma = a + b;
  else
    suma = a – b;
endif
...
```

Alături de forma arătată mai sus, se utilizează şi două forme derivate: (1) selecția simplă (*if … then*), (2) selecția multiplă (*case*).

Selecția simplă este selecția care are numai o parte *A*, care se execută atunci când rezultatul testului logic *L* are o anumită valoare. Pentru cealaltă valoare a testului logic se execută următoarea instrucțiune din program (Figura 5.3). *Selecția multiplă* permite alegerea unei părți din mai multe posibile (Figura 5.4), folosind pentru selecție un *index*. Dintre toate părțile se va executa aceea pentru care valoarea expresiei corespunzătoare este identică cu valoarea indexului.




```
if L then
   A;
endif
```

Figura 5.3. Selecția simplă.

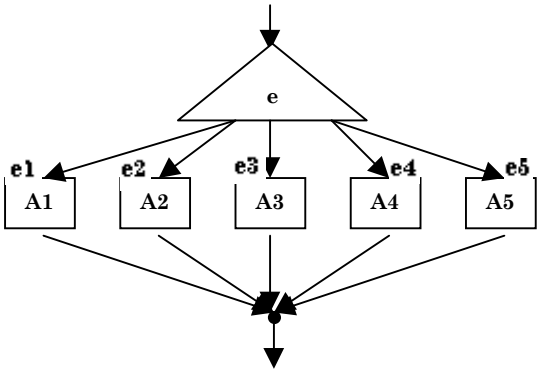


Figura 5.4. Selecția multiplă.

2.2.3. Iterația

Iterația este o structură cu o singură intrare şi o singură ieşire formată dintr-o singură *parte* (*A*) ce se execută în funcție de rezultatul unui *test logic L*. Există două forme de iterație, iterație cu testul la început, numită *do while* (Figura 5.5) şi iterație cu testul la sfârşit, numită *repeat until* (Figura 5.6).

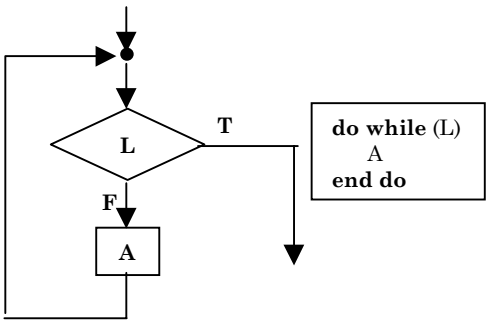


Figura 5.5. Schema logică şi scrierea în pseudocod a iterației *do while*.

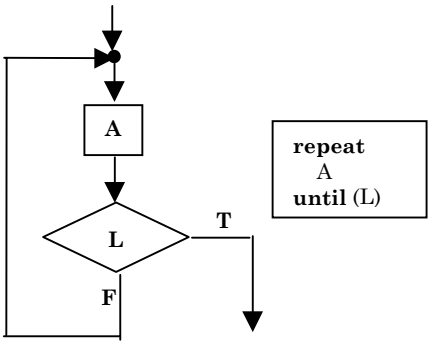


Figura 5.6. Schema logică şi scrierea în pseudocod a iterației *repeat until*.

Modul de execuție a iterației *do while* este următorul: cu datele din memorie se evaluează expresia *L*; dacă rezultatul este *TRUE* se execută *A*, se evaluează *L*, dacă *L* este *TRUE* se execută *A* ş.a.m.d., până când rezultatul devine *FALSE* şi atunci iterația ia sfârşit. Dacă la prima evaluare a expresiei logice rezultatul este *FALSE* iterația ia sfârşit, iar partea *A* nu se execută niciodată. Pentru ca iterația să se termine într-un număr finit de paşi este necesar ca partea *A* să modifice valoarea unei variabile ce apare şi în testul *L*, astfel incât să fie posibil ca după un număr finit de paşi *L* să capete valoarea *FALSE*. Daca această condiție nu se indeplineşte avem o buclă infinită.

Modul de execuție a iterației *repeat until* este următorul: se execută *A*, se evaluează expresia logică *L*, dacă rezultatul este *FALSE* se execută *A*, se evaluează *L* ş.a.m.d. până ce rezultatul devine *TRUE*, când iterația ia sfârşit. Partea iterată se execută în acest caz cel puțin odată. Evident şi buclele (iterațiile) *repeat until* pot fi infinite, dacă la execuția ei, după un număr finit de paşi, expresia logică *L* nu poate căpăta valoarea *TRUE*. În continuare sunt prezentate exemple de astfel de instrucțiuni compuse.

Exemplu:

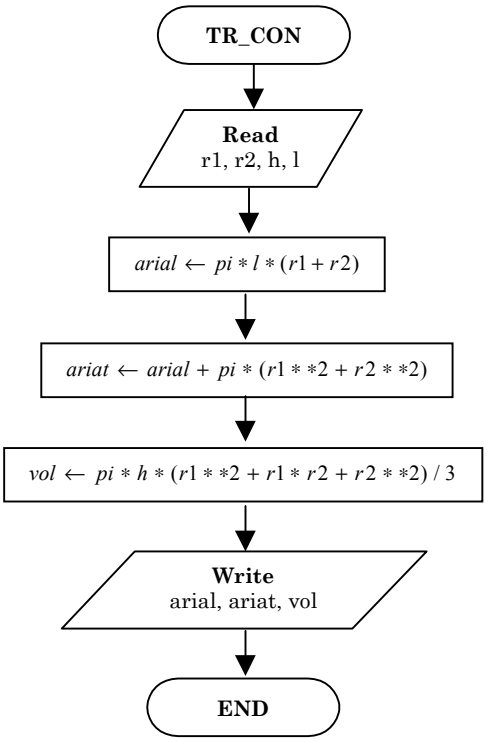
```
...
i ← vinitial;
do while ( i ≤ 100 )
    s1 ← s1 + x ;
    s2 ← s2 + x * x ;
    i ← i + p;
enddo
...
```

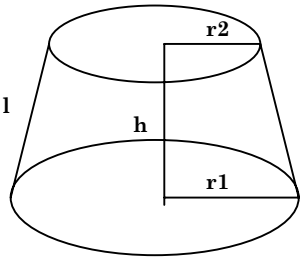
Exemplu:

```
...
i ← 1;
repeat
    s1 ← s1 + x * *2 ;
    s2 ← s2 + x + y ;
    i ← i + 1;
until ( i ≤ n )
...
```

3. ALGORITMUL DE CALCUL AL ARIEI LATERALE, ARIEI TOTALE ŞI VOLUMULUI UNUI TRUNCHI DE CON

Schema logică şi programul în pseudocod sunt prezentate în Figura 5.7.





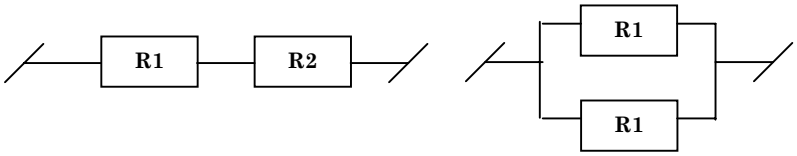

```
 program tr_con
! declararea variabilelor din program
  real r1, r2, h, l, arial, ariat, vol, pi;
! citirea datelor de intrare
  read r1, r2, h, l;
! calculul ariei laterale, ariei totale,
! volumului
  arial ⟵ pi*l*(r1+r2);
  ariat ⟵ arial+pi*(r1**2*r2**2);
  vol ⟵ pi*h*(r1**2+r1*r2+r2**2)/3;
! scrierea datelor de ieşire
  write arial, ariat, vol;
  end
```

a b

Figura 5.7. Algortimul de calcul al ariei laterale, ariei totale, volumului unui trunchi de con: a-schema logică; b-program în pseudocod.

4. ALGORITMUL DE CALCUL AL REZISTENȚEI ECHIVALENTE A UNUI CIRCUIT

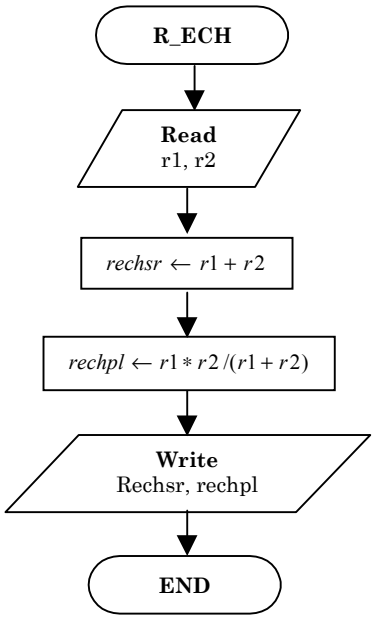
Să se realizeze un algoritm pentru calculul rezistenței echivalente a unui circuit format din două rezistoare legate fie în serie, fie în paralel (Figura 5.8).



a b

Figura 5.8. Circuit cu rezistențe: a-legate în serie; b-legate în paralel.

Schema logică şi programul în pseudocod sunt prezentate în Figura 5.9.



a

$$R_{serie} = R_1 + R_2$$

$$R_{paralel} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}}$$

```
  program r_ech
! declararea variabilelor
  real r1, r2, rechsr, rechpl;
! citirea datelor de intrare
  read r1, r2;
! calculul rezistenței echivalente
  rechsr ← r1+r2;
  rechpl ← r1*r2/(r1+r2);
! scrierea datelor de ieşire
  write rechsr, rechpl;
  end
```

b

Figura 5.9. Schema logică şi programul în pseudocod pentru calculul rezistenței echivalente.

5. ALGORTIMUL DE CALCUL AL UNEI SUME

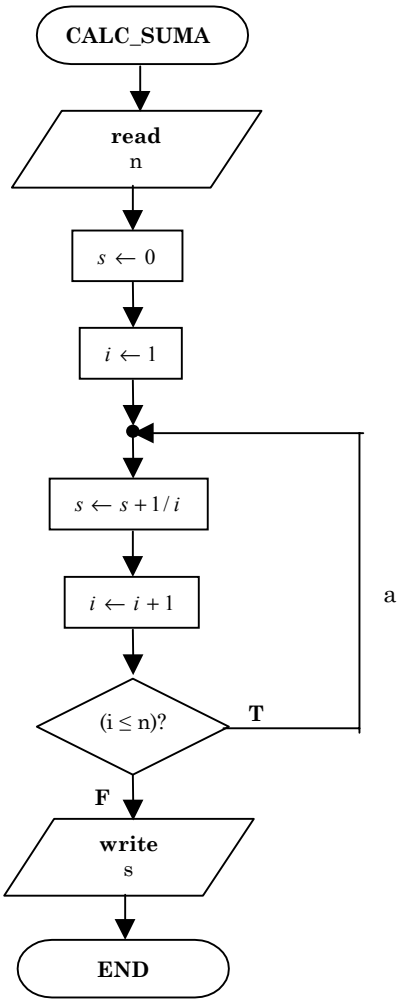
Să se descrie un algoritm pentru calculul sumei $\sum_{i=1}^{n} \frac{1}{i}$.

Pentru acest program singura dată de intrare este variabila de tip întreg *n*. Suma celor *n* termeni *1/i* este singura dată de ieşire, variabilă de tip real. Variabila *i* este o variabilă intermediară cu care se calculează fiecare termen al sumei şi se asigură încheierea calculului sumei într-un număr finit de paşi. Schema logică şi programul în pseudocod sunt prezentate în Figura 5.10.

6. ALGORITMUL PENTRU SORTAREA A DOUĂ NUMERE ÎN ORDINE CRESCĂTOARE

Fie două numere întregi *m* şi *n*. Să se scrie un algoritm pentru sortarea în ordine crescătoare a celor două numere.

Programul are două date de intrare, variabilele de tip întreg *m* şi *n*, reprezentate de cele două numere, două date de ieşire *nrmin* şi *nrmax*, varabile de tip întreg (Figura 5.11).

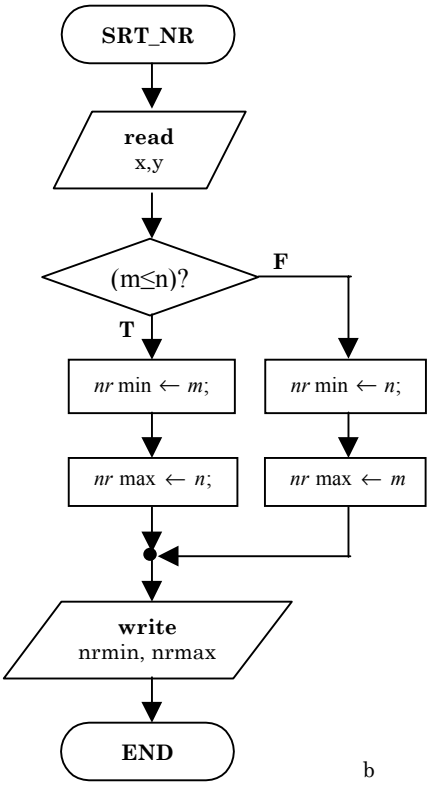


a

```
  program calc_suma
! declararea variabilelor
  integer n, i;
  real suma;
! citirea datelor de intrare
  read n;
! inițializarea variabilelor
  s ⟵ 0;
  i ⟵ 1;
! iterație condiționată posterior
! pentru calculul sumei
  repeat
     s ⟵ s + 1/ i;
!    instrucțiune de atribuire
!    necesară trecerii la următorul
!    termen
     i ⟵ i + 1;
  until ( i ≤ n )
! scrierea datelor de iesire
```

b

Figura 5.10. Schema logică şi programul în pseudocod pentrul calculul unei sume.




```
  program srt_nr
! declararea variabilelor
  integer n, m, nrmin, nrmax;
! citirea datelor de intrare
  read m, n;
! sortarea în ordine crescătoare
  if (m ≤ n) then
       nrmin ⟵ m;
       nrmax ⟵ n;
     else
       nrmin ⟵ n;
       nrmax ⟵ m;
  endif
  write nrmin, nrmax;
  end
```

a

b

Figura 5.11. Descrierea în pseudocod şi schema logică a algoritmului de sortare a două numere.

LUCRAREA 6

ETAPELE REALIZĂRII UNUI PROGRAM. PRIMUL PROGRAM FORTRAN

1. NOȚIUNI INTRODUCTIVE

Limbajele de programare reprezintă unul din mijloacele principale de comunicație om-maşină. Prin *limbaj de programare* se înțelege, în general, orice limbaj folosit pentru descrierea algoritmilor şi a structurilor de date. Elementul constitutiv al oricărui limbaj de programare este *instrucțiunea*, care reprezintă exprimarea într-o formă riguroasă a unei cereri de utilizare a unei resurse a unui sistem de calcul. Fiecare instrucțiune precizează tipul operației, locația de stocare a operanzilor şi locația pentru stocarea rezultatelor. O *succesiune de instrucțiuni*, care definesc în mod univoc un algoritm de rezolvare a unei probleme, constituie un *program*.

Primele calculatoare executau programe scrise în *cod maşină*, adică instrucțiunile erau scrise în formă numerică. Pentru reprezentări externe calculatorului, instrucțiunile se scriau folosind şiruri de cifre binare, octale sau hexazecimale. În acelaşi mod erau reprezentate şi datele ce erau prelucrate de aceste programe. Folosirea acestui mod de scriere a programelor a avut foarte multe neajunsuri, în special legate de propagarea erorilor, utilizare greoaie, documentare, modificare şi întreținere dificile, inexistența *portabilității* programelor (posibilitatea execuției aceluiaşi program pe sisteme de calcul diferite). Ca urmare, au apărut *limbajele de asamblare*, care deşi au rămas în continuare dependente de componența hardware, au permis apariția primelor elemente specifice programării simbolice (coduri pentru instrucțiuni, adresarea simbolică, utilizarea microinstrucțiunilor, acces la biblioteci de subprograme etc).

Dificultățile impuse de utilizarea limbajelor maşină sau a limbajelor de asamblare în dezvoltarea programelor, au determinat apariția în anii '50 ai secolului trecut a unor limbaje apropiate de limbajul natural, dar care respectau precizia şi modul direct de adresare a limbajului maşină. Aceste limbaje au căpătat denumirea de limbaje de nivel înalt. *Limbajele de nivel înalt* sunt limbajele de programare pentru care tipurile de date, operațiile de prelucrare şi control şi, în consecință programele nu sunt dependente de sistemul de calcul pe care se execută, de modul de reprezentare internă a datelor şi a operațiilor specifice acestuia Avantajele utilizării limbajelor de nivel înalt în comparație cu cele anterioare sunt: (1) naturalețe (apropiere de limbajul natural şi matematic), (2) uşurința de înțelegere şi utilizare, (3) portabilitate, (4) eficiență în scriere.

Creşterea numărului de limbaje de programare a determinat clasificarea acestora după diverse criterii. O astfel de clasificare împarte limbajele de programare în clase: (1) *limbaje procedurale* sau *algoritmice*, care fac parte din categoria limbajelor de

nivel înalt (*Fortran, Algol, Pascal, Basic* etc), (2) *limbaje neprocedurale* sau limbaje de nivel foarte înalt (succesiunea instrucțiunilor scrise în program influențează în mică măsură succesiunea executării lor), (3) *limbaje specializate* sau *orientate pe problemă*, (4) *limbaje conversaționale*, care asigură dialogul utilizator-calculator pe parcursul execuției programului), (5) *limbaje ale inteligenței artificiale*.

Specificarea oricărui limbaj presupune definirea sintaxei, semanticii şi pragmaticii limbajului. *Sintaxa* (studiul structurii limbajului) reprezintă un set de reguli ce guvernează alcătuirea mulțimii propozițiilor din limbaj. Definirea riguroasă a sintaxei impune folosirea unor elemente de teoria limbajelor formale: *vocabular, propoziție, frază, gramatică* etc. Utilizarea efectivă a acestor elemente presupune folosirea unui limbaj special destinat exprimării regulilor gramaticale, numit *metalimbaj*. Cel mai cunoscut este *metalimbajul BNF* (*Backus Naur Form*). *Semantica* este reprezentată de un set de reguli ce determină sensul, semnificația propozițiilor într-un limbaj. Dacă în cazul sintaxei principala problemă este cea a specificării mulțimii propozițiilor corecte, în cazul definirii semanticii problema principală este cea a specificării semnificației propozițiilor corecte. *Pragmatica* este o mulțime de reguli ce descriu formarea efectului propozițiilor limbajului pentru un anumit receptor. În cazul limbajelor de programare pragmatica are legătură cu studiul eficienței programelor, independenței față de maşină etc.

Primul limbaj de programare de nivel înalt standardizat a apărut în anul 1954 şi s-a numit *FORTRAN* (*FORmula TRANslation*). Limbajul Fortran este un limbaj procedural, conversațional, orientat către rezolvarea problemelor din domeniile tehnico-inginereşti.

2. ELEMENTELE DE BAZĂ ALE LIMBAJULUI FORTRAN

Elementele de bază ale limbajului Fortran sunt:

- *Setul de caractere*: literele mari şi mici ale alfabetului limbii engleze, cifrele arabe (sistemului de numerație zecimal) şi caractere speciale; cu acest set de caractere, respectând sintaxa specifică limbajului, se pot forma entități cum ar fi: cuvinte cheie, nume simbolice (identificatori), etichete, constante, expresii, instrucțiuni etc;
- *Cuvintele cheie* sunt cuvinte standardizate folosite în formarea şi definirea instrucțiunilor (*dimension, real, integer, data, read, write, print, do, enddo, if, then, else, endif* etc);
- *Nume simbolice*, şiruri de caractere atribuite de programator pentru unități de program şi variabile;
- *Etichetele* sunt numere de maximum cinci cifre ce desemnează o instrucțiune în cadrul programului cu scopul de a putea face referință la ea;

- *Constantele* sunt date ce nu îşi schimbă valoarea în timpul execuției programului;
- *Expresiile* sunt formate din operanzi (date de prelucrat) şi operatori (operații de prelucrare).

Programele scrise în limbaj Fortran sunt formate din construcții sintactice numite *instrucțiuni* şi opțional *comentarii*. Instrucțiunile descriu datele prelucrate de program şi operațiile algoritmului de rezolvare a problemei.Instrucțiunile pot fi *executabile* şi *neexecutabile*. Cele executabile descriu operațiile ce trebuie executate (citire, scriere, atribuire, control, operații intrare/ieşire), iar cele neexecutabile descriu datele utilizate şi caracteristicile acestora, tipul unităților de program etc.

3. ETAPELE REALIZĂRII UNUI PROGRAM

Procesul dezvoltării unui program executabil necesită câteva etape:

- *Editarea fişierului sursă* scris într-un limbaj de programare;
- *Compilarea* fişierului sursă, ce are ca rezultat obținerea *fişierului obiect*, adică programul în limbaj binar, specific calculatorului;
- *Generarea programului executabil*, sau altfel spus combinarea dintre fişierului obiect, obținut anterior, cu fişiere obiect din bibliotecile calculatorului;
- *Lansarea* în execuție a programului executabil, ce va avea ca efect prelucrarea datelor de intrare, în urma căreia se vor obține datele de iesire.

Editarea fişierului sursă se face utilizând un editor de texte. *Numele fişierului sursă* va fi urmat, obligatoriu, de extensia FOR. Precizarea extensiei este necesară prelucrării ulterioare, pentru a obține fişierul obiect. Textul programului este divizat în unități fizice numite *linii* ale fişierului sursă. Pentru fişierele sursă scrise în limbajul de programare FORTRAN în *formă fixă*, liniile au o lungime de 80 coloane. În coloanele 1-5 se scriu *etichetele* instrucțiunilor. Pentru scrierea instrucțiunilor se folosesc coloanele 7-72. Dacă în coloana 6 este scris orice caracter FORTRAN cu excepția caracterului *blanc* (spațiu liber) sau 0, atunci acea linie este o continuare a liniei precedente. Pe o linie se găseşte o singură instrucțiune, ce poate fi continuată pe mai multe linii. O instrucțiune nu poate avea mai mult de 20 de linii. Coloanele 73-80 sunt ignorate de către compilator. Caracterul *C* sau * în coloana 1 indică faptul că întreaga linie este un comentariu. De asemenea, caracterul *!*, dacă nu face parte dintr-o constantă caracter, marchează începutul unui comentariu, ce se termină cu sfârşitul liniei. Comentariile sunt ignorate în procesul ulterior de prelucrare, având doar scopul de a face programul mai inteligibil.

Compilarea fişierului sursă este operaţia de prelucrare a acestuia prin care programul este rescris (*tradus*) în limbaj cod maşină. Traducerea fişierului sursă în limbaj cod maşină se realizează cu ajutorul unui program complex numit *compilator*. Acest program are ca intrare fişierul sursă FORTRAN, iar ca ieşire *fişierul obiect*.

Generarea programului executabil se face prin intermediul unui program numit *editor de legături* (*linker*). Editorul de legături prelucrează fişierul obiect în scopul construirii programului executabil, pentru ca acesta să poată fi executat pe calculatorul respectiv.

După realizarea programului executabil, acesta poate fi *lansat în execuţie* (activat) ca orice alt nume prin numele său.

Editarea, compilarea şi editarea legăturilor necesită utilizarea unor *programe de servicii* dedicate unor astfel de lucrări. În prezent există pachete de programe integrate pentru dezvoltarea de programe. Unul dintre aceste pachete de programe este şi *Developer Studio*.

Acest produs software, care este un mediu de dezvoltare a programelor, lucrează sub sistemul de operare Windows şi foloseşte, evident, o interfaţă grafică cu programatorul. Lansarea în execuţie a acestui mediu de programare se poate face fie prin dublu click cu mouse-ul pe pictograma corepunzătoare din fereastra Windows, fie prin apăsarea butonului *Start* din linia de stare a ferestrei Windows, deschiderea meniului *Programs*, alegerea submeniului *Visual Fortran*, deschiderea acestuia şi alegerea comenzii *Developer Studio*.

Interfaţa grafică a produsului Developer Studio cuprinde următoarele elemente:

- Linia meniului principal;
- Pictogramele corepunzătoare unor comenzi existente în submeniurile meniului principal;
- Fereastră corespunzătoare spaţiului de lucru (*Workspace),*în care se regăsesc informaţii privind fişierele asociate cu un anumit program; spaţiul de lucru se deschide în mod automat la compilarea fişierului sursă; la întrebările sistemului se apasă butonul *Yes*;
- Fereastră corespunzătoare fişierului text, ce va stoca programul FORTRAN propriu-zis; un fişier text se crează fie prin alegerea comenzii *New…* din submeniul *File*, fie prin realizarea unui click cu mouse-ul pe pictograma corespunzătoare; fişierele text ce conţin programe scrise în limbaj FORTRAN trebuie salvate pe discul fix având în mod necesar extensia *FOR*, altfel nu vor fi compilate. Pentru a realiza acest lucru se alege şi lansează comanda *Save As…* (fie din submeniul *File*, fie prin realizarea unui click pe pictograma corespunzătoare); prin activarea comenzii se deschide o fereastră de dialog ce ne permite să alegem folderul (subdirectorul) unde stocăm fişierul, numele şi extensia acestuia, după care

se apasă butonul *Save*; la redactarea textului programului se va ține cont de regulile de scriere a fişierelor sursă FORTRAN arătate mai sus;

- ❖ Fereastră corespunzătoare afişării rezultatelor obținute în urma operației de compilare, respectiv editare a legăturilor (generării fişierului executabil);
- ❖ Meniul *Build* conține comenzile necesare operației de compilare si generare a fişierului executabil; acestea sunt *Compile* şi *Build name.exe*; aceste comenzi au şi pictograme corespunzătoare; lansarea în execuție se face cu comanda *Execute* (sau pictograma corespunzătoare).

Dacă compilarea fişierului sursă s-a efectuat cu succes (sistemul nu indică erori de sintaxă) se crează fişierul obiect ce are acelaşi nume cu fişierul sursă, dar cu extensia *OBJ*. Fişierul obiect se obține doar în cazul în care toate instrucțiunile fişierului sursă au respectat întocmai regulile sintactice ale limbajului de programare. Dacă în fişierul sursă există erori de sintaxă, în urma compilării este generat un fişier text ce cuprinde o listă a erorilor de sintaxă şi informații minime asupra tipului şi locului acestora în program. După corectarea lor, prin modificarea textului fişierului sursă, acesta din urmă se compilează din nou. Procesul se repetă până nu mai există mesaje despre erorile de sintaxă.

După compilare se lansează comanda de generare a fişierului executabil. Realizarea fişierului executabil ne permite executarea programului.

4. PRIMUL PROGRAM FORTRAN

4.1. Declararea tipului variabilelor scalare

O variabilă scalară este un obiect notat cu un *nume* (*vname*). Pentru a specifica tipul şi unele atribute ale variabilelor ce descriu modul de prelucrare a acestora în program se folosesc *instrucțiune de declarare a tipului*, care sunt instrucțiuni neexecutabile.

Limbajul Fortran utilizează cinci tipuri de date scalare: *intreg, real, complex, logic, caracter*. Pentru fiecare tip există un cuvânt cheie: *integer, real, complex, logical, character*.

Sintaxa formală a unei instrucțiuni de declarare a tipului variabilelor scalare este:

spec_tip [[, *atrib*]…::] *lista*

în care **spec_tip** este unul din cuvintele cheie de mai sus, care indică tipul variabilei scalare, *atrib* este un atribut, iar *lista* este o listă de nume de variabile scalare separate

prin separatorul *virgulă*.

4.2. Instrucțiuni de citire, scriere, atribuire

Citirea şi scrierea cu formatarea condusă de listă. În limbajul de programare FORTRAN există posibilitatea să se asigure conversia valorilor din forma externă de codificare, specifică limbajului, în forma de codificare specifică memoriei internă şi invers, fără a furniza în mod explicit modul de afişare sau imprimare descris într-o specificație de format. Acest lucru este posibil prin editarea condusă de lista, numită şi *formatare implicită* sau *formatare condusă de listă*.

Instrucțiunea de citire cu formatare condusă de listă are sintaxa:

read*, [*vname*[,*vname*]...]

Asteriscul (*) înseamnă ca formatul este lăsat la alegerea celui ce pregăteşte fişierul de intrare. Modul de execuție este următorul: din următoarea înregistrare a fişierului de intrare creat de la tastatură şi afişat pe ecranul monitorului, procesorul citeşte valori pe care le stochează în memorie la adresele de memorie ce corespund variabilelor din lista de intrare. Este necesar ca datele existente în înregistrare să fie de acelaşi tip cu variabilele din lista de intrare.

Există următoarele reguli şi restricții: (1) înregistrarea este formată din valori şi separatori de valori, (2) dacă lista nu are nici un element, se sare o înregistrare din fişierul de intrare.

Valorile permise sunt:

- **zero** - o valoare nulă ce se scrie astfel **,** **,** (între separatori nu este scrisă o valoare);
- **c** - constantă;
- **r*c** - constanta c este repetată de r ori;
- **r*** - valoare nulă repetată de r ori.

Separatorii permişi sunt:

- **,** - o virgulă, opțional precedată sau urmată de blancuri unul după altul;
- **/** - un slash, opțional precedat sau urmat de blancuri unul după altul;
- **" "** - un blanc între două valori non blanc.

Înregistrarea se crează de către programator de la tastatură şi apare pe ecranul monitorului.

Instrucțiunea de scriere cu formatare condusă de listă are sintaxa:

print*, [*entitate*[,*entitate*]...]

Asteriscul (*) în instrucțiunea PRINT înseamnă formatare implicită, adică formatul datelor de ieşire este lăsat pe seama computerului. Modul de execuție este următorul: în fişierul de ieşire, ecranul terminalului, se crează o înregistrare, adică se scrie o nouă linie, linie ce conține în ordinea de la stânga la dreapta valorile entităților din lista de ieşire. Dacă lista nu are nici un element în fişierul de ieşire se scrie o înregistrare vidă, ce are doar blancuri.

Instrucțiunea de atribuire scalară. Instrucțiunii de atribuire simple din programarea structurată, în Fortran, îi corespunde o instrucțiune executabilă numită *instrucțiune de atribuire scalară* ce are următoarea sintaxă:

var = expr

în care *var* este un nume de variabilă, iar *expr* o expresie. Regulile de sintaxă impun ca între tipul variabilei şi tipul expresiei să fie acelaşi. În funcție de tipul datelor scalare avem instrucțiune de atribuire scalară numerică, logică sau caracter.

4.3. Program Fortran pentru calculul ariei laterale, ariei totale şi volumului unui trunchi de con drept

În continuare este arătat programul scris în limbaj Fortran, în formă fixă, pentru algoritmul de calcul al ariei laterale, ariei totale şi volumului unui trunchi de con drept. Schema logică şi programul în pseudocod pentru acest algoritm sunt prezentate în *Lucrarea 5*.

1÷5	6	7÷72	73÷80
		program tr_con	
c		declararea variabilelor din program	
		implicit none	
		real r1, r2, h, l, arial, ariat, vol, pi	
c		citirea datelor de intrare	
		read*, r1, r2, h, l	
c		calculul ariei laterale, ariei totale, volumului	
		pi = 3.14	
		arial = pi*l*(r1 + r2)	
		ariat = arial + pi*(r1**2 + r2**2)	
		vol = pi*h*(r1**2 + r1*r2 + r2**2)/3.	
c		scrierea datelor de ieşire	
		print*, arial, ariat, vol	
		end	

LUCRAREA 7

CONSTRUCȚIA IF. PROGRAMUL SEMN_NR

1. CONSTRUCȚIA IF

O constructie IF selectează pentru execuție cel mult un bloc de instrucțiuni şi construcții. Sintaxa instrucțiunii este:

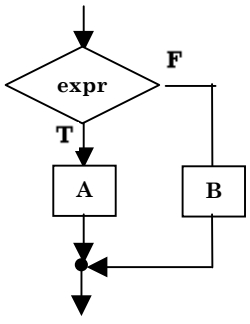
```
if (expr1) then
    bloc1
  [else if (expr2) then
        bloc2]...
      [else
        bloc3]
endif
```

Aici *expr1*, *expr2* sunt expresii logice scalare, *bloc1*, *bloc2*, *bloc3* reprezintă fiecare un *bloc de instrucțiuni*. Trebuie luate in considerație următoarele reguli şi restricții:

- ❖ se execută cel mult unul din blocurile construcției; este posibil ca nici un bloc să nu se execute;
- ❖ dupa instrucțiunea ELSE nu sunt permise instrucțiuni ELSE IF;
- ❖ sunt interzise salturile la o instrucțiune ELSE IF sau la o instrucțiune ELSE;
- ❖ in interiorul unei construcții IF este permis saltul la END IF din oricare din blocurile construcției IF; saltul dinafară la END IF este permis, dar este considerat o caracteristică depăşită.

Construcția IF se execută astfel: expresiile logice sunt evaluate în ordine până ce se gaseşte una adevarată; atunci se execută primul bloc ce urmează după prima expresie adevarată şi execuția *construcției IF* se termină. Expresiile logice adevărate care urmează nu mai au efect. Se poate întâmpla ca nici una din expresiile logice ale construcției să nu fie adevărată. În acest caz se execută blocul ce urmează după instrucțiunea ELSE dacă există unul; în caz contrar nu se execută nici unul din blocurile construcției.

Cazul cel mai simplu al *construcției IF* este selecția cu schema logică arătată în Figura 7.1, care în FORTRAN se poate codifica ca mai jos.




```
if (expr) then
   A;
 else
   B;
endif
```

Figura 7.1. Construcția IF în Fortran.

Cazul unei *construcții IF* ce conține o instrucțiune *ELSE IF* este în esență o selecție ce conține o alta selecție (una din părți este o alta selecție), schema logică fiind arătată în Figura 7.2.

Este posibilă şi o altă codificare în FORTRAN pentru construcția IF anterioară, în care unul din blocuri să fie o construcție IF. Codificarea în FORTRAN pentru cele două variante este prezentată mai jos.

```
Varianta 1
if (expr1) then
      a
   else if (expr2) then
         b
       else
         c
end if


Varianta 2
if (expr1) then
      a
    else
      if (expr2) then
            b
         else
            c
       end if
end if
```

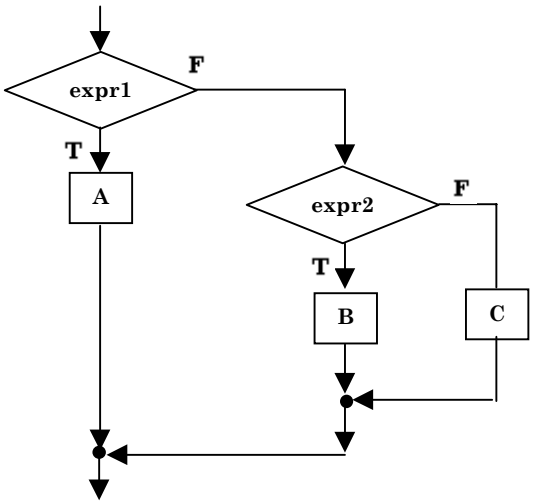


Figura 7.2. Construcția IF…ELSE IF.

Cazul unei construcții IF cu doua instrucțiuni ELSE IF corespunde schemei logice din Figura 7.3, unde A, B, C, D sunt blocuri de instrucțiuni sau construcții. Codificarea în FORTRAN este următoarea:

```
if (expr1) then
      a
   else if (expr2) then
            b
        else if (expr3) then
            c
          else
            d
end if
```

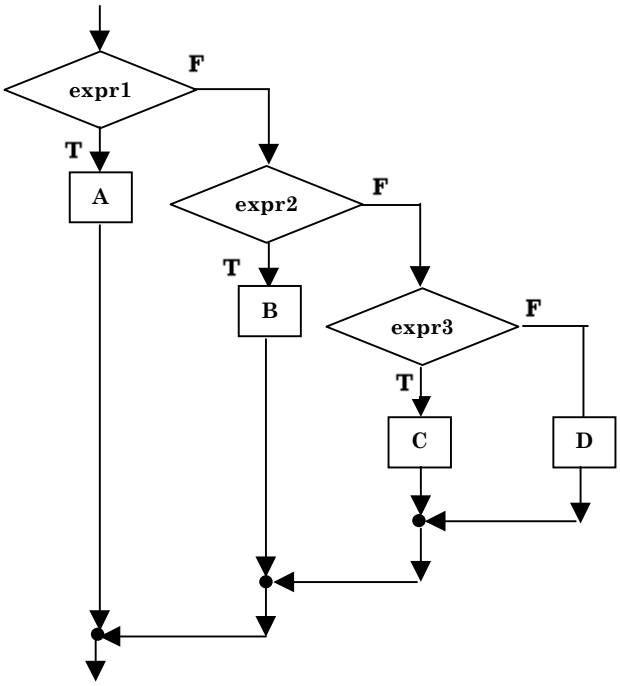


Figura 7.3. Schema logică a construcției If…ELSEIF…ELSEIF

2. PROGRAMUL SEMN_NR

Schema logică şi programul în pseudocod a algoritmului pentru determinarea semnului unui număr întreg sunt prezentate în Figura 7.4.

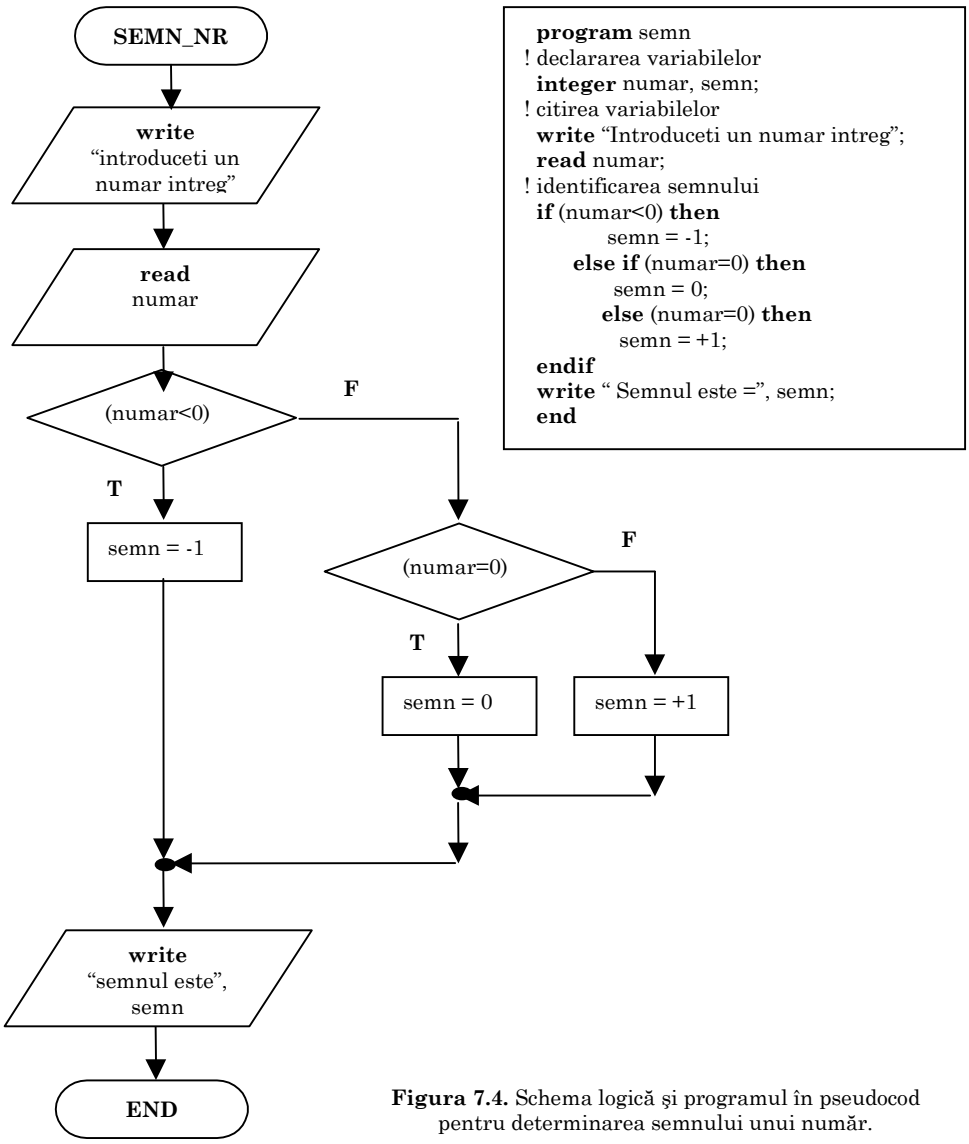


Figura 7.4. Schema logică şi programul în pseudocod pentru determinarea semnului unui număr.

Programul Fortran ce descrie acest algoritm este prezentat în continuare.

1÷5	6	7÷72	73÷80
		program semn_nr	
c		declararea variabilelor din program	
		implicit none	
		integer numar, semn	
c		crearea unui dialog	
		print*, " Intoduceti un numar intreg"	
c		citirea datelor de intrare	
		read*, numar	
c		alegerea semnului numarului	
		if (numar.lt.0.) **then**	
		semn = -1	
		elseif (numar.eq.0) **then**	
		semn = 0	
		else	
		semn = +1	
		endif	
c		scrierea datelor de ieşire	
		print*, " semnul este ", semn	
		end	

O variantă a acestui program este următoarea:

1÷5	6	7÷72	73÷80
		program semn_nr	
c		declararea variabilelor din program	
		implicit none	
		integer numar	
		character*10 semn	
c		crearea unui dialog	
		print*, " Intoduceti un numar intreg"	
c		citirea datelor de intrare	
		read*, numar	
c		alegerea semnului numarului	
		if (numar.lt.0.) **then**	
		semn = " negativ "	
		elseif (numar.eq.0) **then**	
		semn = " 0 "	
		else	
		semn = "pozitiv"	
		endif	
c		scrierea datelor de ieşire	
		print*, " semnul este ", semn	
		end	

LUCRAREA 8

CONSTRUCȚIA CASE. PROGRAMELE ARIE ŞI TIP_TASTA

1. CONSTRUCȚIA CASE

Construcția CASE selectează pentru execuție cel mult un bloc, însă schema de selectare a blocului este diferită de cea a construcției IF. Sintaxa construcției CASE este prezentată în continuare.

```
select case (expr)
   [case (select)
         bloc]...
   [case default
         bloc]
end select
```

Aici *expr* este o expresie de tip întreg, character sau logic. Selectorul *select* este o listă formată din valori singure sau domenii de valori de acelaşi tip cu *expr*. Domeniul de valori este format din două valori separate prin **":"** şi reprezintă un domeniu de valori cuprins între valorile ce îl definesc inclusiv capetele.

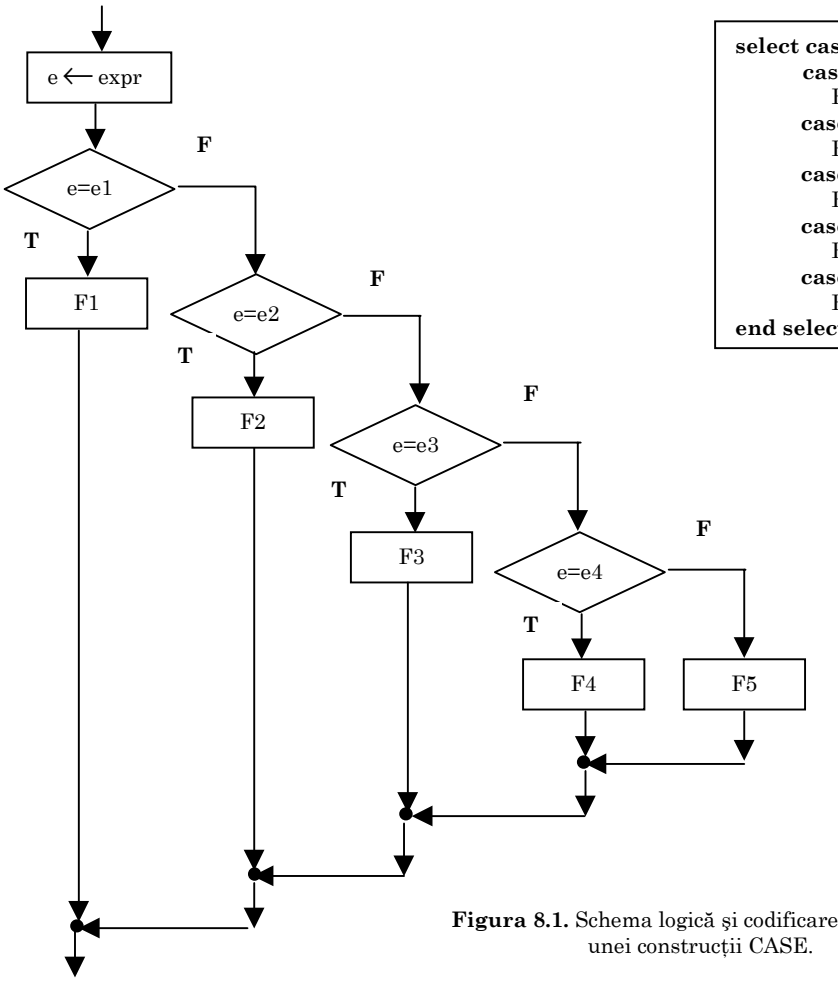
Construcția CASE se execută astfel: se evaluează expresia *expr* din instrucțiunea SELECT CASE; apoi sunt examinate expresiile fiecărui selector *select* din instrucțiunile CASE pâna ce se gaseşte una cu acceaşi valoare ca *expr*, sau cu un domeniu ce include valoarea lui *expr*, se execută blocul ce urmează acestei instrucțiuni CASE şi execuția construcției CASE se termină. Nu trebuie să existe mai mult decât o instrucțiune CASE care să poată avea o valoare a selectorului care să se potrivească cu valoarea lui *expr*. Dacă nici o instrucțiune CASE nu se potriveşte la valoarea expresiei expr şi există o instrucțiune CASE DEFAULT se execută blocul ce urmează lui CASE DEFAULT. Instrucțiunea CASE DEFAULT este opțională.

Construcția Case este o caracteristică nouă a limbajului Fortran, existentă începând cu versiunea *Fortran 90*. În acest caz, selecția se bazează pe valoarea unei expresii scalare aflată în instrucțiunea SELECT CASE la începutul construcției. Valoarea acestei expresii se numeşte *index*. Instrucțiunea ce conține cuvintele cheie SELECT CASE se numeşte *instrucțiunea SELECT CASE*. Instrucțiunea începând cu cuvântul cheie CASE este numită *instrucțiunea CASE*. Instrucțiunea începând cu cuvintele cheie END SELECT este numită *instrucțiunea END SELECT*. O listă de domenii de valori sau cuvântul cheie DEFAULT se numeşte *selector* (*selector case*). Semnificația sintaxei domeniului de valori ale unui selector este următoarea:

- *val*, selecția se face dacă *expr* = *val*;
- *:val,* selecția se face dacă *expr* ≤ *val*;
- *val:*, selecția se face dacă *expr* ≥ *val*;
- *val_1:val_2*, selecția se face dacă *val_1* ≤ *expr* ≤ *val_2*.

Există o serie de reguli şi restricții referitore la domeniile de valori ale selectoarelor.

Forma scrisă în limbaj FORTRAN a construcției CASE, al cărei algoritm este descris de schema logică din Figura 8.1, este prezentată în continuare.




```
select case (e)
        case (e1)
           F1
        case (e2)
           F2
        case (e3)
           F3
        case (e4)
           F4
        case default
           F5
end select
```

Figura 8.1. Schema logică şi codificarea Fortran a unei construcţii CASE.

2. PROGRAMELE ARIE ŞI TIP_TASTA

Folosind o construcţie CASE, s-a realizat un algoritm pentru calculul ariei unei figuri geometrice plane (pătrat, cerc, dreptunghi şi triunghi echilateral) şi s-a codificat acest algoritm în limbaj FORTRAN. Indexul construcţiei CASE este variabila caracter *figura*. Posibilele valori ale indexului sunt constantele caracter *P*, pentru pătrat, *C*, pentru cerc, *T*, pentru triungiul echilateral, *D*, pentru dreptunghi. Pentru alte valori ale indexului se activează selectorul DEFAULT.

Folosind o construcţie CASE s-a realizat şi un program pentru selectarea pe grupe de caractere a tastelor alfanumerice tipăribile ale unei tastaturi. Indexul construcţiei CASE în acest caz este variabila caracter *tasta*. Domeniile de valori ale selectorilor sunt constantele caracter (a, e, i, o ,u), pentru categoria *vocale*, constantele

întregi (0,…,9), pentru categoria *cifre*, constantele caracter (b, c, d, f, g, h, j,…,n, p,…,t, v,…,z) pentru categoria *consoane*. Pentru alte caractere tipăribile se activează selectorul DEFAULT.

Cele două programe sunt prezentate mai jos.

1÷5	6	7÷72	73÷80

```
program arie
implicit none
character*1 figura
real raza, pi, aria, latura
real lungime, latime
print*,'figura? alegeti p = patrat'
print*,          'c = cerc'
print*,          't = triunghi echilateral'
print*,          'd = dreptunghi'
read*, figura
select case (figura)
      case ('c')
            print*,'raza cercului?'
            read*, raza
            pi = 4.0*atan(1.)
            aria = pi*raza*raza
            print*,'aria= ',aria
       case ('p')
            print*,'latura patratului ?'
            read*, latura
            aria = latura*latura
            print*,'aria= ',aria
      case ('d')
            print*,'lungimea?'
            read*,lungime
            print*,'latime?'
            read*, latime
            aria=lungime*latime
            print*,'aria= ',aria
      case ('t')
            print*,'latura?'
            read*,latura
            aria= 0.25*latura8latura*sqrt(3.)
            print*,'aria= ',aria
      case default
            print*,'eroare. introduceti doar una din literele p,c,t sau d'
end select
end
```

1÷5	6	7÷72	73÷80
		program tip_tasta **implicit none** **character***1 tasta **print*,'** alegeti o tasta' **read***, tasta **select case** (tasta) **case** ('a', 'e','i','o','u') **print***, tasta, " este o vocala" **case** ('0':'9') **print***,tasta, " este o cifra" **case** ('b':'d','f':'h','j':'n','p':'t','v':'z') **print***,tasta, " este o consoana" **case** (" ") **print***, tasta, " este spatiu liber" **case default** **print***, tasta, " este un caracter special" **end select** **end**	

LUCRAREA 9

CONSTRUCŢIILE DO SIMPLĂ ŞI DO WHILE. PROGRAMELE PRODUS_SUMA_MEDIA, TAB_FUNCTIE ŞI FUN_TRIG

1. CONSTRUCŢIA DO SIMPLĂ. CONSTRUCŢIA DO WHILE

În limbajul de programare FORTRAN structurile iterative se implementează prin *construcţia DO*. Sintaxa *construcţiei DO cu bloc* este:

```
do control_bucla
      bloc
end do
```

Blocul conţine *zero* sau *mai multe instrucţiuni* şi *construcţii* a căror execuţie repetată este comandată de controlul buclei.

Instrucţiunea ce începe cu cuvântul cheie *DO* se numeşte *instrucţiunea DO*. Instrucţiunea ce începe cu cuvintele cheie *END DO* se numeşte *instrucţiunea END DO*.

Blocul de instrucţiuni *bloc* se mai numeşte *domeniul buclei DO*, ce poate conţine şi construcţii DO, construcţii IF, construcţii CASE sau alte construcţii; este necesar ca orice construcţie interioară să fie înglobată complet în construcţia exterioară. Dacă domeniul buclei DO conţine o altă construcţie DO se spune că buclele DO sunt imbricate.Modul de execuţie al constructiei DO este arătat în schema logică din Figura 9.1.

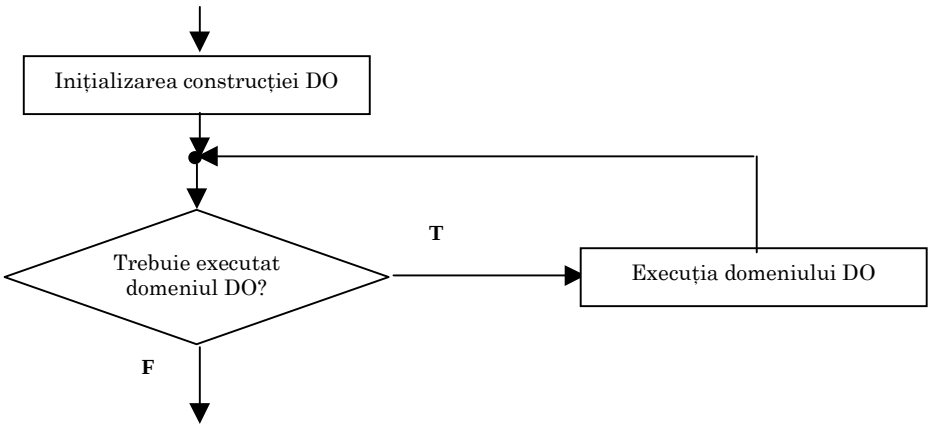


Figura 9.1. Schema logică a construcţiei DO.

Există două forme de bază ale construcţiei DO: (1) construcţia DO bloc; (2) construcţia DO non-bloc. Practica programării moderne recomandă utilizarea construcţiei DO bloc.

O *construcţie DO bloc* este o construcţie DO ce se termină cu o instruţiune END DO sau CONTINUE ce nu face parte dintr-o altă construcţie DO. Execuţia unei bucle

DO este controlată de o variabilă DO a cărei valoare este incrementată se un anumit număr de ori prevăzut în instrucțiunea DO. Acest tip de construcție DO este numită şi *construcția DO cu contor al iterației*. Mai există două metode de control al execuției buclei DO specifice *construcției DO WHILE* şi *construcției DO simple*.

Există două instrucțiuni speciale ce pot să apară în domeniul unei construcții DO şi care pot să altereze execuția secvențială a instrucțiunilor şi construcțiilor blocului. Una este instrucțiunea EXIT, iar cealaltă instrucțiunea CYCLE.

Instrucțiunea EXIT cauzează terminarea imediată a execuției construcției DO şi transmite controlul la prima instrucțiune executabilă dupa END DO. Sintaxa acestei instrucțiuni este: **exit**.

Instrucțiunea CYCLE, spre deosebire de instrucțiunea EXIT, ce termină complet execuția unei construcții DO, întrerupe execuția domeniului şi cauzează începerea unui nou ciclu de execuție al construcției DO. Sintaxa instrucțiunii CYCLE este: **cycle**.

La întreruperea execuției unei construcții DO de către o instrucțiune CYCLE, variabila de control a iterației este actualizată şi începe procesul de execuție al iterației următoare.

```
do
      bloc
end do
```

În cazul construcției DO simple lipseşte controlul buclei, execuția construcției DO terminându-se în mod explicit prin execuția unei instrucțiuni din interiorul domeniului construcției. Sintaxa construcției DO simple este cea arătată alăturat.

```
do while (expr)
      bloc
end do
```

Forma *DO WHILE* a construcției DO permite execuția repetată a unui bloc atât timp cât valoarea unei expresii logice rămâne adevarată. Sintaxa instrucțiunii DO WHILE este arătată mai jos. Aici *expr* este o expresie logică scalară.

Execuția unei construcții DO WHILE se face astfel: se evaluează expresia logică *expr*; dacă valoarea expresiei este .TRUE. se execută blocul. După execuția ultimei instrucțiuni a blocului, execuția revine la instrucțiunea DO WHILE, expresia logică *expr* este din nou evaluată şi ciclul se repetă. Dacă valoarea expresiei logice *expr* este .FALSE. blocul nu se mai execută şi controlul trece la prima instrucțiune executabilă după END DO.

Construcția DO WHILE implementează în FORTRAN iterația cu testul la început, adică iterația do while.

2. PROGRAMELE PRODUS_SUMA_MEDIA, TAB_FUNCTIE, FUN_TRIG

Folosind o construcție DO simplă s-a scris un program Fortran pentru calculul produsului, sumei şi mediei unui şir de numere reale. Numărul factorilor (termenilor)

este ales de utilizator în timpul execuției acestuia. Programul este prezentat în continuare.

1÷5	6	7÷72	73÷80
		program produs_suma_media **implicit none** **integer** i **real** x, produs, suma,media **character***1 raspuns **print***, "Primul numar" **read***, x suma=x produs=x i=1 b1:do i=i+1 **prin**t*, " Numarul", i **read***, x suma=suma+x produs=produs*x media=suma/float(i) **print***, "Continuati? [y/n]" **read***, raspuns **if** (raspuns.**eq**."Y".**or**.raspuns.**eq**."y") **then** **cycle** b1 **else** **exit** **endif** **enddo** b1 **print***, "produsul= ", produs **print***, "suma = ", suma **print***, "media= ", media **end**	

Folosind o construcție DO WHILE s-a realizat un program FORTRAN de tabelare a valorilor unei funcții reale de variabilă reală, pe un interval [a,b], pentru *n* valori ale argumentului, numit *tab_functie*. Funcția este dată de relația următoare:

$$F(x) = 2 \cdot x^3 - x + 4.5$$

Pentru tabelarea valorilor funcțiilor trigonometrice *sin* şi *cos* pe un interval, s-a realizat un program FORTRAN care cuprinde construcții DO WHILE, numit *fun_trig*.

1÷5	6	7÷72	73÷80
		program tab_fun **implicit none** **integer** n **real** x, xinit, xfinal, pas, fx **print***,'introduceti numarul de puncte' **read***, n **print***, 'introduceti capetele intervalului' **read***, xinit, xfinal pas = (xfinal-xinit)/float(n-1) x=xinit **do while** (x.le.xfinal) fx = 2.0*x**3 - x + 4.5 **print***, 'x= ',x, 'f(x)= ', fx x = x + pas **end do** **end**	

1÷5	6	7÷72	73÷80
	*	**program** fun_trig **implicit none** **integer** i, n **real** pi,x_grad,x_rad,pas_grad, pas_rad,xinit_grad, xinit_rad **real** xfinal_grad, xfinal_rad,y_sin, y_cos **character***10, raspuns **print***, " Numarul de valori ale argumentului" **read***, n **print***, "Valoarea argumentului este in radiani sau in grade? [radiani/grade]" **read***,raspuns pi=4.*atan(1.) **if** (raspuns.eq."radiani") **then** **print***, "Valoarea initiala si finala in radiani" **read***, xinit_rad, xfinal_rad pas_rad=(xfinal_rad-xinit_rad)/float(n-1) x_rad=xinit_rad **print***, "Doriti indicarea argumentului si in grade?[Y/N]" **read***, raspuns **if** (raspuns.**eq**."Y".**or**.raspuns.**eq**."y") **then** **print***, " x, [rad] ","x, [grad] ", "y_sin ","y_cos " **do while** (x_rad.**le**.xfinal_rad) y_sin=sin (x_rad) y_cos=cos(x_rad)	

```
                    x_grad=x_rad*180./pi
                    print*, x_rad, "     ",x_grad,"     ",y_sin,"    ", y_cos
                    x_rad=x_rad+pas_rad
                enddo
            else
                print*, "    x, [rad]          ", "y_sin          "," y_cos"
                do while (x_rad.le.xfinal_rad)
                    y_sin=sin (x_rad)
                    y_cos=cos(x_rad)
                    print*, x_rad, "     ",y_sin,"    ",y_cos
                    x_rad=x_rad+pas_rad
                 enddo
        endif
    else
        print*, "Valoarea initiala si finala in grade"
        read*, xinit_grad, xfinal_grad
        xinit_rad=xinit_grad*pi/180.
        xfinal_rad=xfinal_grad*pi/180.
        pas_rad=(xfinal_rad-xinit_rad)/float(n-1)
        x_rad=xinit_rad
        print*, "Doriti indicarea argumentului si in radiani?[Y/N]"
        read*, raspuns
        if (raspuns.eq."Y".or.raspuns.eq."y") then
            print*, "   x, [rad]       ","x, [grad]       ",   "y_sin      ","    y_cos   "
            do while (x_rad.le.xfinal_rad)
                    y_sin=sin (x_rad)
                    y_cos=cos(x_rad)
                    x_grad=x_rad*180./pi
                    print*, x_rad, "     ",x_grad,"     ",y_sin,"    ", y_cos
                    x_rad=x_rad+pas_rad
            enddo
        else
            print*, "    x, [grad]          ", "y_sin          "," y_cos"
            do while (x_rad.le.xfinal_rad)
                    y_sin=sin (x_rad)
                    y_cos=cos(x_rad)
                    x_grad=x_rad*180./pi
                    print*, x_grad, "     ",y_sin,"    ",y_c
                    x_rad=x_rad+pas_rad
            enddo
        endif
    endif
    end
```

LUCRAREA 10

TABLOURI. CONSTRUCȚIA DO CU CONTOR AL ITERAȚIEI. PROGRAMUL MASURATORI

1. TABLOURI. ELEMENTE DE TABLOU

În programare, o mulțime ordonată de date de acelaşi tip se poate organiza ca o entitate numită *tablou*. Tabloul este o dată structurată şi omogenă. Pentru computer tabloul reprezintă un grup de celule de memorie situate, una dupa alta, grup căruia i se asociază un nume, numele tabloului. Celulele tabloului, numite elementele tabloului, sunt referite print-un indice ce se asociază numelui. Notațiile elementelor de tablou reprezintă adresele celulelor în care se stochează valorile elementelor tabloului.

În limbajele de programare se acceptă tablouri cu o dimensiune, cu două dimensiuni sau cu mai multe dimensiuni.

Pentru a informa compilatorul că o anumită entitate reprezintă un tablou trebuie specificate *atributele* caracteristice ale acestuia. Pentru a specifica care sunt atributele unei entități se folosesc *intrucțiuni de declarare* sau *declarații*.

În Fortran există mai multe forme de tablouri. În continuare ne vom referi la *tablouri cu forma explicită*. Aceste tablouri au două atribute: *tip*, *dimensiune*.

Declararea tipului se face cu ajutorul cuvintelor cheie folosite şi pentru declararea variabilelor scalare. Declararea dimensiunilor unui tablou este obligatorie şi se face cu ajutorul unor instrucțiuni neexecutabile ce conțin cuvinte cheie cuprinse într-o *specificație de tablou*. Un tablou cu formă explicită are limitele specificate pentru fiecare dimensiune.

Dimensiunile unui tablou pot fi specificate în două moduri: (1) cu o instrucțiune DIMENSION, când sunt indicate numărul dimensiunilor şi numărul valorilor pentru fiecare indice, (2) cu o instrucțiune de declarare a tipului în care este inclus şi specificatorul pentru declararea dimensiunilor.

Instrucțiunea de declarare a dimensiunilor are forma:

DIMENSION [::] *array*(spec)[,*array*(spec)]...

adică după cuvântul DIMENSION urmează o listă de specificatori de tablou. Instrucțiunea DIMENSION nu este executabilă; ea informează compilatorul că anumite variabile sunt tablouri şi compilatorul rezervă în memorie spațiul necesar stocării elementelor de tablou. În memorie elementele unui tablou unidimensional se stochează unul dupa altul, astfel că elementul de tablou n+1 urmează elementului n, ş.a.m.d. În cazul tablourilor multidimensionale elementele sunt organizate ca un şir liniar deoarece memoria computerului are o singură dimensiune. În FORTRAN elementele de tablou se aranjează în memorie potrivit regulei "ordinea coloanei majore", ceea ce înseamnă că un

tablou se stocheaza în memorie astfel că cel mai din stânga indice să varieze cel mai rapid. Orice tablou este descris printr-o *specificație de tablou* formată din *declaratori de dimensiune*. Numărul de declaratori de dimensiune defineşte *rangul* tabloului sau altfel spus numărul de dimensiuni al tabloului. Limbajul de programare FORTRAN acceptă tablouri cu cel mult 7 dimensiuni.

Forma explicită a unui declarator de dimensiune este:

[lower:] upper

unde *lower* şi *upper* sunt *expresii de specificație*, care sunt expresii scalare sau constante întregi ce arată valoarea minimă, respectiv maximă a indicelui. Valorile lui *lower* şi *upper* pot fi pozitive, negative sau zero. Dacă valoarea minimă este omisă, valoarea ei implicită se consideră 1.

Instrucțiunea de declarare a tipului şi dimensiunilor are forma:

type [DIMENSION] [::] *array*(spec) [,*array*(spec)]…

adică este formată din cuvântul cheie *type* ce arată tipul (specificator de tip) şi o instrucțiune de declarare a tipului.

Elementul de tablou specifică un element al tabloului. Elementul de tablou se scrie astfel:

array (s[,s]...)

unde (*s[,s]…*) reprezintă o listă de valori corespunzătoare numărului declaratorilor de dimensiune.

Cu elementele de tablou se pot efectua toate operațiile pentru variabile scalare (citire, scriere, operații aritmetice sau logice). Una din proprietățile ce conferă flexibilitate prelucrării datelor reprezentate prin tablouri este posibilitatea de a efectua calcule asupra indicilor, adică posibilitatea ca indicii să fie expresii aritmetice.

2. CONSTRUCȚIA DO CU CONTOR AL ITERAȚIEI

Cele mai generale construcții DO cu contor al iterației sunt cele în care valoarea inițială, cea finală şi pasul sunt expresii. Fie *i* variabila ciclului, *e1* expresia pentru valoarea inițiala a lui *i*, *e2* expresia pentru valoarea finală a lui *i* şi *e3* expresia pentru incrementul variabilei DO (controlului buclei). În acest caz este util să se introducă o variabilă auxiliară i_c, numită *variabilă contor*, ce controlează execuția iterației. Schema logică a unui astfel de construcții DO este dată în Figura 10.1. Pentru forma bloc, construcția DO cu contor al iterației are sintaxa prezentată în continuare.

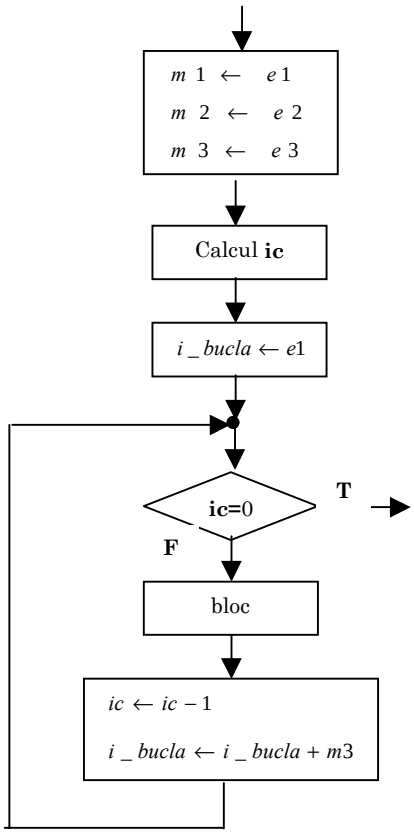


Figura 10.1. Schema logică a construcției DO cu contor al iterației.

```
DO i_bucla = e1,e2,e3
        bloc
END DO
```

Instrucțiunea DO are sintaxa:

DO *i_bucla = e1,e2[,e3]*

unde *i_bucla* este o variabilă întreagă sau reală, *e1, e2* sunt expresii întregi sau reale, iar *e3* este o expresie întreagă sau reală nenulă. Dacă *e3* este egală cu *1* se poate omite scrierea ei.

În limbajul de programare FORTRAN, construcția DO cu contor al iterației, utilizată pentru operații de intrare/ieşire, se poate înlocui printr-o singură instrucțiune de intrare/ieşire echivalentă, ce are următoarea sintaxa:

read*, (x(i), i=1,n)

write*, (x(i), i=1,n)

print*, (a(i), i=1,n)

unde (*x(i), i=1,n*) se numeşte *listă DO implicită*. În instrucțiunile de citire sau de scriere o listă DO implicită poate fi element al unei liste de intrare sau de ieşire pentru instrucțiunile de citire sau de scriere.

Forma generala a unei liste DO implicite este:

(*list*, v=e1,e2[,e3])

unde *list* este o listă de intrare/ieşire ce conține de obicei *elemente de tablou*, *v* este o variabilă întregă, variabila de control, *e1* este valoarea inițială, *e2* valoarea finală iar *e3*

este incrementul. Dacă *e1, e2, e3* sunt expresii, atunci ele sunt expresii aritmetice întregi.

În continuare se prezintă programul FORTRAN scris pentru algoritmul de calcul al mediei şi dispersiei unui set de *n* date experimentale obținute prin măsurători directe. Media şi dispersia se determină folosind următoarele relații:

$$x_{mediu} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - x_{mediu})^2}{n-1}}$$

1÷5	6	7÷72	73÷80
		program masuratori **implicit none** **integer** i,n **parameter** (n=50) **real** x(n), suma, xmediu, s2abat,sigma i=1 **do while** (i.le.n) **read***, x(i) i=i+1 **end do** suma = 0. i=1 **do while** (i.le.n) suma = suma + x(i) i=i+1 **end do** xmediu=suma/float(n) s2abat=0. **do** k = 1,n s2abat = s2abat + (x(k) - xmediu) **2 **end do** sigma=**sqrt**(s2abat/float(n-1)) **print***, 'x mediu= ',xmediu **print***, 'abaterea standard= ',sigma **end**	

LUCRAREA 11

SUBPROGRAME. PROGRAMELE CENTRU_DE_MASA ŞI SORTARE

1. SUBPROGRAME

1.1. Proceduri funcție

O *funcție* este o procedură ce se foloseşte de obicei ca primitivă într-o expresie. În FORTRAN o funcție externă se scrie in forma:

```
[spec_tip] FUNCTION nume ([lista_de_nume_de_argumente_formale])
        [partea_de_specificatie]
        [partea_de_executie]
    END
```

Astfel o funcție externă este o unitate de program ce începe cu instrucțiunea FUNCTION şi se termină cu instrucțiunea END. Corpul funcției poate conține orice instrucțiune cu excepția instrucțiunilor BLOCK DATA, FUNCTION, INTERFACE TO, PROGRAM sau SUBROUTINE. În instrucțiunea FUNCTION *nume* este numele funcției ales de către programator. Numele funcției nu poate să apară în instrucțiunile AUTOMATIC, COMMON, DATA, EQUIVALENCE, INTRINSIC, NAMELIST sau SAVE.

Lista *lista_de_nume_de_argumente_formale* este formată din nume de variabile, nume de tablouri, nume de variabile de structură, nume de proceduri externe sau nume de funcții intrinseci. Tipul funcției poate fi specificat prin specificația de tip *spec_tip*. Dacă *spec_tip* lipseşte, tipul funcției este determinat de instrucțiunile IMPLICIT ce se aplică. Dacă nu există instrucțiuni IMPLICIT tipul funcției se determină prin convenția implicită asupra tipului variabilelor. Tipurile argumentelor se stabilesc prin instrucțiunile din *partea_de_specificație*: IMPLICIT, EXTERNAL, DIMENSION sau instrucțiuni de declarare a tipului. Numele funcției se comportă ca o variabilă; în *partea_de_execuție* într-un anume punct variabilei *nume* trebuie să i se atribuie o valoare. Această valoare este cea returnată unității de program ce apelează funcția.

Lista de argumente formale furnizează modul de comunicație al datelor cu funcția; argumentele, de obicei, servesc drept date de intrare pentru funcție. Rezultatul funcției este furnizat expresiei ce conține apelul funcției.

Apelul unei funcții se poate face în orice unitate de program printr-o referință de funcție, care de obicei, este primitivă intr-o expresie. Forma referinței de funcție este:

nume_functie ([lista_de_argumente_de_lucru])

Lista_de_argumente_ de_lucru trebuie să conțină același număr de argumente ca și instrucțiunea de definiție FUNCTION.

Argument_de_lucru poate fi constantă, variabilă, expresie, element de tablou, nume de funcție intrinsecă. Argumentele formale şi cele de lucru trebuie să reprezinte același tip de date.

Execuția procedurii funcție se face astfel:

- argumentele de lucru ce sunt expresii se evaluează;
- argumentele de lucru se asociază cu argumentele formale corespunzatoare;
- se execută corpul functiei;
- controlul revine la unitatea de program ce a făcut apelul (cea care conține referințe de funcție), iar rezultatul funcției (valoarea funcției) este furnizat expresiei ce conține apelul funcției.

1.2. Proceduri subrutine

În FORTRAN o subrutină se scrie în forma:

```
SUBROUTINE nume ([lista_de_argumente_formale])
  [partea_de_specificatie]
  [partea_de_executie]
END
```

Altfel spus, o subrutină este o unitate de program ce începe cu instrucțiunea SUBROUTINE şi se termină cu instrucțiunea END. O subrutină poate conține orice instrucțiune cu excepția instrucțiunilor BLOCK DATA, FUNCTION, INTERFACE TO, PROGRAM sau altă instrucțiune SUBROUTINE. În instrucțiunea SUBROUTINE *nume* reprezintă numele subrutinei ales de către programator, iar *lista_de__argumente_formale* este o listă de nume ce pot fi nume de variabile, nume de tablouri, nume de variabile de structură, nume de proceduri externe sau funcții intrinseci.

Lista de argumente stabileşte numărul de argumente formale ale subrutinei. Tipul argumentelor formale se declară prin instrucțiuni IMPLICIT, EXTERNAL, DIMENSION sau de declarare a tipului, instrucțiuni ce se scriu în partea_de_specificatie a subrutinei. Numele argumentelor formale nu pot să apară în instrucțiunea AUTOMATIC, COMMON, DATA, EQUIVALENCE, INTRINSIC sau SAVE.

În *partea_de_execuție* a subrutinei trebuie să existe cel puțin o instrucțiune RETURN, ce are următoarea sintaxa:

RETURN

Instrucțiunea RETURN este o instrucțiune executabilă a cărei acțiune este returnarea controlului execuției unei subrutine sau unei funcții la unitatea de program apelantă. Într-un subprogram instrucțiunea END are acelaşi efect ca şi instrucțiunea RETURN.

Apelul unei subrutine se face cu o instrucțiune CALL, ce are urmatoarea sintaxa:

CALL nume ([lista_de_argumente_de_lucru])

unde *nume* este numele subrutinei. *Lista_de_argumente_de_lucru* trebuie să conțină acelaşi număr de argumente ca cel din instrucțiunea SUBROUTINE. *Argument_de_lucru* poate fi: constantă, expresie, tablou, element de tablou, nume de subrutină sau de funcție externă, nume de funcție intrinsecă. Argumentele de lucru şi cele formale trebuie să reprezinte acelaşi tip de date.

Execuția instrucțiunii CALL se face astfel:

- argumentele de lucru ce sunt expresii se evaluează;
- argumentele de lucru se asociază cu argumentele formale corespunzatoare;
- se execută corpul subrutinei
- controlul revine la unitatea de program ce a facut apelul.

Argumentul formal pentru care argumentul de lucru este o variabilă (sau nume de tablou) se zice că este apelat prin referință, iar argumentul formal pentru care argumentul actual este o constantă sau o expresie mai complexă decât o variabilă, se zice că este apelat prin valoare.

2. PROGRAMELE CENTRU_DE_MASA ŞI SORTARE

Cu scopul de a determina coordonatele centrului de masă pentru diferite sisteme de puncte materiale s-a scris un program FORTRAN (centru_de_masa), prezentat în continuare. Se consideră că sistemul de puncte materiale este format din n puncte, ce au coordonatele x_i, y_i, z_i şi masele m_i, iar $i = 1,n$.

Coordonatele centrului de masă sunt date de formulele următoare:

$$x_{cm} = \frac{\sum_{i=1}^{n} m_i \cdot x_i}{\sum_{i=1}^{n} m_i}; \qquad x_{cm} = \frac{\sum_{i=1}^{n} m_i \cdot x_i}{\sum_{i=1}^{n} m_i}; \qquad x_{cm} = \frac{\sum_{i=1}^{n} m_i \cdot x_i}{\sum_{i=1}^{n} m_i}.$$

1÷5	6	7÷72	73÷80
		program centru_de_masa **implicit none** **integer** i, n **parameter** (n=10) **real** x(n), y(n), z(n), m(n) **real** centru, xcm, ycm, zcm **do** i=1,n **read***, m(i),x(i),y(i),z(i) **end do** xcm = centru (n,x,m) ycm = centru (n,y,m) zcm = centru (n,z,m) **print***, 'coordonatele centrului de masa a sistemului' **print***, 'xcm= ', xcm,'ycm= ',ycm, 'zcm= ',zcm **end** **real function** centru(n,u,v) **implicit none** **integer** i, n **parameter** (n=10) **real** u(n), v(n), su1, su2 su1 = 0.0 su2 = 0.0 **do** i=1,n su1 = su1 + v(i) su2 = su2 + u(i)*v(i) **end do** centru = su/masa **return** **end**	

Pentru a exemplifica modul de utilizare a unei subrutine s-a realizat programul Fortran, numit *sortare*, care sortează în ordine crescătoare trei numere reale. Programul principal citeşte datele de intrare (trei numere reale), le sortează în ordine crescătoare prin apelarea subrutinei *schimb* şi afişează numerele în ordine crescătoare. Subrutina *schimb* ordonează crescător numerele prin schimbarea ordinii inițiale. Acest program ce cuprinde două unități de program este prezentat în continuare.

1÷5	6	7÷72	73÷80
		program sortare **implicit none** **real** x1, x2, x3	

```
print*, 'introduceti cele trei numere reale x1, x2, x3'
read*, x1, x2, x3
if (x1.gt.x2) then
    call schimb(x1,x2)
end if
if (x1.gt.x3) then
    call schimb(x1,x3)
end if
if (x2.gt.x3) then
    call schimb(x2,x3)
end if
print*, 'numerele in ordinea crescatoare sunt:'
print*, x1, x2, x3
end

subroutine schimb (a,b)
implicit none
real a,b,temp
temp=a
a=b
b=temp
return
end
```

LUCRAREA 12

INSTRUCȚIUNEA EXTERNAL. PROGRAMELE INTEGRALA ŞI SEMN_FUNCTIE

1. INSTRUCȚIUNEA EXTERNAL

Dacă într-un apel de subrutină sau de funcție (definite de utilizator) argumentele de lucru sunt numele unor subrutine sau a unor funcții, acestea trebuie declarate cu o instrucțiune EXTERNAL.

Instrucțiunea EXTERNAL trebuie să fie scrisă în unitatea de program ce conține apelul. In acest fel compilatorul este informat că anumite nume sunt nume de proceduri externe şi nu sunt nume de tablouri sau de variabile. Instrucțiunea EXTERNAL are sintaxa:

EXTERNAL nume[,nume]...

unde *nume* este nume de subrutină externă sau funcție.

2. PROGRAMELE INTEGRALA ŞI SEMN_FUNCTIE

Să se evalueaze valoarea unei integrale definite, ce are ca integrant o funcție de o singură variabilă, prin metoda trapezelor, folosind un program FORTRAN ce descrie algoritmul acestei metode. Funcția de integrat este definită prin:

$$f(x) = x^2 \cdot \sin(x) \cdot e^x$$

Problema integrării numerice a unei funcții f constă în aflarea unei valori aproximative a integralei:

$$\int_a^b f(x) \cdot dx$$

cu ajutorul unui număr finit de valori ale funcției considerate. Pentru a obține o aproximare a valorii date cu o precizie impusă se utilizează formule de cuadratură iterate. Aceste metode presupun parcurgerea următoarelor etape: (1) se împarte intervalul de integrare $[a,b]$ într-un număr convenabil de subintervale egale, (2) pe fiecare din acestea se aproximează integrala cu ajutorul unei formule de cuadratură simple, (3) se sumează toate valorile obținute pe subintervale, obținând

astfel o aproximație a integralei pe întregul interval. Una din formulele cele mai utilizate este *formula trapezelor* dată de relația:

$$\int_a^b f(x)dx = \frac{h}{2}\left\{\left[f(a) + 2 \cdot \sum_{i=1}^{n-1} f(a + i \cdot h) + f(b) \right] - \frac{(b-a) \cdot h^2}{12} \cdot f''(\xi) \right\}$$

în care $h = (b-a)/n$ reprezintă *pasul rețelei de puncte*, iar n este numărul de intervale. În acest caz, $x_0 = a$ şi $x_n = b$.

Cu ajutorul termenului $\left| -\frac{(b-a) \cdot h^2}{12} \cdot f''(\xi) \right|$ se poate estima numărul minim de intervale astfel încât eroarea să nu fie mai mică decât o valoare adoptată anterior evaluării integralei. Dacă se înlocuieşte h şi se adoptă o eroare minimă *err* se obține condiția:

$$\left| -\frac{(b-a)^3}{12 \cdot n^2} \cdot f''(\xi) \right| \le err$$

din care se obține:

$$\sqrt{\left| -\frac{(b-a)^3}{12 \cdot err} \cdot f''(\xi) \right|} \le n$$

În cazul funcției considerate $f'''(x) = (x^2 + 2 \cdot x + 2) \cdot e^x$, care este o funcție strict crescătoare şi continuă, astfel încât valoarea maximă pe un interval este atinsă pentru valoarea maximă a argumentului x. Dacă înlocuim în ultima relație valoarea derivatei a doua se obține:

$$\sqrt{\left| -\frac{(b-a)^3}{12 \cdot err} \cdot (b^2 + 2 \cdot b + 2) \cdot e^b \right|} \le n$$

Programul FORTRAN prezentat în continuare are ca date de intrare limitele intervalului de integrare (*a,b*), eroarea de estimare a integralei (*err*) şi numărul de intervale adoptat (*n*) astfel încât să fie atinsă eroarea *err*. Programul face o primă estimare a numărului minim de intervale (*nmin*), iar utilizatorul va folosi această valore pentru a alege valoarea pentru *n*.

1÷5	6	7÷72	73÷80
		program integrala **implicit none** **real** a, b, val, trapez, functie, err **integer** n, nmin **external** functie **print***, " Care este intervalul de integrare ([a,b])?" **read***, a, b **print***, " Care este eroarea de calcul adoptata (err)?" **read***, err nmin= aint (**sqrt**(((b-a)**3*(b**2+2*b+2))/(12.*err))) **print***, " Numarul minim de intervale pentru o eroare de ", err,"	
	*	este: ", nmin	
		print*, " Care este numarul de intervale adoptat?" **read***, n val = trapez(functie,a,b,n) **print***,'valoarea integralei este: ',val **end** **real function** trapez(f,a,b,n) **implicit none** **real** a, b, h, suma, x, f **integer** i, n h = (b-a)/float(n) suma = 0.5*(f(a)+f(b)) **do** i=1,n-1 suma = suma + f(a+float(i)*h) **end do** trapez = h*suma **return** **end** **real function** functie(x) **implicit none** **real** x functie = x*x*exp(x)-x **return** **end**	

Programul *Semn_functie* determină dacă există o schimbare de semn pentru o funcție reală *f(x)* de variabilă reală *x* pe un interval [*a,b*] al argumentului, între două valori succesive. Programul este alcătuit din patru unități de program: programul principal, două subprograme *subroutine* şi un subprogram *function*. Programul principal

realizează citirea datelor de intrare, determină valorile argumentului *x* pentru care se calculează valorile funcției şi apelează subprogramele subroutine. Subrutina *schimbare* determină valorile succesive între care există schimbarea de semn şi le memorează în două tablouri unidimensionale. Pentru calculul valorilor funcției apelează subrogramul *funție*. Subrutina *omatrice* afişează subintervalele pe care există schimbări de semn. Programul este prezentat mai jos.

1÷5	6	7÷72	73÷80
		program semn_functie **implicit none** **integer** i,n, nmax **integer** l, m, mmax **parameter** (nmax=1000, mmax=2) **real** xinitial, xfinal, x(nmax) **real** a(nmax), b(nmax), rezultat(nmax,mmax), functie **external** functie **print***,"Care este intervalul pe care se verifica semnul functiei?" **read***, xinitial, xfinal **print***, "Pentru cate valori se face evaluarea functiei(nmax=1000)?" **read***, n **print***, "Se calculeaza cele ",n," valori ale argumentului. Asteptati!" **do** i=1,n x(i)=xinitial+((xfinal-xinitial)/float(n-1))*float(i-1) **enddo** **call** schimbare(nmax,n,x,functie,l,m,a,b) **if** ((l.**eq**.0).**and**.(m.**eq**.0)) **then** **print***, " Functia nu are schimbare de semn pe intervalul considerat" **else** **call** omatrice(nmax,mmax,n,l,m,a,b,rezultat) **endif** **end** **subroutine** schimbare(nmax,n,x,functie,l,m,a,b) **integer** i, j, k,l, m, n, nmax **real** x(nmax), a(nmax), b(nmax), functie j=0 k=0 **do** i=1,(n-1) **if** ((functie(x(i))*functie(x(i+1))).**le**.0.) **then** j=j+1 k=k+1 a(j)=x(i)	

```
        b(k)=x(i+1)
    endif
    l=j
    m=k
enddo
return
end

subroutine omatrice(nmax,mmax,n,l,m,a,b,rezultat)
integer i, j, nmax, mmax, n, l, m
real x(nmax), a(nmax), b(nmax), rezultat(nmax,mmax)
character*1 matrice
print*, " Matricea rezultat este: "
read*, matrice
do j=1,2
    if (j.eq.1) then
        do i=1,l
           rezultat(i,j)=a(i)
        enddo
      else
        do i=1,m
           rezultat(i,j)=b(i)
        enddo
    endif
k=j
enddo
print*, " Matricea ",matrice, "(",l,",",k,") este "
do i=1,l
    print*,"          ", (rezultat(i,j), j=1,2)
enddo
return
end

real function functie(x)
real x
functie=x**3-0.5*x**2+2.*x-1.5
return
end
```

LUCRAREA 13

FIŞIERE. INSTRUCȚIUNILE OPEN ŞI CLOSE. PROGRAMELE IMPAR ŞI TRANSMATR

1. FIŞIERE

Un *fişier* este o colecție de date înregistrate pe un suport extern de informație. Fiecare fişier se asociază unei unități de intrare/ieşire. Într-un fişier datele sunt grupate în *înregistrări*. Limbajul Fortran consideră o înregistrare ca o linie la terminal, o linie la imprimare sau o înregistrare logică pe un suport magnetic. O înregistrare este formată dintr-un şir de date. Înregistrarea poate fi reprezentată printr-o succesiune de *câmpuri*, ce conține fiecare valoarea unei date. Valorile înregistrărilor pot fi formatate sau neformatate. O înregistrare formatată se scrie şi se citeşte cu o instrucțiune de intrare/ieşire formatată, iar o înregistrare neformatată se citeşte şi se scrie cu o instrucțiune de intrare/ieşire neformatată. Există două tipuri de înregistrări: ce conțin date şi o înregistrare specială end-of-file.

În FORTRAN se face distincție între fişierele localizate pe un fişier extern şi fişierele stocate în memorie, accesibile programului. Astfel, deosebim două tipuri de fişiere: *fişiere externe* şi *fişiere interne*.

Fişierele externe sunt stocate pe dispozitivele periferice: unitate de bandă magnetică, unitate de discuri, terminal. Pentru fiecare fişier extern există un set de metode de acces permise, un set de forme permise(formatat sau neformatat), un set de acțiuni permise şi un set de lungimi ale înregistrărilor permise. Stabilirea acestor caracteristici este determinată de către cererile de utilizare ale fişierului şi caracteristicile sistemului de operare.

Fişierele interne sunt stocate, spre deosebire de cele externe, în memoria internă a computerului.

Fişierele FORTRAN au nume. Numele unui fişier intern este numele unei variabile caracter sau a tabloului caracter a cărui valori constituie înregistrările fişierului. Numele unui fişier extern, sau cum se mai numeşte specificația de fişier, este un şir de caractere pe care sistemul de operare îl recunoaşte ca nume de fişier. Dacă nu se specifică o cale de acces sistemul de operare presupune că fişierul se găseşte în directorul curent.

Pentru majoritatea operațiilor de I/O un fişier se identifică printr-un specificator de unitate. Pentru FORTRAN Microsoft specificatorul de unitate este, fie o expresie de tip întreg, fie un asterisc (*). În FORTRAN se spune că un fişier există dacă el există ca fişier la care un program poate avea acces.

Un fişier se zice că este conectat la un program, dacă este asociat cu un specificator de unitate cunoscută sistemului. Conectarea (deschiderea unității) se face

cu ajutorul instrucțiunii OPEN. Deconectarea (închiderea unității) se face cu instrucțiunea CLOSE.

Anumite fişiere sunt preconectate, ele sunt cunoscute procesorului şi întotdeauna sunt disponibile în decursul execuției programului. FORTRAN Microsoft are 4 unități preconectate:

- " ***** " - întotdeauna reprezintă tastatura şi ecranul;
- " **0** " - inițial reprezintă tastatura şi ecranul;
- " **5** " - inițial reprezintă tastatura;
- " **6** " - inițial reprezintă ecranul.

Unitatea asterisc (*) nu poate fi conectată la nici un alt fişier; încercarea de a închide această unitate cauzează o eroare în timpul compilării.

Cu o instrucțiune OPEN unitățile 0, 5 si 6 pot fi conectate la orice alt fişier. Dacă închidem unitățile 0, 5 si 6 ele se reconectează în mod automat la tastatură sau la ecran. La un anumit moment dat, doar un singur fişier poate să fie conectat la o unitate şi invers. Un fişier ce nu este conectat la o unitate nu poate fi folosit în nici o instrucțiune I/O, cu excepția instrucțiunilor OPEN, CLOSE sau INQUIRE.

Există două metode de acces la înregistrările unui fişier: acces secvențial şi acces direct. După metodele de acces fişierele se împart în fişiere secvențiale şi fişiere directe.

2. INSTRUCȚIUNEA OPEN

Instrucțiunea OPEN se foloseşte pentru a conecta un fişier extern la o unitate, a crea un fişier care este preconectat, a crea un fişier şi a-l conecta la o unitate, a schimba unele proprietăți ale conectării. Astfel, pe lângă realizarea conectării propriu-zise, instrucțiunea OPEN se foloseşte şi pentru determinarea proprietăților conectării. Sintaxa instrucțiunii este:

OPEN ([UNIT=],u,olist)

unde *u* este o expresie scalară de tip întreg ce specifică numărul unității fişierului extern, iar *olist* este o listă de specificatori opționali. Un specificator nu poate să apară mai mult decât o dată. Pentru specificatori toate entitățile sunt scalare şi toate caracterele sunt de tipul implicit. Câțiva dintre specificatorii opționali sunt prezentați în continuare:

- FILE = *nume_fisier*, permite definirea numelui fişierului; este obligatoriu dacă lista de specificatori conține specificatorul STATUS cu valorile OLD, NEW sau REPLACE, însă nu trebuie să apară când valoarea acestui specificator este SCRATCH;

- STATUS = *expr*, unde *expr* este o expresie caracter ce poate lua valorile 'OLD', 'NEW', 'REPLACE','UNKNOWN' (default) sau 'SCRATCH'; fişierele SCRATCH sunt fişiere temporare şi nu trebuie numite;
- FORM = *expr*, unde *expr* este o expresie caracter ce defineşte tipul înregistrărilor fişierului; expresia caracter are una din valorile 'FORMATTED' sau 'UNFORMATTED' ;
- ACCESS = *expr*, unde *expr* este o expresie caracter ce specifică tipul de acces pentru transferul datelor; expresia caracter poate furniza rezultatele 'DIRECT' sau 'SEQUENTIAL' (implicit)
- IOSTAT = *ios*, unde *ios* este o variabilă de tip întreg ce permite obținerea de date despre modul de execuție a instrucțiunilor de I/O.
- RECL = *expr*, unde *expr* este o expresie întreagă cu valoare pozitivă ce specifică lungimea fiecărei înregistrări; acest parametru este obligatoriu pentru fişierele în acces direct;
- ERR = *et*, unde *et* este eticheta instrucțiunii executabile din aceeaşi unitate de program cu OPEN la care se transferă controlul, dacă există o eroare de I/O.

3. INSTRUCȚIUNEA CLOSE

Instrucțiunea CLOSE se foloseşte pentru a deconecta un fişier ce a fost conectat cu OPEN. Sintaxa instrucțiunii CLOSE este:

CLOSE ([UNIT=] u, clist)

unde *u* este o expresie întreagă ce specifică o unitate externă, iar *clist* o lista de specificatori opționali:

- ERR = *et,* realizează transferul la instrucțiunea cu eticheta *et* din aceeaşi unitate de program cu CLOSE, dacă există o eroare I/O;
- IOSTAT = *ios*, unde *ios* este o variabilă întreagă ce returnează valoarea zero dacă execuția este fără eroare sau o valoare pozitivă dacă există erori;
- STATUS = *expr*, unde *expr* este o expresie caracter ce poate lua valoarea 'KEEP' sau 'DELETE'; în afara fişierelor 'SCRATCH', valoarea implicită este 'KEEP'.

2. PROGRAMELE IMPAR ŞI TRANSMATR

Programul Fortran *Impar*, prezentat în continuare, crează un fişier ce conține toate numerele impare cuprinse între 0 si 999. Programul *Transmatr* determină transpusa unei matrici.

1÷5	6	7÷72	73÷80
		program impar	
		implicit none	
		integer impar_max, impar_urmator	
		parameter (impar_max=99)	
		character*4 fmt	
		fmt="(i3)"	
		open (unit=10, file='nrimp', form='formatted'	
	&	access='sequential',status='unknown')	
		do impar_urmator=1,impar_max,2	
		write(10,fmt) impar_urmator	
		end do	
		endfile (unit=10)	
		close (unit=10)	
		end	

1÷5	6	7÷72	73÷80
		program transmatr	
		implicit none	
		integer nmax, mmax, eroare	
		parameter (mmax=100, nmax=100)	
		integer i, j, m, n	
		real a(mmax, nmax), transpusa(nmax,mmax)	
		print*, " Cate linii (mmax=100) si cate coloane (nmax=100) are ma	
	*	tricea A(mmax,nmax)?"	
		read*, m,n	
		print*, " Elementele matricii A pe fiecare linie"	
		do i=1,m	
		read*, (a(i,j), j=1,n)	
		enddo	
		do i=1,m	
		do j=1,n	
		transpusa(j,i)=a(i,j)	
		enddo	
		enddo	
		open (unit=10, file="transpusa.dat", status="replace",	
	*	form="formatted", iostat=eroare, err=100)	
		do j=1,n	
		write (10,*) (transpusa (j,i), i=1,m)	
		enddo	
		close (unit = 10)	
100		**print***, " Eroare = ", eroare	
		print*, "Program terminat"	
		end	

LUCRAREA 14

INTRUCȚIUNI DE TRANSFER. PROGRAMELE SUBSIR ŞI EL_MAX_MIN_MED

1. INSTRUCȚIUNI DE TRANSFER A DATELOR

1.1. Instrucțiunile READ, WRITE, PRINT

Instrucțiunile de transfer a datelor sunt READ, WRITE si PRINT. Sintaxa acestor instrucțiuni este:

READ (*lista_cu_speci_de_control_io*) [*lista_de_elemente_input*]

READ *fmtspec* [,*lista_de_elemente_input*]

WRITE (*lista_cu_specif_de_control_io*) [*lista_de_elemente_output*]

PRINT *fmtspec* [,*lista_de_elemente_output*]

Aici *fmtspec* este *specificatorul de format* definit ca:

fmtspec= {string|label|*}

unde *fmtspec* este o expresie caracter ce reprezintă specificația de format, *label* este eticheta unei instrucțiuni ce indică formatarea explicită, iar *asteriscul* indică formatarea comandată de listă. Specificațiile de control I/O sunt:

- [UNIT=] u, unde *u* este specificatorul de unitate;
- [[FMT=] fmtspec], unde *fmtspec* este specificatorul de format;
- [[NML=] nmlspec], unde *nmlspec* este specificatorul grupului NAMELIST;
- [REC =rcl], unde *rcl* este o expresie de tip întreg ce reprezină numărul înregistrării. Se foloseşte doar în acces direct.
- [IOSTAT = ios], unde *ios* este o variabila întreagă;
- [ERR = et], unde *et* este eticheta unei instrucțiuni din acceaşi unitate de program;
- [END = endlabel], unde *endlabel* este eticheta unei instrucțiuni din acceaşi unitate de program.

Dacă se omite UNIT=, primul parametru trebuie să fie *u*. Dacă se omite FMT= sau NML=, al doilea parametru trebuie să fie *fmtspec* sau *nmlspec*. Ceilalți parametri pot să apară în orice ordine. Nu este permis să existe şi specificatorul de format şi

specificatorul grupului NAMELIST.

Lista_de_elemente_input este formată din variabile, elemente de tablou, elemente de structura sau liste implicite DO. Lista_de_elemente_output este formată din expresii (în particular, variabile) sau liste DO implicite.

Dacă în cazul unei operații I/O apare o eroare sau dacă se întâlneşte înregistrarea EOF, acțiunea pe care o ia procesorul depinde de prezenta opțiunilor ERR, IOSTAT şi END. În cazul instrucțiunii READ nu se consideră eroare evenimentul atingerii înregistrării EOF. Deoarece instrucțiunea PRINT este singura instrucțiune I/O care nu acceptă opțiunile ERR, END şi IOSTAT, o eroare în decursul execuției instrucțiunii PRINT este întotdeauna o eroare run-time.

Pentru I/O formatat fişierul este format din caractere. Aceste caractere sunt convertite în reprezentari convenabile pentru stocarea în memoria calculatorului în timpul operațiilor de intrare şi convertite de la reprezentarea internă la caractere. Atunci când un fişier este accesat secvențial înregistrările sunt procesate în ordinea în care ele apar în fişier. Input-output ce avansează înseamnă că fişierul este poziționat după sfârşitul ultimei înregistrări citite sau scrise atunci când operația I/O s-a terminat.

Sintaxa instrucțiunilor cu acces secvențial formatat este:

```
READ ([UNIT=] u,[FMT=]fmtspec[,IOSTAT=ios][,ERR=errorlabel]
                [,END=endlabel]) [ilist]

WRITE ([UNIT=] u,[FMT=]fmtspec[,IOSTAT=ios][,ERR=errorlabel])
                 [olist]

PRINT fmtspec [,olist]
```

Parametrul FMT poate fi omis doar dacă omitem opțiunea UNIT=; *fmtspec* poate fi eticheta unei instrucțiuni format, o expresie caracter a cărei valoare este o specificație de format sau un asterisc ce indică o formatare condusă de listă. În sintaxa generală, prezentată mai sus, *ilist* reprezintă lista elementelor de intrare, iar *olist* lista elementelor de ieşire.

1.2.Operații I/O care nu avansează. Specificatorul ADVANCE

Specificatorul ADVANCE ce poate exista în sintaxa instrucțiunilor READ şi WRITE permite ca operațiile I/O dintr-un (într-un) fişier să se efectueze fără a avansa poziția la următoarea înregistrare. Operațiile I/O care nu avansează se pot folosi doar pentru fişiere externe care sunt formatate în mod explicit şi sunt conectate pentru acces secvențial. Sintaxa unor instrucțiuni ce folosesc specificatorul ADVANCE este următoarea:

READ ([UNIT=]u[,[FMT=]fmtspec],ADVANCE=expr,[SIZE=var][,EOR=et]
[,IOSTAT=ios][,ERR=et][,END=et]) [lista_ent_input]

WRITE ([UNIT=]u[,[FMT=]fmtspec],ADVANCE=expr, [,IOSTAT=ios]
[,ERR=et]) [lista_ent_input]

Valoarea entității *expr* din specificatorul ADVANCE trebuie să fie NO pentru a nu se produce avansul. *Format* poate fi eticheta unei instrucțiuni FORMAT sau o expresie caracter a cărei valoare este specificația de format. *Format* nu poate fi un asterisc deoarece ar indica o formatare condusă de listă.

2. PROGRAMELE SUBSIR ŞI EL_MAX_MIN_MEDIA

Cu programul FORTRAN *subsir* se determină elementul maxim din subşirul format din termenii de rang impar ai unui şir inițial de *n* numere reale şi elementul minim al subşirului format din termenii de rang par ai aceluiaşi şir, precum şi rangurile acestor elemente în şir.

Programul FORTRAN *el_max_min_media* se determină elementul cu valoarea maximă, respectiv, elementul cu valoarea minimă pentru un şir de *n* numere reale şi media aritmetică a acestora.

Cele două programe sunt prezentate în continuare.

1÷5	6	7÷72	73÷80
		program subsir	
		implicit none	
		integer i,n,nmax,rang_max,rang_min	
		parameter (nmax=20)	
		real x(nmax), xmin, xmax	
		open (unit=1, file='sir.dat', form='formatted',	
	*	access='sequential', status='old')	
		open (unit=2, file='subsir.dat', form='formatted',	
	*	access='sequential', status='unknown')	
		read (1,10) n	
	10	**format** (i3)	
		read (1,20) (x(i),i=1,n)	
	20	**format** (10f8.3)	
		xmax=x(1)	
		do i=1,n,2	
		if (xmax.**le**.x(i)) **then**	
		xmax=x(i)	

```
      rang_max=i
   end if
end do
write (2,30)
30 format ('*************************************************************
* ****************************************')
write (2,40) xmax, rang_max
40 format (/, 'elementul maxim al subsirului cu termeni de rang
* impar are valoarea= ',f8.3,' si rangul= ', i3)
xmin=x(2)
do i=2,n,2
   if (xmin.ge.x(i)) then
      xmin=x(i)
      rang_min=i
   end if
end do
write (2,50)
50 format ('*************************************************************
* ****************************************')
write(2,60) xmin, rang_min
60 format (/, 'elementul minim al subsirului cu termeni de rang
* par are valoarea= ',f8.3,' si rangul= ', i3)
write (2,70)
70 format ('*************************************************************
* ****************************************')
end
```

```
1÷5  6  7÷72                                                          73÷80
     program el_max_min_med
c    acest program determina elementul maxim, elementul minim si  media
c    aritmetica pentru un sir de n numere reale.
c    input
c    n=numarul elementelor sirului
c    x(i)=tablou unidimensional de dimensiune n ce contine elementele siru lui
c    output
c    x(i)=tablou unidimensional de dimensiune n ce contine elementele siru-
c    lui dat initial
c    k=indicele elementului din sirul dat ce are valoarea maxima
c    xmax=valoarea maxima a elementelor sirului dat
c    l=indicele elementului din sirul dat ce are valoarea minima
c    xmin=valoarea minima a elementelor sirului dat
c    sxm=media aritmetica a elementelor sirului
```

```
      implicit none
      integer i,k,l,n,nm
      parameter (nm=20)
      real x(nm),xmax,xmin,s,sxm
      character*10 fmt1
      fmt1='(a\)'
      write (*,fmt1,ADVANCE='NO') ' Numarul elementelor vectorului n= '
      read*,n
      print*,' introdu elementele vectorului x '
      read*,(x(i),i=1,n)
      xmax=x(1)
      k=0
      do i=1,n
         if (x(i).ge.xmax) then
            xmax=x(i)
            k=i
         endif
      enddo
      xmin=x(1)
      l=0
      do i=1,n
         if (x(i).le.xmin) then
            xmin=x(i)
            l=i
         endif
      enddo
      s=0.0
      do i=1,n
         s=s+x(i)
      enddo
      sxm=s/float(n)
      write (*,10) n
10    format (/' vectorul x(',i3,') este ')
      write(*,20) (x(i),i=1,n)
20    format(/,8(1x,f9.3))
      write(*,30) k,xmax
30    format(/' elementul cu valoarea maxima este x(',i3,')= ',f10.3)
      write(*,40) l,xmin
40    format(/' elementul cu valoarea minima este x(',i3,')= ',f10.3)
      write(*,50) sxm
50    format(/' media aritmetica a elementelor este= ',f10.3)
      end
```

LUCRAREA 15

ALGORITMUL DE INTERPOLARE AITKEN-NEVILLE. PROGRAMUL INTERPOLARE

1. INTRODUCERE

Interpolarea este una din metodele importante de aproximare a valorilor funcțiilor. A interpola înseamnă a rezolva o problemă de tipul:

Dacă pentru o funcție de n variabile $F:D \longrightarrow R$ $(D \subset R^n)$ *se cunosc valorile într-un număr de puncte* $(x_1^i, x_2^i, x_3^i, ..., x_n^i) \in D, i = 1,2,...,k$, *să se determine valoarea ei într-un punct dat* $(x_1^0, x_2^0, x_3^0, ..., x_n^0) \in D$.

Evident, în această formulare generală problema este nedeterminată, deoarece funcția F poate avea o valoare arbitrară în punctul dat. De aceea se va presupune că funcția F posedă anumite proprietăți de continuitate şi diferențiabilitate.

Considerațiile următoare sunt valabile pentru o funcție reală de o singură variabilă reală:

Fie funcția $f:I \longrightarrow R$, *unde* $I \subset R$ *este un interval şi fie n+1 puncte aparținând intervalului I* ($x_0, x_1, x_2, \ldots, x_n$), *numite* ***noduri de interpolare***. *Se presupune că sunt cunoscute valorile funcției f în nodurile de interpolare,* $y_i = f(x_i)$, $i = 0,1,\ldots,n$. *Se pune problema construirii unei funcții g, ce aparține unei clase specificate de funcții, care în nodurile de interpolare satisface anumite condiții ce o "apropie" de funcția f. Funcția g se numeşte* ***funcţia de interpolare*** *a funcției f.*

Clasa polinoamelor este una din clasele de funcții de interpolare des utilizate deoarece în urma operațiilor algebrice cât şi a operațiilor de derivare şi integrare obținem ca rezultat tot un polinom. De asemenea, o funcție continuă poate fi aproximată uniform pe un interval compact cu un polinom, conform *teoremei de aproximare a lui Wierstrass*.

Se cunosc mai multe formule de interpolare, cum ar fi formula de interpolare a lui Lagrange, formula de interpolare a lui Hermite, formula de interpolare a lui Newton.

Fie $f: I \longrightarrow R$ o funcție definită pe intervalul $I \subset R$ şi nodurile de interpolare $x_0, x_1, x_2, \ldots, x_n$, n puncte distincte ale intervalului I.

Teoremă. Există un polinom unic P de grad cel mult n, care în nodurile de interpolare satisface condițiile $P(x_i) = f(x_i)$, $i = 0,1,2,\ldots,n$.

Definiție. Polinomul a cărei existență şi unicitate este dată de teorema precedentă se numeşte *polinomul de interpolare* a funcției f pe nodurile $x_0, x_1, x_2, \ldots, x_n$.

Polinomul de interpolare (care este unic) poate fi scris în forme diferite, utile atât din punct de vedere teoretic cât şi practic. Dacă nu se impune condiția ca gradul polinomului $P(x)$ să fie cel mult n, se pot determina o infinitate de polinoame ce au valoarea $f(x_i)$ în punctul x_i, pentru orice $i = 0,1,2\ldots,n$.

2. FORMULA DE INTERPOLARE A LUI LAGRANGE

Următoarea formă a polinomului de interpolare se numeşte *forma lui Lagrange* a polinomului de interpolare sau *polinomul lui Lagrange:*

$$P(x)=\sum_{i=0}^{n} f(x_i)\cdot L_i(x)$$

în care $L_i(x)$ este definit de relația:

$$L_i(x)=\frac{(x-x_0)\cdot(x-x_1)\cdots(x-x_{i-1})\cdot(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)\cdot(x_i-x_1)\cdots(x_i-x_{i-1})\cdot(x_i-x_{i+1})\cdots(x_i-x_n)}$$

Se notează cu $L(x_0, x_1, x_2, \ldots, x_n; f|x)$ polinomul de interpolare asociat funcției f pe nodurile $x_0, x_1, x_2,\ldots,x_n$, sub forma lui Lagrange.

Definiție. Funcția $R_n : I \to R$ definită prin $R_n = f(x) - L(x_0, x_1, x_2, \ldots, x_n; f|x)$, adică eroarea cu care $f(x)$ este aproximată de valoarea polinomului lui Lagrange în punctul x, se numeşte *restul de ordin n* asociat funcției f pe nodurile $x_0, x_1, x_2,\ldots,x_n$.

Din relația de definiție a *restului de ordin n* se obține:

$$f(x) = L(x_0, x_1, x_2, \ldots, x_n; f|x) - R_n, x\in I$$

care este *formula de interpolare a lui Lagrange.*

3. ALGORITMUL DE INTERPOLARE AITKEN-NEVILLE

Teoremă. Polinomul lui Lagrange $L(x_0, x_1, x_2, \ldots, x_n; f|x)$ satisface următoarea relație de recurență:

$$L(x_0,x_1,\ldots,x_n;f\mid x)=\frac{(x-x_1)\cdot L(x_1,x_2,\ldots,x_n;f\mid x)-(x-x_n)\cdot L(x_0,x_2,\ldots,x_{n-1};f\mid x)}{(x_n-x_0)}$$

Această formulă oferă o regulă pentru condtruirea pas cu pas a valorilior polinoamelor lui Lagrange într-un punct dat:

x_0	$x - x_0$	$L(x_0;f\|x)$				
x_1	$x - x_1$	$L(x_1;f\|x)$	$L(x_0,x_1;f\|x)$			
x_2	$x - x_2$	$L(x_2;f\|x)$	$L(x_1,x_2;f\|x)$	$L(x_0,x_1,x_2;f\|x)$		
x_3	$x - x_3$	$L(x_3;f\|x)$	$L(x_2,x_3;f\|x)$	$L(x_1,x_2,x_3;f\|x)$		
…	…	…	…	…	…	
x_i	$x - x_i$	$L(x_i;f\|x)$	$L(x_{i-1},x_i;f\|x)$	$L(x_{i-2},x_{i-1},x_i;f\|x)$	…	$L(x_0,x_1,x_2,\ldots,x_i;f\|x)$

Dacă ultima valoare aproximativă $L(x_0,x_1,x_2,\ldots,x_i; f|x)$ nu aproximează suficient de bine se ia un nou nod x_{i+1} şi se formează linia corespunzătoare acestuia. Se compară din nou valoarea obținută $L(x_0,x_1,x_2,\ldots,x_{i+1}; f|x)$ cu cea anterioară. Acest procedeu se repetă până când este îndeplinită o condiție de eroare absolută între cele două valori succesive.

Acest procedeu de evaluare a valorilor polinoamelor lui Lagrange într-un punct dat se numeşte *metoda Aitken-Neville*.

Algoritmul Aitken-Neville. Pentru obținerea unui algoritm se face următoarea notație:

$$L_{ij}(x) = L(x_0,x_1,x_2,\ldots,x_i; f|x),\ i < j$$

Conform unei relații anterioare se poate scrie:

$$L_{ij}(x) = \frac{(x - x_i)\cdot L_{i-1,\,j-1}(x) - (x - x_{i-j})\cdot L_{i,\,j-1}(x)}{(x_{i-j} - x_i)}$$

Tabelul anterior al valorilor polinomului de interpolare se poate scrie, cu convenția $L_{ij}(x) = f(x_i)$ pentru orice i, astfel:

x_0	L_{00}				
x_1	L_{10}	L_{11}			
x_2	L_{20}	L_{21}	L_{22}		
x_3	L_{30}	L_{31}	L_{32}		
…	…	…	…	…	
x_i	L_{i0}	L_{i1}	L_{i2}	…	L_{ii}

Descris în limbaj natural algoritmul de interpolare Aitken-Neville se scrie astfel:

1. atribuie $L_{00} = f(x_0)$;
2. atribuie $i = 1$;
3. atribuie $L_{i0} = f(x_i)$;
4. Calculează L_{ij} cu relația de mai sus, $j = 0,1,\ldots, i$;
5. Dacă L_{ii} este o aproximație suficient de bună, se trece la punctul 7; în caz contrar, se trece la punctul 3;
6. atribuie $i = i + 1$; treci la pasul 3;
7. atribuie $f(x) = L_{ii}$;
8. algoritmul se încheie.

Pentru a decide dacă L_{ii} este o aproximație suficient de bună a valorii funcției f în punctul x se poate folosi condiția:

$$|L_{ii}(x) - L_{i-1,j-1}(x)| < e$$

în care e este o toleranță dată pentru eroarea absolută între două valori sucesive ale polinomului de interpolare.

4. PROGRAMUL INTERPOLARE

În continuare este prezentat un program FORTRAN care implementează algoritmul de interpolare Aitken-Neville.

Programul are următoarele date de intrare:

- *n*, numărul nodurilor de interpolare;
- valorile argumentului *x*, (*x(i),i=1,n*), corespunzătoare celor *n* noduri de interpolare, reprezentate sub forma unui tablou unidimensional real cu *n* elemente;
- valorile funcției în nodurile de interpolare, (*fx(i), i = 1,n*), tablou unidimensional real cu *n* elemente;
- valoarea argumentului *x* în punctul în care se aproximează funcția prin interpolare, *x_eval*, variabilă reală scalară;
- toleranța erorii absolute e, *epsilon*, variabilă reală scalară.

Programul furnizează următorul rezultat:

- valoarea funcției *f(x_eval)* în punctul considerat.

1÷5	6	7÷72	73÷80
		program interpolare **implicit none** **integer** i, j, n, nmax **parameter** (nmax=50) **real** x(nmax), fx(nmax), l(nmax, nmax+1), x_eval, epsilon **character***10 fmt fmt="(a\)" **write (***,fmt,ADVANCE="NO") " Numarul nodurilor de interpolare este = " **read***, n **print***, " Valorile argumentului corespunzătoare nodurilor" **read***, (x(i), i=1,n) **print***, "Valorile functiei in nodurile de interpolare" **read***, (fx(i), i=1,n) **write (***,fmt,ADVANCE="NO") " Valoarea punctului de evaluare a functiei= " **read***, x_eval **write (***,fmt,ADVANCE="NO") " Eroarea de calcul epsilon= " **read***, epsilon l(1,1) = fx(1)	

```
do i=2,n
   l(i,1)=fx(i)
   do j=2,i
      l(i,j)=((x_eval-x(i))*l(i-1,j-1)-(x_eval-x(i-j+1))*l(i,j-1))/(x(i-j+1)-x(i))
      if (l(i,j)-l(i-1,j-1)).le.epsilon) then
            print*, " F(", x_eval,")= ", l(i,j)
            stop
      endif
   enddo
enddo
print*, " F(", x_eval,")= ", l(i,j), " fara a avea o toleranta sub epsilon"
end
```

LUCRAREA 16

TESTAREA PROGRAMELOR

1. INTRODUCERE

Proiectarea este un proces de concepție a unei soluții posibile pentru problema propusă spre rezolvare, bazat pe următoarele procedee generale: analiză, sinteză, abstractizare, elaborare, reprezentare, evaluare şi operaționalizare. În cazul produselor software, proiectarea este activitatea care începe cu definirea cerințelor şi specificațiilor produsului, continuă cu detalierea şi transformarea acestora până la definirea structurii unei soluții. Această soluție trebuie să fie reprezentată într-un limbaj astfel încât proiectul obținut să poată servi mai departe la construirea sau elaborarea propriu-zisă a produsului software (codificarea, testarea). Proiectul rezultat trebuie să precizeze exact structura produsului, adică componentele (de tip program sau de tip date), relațiile între componente şi interacțiunea produsului cu mediul.

Proiectarea oricărui produs se descompune în două subactivități:

- *proiectarea de ansamblu*, care identifică structura generală a problemei şi scopul pentru care se crează noul produs. În acest fel sunt identificate componentele sistemului, relațiile dintre ele, algoritmii ce vor fi utilizați, datele comune ale componentelor;
- *proiectarea de detaliu*, care presupune detalierea componentelor, definirea şi precizarea algoritmilor, structurile de date, interfețele dintre componente şi modul lor de implementare.

Majoritatea metodelor de proiectare a programelor admit modularizarea ca principiu general de obținere a soluției. Una din metodele des folosite pentru proiectarea programelor mari este proiectarea structurată.

Obiectivul proiectării structurate este realizarea unui sistem eficient, fiabil, uşor de întreținut şi modificat, fiind asigurat prin:

- proiectarea pornind de la structură funcțională;
- abordarea complexității problemei într-o manieră care conduce la tratarea individuală a subproblemelor;
- reutilizarea de module realizate anterior;
- folosirea unor criterii de evaluare a calității proiectării.

Proiectarea structurată ajuta programatorul în obținerea unui program modular sau a unui ansamblu de programe modulare.

O dată ce un program a fost alcătuit trebuie să ne asiguram că el îndeplineşte funcția pe care o dorim şi nu alta. Pentru analiza metodelor de testare şi depanare o importanță deosebită o are sistematizarea erorilor posibile.

Erorile ce pot apărea în etapele de proiectare şi codificare pot fi grupate în următoarele clase:

- *erori daterminate de alegerea şi descrierea algoritmului*; aceste erori sunt generate de un algoritm incorect, un algoritm corect dar inadecvat problemei; validarea incorectă şi/sau incompletă a datelor de intrare etc;
- *erori în definirea şi utilizarea datelor*; aceste erori pot avea drept cauze neinițializarea tuturor variabilelor, formate improprii pentru instrucțiunile de citire, definirea incorectă a câmpurilor în fişierele de intrare etc;
- *erori produse prin nerespectarea regulilor de folosire a operatorilor şi operanzilor*; expresii complicate cu posibilități reduse de control a corectitudinii evaluării, erori de conversie a operanzilor etc;
- *erori produse de tehnica de programare*;
- *erori produse din neatenție*;
- *erori în manipularea textului programului*;

Cea mai mare parte a erorilor enumerate mai sus sunt depistate în *etapele de compilare şi editarea legăturilor*.

În timpul execuției programelor pot apare erori de tipul:

- *erori de echipament*, ce sunt determinate de caracteristicile sistemului de calcul (aritmetica numerelor, modul de implementare a datelor scalare sau a celor structurate etc);
- *erori de transfer de date către periferice*.

Cele mai cunoscute strategii de testare a unui produs program complex sunt *testarea de sus în jos* (*top-down*), *testarea de jos în sus* (*bottom-up*), *metoda mixtă*. Testarea trebuie aplicată fiecărui modul în parte, interfeței (legăturii) dintre module şi produsului program în ansamblu.

Scopul testării programelor este *depistarea şi eliminarea erorilor*. Cea mai simplă tehnică de analiză a corectitudinii programelor este aşa-numita *testare prin exemple concrete*. În acest caz, testarea reprezintă operația de verificare a corectitudinii unui program prin furnizarea de date date de intrare şi compararea rezultatelor cu datele de ieşire cunoscute. Tehnica testării prin observarea comportării programului în timpul execuției programului cu datele de test este cunoscută sub denumirea de testare experimentală a programelor. Ideea testării experimentale este descrisă în continuare. Dispunem de un program executabil ce a fost scris într-un limbaj de programare, de exemplu FORTRAN. Alegem o problemă test ce se bucura de proprietatea că îi cunoaştem dinainte răspunsul pentru anumite date de intrare. Cu datele de intrare

respective se ruleaza programul pe calculator fiind necesar a se obține în final rezultatele testelor de control. Tehnica testarii experimentale este una din cele mai răspândite tehnici de testare a programelor tehnico-stiințifice. Programele performante în literatura de specialitate, de regulă, conțin în documentația lor o colecție de probleme test alcătuită din:

- set de date de intrare şi rezultate corespunzătoare;
- set de test drivers, programe de antrenare pentru rezolvarea problemelor test.

Problemele test trebuie să fie alese în aşa fel încât să fie testate prin calcul toate drumurile posibile din schema logică. Un program preluat din literatura de specialitate se consideră corect implementat pe un computer sau cum se mai spune validat, dacă el dă rezultate satisfăcătoare la rularea problemelor test. Trebuie remarcat faptul că testarea experimentală a unui program nu este echivalentă cu demonstrația corectitudinii programului.

Stabilirea datelor de test se poate face având în vedere fie *specificația problemei* fie *textul programului*. Prima metodă de stabilire a datelor de test este adecvată problemelor simple sau în testarea modulelor. În al doilea caz pentru a stabili datele de test se ține seama de instrucțiunile ce trebuie executate.

Adesea este imposibil să se execute programul cu toate datele de test posibile. În această situație apare problema alegerii acelei submulțimi de date care să depisteze cu probabilitate maximă erorile prezente în program. Testarea minimă care trebuie făcută constă în realizarea unui număr de execuții ale programului care să asigure execuția fiecărei instrucțiuni cel puțin odată.

Este necesară şi testarea robusteței programului. Aceasta înseamnă o buna comportare a programului atunci când datele de intrare sunt intenționat greşite, iar problema nu are sens. Un program robust nu trebuie să fie afectat de datele de intrare eronate. Comportarea cea mai normală în astfel de situații ar fi semnalarea unor mesaje de eroare corespunzătoare.

Detectarea unor erori, atât în faza de compilare cât şi în cea de testare necesită *depanarea* programului. Depanarea este procesul de transformare a unui program sau a unei părți de program pentru a înlătura o eroare detectată. Sistemele de operare sau mediile de programare moderne oferă pachete de programe care asistă programatorul în depanarea programului, prin execuția lor pas cu pas şi vizualizarea valorilor curente ale variabilelor (depanare dinamică).

2. TESTAREA PROGRAMULUI INTERPOLARE

Testarea programului interpolare se va face prin metoda testării experimentale.

Fie funcțiile:

$$f(x) = 2 \cdot x - 5$$

$$g(x) = x^2 - x + 1$$

Datele de test sunt formate dintr-un număr n de valori ale argumentului şi o valoare a argumentului diferită de cele alese pentru noduri ca punct de evaluare a funcției, pentru care trebuie calculată şi valoarea funcției. Se consideră că programul a trecut testul dacă valoarea funcției obținută prin evaluarea expresiei corespunzătoare în punctul considerat diferă de valoarea funcției obținută prin interpolare cu o valoare *eps*. În vederea testării, programul *Interpolare* va fi rescris astfel încât să poată fi calculate valorile celor două funcții în punctul de evaluare şi să se poată verifica dacă diferența dintre valoarea obținută prin calcul şi cea obținută prin interpolare este mai mare decât *eps*. Atunci când condiția $\left| f(x_{evaluare})_{calcul} - f(x_{evaluare})_{interpolare} \right| \le eps$ este îndeplinită se va afişa mesajul *Programul Interpolare a trecut testul*, iar dacă nu este îndeplinită mesajul *Pentru eps=valoare, programul Interpolare nu a trecut testul*.

1÷5	6	7÷72	73÷80
		program test_interpolare	001
		implicit none	002
		integer i, j, n, nmax	003
		parameter (nmax=50)	004
		real x(nmax), fx(nmax), l(nmax, nmax+1), x_eval, errcalcul, epsilon	005
		real fx, gx	006
		character*10 fmt	007
		fmt="(a)"	008
		write (*,fmt,ADVANCE="NO") " Numarul nodurilor de interpolare este = "	009
		read*, n	010
		print*, " Valorile argumentului corespunzătoare nodurilor"	011
		read*, (x(i), i=1,n)	012
		print*, "Valorile functiei in nodurile de interpolare"	013
		read*, (fx(i), i=1,n)	014
		write (*,fmt,ADVANCE="NO") " Valoarea punctului de evaluare a functiei= "	015
		read*, x_eval	016
		write (*,fmt,ADVANCE="NO") " Eroarea de calcul errcalcul= "	017
		read*, errcalcul	018
		write (*,fmt,ADVANCE="NO") "Valoarea maximă a diferentei, epsilon= "	019
		read*, epsilon	020
		do i=1,n	021
		fx(i)= functie(x(i))	022

```
      enddo                                                                                    023
      call interpolare(n,x,fx,x_eval,errcalcul,ierr,fx_interpolat)                             024
      if (ierr.eq.1) then                                                                      025
         print*, " F(", x_eval,")= ", fx_interpolat, " cu o toleranta mai mica decat: “,       026
     *  errcalcul                                                                             027
        else                                                                                   028
         print*, " F(", x_eval,")= ",fx_interpolat, " fara a avea o toleranta mai mica        029
     *  decat: ",errcalcul                                                                    030
      endif                                                                                    031
      val_fx=functie(x_eval)                                                                   032
      if (abs(val_fx-fx_interpolat).le.epsilon) then                                           033
         print*, " Programul Interpolare a trecut testul pentru eps= ", epsilon                034
        else                                                                                   035
         print*, " Programul Interpolare nu a trecut testul pentru eps= ", epsilon             036
      endif                                                                                    037
      end                                                                                      038

      subroutine interpolare(n,x,fx,x_eval,errcalcul,ierr,fx_interpolat)
      implicit none
      integer i, j, ierr, n
      real x_eval
      real x(n), fx(n), errcalcul, fx_interpolat, l(n,n+1)
      do i=1,n
         l(i,1)=fx(i)
            do j=2,i
               l(i,j)=((x_eval-x(i))*l(i-1,j-1)-(x_eval-x(i-j+1))*l(i,j-1))/(x(i-j+1)-x(i))
               if ((l(i,j)-l(i-1,j-1)).le.errcalcul) then
                     fx_interpolat=l(i,j)
                     ierr=1
                     return
                endif
            enddo
      enddo
      fx_interpolat=l(i,j)
      return
      end

      real function functie(x)
      implicit none
      real x
      functie=2.*x-5.
      return
      end
```

Pentru testarea programului Interpolare în cazul celei de a doua funcții se va realiza un nou program, identic cu cel anterior cu excepția subprogramului *functie* în care trebuie modificată expresia scalară numerică corespunzătoare funcției. Noul subprogram *functie* este arătat mai jos.

1÷5	6	7÷72	73÷80
		real function functie(x) **implicit none** **real** x functie=x**2-x+1 **return** **end**	

Din cele arătate mai sus se observă că pentru a putea testa programul *interpolare* trebuie realizat un program de testare dedicat. Dacă dorim să testăm programul pentru funcții diferite trebuie rescris programul.

Scrierea programului de testare şi modificarea acestuia se simplifică, deşi procedeul este asemănător ca cel descris mai sus, dacă există biblioteci de programe ce conțin subprograme funcție şi programul pe care dorim să-l testăm. Produsul software *Developer Studio* permite crearea bibliotecilor de programe.

3. CREAREA ŞI UTILIZAREA BIBLIOTECILOR DE PROGRAME

Pentru crearea unei biblioteci de programe trebuie procedat în felul următor:

- se activează programul *Developer Studio*;
- se activează opțiunea *New* a meniului *File*; prin activarea acestei opțiuni se deschide o casetă de dialog;
- se alege proiectul *Win 32 Static Library*, se indică numele bibliotecii, calea în sistemul de fişiere şi se activează butonul *OK*; se deschide un nou *Workspace* pentru noul proiect cu numele bibliotecii;
- se activează opțiunea *Add to Project* a meniului *Project*; se deschide un nou submeniu şi se alege opțiunea *File*; se deschide o casetă de dialog care permite alegerea fişierului sau fişierelor ce se vor adăuga proiectului;
- se selectează fişierul (fişierele) şi se activează butonul *OK*;
- se activează opțiunea *Build name.lib* a meniului *Build*; se aşteaptă până la încheierea procesului de creare a bibliotecii.

Pentru a adăuga noi programe la o bibliotecă existentă se deschide mai întâi spațiul de lucru (opțiunea *Open Workspace*) cu numele identic cu cel al bibliotecii şi se parcurg ultimile trei etape descrise mai sus.

După creare, pentru a putea apela subprogramele dintr-o bibliotecă trebuie procedat astfel:

- după compilarea programului principal, folosind opțiunea *Add to Project* a meniului *Project* se adaugă spațiului de lucru corespunzător proiectului biblioteca de programe dorită;
- se generează programul executabil cu opțiunea *Build name.exe* a meniului *Build*.

Pentru a exemplifica crearea şi utilizarea bibliotecilor de programe, în vederea testării programului *interpolare* s-a creat biblioteca *testinterp.lib* ce conține trei subprograme dintre care două subprograme funcție, corespunzătoare celor două funcții indicate mai sus, şi o subrutină ce implementează algoritmul de interpolare Aitken-Neville. Cele trei subprograme sunt identice cu cele de mai sus.

Programul pentru testare, când folosim biblioteca *testinterp* va fi format doar din programul principal ce va conține şi instrucțiunile de apel corespunzătoare subrutinei *interpolare*, respectiv subprogramului *functie**i*** unde *i* este numărul *1* sau *2,* ce indică funcția pentru care testăm subrutina *interpolare.*

Programul *test_interpolare* este identic cu cu programul principal arătat mai sus, cu excepția instrucțiunii de atribuire de la linia *022* , care devine:

		fx(i)= functie1(x(i))	022

când se testează subrutina cu prima funcție, sau:

		fx(i)= functie2(x(i))	022

când se testează subrutina cu a doua funcție.

LUCRAREA 17

RELAȚII DE RECURENȚĂ

1. INTRODUCERE

O categorie importantă de probleme necesită pentru a fi rezolvate utilizarea unor *relații de recurență*. Utilizarea relațiilor de recurență este posibilă prin implememtarea acestora în algoritmul de rezolvare sub forma unor structuri iterative. Din punct de vedere matematic, aplicarea unei relații de recurență poate genera o secvență finită sau infinită de calcule, dar algoritmul ce o implementează se realizează într-un număr finit de paşi. Folosirea unei relații de recurență presupune existența unor date inițiale. Astfel, o relație de forma:

$$x_n = f(x_{n-1})$$

defineşte o relație de recurență într-un singur pas şi necesită cunoaşterea unei singure date inițiale, de exemplu x_0.

O relație de recurență mai generală, de tipul:

$$x_n = f(x_{n-1}, x_{n-2}, ..., x_{n-r})$$

defineşte o relație de recurență în r paşi ce necesită r valori inițiale, de exemplu $x_0, x_1, \ldots, x_{r-1}$.

2. CALCULUL COEFICIENȚILOR BINOMIALI

Se presupune că trebuie calculați coeficienții binomiali $C_n^0, C_n^1, ..., C_n^p$ în care n şi p sunt întregi pozitivi cunoscuți. Relația de recurență existentă între doi termeni succesivi este:

$$C_n^k = \frac{n-k+1}{k} \cdot C_n^{k-1},$$

iar valoarea inițială este $C_n^0 = 1$.

Aplicarea relației de recurență permite calculul succesiv al coeficienților binomiali $C_n^0, C_n^1, ..., C_n^p$, folosind o singură valoare inițială, fiind deci o relație de

recurență într-un singur pas.

Relația de recurență se codifică în limbajul Fortran folosind o instrucțiune de atribuire de forma:

```
coef_binom = (n – k + 1)*coef_binom / k
```

care face parte din blocul de instrucțiuni al unei construcții DO cu contor al iterației. Înaintea execuției construcției DO trebuie inițializate valoarea inițială a variabilei *coef_binom*, precum şi cea a variabilei *k* a buclei DO. Secvența de instrucțiuni necesare este următoarea:

```
coef_binom = 1
k = 0
do k = 1, p
    coef_binom = (n – k + 1)*coef_binom / k
enddo
```

În continuare se pune problema modului de reprezentare şi stocare a datelor de intrare şi mai ales a datelor de ieşire. Datele de intrare sunt *n* şi *p*, variabile scalare de tip întreg. Datele de ieşire sunt *k*, variabilă scalară de tip întreg şi valoarea corespunzătoare a coeficientului binomial, *coef_binom*. Coeficienții binomiali pot fi fie o variabilă scalară de tip întreg, sub numele căreia se memorează succesiv valoarea fiecărei coeficient, fie un tablou unidimensional care memorează valorile tuturor coeficienților binomiali. În primul caz transferul datelor din memorie trebuie efectuat imediat după evaluarea valorii fiecărui coeficient, iar în al doilea caz transferul se poate face după calculul tuturor valorilor coeficienților.

Folosirea unei singure variabile scalare *coef_binom* face ca pentru stocarea rezultatelor să fie necesară o singură celulă de memorie, transferul datelor de ieşire efectuându-se imediat după execuția instrucțiunii de atribuire, însă în acest fel se poate calcula teoretic oricare coeficient binomial. Utilizarea unui tablou unidimensional static impune restricții asupra numărului de coeficienți ce pot fi calculați prin numărul maxim de elemente pe care îl poate avea tabloul unidimensional. O soluție este utilizarea unui *tablou alocabil* căruia i se alocă dinamic spațiul de memorie corespunzător, în timpul execuției programului.

O importantă restricție asupra numărului de coeficienți ce pot fi calculați, restricție valabilă pentru ambele variante de programe, este dată de constantele calculatorului, adică valoarea maximă a unui număr ce poate fi reprezentat pe un anumit calculator. În cazul declarării variabilelor ca fiind de tip întreg pot fi calculați pe un calculator personal coeficienții până la maxim coeficientul C_{30}^{14}. În continuare sunt

prezentate cele trei variante ale programului pentru calculul coeficienților binomiali.

```
      program coef_binomial
      implicit none
      integer k, n, p
      integer coef_binom
      character*10 fmt
      fmt="(a)"
      write (*, fmt, advance="no") " Valorile pentru n si p: "
      read*, n,p
      open(unit=1, file="coefbinom.txt",form="formatted",access="sequent
     * ial",status="unknown")
      coef_binom=1
      k=0
      write(1,10) k, coef_binom
      do k=1,p
            coef_binom= (n-k+1)*coef_binom/k
            write(1,10) k, coef_binom
      enddo
10    format(3x,i3,4x,i12)
      close (unit=1)
      end
```

```
      program coef_binomial
      implicit none
      integer k, n, p, pmax
      parameter (pmax=50)
      integer coef_binom(pmax)
      character*10 fmt
      fmt="(a)"
      write (*, fmt, advance="no") " Valorile pentru n si p: "
      read*, n,p
      open(unit=1, file="coefbis.txt",form="formatted",access="sequent
     * ial",status="unknown")
      coef_binom(0)=1
      k=0
      do k=1,p
            coef_binom(k)= (n-k+1)*coef_binom(k)/k
      enddo
```

```
      do k=0,p
            write(1,10) k, coef_binom(k)
10          format(3x,i3,4x,i12)
      enddo
      end
```

```
1÷5  6  7÷72
     *   program coef_binomial
         implicit none
         integer k, n, p
         integer, allocatable :: coef_binom(:)
         character*10 fmt
         fmt="(a)"
         write (*, fmt, advance="no") " Valorile pentru n si p: "
         read*, n,p
         open(unit=1, file="coefbinom.txt",form="formatted",access="sequent
         ial",status="unknown")
         allocate (coef_binom(p+1))
         coef_binom(0)=1
         k=0
         do k=1,p
               coef_binom(k)= (n-k+1)*coef_binom(k)/k
         enddo
         do k=0,p
               write(1,10) k, coef_binom(k)
         enddo
10       format(3x,i3,4x,i12)
         close (unit=1)
         end
```

LUCRAREA 18

PROCEDURI RECURSIVE. PROGRAMUL PUTERI

1. INTRODUCERE

O *relație de recurență* este o formulă prin care un termen al unui şir se poate calcula din unul sau mai mulți termeni precedenți.

Fie primii k termeni ai unui şir (k număr natural), şi anume $u_1, u_2, u_3,\ldots,u_k,\ldots$ sau altfel scris $\{u_k\}_{k\geq 1}$. O relație de forma:

$$u_{n+k} = a_1 \cdot u_{n+k-1} + a_2 \cdot u_{n+k-2} + \ldots + a_k \cdot u_k, 1 \leq m \leq n \quad (1)$$

se numeşte *relație recursivă de ordin k*.

Algoritmii care implementează o (relație de) recurență se numesc *algoritmi recursivi*. Se afirmă că orice *algoritm recursiv* se poate transforma într-un *algoritm iterativ*.

Limbajul Fortran acceptă *proceduri recursive, funcții recursive* şi *subrutine recursive*. Utilizarea unor algoritmi recursivi în locul unor algoritmi nerecursivi are ca dezavantaj creşterea timpului de execuție, ceea ce înseamnă un consum mai mare de resurse (*timp procesror*), însă are avantajul unei descrieri simple şi concise.

2. RELAȚII DE RECURENȚĂ. EXEMPLE

Fie şirul dat prin progresia geometrică definită astfel:

$$u_1 = a,\ u_2 = aq,\ u_3 = aq^2,\ u_4 = aq^3,\ \ldots,\ u_n = aq^n,\ldots \quad (2)$$

pentru care relația recursivă este:

$$u_n = qu_{n-1} \quad (3)$$

în care $k = 1$ şi $a_1 = q$. Relația 3 este o *relație de recurență de ordinul întâi*.

Fie şirul dat prin progresia aritmetică definită în continuare:

$$u_1 = a,\ u_2 = a + r,\ u_3 = a + 2r,\ \ldots,\ u_n = a + (n-1)r,\ \ldots \quad (4)$$

Între doi termeni consecutivi există relația:

$$u_{n+1} = u_n + r \tag{5}$$

care însă nu este de forma (1). Pentru a obține o astfel de relație considerăm relații de tipul 5 pentru doi termeni consecutivi:

$$u_{n+2} = u_{n+1} + r,\ u_{n+1} = u_n + r$$

iar prin diferență rezultă:

$$u_{n+2} = 2u_{n+1} - u_n \tag{6}$$

pentru care $k = 2$, $a_1 = 2$, $a_2 = -1$. Relația 6 este o *relație de recurență de ordinul doi.*

Fie şirul pătratelor numerelor naturale:

$$u_1 = 1,\ u_2 = 2^2,\ u_3 = 3^2,\ \ldots,\ u_n = n^2,\ u_{n+1} = (n+1)^2,\ \ldots$$

Termenul u_{n+1} poate fi scris sub forma $u_{n+1} = n^2 + 2n + 1$ care este echivalentă cu $u_{n+1} = u_n + 2n + 1$. Următorul termen poate fi scris folosind o relație asemănătoare: $u_{n+2} = u_{n+1} + 2n + 3$. Dacă se scad cele două relații se obține:

$$u_{n+2} = 2u_{n+1} - u_n + 2$$

Dacă se măreşte n cu o unitate se obține:

$$u_{n+3} = 2u_{n+2} - u_{n+1} + 2$$

Prin scăderea celor două relații de mai sus se obține:

$$u_{n+3} = 3u_{n+2} - 3u_{n+1} + u_n \tag{7}$$

pentru care $k = 3$, $a_1 = 3$, $a_2 = -3$, $a_3 = 1$. Relația 7 este o *relație de recurență de ordinul trei.*

3. PROGRAMUL PUTERE

Fie două numere naturale n şi m. Se pune problema să se calculeze puterile n^m. Fie şirul puterilor lui n definit astfel:

$$p_0 = n^0,\ p_1 = n^1,\ p_2 = n^2,\ \ldots,\ p_m = n^m,\ \ldots$$

Termenul p_k poate fi scris sub forma $p_m = np_{m-1} = nn^{m-1} = n^m$. Se obține o relație de recurență de ordinul întâi pentru care $k = 1$, $\alpha_1 = n$.

Descrierea în pseudocod a algoritmului iterativ, respectiv a algoritmului recursiv, pentru calculul puterilor numărului natural n este prezentată în continuare:

```
algoritm iterativ

real function putere (n,k);
integer i,k,n;
putere ⟵ 1;
do i = 1,k
    putere ⟵ n*putere;
enddo
end
```

```
algoritm recursiv

recursive real function putere (n,k);
integer k,n;
if (k = 0) then
    putere_n ⟵ 1;
  else
    putere_n ⟵ n*putere(n,k-1);
endif
end
```

Un alt algoritm pentru rezolvarea problemei considerate, utilizabil pentru valori *mari* ale lui k, prin care scade numărul de înmulțiri şi se reduce consumul de resurse, este prezentat mai jos.

Un alt mod de calcul pentru puterile numărului n este următorul:

$$n^k = \begin{cases} \left(n^{k/2}\right)^2 & \text{daca } k \text{ este par} \\ \left(n^{(k-1)/2}\right)^2 & \text{daca } k \text{ este impar} \end{cases}$$

Descrierea în pseudocod a a algoritmului recursiv este următoarea:

```
algoritm recursiv

recursive real function putere (n,k);
integer k,n;
if (k = 0) then
    putere_n ⟵ 1;
  else
    if ( mod(k,2) = 0 ) then
      putere_n ⟵ putere(n, k/2) * putere(n, k/2);
     else
      putere_n ⟵ n*putere(n,(k-1)/2)*putere(n,(k-1)/2);
    endif
endif
end
```

Pentru a compara timpii de execuție a algoritmului iterativ şi a celui recursiv s-a apelat de două ori subrutina intrinsecă *cpu_time* (*time*), care returnează timpul

procesorului, în secunde.

O problemă importantă a programelor formate din mai multe unități de program este corectitudinea apelului unei proceduri externe. Datele necesare compilatorului pentru a genera un apel corect al unei proceduri sunt furnizate de o colecție de proprietăți ale procedurii care sunt cunoscute sub denumirea de *interfața procedurii*. Dacă întreaga colecție de proprietăți este accesibilă compilatorului atunci când acesta întâlneşte un apel de procedură înseamnă că se utilizează o *interfață explicită*. Atunci când într-o unitate de program întâlneşte un apel de procedură externă compilatorul nu posedă un mecanism pentru a accesa codul obiect al procedurii care a fost compilat în mod separat. În acest caz, se utilizează o *interfață implicită*. Interfețele implicite sunt o sursă importantă de erori, mai ales în programele mari cu multe unități de program şi apeluri frecvente.

Limbajul Fortran 90 permite specificarea de interfețe explicite pentru procedurile externe prin intermediul unui *bloc interfață*. Exemple de utilizare a blocurilor interfață sunt prezentate în programele de mai jos, care implementează algoritmii de mai sus.

1÷5	6	7÷72	73÷80

```
program puteri     ! varianta iterativa
implicit none
integer:: n, k
real :: t1, t2
interface
   real function putere(n,k)
        integer, intent (in)::n,k
        integer::i
   end function putere
end interface
write (*,"(A)",advance="no") " Numarul n = "
read*,n
write (*,"(A)",advance="no") " Exponentul k = "
read*,k
call cpu_time(t1)
print*, " Puterea ",k," a numarului ",n," este ", putere(n,k)
call cpu_time(t2)
print*, " Timpul de executie a programului este: ", (t2-t1), " secunde"
end

real function putere(n,k)
implicit none
integer, intent (in) :: n,k
integer :: i
```

```
putere=1
do i=1,k
    putere=n*putere
enddo
return
end
```

```
program puteri     ! varianta recursiva 1
implicit none
integer:: n, k
real :: t1, t2
interface
   recursive real function putere(n,k)
       integer, intent (in)::n,k
   end function putere
end interface
write (*,"(A)",advance="no") " Numarul n = "
read*,n
write (*,"(A)",advance="no") " Exponentul k = "
read*,k
call cpu_time(t1)
print*, " Puterea ",k," a numarului ",n," este ", putere(n,k)
call cpu_time(t2)
print*, " Timpul de executie a programului este: ", (t2-t1), " secunde"
end

recursive real function putere(n,k) results (puteri_n)
implicit none
integer, intent (in) :: n,k
if (k.eq.0) then
   puteri_n = 1
 else
   puteri_n=n*putere(n,k-1)
endif
end
```

```
program puteri     ! varianta recursiva 2
implicit none
integer:: n, k
real :: t1, t2
```

```
interface
    recursive real function putere(n,k)
        integer, intent (in)::n,k
    end function putere
end interface
write (*,"(A)",advance="no") " Numarul n = "
read*,n
write (*,"(A)",advance="no") " Exponentul k = "
read*,k
call cpu_time(t1)
print*, " Puterea ",k," a numarului ",n," este ", putere(n,k)
call cpu_time(t2)
print*, " Timpul de executie a programului este: ", (t2-t1), " secunde"
end

recursive real function putere(n,k) results (puteri_n)
implicit none
integer, intent (in) :: n,k
if (k.eq.0) then
    puteri_n = 1
 else
   if (mod(k,2).eq.0) then
        puteri_n =putere(n,k/2)*putere(n,k/2)
     else
        puteri_n =n*putere(n,(k-1)/2)*putere(n,(k-1)/2)
   endif
endif
end
```

LUCRAREA 19

EXPRESII TABLOU. PROGRAMUL GRAM_SCHMIDT

1. INTRODUCERE

Fie ***V*** un spaţiu vectorial peste un câmp ***K***. O mulţime ***B*** de vectori din ***V*** se numeşte *bază* pentru ***V*** dacă ***B*** este liniar independentă şi generează spaţiul vectorial ***V***.

Un spaţiu vectorial pe care s-a definit un *produs scalar* se numeşte *spaţiu vectorial euclidian*. Pentru spaţiul vectorial euclidian $\boldsymbol{R^3}$ produsul scalar este definit prin funcţia reală:

$$(\boldsymbol{x}, \boldsymbol{y}) = x_1 \cdot y_1 + x_2 \cdot y_2 + x_3 \cdot y_3 \tag{1}$$

Pentru spaţiul vectorial euclidian ***V***, prin *norma euclidiană* se înţelege funcţia definită prin:

$$\|v\| = \sqrt{(\boldsymbol{x}, \boldsymbol{x})} \tag{2}$$

Doi vectori din spaţiul vectorial euclidian ***V*** se numesc *ortogonali* dacă produsul lor scalar este nul. O submulţime a spaţiului vectorial se numeşte ortogonală dacă vectorii săi sunt ortogonali doi câte doi. O mulţime ortogonală se numeşte *ortonormată* dacă fiecare element al său are norma egală cu unitatea. Orice mulţime ortogonală dintr-un spaţiu vectorial euclidian formată din elemente nenule este *liniar independentă*. Dacă spaţiul vectorial euclidian are dimensiunea *n*, atunci orice mulţime ortogonală care conţine *n* elemente nenule este o bază a spaţiului vectorial.

Pentru un spaţiu vectorial euclidian din orice mulţime liniar independentă de vectori din spaţiu vectorial se poate construi o mulţime ortonormată, mulţime ortogonală ale cărei elemente au norma 1.

Procedeul prin care se trece de la o mulţime liniar independentă la o mulţime ortonormată se numeşte *procedeul de ortogonalizare Gram-Schmidt*.

Fie ***V*** un spaţiu euclidian cu dimensiunea *3*, iar (v_1, v_2, v_3) este o bază a lui ***V***. Există o bază ortonormată a lui ***V***, (e_1, e_2, e_3), care generează acelaşi spaţiu ca şi baza (v_1, v_2, v_3).

Mai întâi se construieşte o mulţime ortogonală după care se vor norma elementele acestei mulţimi. Mulţimea ortogonală (w_1, w_2, w_3) se obţine din elementele bazei (v_1, v_2, v_3) în felul următor:

- se consideră $w_1 = v_1$;

- $w_2 = v_2 + k_1 w_1$, în care vectorul w_2 nu este nul deoarece vectorii bazei sunt liniar independenți;
- din condiția de ortogonalitate a vectorilor noii baze rezultă:
  - $0 = (w_2,w_1) = (v_2 + k_1 w_1, w_1)$, din care $k_1 = -(v_2,w_1)/(w_1,w_1)$;
- vectorul $w_2 = v_2 - ((v_2,w_1)/(w_1,w_1))w_1$;
- $w_3 = v_3 + k_1 w_1 + k_2 w_2$, în care vectorul w_3 nu este nul deoarece vectorii bazei sunt liniar independenți;
- din condițiile de ortogonalitate a vectorilor noii baze rezultă:
  - $0 = (w_3,w_1) = (v_3,w_1) + k_1(w_1, w_1)$ şi $0 = (w_3,w_2) = (v_3,w_2) + k_2(w_2, w_2)$, din care se obțin $k_1 = -(v_3,w_1)/(w_1,w_1)$, $k_2 = -(v_3,w_2)/(w_2,w_2)$;
- vectorul $w_3 = v_3 - ((v_3,w_1)/(w_1,w_1))w_1 - ((v_3,w_2)/(w_2,w_2))w_2$.

Mulțimea ortonormată se construieşte prin următorul procedeu:

$$e_i = \frac{w_i}{\|w_i\|}, \text{ i} = 1, 2, 3 \qquad (3)$$

2. PROGRAMUL GRAM-SCHMIDT

Fie $\boldsymbol{R^3}$ spațiul vectorial euclidian canonic cu trei dimensiuni. Se pune problema găsirii unei baze ortonormate cunoscând o bază oarecare a spațiului vectorial euclidian. Pentru aceasta s-a scris programul Fortran numit *gram-schmidt*.

Programul conține şi instrucțiuni de atribuire ce folosesc *expresii tablou*. Limbajul Fortran acceptă şi expresii tablou care conțin operanzi tablouri.

Unui tablou i se poate aplica o operație unară. Rezultă un tablou cu aceeaşi formă, fiecare element al tabloului rezultat are valoarea corespunzătoare operației unare aplicate elementului respectiv.

Programul *gram-schmidt* este arătat în continuare.

1÷5	6	7÷72	73÷80
		program gram_schmidt !aplicabil pentru o baza in R^3 **implicit none** **integer**:: i,n **real***8 :: u(3), v(3), w(3) **real***8 :: x(3), y(3), z(3) **real***8 :: d(3), e(3), f(3) **data** n /3/ **real***8 :: produs, a, b, c, norma_d, norma_e, norma_f **print***,"Introduceti componentele fiecarui vector al bazei (u,v,w)" **print***, " Vectorii trebuie sa fie liniar independenti"	

```
read*, (x(i), i=1,n)
read*, (y(i), i=1,n)
read*, (z(i), i=1,n)
u=x
v=y-(produs(n,y,u)/produs(n,u,u))*u
w=z-(produs(n,z,u)/produs(n,u,u))*u-(produs(n,z,v)/produs(n,v,v))*u
d=u/dsqrt(produs(n,u,u))
e=v/dsqrt(produs(n,v,v))
f=w/dsqrt(produs(n,w,w))
print*
print*, "         Baza ortonormata este urmatoare (d,e,f)"
print*
print*,"              d1,                    d2,                    d3"
print*,(d(i), i=1,n)
print*
print*,"              e1,                    e2,                    e3"
print*,(e(i), i=1,n)
print*
print*,"              f1,                    f2,                    f3"
print*,(f(i), i=1,n)
print*
print*, " Se verifica daca vectorii sunt ortogonali"
a=produs(n,d,e)
if (a.le.1.d-10) then
      a=0.d0
endif
b=produs(n,d,f)
if (b.le.1.d-10) then
      b=0.d0
endif
c=produs(n,e,f)
if (c.le.1.d-10) then
      c=0.0
endif
print*
print*, " Produsul scalar (d,e) este= ", a
print*
print*, " Produsul scalar (d,f) este= ", b
print*
print*, " Produsul scalar (e,f) este= ", c
print*
norma_d= dsqrt(produs(n,d,d))
```

```
norma_e= dsqrt(produs(n,e,e))
norma_f= dsqrt(produs(n,f,f))
print*, " Se verifica daca vectorii sunt normati"
print*
print*, " Norma vectorului d este= ",norma_d
print*
print*, " Norma vectorului e este= ",norma_e
print*
print*, " Norma vectorului f este= ",norma_f
end

real*8 function produs(n,u,v)
implicit none
integer :: i,n
real*8 :: u(n), v(n)
produs=0.
do i=1,n
      produs=produs+u(i)*v(i)
enddo
return
end
```

LUCRAREA 20

UTILIZAREA BIBLIOTECILOR DE PROGRAME. PROGRAM RADACINA

1. INTRODUCERE

1. BIBLIOTECI DE PROGRAME

Dacă inițial un programator scria o mulțime de programe pentru rezolvarea problemelor concrete, prin creşterea complexității problemelor abordate şi a tehnicilor utilizate, s-a impus schimbarea modului de rezolvare problemelor tehnico-ştiințifice cu ajutorul calculatorului. Un mare progres a fost reprezentat de dezvoltarea unor *biblioteci de programe*, ce conțin subprograme (*rutine*) concepute, scrise şi documentate într-un cadru unitar, testate complet şi destinate unei mari comunități de utilizatori.

Există două tipuri de biblioteci de programe: (1) *biblioteci generale*, care acoperă o marie arie tematică din matematică şi statistică, (2) *bibliotecile specializate*, care sunt destinate rezolvării unor anumite categorii de probleme, cum ar fi analiza şi rezolvarea sistemelor liniare, rezolvarea sistemelor de ecuații neliniare etc.

Folosirea bibliotecilor de programe implică realizarea de către programator, pe baza documentației, a programelor proprii ce fac apel la componente ale bibliotecii, rezolvând astfel într-un mod eficient probleme concrete.

Calitatea bibliotecilor de programe se manifestă atât la nivelul componentelor, cât şi la nivelul întregului, prin următoarele atribute: fiabilitate, robustețe, structurare, utilizabilitate, validitate şi portabilitate.

SLATEC este o biblioteca generală de programe ce conține peste 1000 de rutine scrise în limbajul FORTRAN. Acestea sunt rodul cercetărilor intreprinse de-a lungul anilor '80 la institute ştiințifice de mare prestigiu din Statele Unite (Air Force Weapons Laboratory - Albuquerque, Lawrence Livermore National Laboratory-Livermore, Los Alamos National Laboratory-Los Alamos, National Bureau of Standards-Washington, National Magnetic Fusion Energy Computer Center-Livermore, Oak Ridge National Laboratory-Oak Ridge, Sandia National Laboratories-Alburquerque&Livermore). Rutinele sunt concepute într-o formă destul de portabilă, ceea ce a permis implementarea lor pe sisteme din categoria PC, testarea implementării corecte făcându-se cu ajutorul testelor ce însoțesc biblioteca. Domeniile acoperite de rutine sunt deosebit de largi, printre acestea putându-se menționa: Calculul funcțiilor elementare şi speciale, Operații cu vectori şi matrici, Rezolvarea sistemelor liniare, Valori şi vectori proprii, Descompunerea matricilor, Sisteme supra- si sub-determinate, Interpolare, Rezolvarea ecuațiilor neliniare, Optimizare, Diferențiere şi integrare numerică, Ecuații diferențiale

ordinare, Ecuații cu derivate parțiale, Transformate Fourier rapide, Aproximare, Generare de numere pseudo-aleatoare, Sortare etc.

2. REZOLVAREA NUMERICĂ A ECUAȚIILOR

Fie o funcție $f : D \subset R \to R$. Prin soluție a ecuației $f(x) = 0$ se înțelege un element x_0 al mulțimii D pentru care este valabilă proprietatea că $f(x_0) = 0$. Obținerea unei soluții x_0 a ecuației de mai sus, în general, nu este posibilă prin metode analitice. Aproximarea prin metode numerice a rădăcinii unei ecuații rămâne una din alternativele des utilizate pentru rezolvarea unor probleme de tipul celei prezentate.

Dacă *f(x)* este un polinom, ecuația se numeşte *ecuație algebrică*, în caz contrar fiind denumită *ecuație transcendentă*.

În rezolvarea numerică a ecuației de mai sus se disting două etape importante: (1) *separarea rădăcinilor*, (2) *determinarea aproximativă a unei rădăcini* şi evaluarea erorii de calcul.

Metodele obişnuite de separare a rădăcinilor unei ecuații sunt *şirul lui Rolle* sau *şirul lui Sturm*.

Pentru ecuațiile algebrice, teorema fundamentală a algebrei asigură existența unui număr de rădăcini (reale sau complexe), eventual multiple, egal cu gradul polinomului. Datorită proprietăților polinoamelor există metode specifice de determinare aproximativă a rădăcinilor unei ecuații algebrice deosebit de eficiente. Metodele sunt de două tipuri: (1) *metode globale*, cu care se determină toate rădăcinile ecuației, (2) *metode locale* prin care se determină aproximativ, folosind procedee iterative, câte o singură soluție aproximativă localizată anterior.

Din clasa metodelor locale de aproximare a rădăcinilor unei ecuații fac parte şi metodele iterative: (1) *metoda bisecției* (*înjumătățirii intervalului*), (2) *metoda coardei*, (3) *metoda tangentei* (*Newton-Raphson*) etc.

3. PROGRAMUL RADACINA

Se pune problema aflării aproximative a rădăcinii ecuației de mai jos, pe intervalele (1,3), (-1,0). Pentru aceasta se utilizează rutina DFZERO din biblioteca matematica SLATEC.

Ecuația este următoarea:

$$x^3 + 2 \cdot x - x = 0$$

Pentru a putea realiza un apel corect al rutinei *dfzero* se va prezenta mai întâi documentația de utilizare a acesteia.

Linia de definiție a rutinei este următoarea:

SUBROUTINE DFZERO(F,B,C,R,RE,AE,IFLAG)

Subrutina DFZERO determină un zero al funcției F(x) între valorile date B si C, cu o toleranță specificată prin criteriile de oprire a algoritmului (DABS(B-C).LE.2*(RW*DABS(B)+AE)). Metoda utilizată este o eficientă combinație a metodei secantei si bisecției.

Descrierea argumentelor:

- ❖ F, B, C, R, RE si AE sunt parametrii de intrare in dubla precizie.
- ❖ B, C si IFLAG sunt parametrii de iesire. B si C sunt dubla precizie, iar IFLAG de tip intreg).

- F - numele unei proceduri FUNCTION ce determină valorile, în dublă precizie, pentru funcția F(x). Argumentul funcției F trebuie declarat de tip dublă precizie;
- B - unul din capetele intervalului (B,C), atât ca parametru de intrare cât şi ca parametru de ieşire. Valoarea lui B la ieşire este cea mai bună aproximație a rădăcinii funcției;
- C - al doilea capăt al intervalului (B,C), atât la intrare cât şi la ieşire;
- R - o estimare a zeroului funcției F care poate ajuta la creşterea vitezei de convergență. Dacă F(B) şi F(R) au semne inverse, o rădăcină va fi găsită în intervalul (B,R); dacă F(R) şi F(C) au semne inverse, o rădăcină se va afla în intervalul (R,C); altfel, intervalul (B,C) va fi investigat pentru o posibilă rădăcină. Când nici o estimare mai bună nu este cunoscută, este recomandat ca R să fie ales fie B fie C;
- RE - eroarea relativă utilizată pentru RW în criteriile de oprire. Dacă valoarea cerută RE este mai mică decât precizia maşinii, atunci pentru RW se dă o valoare aproximativ egală cu precizia maşinii;
- AE - eroarea absolută utilizată în criteriile de oprire. Dacă intervalul (B,C) conține originea, atunci o valoare nenulă se alege pentru AE;
- IFLAG - un cod de stare. Utilizatorul trebuie să verifice IFLAG după fiecare apel. Controlul este cedat utilizatorului în toate cazurile. XERROR nu furnizează mesaje de eroare în aceste cazuri. Valorile pentru IFLAG returnate după apelul rutinei au următoarea semnificație:
    1. B satisface toleranța cerută de un zero al funcției. Intervalul (B,C) s-a redus la toleranța cerută, funcția schimbă semnul în (B,C) şi F(x) descreşte în valoare când (B,C) se restrânge;
    2. F(B) = 0. Oricum, intervalul (B,C) nu poate fi restrâns la toleranța cerută;

3. B poate fi apropiat unui punct singular al funcției F(x). Intervalul (B,C) s-a restrâns la toleranța cerută, funcția işi schimbă semnul în (B,C), dar F(x) creşte în valoare când (B,C) descreşte;
4. nu s-a găsit nici o schimbare de semn a funcției, deşi intervalul (B,C) s-a restrâns la toleranța cerută. Utilizatorul trebuie să examineze acest caz şi să decidă când B este apropiat unui minim local sau B este apropiat de o rădăcină multiplă sau nici unul din aceste cazuri;
5. prea multe evaluări ale funcției (> 500).

Pentru a determina rădăcinile ecuației de mai sus, pe intervalele considerate, cu ajutorul subrutinei DFZERO, se consideră funcția:

$$F(x) = x^3 + 2 \cdot x - x$$

pentru care se determină zerourile pe aceleaşi intervale ale variabilei x.

1÷5	6	7÷72	73÷80
		program radacina **implicit none** **integer** :: iflag **real***8 :: f, b, c, r, re, ae **external** f **data** re/1.e-6/ae/1.e-5/ **write**(*,10) **format** (/,' introduceti capetele intervalului: ') **read***, b, c r=b **call** dfzero(f,b,c,r,re,ae,iflag) **if** ((iflag.**eq**.1).**or**.(iflag.**eq**.2)) **then** **write** (*,20) b **format** (/,' radacina ecuatiei este: ',e12.6) **else** **print***, 'iflag = ', iflag **end if** **end** **real***8 **function** f(x) **real***8 :: x f=x**3 + 2*x**2 - x **return** **end**	

BIBLIOGRAFIE

1. Băduț, M., *Calculatorul în trei timpi*, Editura Polirom, Iaşi, 2001.
2. Balan, Draga-Maria, Balan, G., *Windows, Word for Windows, Excel*, Editura Promedia Plus Computers, Cluj-Napoca, 1995.
3. Bârsan, T., Burdujan, I., Vrabie, I., *Metode numerice,* Rotaprint I.P.Iaşi, 1990.
4. Boian, Fl.-M., *Sisteme de operare interactive*, Editura Libris, Cluj-Napoca, 1994.
5. Cârstea, Mihaela, Diamandi, I., *Calculatorul pe înțelesul tuturor*, Editura Agni, Bucureşti, 1995.
6. Crstici, B., ş.a., *Matematici speciale*, Editura Didactică şi Pedagogică, Bucureşti, 1981.
7. Dodescu, Gh., Odăgescu, I., Năstase, P., Copos, Cristina, *Compendiu de programare a mini/micro calculatoarelor*, Editura Enciclopedică, Bucureşti, 1993.
8. Frențiu, M., Pârv, B., *Elaborarea programelor*, Editura Promedia, Cluj-Napoca, 1994.
9. Gheorghiu, Anca, *Programarea calculatoarelor electronice*, Editura Victor, Bucureşti, 2003.
10. Kovacs, S., Kovacs, Antonia, *Un PC pentru fiecare*, Editura Microinformatica, Cluj-Napoca, 1993.
11. Mitrana, V., *Provocarea algoritmilor*, Editura Agni, Bucureşti, 1994.
12. Mocanu, M., Marian, Gh., Bădică, C., Bădică, Carmen, *333 probleme de programare*, Editura Teora, Bucureşti, 1993.
13. Patriciu, V.-V.,*Sisteme de operare pentru mini şi microcalculatoare*, Editura Militară, Bucureşti, 1992.
14. Petruş, O., *Fortran 90/95 – Limbaj şi tehnici de programare*, Universitatea Tehnică Iaşi, 2001.
15. Popovici, P., Cira, O., *Rezolvarea numerică a ecuațiilor neliniare*, Editura Signata, Timişoara, 1992.
16. Roşculeț, M., *Algebră liniară, geometrie analitică şi geometrie diferențială*, Editura Tehnică, Bucureşti, 1987.
17. Roşculeț, M., *Analiză matematică*, Editura Tehnică, Bucureşti, 1996.
18. Şerbănați, L.-D., *Limbaje de programare şi compilatoare*, Editura Academiei Române, Bucureşti, 1987.
19. Sima, V., Varga, A., *Practica optimizării asistate de calculator*, Editura Tehnică, Bucureşti, 1986.
20. Udrişte, C., Bucur, C., *Probleme de matematici şi observații metodologice*, Editura Facla, Timişoara, 1980.
21. Vraciu, G., Popa, A., *Metode numerice cu aplicații în tehnica de calcul*, Editura Scrisul românesc, Craiova, 1982.