MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

Daniela Oprescu **Liana Bejan Ienulescu**

INFORMATICĂ

varianta C++

manual pentru clasa a XI-a

• filiera *teoretică*, profil *real*, specializarea *matematică-informatică*
• filiera *vocațională*, profil *militar MApN*, specializarea *matematică-informatică*

NICULESCU

Manualul a fost aprobat prin Ordinul Ministrului Educației și Cercetării nr. 4742 din 21.07.2006, în urma evaluării calitative organizate de către Consiliul Național pentru Evaluarea și Difuzarea Manualelor și este realizat în conformitate cu programa analitică aprobată prin Ordinul Ministrului Educației și Cercetării nr. 3252 din 13.02.2006.

Referenți științifici:
Prof. gr. I **Brândușa Bogdan**
Prof. gr. I **Emma Gabriela Dornescu**

Contribuția autoarelor la realizarea manualului:
Prof. gr. I **Daniela Oprescu**, distinsă cu premiul „Gheorghe Lazăr" clasa I (2006): capitolele 1, 2, 3, 4, 6, 7, 8
Prof. gr. I **Liana Bejan Ienulescu**: capitolul 5

Descrierea CIP a Bibliotecii Naționale a României
OPRESCU, DANIELA
Informatică - varianta C++: manual pentru clasa a XI-a / Daniela Oprescu, Liana Bejan Ienulescu. – București: Editura NICULESCU ABC, 2006
ISBN-10: 973-87842-4-7 ISBN-13: 978-973-87842-4-6

I. Bejan Ienulescu, Liana

004(075.35)

Adresa: B-dul Regiei 6D
060204 – București, România
Tel: (+40)21-312.97.82
(+40)21-312.97.84
Tel/Fax: (+40)21-312.97.83
Call center: (+40)21-314.88.55
E-mail: club@niculescu.ro
Internet: www.niculescu.ro

Redactor: *Georgeta Vîrtic*
Procesare computerizată: *S.C. ARETE COMPUTER DESIGN S.R.L.*

Tipărit la fedprint tipografie
B-dul Tudor Vladimirescu, nr. 31, sector 5, București, ROMÂNIA
Tel.: 411.00.55; 411.47.76 fed@promo.ro

ISBN-13: 978-973-87842-4-6
ISBN-10: 973-87842-4-7

Prezentare

Manualul **C++** respectă programa şcolară în vigoare pentru clasa a XI-a, profilul real, specializarea matematică-informatică.

Programa prevede atât noţiuni pentru formarea competenţelor de programator – tehnici de programare –, cât şi noţiuni pentru formarea competenţelor necesare elaborării algoritmilor – metode de rezolvare a unor clase de probleme.

Având în vedere complexitatea acestui conţinut, autorii au abordat o manieră de prezentare gradată, progresivă, pentru a atenua diferenţa între efortul de asimilare depus în clasele a IX-a şi a X-a în regimul de o oră pe săptămână şi efortul cerut de noul conţinut în regimul de patru ore săptămânal.

Majoritatea capitolelor prevăd recapitulări ale noţiunilor din clasele anterioare sau îndrumări de programare cu scopul de a completa cunoştinţele anterioare cu cerinţele de utilizare a limbajului care nu au putut fi fixate din lipsă de timp.

Fiecare capitol începe prin a-şi prezenta obiectivele şi conţinutul principal. După explicaţiile de bază, capitolul oferă o casetă în care sunt rezumate caracteristicile esenţiale ale noţiunii care face obiectul acelui capitol.

Fiecare noţiune sau grup de noţiuni de bază este urmat de exemple şi exerciţii pentru fixare şi pentru clarificarea eventualelor nelămuriri.

Orice noţiune mai complicată, teoretică sau legată de limbajul de programare, începe prin a fi explicată pe un caz concret. De asemenea, autorii au prevăzut explicaţii pentru a preîntâmpina eventualele confuzii ce se pot instala în cursul unei tratări superficiale a conţinutului.

S-a urmărit finalitatea practică a noţiunilor învăţate în fiecare capitol. În acest sens, au fost introduse – oriunde contextul a permis – aplicaţii interdisciplinare din chimie, biologie, matematică, psihologie, tehnica de calcul, geografie, codificarea informaţiilor.

Exemplele de probleme rezolvate şi programele aferente sunt prezentate gradat în cadrul capitolului – în ordinea dificultăţii – şi fiecare precizează un scop didactic. S-a avut în vedere faptul că disciplina „Tehnologia Informaţiei şi a Comunicaţiilor – *Sisteme de gestiune a bazelor de date*" are prevăzute puţine ore de lucru în laborator, astfel că s-au elaborat programele în aşa fel încât să suplinească această lipsă. În casetele fiecărui program sunt puse în evidenţă liniile cu prelucrările importante atât pentru obiectivul explicativ al rezolvării respective, cât şi pentru modalitatea de programare. Din acest motiv, este necesar ca fiecare program dat ca rezolvare să fie analizat şi încercat.

Exerciţiile, temele şi testele propuse sugerează profesorilor moduri de alcătuire a materialelor pentru probele de verificare. Manualul prezintă aplicaţii complexe care pot deveni proiecte de an. De asemenea, sunt enunţate – şi dezvoltate până la un punct – teme pentru portofoliu.

Unde conţinutul prezentat impunea o analiză a eficienţei programului sau a metodei de rezolvare, au fost date în paralel ambele rezolvări pentru ca elevul să poată compara şi selecta criterii de eficienţă.

Cuprins

PARTEA I

Datele care intervin într-o problemă

Această parte prezintă noţiunile necesare identificării şi prelucrării datelor ce intervin în rezolvarea unei probleme şi vine în continuarea celor învăţate în clasele anterioare.

Introducere

Date de tip adresă

O variabilă utilizată într-un program este caracterizată de unele atribute de care trebuie să se îngrijească programatorul în momentul când defineşte acea variabilă.

Atributele unei variabile

I. Denumirea **(identificatorul)**.
De regulă, denumirea este aleasă astfel încât să sugereze sarcina acelei variabile.

II. Tipul **– mulţimea din care aceasta primeşte valori**.
Pentru calculator, aceasta este o submulţime a mulţimii matematice corespunzătoare.
De exemplu, tipul **unsigned** defineşte valori în submulţimea de numere naturale: [0, 65535].

III. Aria de acţiune **– domeniul de valabilitate a definirii şi existenţei acelei variabile.**
De exemplu, secvenţa:

```
int k=3; if(a<b) {int k=10; a+=k;} cout<<k;
```

va afişa 3, pentru k cunoscut în secvenţa exterioară lui **if**. Pe k=10 îl cunoaşte numai în secvenţa afirmativă a instrucţiunii **if**.

IV. Adresa din memoria internă **– locul la care este rezervat spaţiul ocupat de valoarea variabilei (în exprimare pe scurt – adresa variabilei)**.

V. Lungimea alocată **variabilei –** numărul de octeţi pe care îi ocupă valoarea acestei variabile în funcţie de tipul ei; spre exemplu, un *char* va ocupa un octet, iar un *int* va ocupa doi octeţi.

Adresa reprezintă *numărul octetului cu care începe zona de memorie alocată variabilei.*

Adresa poate fi privită:
– în mod *absolut* (numărul octetului respectiv este calculat faţă de primul octet fizic al memoriei interne, care are numărul 0), *sau*
– în mod *relativ* (numărul octetului respectiv este calculat faţă de un alt reper decât începutul fizic al memoriei, adică faţă de începutul unei zone de referinţă din memoria internă).

Zonarea memoriei interne se face în *pagini* şi *segmente*.

Interesează mai mult noţiunea de segment, adică o zonă de memorie de capacitate de 64 Ko. O modalitate de alocarea pe segmente a memoriei ocupate de un program executabil este ilustrată în figura 1.1

Declarările de date (constante şi variabile) globale conduc la alocări statice, în *segmentul de date*. Declarările de date locale, de parametri şi de apeluri de subprograme conduc la alocări dinamice în *segmentul de stivă al sistemului*. Implicit, segmentul de date este dimensionat la 65 520 octeţi utilizabili, iar segmentul de stivă al sistemului este dimensionat la 16 384

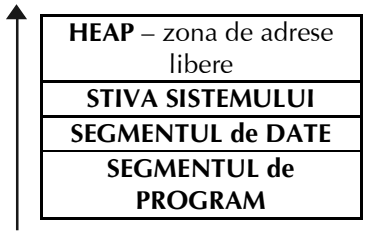


Figura 1.1

octeţi utilizabili[1]. Alte date pot fi stocate în mod dinamic, în zona de adrese libere, **HEAP**, cu ajutorul anumitor comenzi din program.

Operaţiile interne de gestionare a adreselor datelor alocate pentru un program recurg la utilizarea unor regiştri speciali ai microprocesorului:

– **DS** – registrul de adresă de început al segmentului de date;
– **SS** – registrul de adresă de început al segmentului de stivă a sistemului;
– **SP** – registrul de adresă relativă pentru vârful stivei sistemului faţă de baza stivei (considerat 0).

În acest mod, în cadrul unui segment se lucrează cu adrese relative faţă de începutul segmentului, numite deplasamente sau **offset**, iar adresa absolută a unei variabile se calculează de către microprocesor prin însumarea adresei de segment cu adresa **offset**.

Adresa absolută a ultimului element aşezat în stiva sistemului este **(SS)+(SP)**, unde prin scrierea între paranteze se desemnează utilizarea conţinutului respectivelor registre.
Ca reprezentare internă, fiecare adresă absolută ocupă câte 32 de biţi de adresă: primii 16 pentru adresa de segment, iar ultimii 16 pentru offset.
Rezultă că, în total, **o adresă absolută necesită 32 de biţi** pentru a fi memorată (adică patru octeţi).

• Există *două modele de memorie* cu care poate fi compilat un subprogram deoarece acestea influenţează structura adresei de revenire din subprogram. Aceste modele sunt modelul **NEAR** şi modelul **FAR** (explicaţiile aferente depăşesc cadrul manualului).

Tipul de date adresă (*referinţă sau pointer*)

Tipul de date adresă (referinţă, reper, pointer) desemnează *mulţimea de variabile* care pot lua ca valori *adrese* de memorie la care este alocat spaţiul pentru un conţinut de un anume tip numit ***tip de bază***.

Tipul de bază poate fi un tip standard (**int**, **float**, **char** etc.) sau un tip definit de programator prin declaraţia de tip **typedef**.

Figura 1.2

Cu alte cuvinte, *tipul de date adresă* oferă posibilitatea de a acţiona în cadrul programului asupra conţinutului zonelor de memorie alocate utilizând adrese simbolice pentru referirea lor (fără ca programatorul să cunoască efectiv valorile interne ale acelor adrese).

Acest mod de localizare poartă numele de **adresare indirectă** a conţinutului unei zone de memorie.

Facilitatea de a putea acţiona asupra unor zone de memorie prin intermediul adreselor acestora deschide poarta către alocarea dinamică a zonelor de memorie, adică alocarea lor în timpul execuţiei programului. Acest lucru este exploatat atât de *sistemul de operare, în funcţionarea apelurilor de subprograme* (alocarea pe stiva sistemului), cât şi de *programator*, care, prevăzând situaţii în care apar variabile noi pe parcursul rulării programului, are la dispoziţie locul în care să stocheze aceste noi variabile (în zona de adrese libere, **HEAP**).

În figura 1.3 este ilustrată o astfel de situaţie. Când este necesară alocarea unei noi variabile, de exemplu 3,1415… (numărul π) de tipul **double**, pentru stocarea valorii acesteia în **HEAP** se vor repartiza 8 octeţi, iar în segmentul de date se vor repartiza 4 octeţi pentru o variabilă, numită de exemplu **P**, care reţine adresa variabilei **double** din **HEAP.** Fie, de exemplu, în exprimare segment şi **offset**, adresa $0060{:}01A8_{(16)}$ la care trebuie să fie memorată valoarea de tip **double**. Calculată,

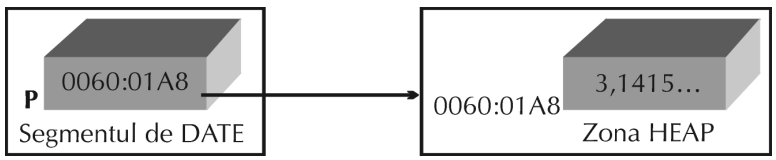


Figura 1.3

[1] Prin opţiuni de compilare impuse programului, se poate stabili segmentul de stivă la orice dimensiune între 1024 şi 65 520 de octeţi.

această adresă are valoarea $007A8_{(16)}$. Variabila **p** este adresa simbolică a zonei de la octetul $007A8_{(16)}$. La octetul de număr absolut $007A8_{(16)}$ începe o zonă de opt octeţi în care este memorată valoarea reală 3,1415…

Definirea variabilelor adresă

Operatorii de referinţă şi de adresă (* şi &)

Definirea unei variabile de tip adresă cere specificarea a două lucruri:
– ***tipul valorii*** ce va fi stocată la adresa simbolică, numit *tip de bază*;
– ***numele variabilei*** care reţine adresa, adică *adresa simbolică* utilizată în program.
Variabila adresă declarată pentru exemplul anterior este:

```
float *p;
```

unde *p* este variabila adresă, iar *float* este tipul de bază.

➢ Operatorul * este **operatorul de referinţă** (sau indirectare) ***către un conţinut de zonă***. Acesta:
– arată că *p* este o variabilă de tip adresă a unei zone care conţine o valoare reală, `float *p;`
– referă conţinutul valoric stocat la adresa **din** p, de exemplu prin construcţia: `*p = 3.1415;`


observă

Caracterul * poate fi scris fie lipit de tipul de bază, fie lipit de variabila adresă sau spaţiat (de exemplu, **int*** a, sau **int** *a, sau **int** * a).

Odată fixat tipul de bază pentru o variabilă adresă, ea nu poate referi adresa unei valori de alt tip diferit de bază.

➢ Operatorul **&**, numit **operator adresă**, oferă modalitatea prin care ***se poate obţine adresa*** unei variabile oarecare, **x**. Adresa obţinută astfel ***se poate înregistra*** într-o variabilă de tip adresă către acelaşi tip de bază ca şi al variabilei **x**.


exemplu

În secvenţa de mai jos se repartizează în segmentul de date variabilele **a** şi **p**, unde **a** este o variabilă simplă întreagă, iniţializată cu valoarea 7, iar **p** este o adresă simbolică a unui loc ocupat de un întreg. După declarările zonelor de lucru, în **p** se încarcă adresa la care a fost repartizat **a**. În consecinţă, operaţia de scriere va afişa valoarea 7 atât ca efect al preluării ei direct din variabila simplă **a**, cât şi ca preluare *a conţinutului adresei memorate în* ***p***. Ultima scriere prezintă pe ecran un rând pe care este o constantă hexazecimală exprimată în **C/C++**, de forma **0xc1c2c3c4**, unde c_i sunt cifre hexazecimale.

De exemplu (fig. 1.4), poate apărea ca valoare de adresă numărul `0xfff4`, ceea ce înseamnă, în baza 10, valoarea de adresă:

$$15*16^3+15*16^2+15*16+4= 65524.$$

Acest lucru înseamnă că valoarea 7 este memorată începând cu octetul 65 524.

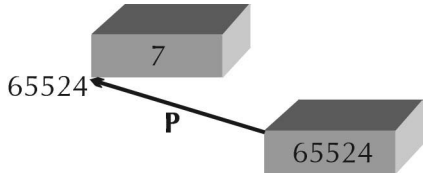


Figura 1.4

```
int a=7, *p;
p=&a;// variabila p primeste adresa variabilei a
cout<<"Valoarea din a="<<a<<endl;
cout<<"Valoarea referita indirect prin adresa din p="<<*p<<endl;
cout<<"Valoarea din p, care este o adresa exprimata in hexazecimal="<<p;
```

Operaţii cu variabilele de tip adresă

• **Atribuire.** Variabilele de tip adresă intră în expresii de atribuire ca orice variabilă, respectându-se însă tipul de bază declarat.


```
char *c,d; //declara variabilele c - de tip adresa catre un caracter
           //si d - de tip caracter
float f=3.14, *afloat=&f; //f - variabila reala, initializata cu 3.14,
                          //iar afloat - variabila adresa catre un
                          //continut real, initializata cu adresa lui f
int a=5,*aa=&a,**aaa=&aa;  //declara variabila întreaga a, initializata cu 5,
                           //variabila adresa catre un continut întreg - aa,
                           //initializata cu adresa lui a si variabila adresa
                           //catre adresa catre un întreg - aaa,
                           //initializata cu adresa aa (dubla indirectare)
```

În exemplul de mai sus se prezintă modul în care este definită adresarea indirectă de două niveluri: se defineşte o adresă simbolică **aaa** în care se încarcă adresa altei adrese simbolice **aa** şi apoi, în aceasta din urmă, se încarcă adresa variabilei propriu-zise, **a**.

Adrese constante. O constantă simbolică de tip adresă este declarată prin construcţia:

```
tip_baza *const = adresa_simbolica;
```


```
int b=5, const * a=&b; //continutul lui a nu se mai poate modifica.
```

În C/C++ adresele din segmentul de date la care sunt alocate **tablourile** (uni- sau multi-dimensionale) sunt *adrese constante* şi sunt *asociate denumirii* tabloului.

Din acest motiv, dacă am avea, de exemplu, declaraţiile: `int a[20],b[20];` operaţia de atribuire globală **a=b** nu este permisă deoarece, intern, operaţia se traduce în încercarea de a schimba adresa lui **a** în adresa lui **b**, ceea ce ar fi un dezastru pentru sistemul de operare. (C/C++ fiind un limbaj de graniţă între limbajul de asamblare şi cel de nivel înalt, operaţiile complexe trebuie rezolvate explicit, nu în mod global).

• **Adunare cu un număr întreg.** Variabila adresă primeşte o nouă valoare rezultată din adunarea unui număr întreg, *n*, la vechea valoare. Cum numărul intervine cu semnul lui algebric, operaţia provoacă o creştere sau o descreştere a valorii anterioare. Intern, operaţia realizează creşterea/descreşterea valorii anterioare cu atâţia octeţi câţi rezultă din produsul lungimii tipului de bază cu numărul întreg *n*, adică:

adresa_nouă=adresa_veche ± sizeof (tip_bază) x n.

• **Comparaţie.** Variabilele adresă pot fi termeni în expresii relaţionale de tip adresă.

• **Parametri.** Variabilele adresă pot fi transmise ca parametri între subprograme ca orice alt tip de parametru (situaţie care va fi întâlnită în capitolul despre subprograme).

• Crearea unui **sinonim al unei variabilei** se poate face explicit prin:

tip_bază & nume_sinonim= nume_variabilă;


```
int a=5; int &b=a;
b=b+7; cout<<b;
      // pentru a s-a creat sinonimul b, pentru care adresa coincide
      // cu lui a;
```

Deoarece:
– pentru structura de tip tablou, limbajul foloseşte denumirea acestuia ca adresă simbolică a începutului zonei compacte în care i s-a alocat spaţiu,
– tabloul este o dată structurată omogenă, elementele sunt de acelaşi tip şi ocupă acelaşi număr de octeţi fiecare,

observă

– pentru un tablou se cunoaşte de la compilare necesarul de spaţiu de memorie din segmentul de date,
– numele tabloului reprezintă şi adresa simbolică de referire la primul element al lui,
atunci rezultă imediat un mod de calcul al adresei oricărui element al tabloului.

exemplu

Dacă se declară: **int** `tabl[10];`
un tablou unidimensional (vector), adresa simbolică este adresa primului element, adică adresa lui `tabl[0]`.

Astfel se explică de ce în C/C++ expresiile de tip indice pentru un element din tablou încep cu valoarea 0: deoarece primul element se află la 0 octeţi distanţă faţă de adresa de început a tabloului.

Ştiind că asupra adreselor se poate aplica adunarea cu un număr întreg, atunci pentru elementul al doilea din tablou calculul de adresă înseamnă `tabl + 1`, deoarece elementul al doilea din tablou este la distanţă de un element faţă de adresa de început.

Generalizând: dacă se doreşte elementul de indice **i**, atunci adresa lui este **tabl+i**. Conţinutul acestui element se poate obţine prin ***(tabl+i)**. Intern, calculul se face prin adăugarea a *i x sizeof(tip_bază)* octeţi la adresa de început a tabloului.

Comutativitatea adunării aplicată calculului de adresă pentru un element de indice **i** poate produce formulările: **tabl[i] = tabl+i = i + tabl = i[tabl].**

rezolvă

1. Fie declaraţiile: `int a=5,*b=&a;`
Alegeţi care dintre exprimările următoare reprezintă conţinutul variabilei `a` şi justificaţi răspunsul prin desen.
a) `&a`; b) `*b`; c) `*&b`; d) `&b`.

2. Stabiliţi ce se va afişa în urma executării secvenţei: `int a=5;*&a=7; cout<<a;`
a) 5; b) eroare la compilare; c) 7; d) un număr hexazecimal.

3. Vectorul `a` conţine în primele patru elemente numerele naturale 0, 1, 2 şi 3. Având declararea: `unsigned *b`; precizaţi, cu justificarea răspunsului, ce se afişează după executarea secvenţei următoare: `b=a+2; cout<<*b;`
a) 3; b) 2; c) număr hexazecimal; d) codul unei erori de execuţie.

4. Vectorul a conţine în primele patru elemente numerele naturale 0, 1, 2 şi 3. Având declararea: `unsigned *c`, precizaţi, cu justificarea răspunsului, ce se afişează după executarea secvenţei următoare: `c=&a[3]; cout<<c<<' '<<*c;`
a) 3; b) număr hexazecimal; c) 3 şi număr hexazecimal; d) 3 3.

5. Se consideră următoarea secvenţă:

```
int a=1,b=2,c=3,*p=&a,*q=&b; *p+=c; *q+=c; cout<<a<<b<<c;
```

Stabiliţi care dintre variantele de mai jos se va afişa şi justificaţi răspunsul ales:
a) 123; b) 456; c) 333; d) 453.

Capitolul 1 Tablouri bidimensionale

În acest capitol veţi învăţa despre:

- Modul de structurare a datelor omogene pe două niveluri ierarhice
- Proiectarea şi parcurgerea unui tablou bidimensional (matrice)
- Prelucrări specifice elementare aplicate acestei structuri

1.1. Recapitulare – tipuri structurate de date

Limbajul **C/C++** oferă instrumente performante de definire şi utilizare a informaţiilor compuse sau structurate, prin **tipurile de structuri** de care dispune:

1. structura de tip tablou;
2. structura de tip şir de caractere;
3. structura de tip articol;
4. structura de tip fişier.

• *Primele trei tipuri* se referă la structurarea datelor în **zone** ale memoriei interne. **Spaţiul** necesar trebuie să fie cunoscut de către compilator, din acest motiv trebuie ţinut cont că dimensiunea maximă a zonei de memorie alocată unei structuri este de 65 520 de octeţi.

• *Al patrulea tip* se referă la structurarea datelor pe suport extern, care, faţă de memoria internă, se poate considera nelimitat şi permanent. În liceu se studiază numai fişierele text.

Tipurile structurate apar prin *compunerea* informaţiilor (datelor) de tip elementar sau structurat, la rândul lui. Din acest motiv, pentru un tip de date structurate *trebuie specificate două caracteristici*:

– **tipul componentelor;**

– **metoda de structurare (metoda de compunere)**.

Deoarece aceste caracteristici, în limita unor restricţii, sunt la alegerea programatorului,

*tipurile structurate **nu** se mai pot numi tipuri standard, recunoscute şi alocate automat de către compilator la simpla apariţie a unui cuvânt rezervat tipului respectiv.*

Dacă pentru o variabilă de tipul **int** sunt alocaţi automat doi octeţi, pentru un grup (tablou unidimensional) de elemente de tip **int** nu este alocată memorie *decât în momentul* în care se declară câte elemente sunt în grupul respectiv. Acest număr maxim de elemente este o informaţie comunicată compilatorului de către programator.

Clasificarea structurilor de date după metoda de structurare

a. Structuri omogene, în care fiecare element are aceleaşi caracteristici ca toate celelalte din grup.

Tablourile fac parte din acest tip de structură. Tablourilor unidimensionale li se mai spune tablouri liniare[1] . Ele sunt repartizate în memorie element cu element, într-o *zonă continuă*.

Adresa de memorie la care începe zona ocupată de tablou este încărcată ca valoare a variabilei ce denumeşte tabloul.

Pentru definirea structurii de tablou este nevoie să se cunoască:

• *tipul elementelor* şi

• *numărul maxim de elemente* care pot apărea în prelucrările cerute de rezolvarea generală a problemei care le foloseşte.

[1] În matematică se numesc vectori desemnând, pe scurt, coordonatele unui versor într-un spaţiu cu *n* dimensiuni. De exemplu, *v* = (3, –1, 0, 2) reprezintă coordonatele unui vector linie în spaţiul cu 4 dimensiuni.

exemplu

O declarare de forma: `long a [20000];`
nu este corectă din punctul de vedere al spaţiului ocupat: 20000 x 4 = 80000 de octeţi depăşesc segmentul de date.

Tipul de structură omogenă permite accesul la fiecare element printr-un acelaşi calcul general de *raportare a locului elementului dorit faţă de începutul tabloului*. Astfel, fiecare componentă are un număr de ordine, ***indice***, prin care se precizează câte elemente sunt înaintea ei faţă de începutul zonei tabloului.

exemplu

Fie declaraţia de tablou unidimensional: `unsigned a[5];` Amplasarea în memorie (fig. 1.5) va folosi o zonă de 5 x 2 octeţi care va începe din locul (adresa) **a**. ***Numele*** tabloului este ***adresa*** lui în

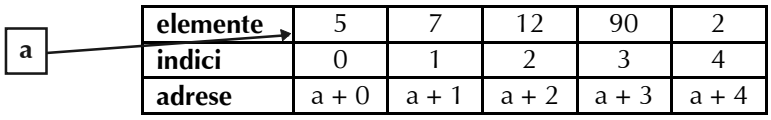


Figura 1.5

exprimare simbolică. Astfel, a+0 este locul (adresa) primului element, iar 0 este indicele lui. Pentru al doilea element, locul (adresa) este a+1, deoarece există un element distanţă de la el până la începutul zonei tabloului. Dacă se doreşte, spre exemplu, utilizarea valorii celui de-al treilea element, atunci se poate scrie a[2] sau *(a+2) şi se va folosi, astfel, valoarea 12 din tablou.

Dacă regula de structură a tabloului impune reperarea elementelor după un singur indice, atunci tabloul se numeşte **unidimensional**; dacă elementele sunt reperate, fiecare, după mai mulţi indici (coordonate), atunci tabloul este **multidimensional** (are mai multe dimensiuni – de exemplu, un tablou ale cărui elemente ar conţine coordonatele unor puncte în spaţiul cu trei dimensiuni, este un tablou tridimensional).

b. Structuri eterogene, în care fiecare element ***poate*** avea alte caracteristici decât celelalte. Aceste structuri se vor descrie în Capitolul 3.

1.2. Tablourile bidimensionale

1.2.1. Metoda de structurare

Un *tablou bidimensional este o* ***structură de tip tablou unidimensional*** *ale cărei* ***componente sunt tablouri*** *unidimensionale de acelaşi tip.*

Din acest punct de vedere se poate înţelege foarte uşor modul în care elementele tabloului bidimensional se stochează în memoria internă. Se va ocupa o zonă compactă de tablouri unidimensionale (elementele) puse cap la cap în continuare, în octeţi succesivi[1] .

exemplu

Dacă se ia în considerare tabloul ***a***, declarat astfel:

```
typedef  unsigned linie[4];
liniea[3];
```

atunci zona ocupată de variabila ***a*** este formată din *trei zone succesive* numerotate cu 0, 1 şi respectiv, 2, de câte 4 valori tip *unsigned* (adică de 4 x 2 = 8 octeţi fiecare), zone care se mai numesc şi *liniile tabloului bidimensional*. Fiecare dintre cele trei linii fiind compusă din câte 4 valori de tip *unsigned*, rezultă că tablolul conţine 12 componente elementare şi ocupă în total 24 de octeţi.
Să presupunem că cele 12 valori sunt chiar numerele de la 0 la 11. Amplasarea lor în memoria internă va fi ca în figura 1.6.

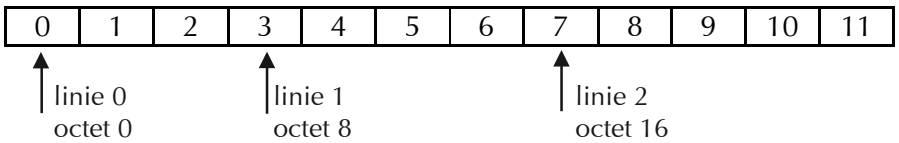


Figura 1.6

[1] Acest mod de aşezare compactă, continuă, în octeţi succesivi, se întâlneşte şi la alte structuri de informaţii şi poartă numele de aşezare în **zonă contiguă**.

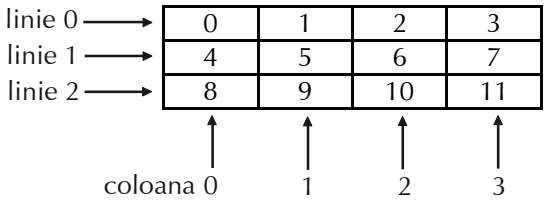


Figura 1.7

Dacă se doreşte prelucrarea unui element, spre exemplu valoarea 4, trebuie să se indice linia din care face parte şi locul lui 4 în cadrul liniei, adică linia de indice 1 şi elementul de indice 0 din cadrul liniei. Deoarece structura formată este fixată, adică ***toate liniile au acelaşi număr de elemente*** (în exemplul luat fiind 4), atunci pentru utilizator ea devine o repartizare dreptunghiulară[1] cu aspect de matrice, sau de tabel omogen (fig. 1.7).

Din figura 1.7 se observă cum aşezarea elementelor linie sub linie generează pe verticală o structură de coloană. Apare astfel evident modul de identificare a unui element prin menţionarea celor două coordonate: ***linia*** şi ***coloana*** în care este amplasat. Deci referirea la valoarea 4 din tablou se va face, în limbajul **C/C++**, prin: **`a[1][0]`.**

1.2.2. Declararea și parcurgerea unui tablou bidimensional

Declararea unui tablou bidimensional se poate face în **C/C++** astfel:

a) Pornind de la definiţia de tablou unidimensional, în care fiecare element este un tablou unidimensional, la rândul său (pe scurt: tabloul bidimensional este un ***vector de vectori***):

```
typedef tip_element   nume_linie[numar_elemente_din_linie];
nume_linie        nume_tablou[numar_linii];
```

unde `numar_elemente_din_linie` şi `numar_linii` sunt **constante întregi pozitive** care reprezintă valorile maxime de alocare a spaţiului necesar.

Declararea:

```
typedef float medii [18];
medii elevi[32];
```

defineşte o structurare a informaţiilor referitoare la mediile generale ale elevilor unei clase, prezentate sub forma unui tabel în care pentru fiecare dintre cei 32 elevi există un rând completat cu cele 18 medii ale sale. Deci variabila **`elevi`** este un tablou unidimensional de 32 de elemente, iar fiecare element al său este de tipul **`medii`**.

b) Pornind de la aspectul de ***spaţiu dreptunghiular***, cum este perceput mai uşor tabloul de către utilizator:

```
tip_element  nume_tablou[numar_linii][numar_coloane];
```

unde `numar_linii` şi `numar_coloane` sunt **constante întregi pozitive** care reprezintă valorile maxime de alocare a spaţiului necesar.

Declararea

```
float clasa[32][18];
```

defineşte aceeaşi structură bidimensională din exemplul de mai sus, dar aici este pus în evidenţă tot grupul – **`clasa`** – pentru care există 32 de linii a câte 18 elemente reale fiecare.

Parcurgerea unui tablou bidimensional presupune trecerea prin fiecare linie a acestuia.
Parcurgerea unei linii din tablou presupune trecerea prin toate elementele ei văzute acum drept coloane.
Referirea unui element al tabloului se face, de obicei, prin exprimarea:

```
nume_tablou[indice_de_linie][indice_de_coloana]
```

t[0][0]	t[0][1]	t[0][2]	…	t[0][n-1]
t[1][0]	t[1][1]	t[1][2]	…	t[1][n-1]
…	…	…	…	…
t[m-1][0]	t[m-1][1]	t[m-1][2]	…	t[m-1][n-1]

Figura 1.8

Vizualizarea în forma dreptunghiulară, matriceală, a unui tablou bidimensional, *t*, cu *m* linii şi *n* coloane va genera configuraţia privind accesul la elementele lui prezentată în figura 1.8.

[1] Un caz particular al tabloului bidimensional este definit în matematică sub noţiunea de **matrice**, prin care se înţelege acea structură în care un element nu poate fi izolat de grupul pe care îl formează cu elementele de pe linia şi coloana pe care este plasat iniţial şi mutat în alt grup, chiar dacă îşi schimbă locul în grup.

1. Dacă, pentru un tablou bidimensional cu *m* linii şi *n* coloane, conţinând elemente de un anume tip şi, pentru un *i* ($0 \le i < m$) şi un *j* ($0 \le j < n$) fixate, dorim să ştim numărul de ordine al unui element dintre cele $m \cdot n$ elemente, atunci se face un simplu calcul:

rang_element = $i \cdot n + j$, unde *rang* ia valori de la *0* la $m \cdot n - 1$.

2. Invers, dacă se cunoaşte *rangul* elementului şi se doreşte aflarea perechii *(i, j)* prin care se desemnează coordonatele elementului în cadrul tabloului, atunci:

$i \leftarrow rang / n$, iar $j \leftarrow rang \ \% \ n$.

Dacă definim prin ***P*** o prelucrare asupra *tuturor* elementelor tabloului, atunci în limbajul de programare se va proiecta o secvenţă de tipul:

```
for ( i = linie_iniţială; i<  linie_finală; i++)
    for ( j=  coloană_iniţială; j< coloană_finală; j++)
        …P…nume_tablou[i][j]
```

În particular, pentru un tablou *a* cu *m* linii şi *n* coloane, secvenţa va fi scrisă:

```
for(i=0;i<m;i++)
    for(j=0;j<n;j++)
        ...P...a[i][j]
```

1.3. Tabloul pătratic – caz particular de tablou bidimensional

În situaţia în care *numărul de linii este egal cu numărul de coloane*, adică **m=n** din notaţiile anterior folosite, se vorbeşte despre un ***tablou pătratic*** sau ***matrice pătratică*** (fig. 1.9).

t[0][0]	t[0][1]	t[0][2]	…	t[0][n-1]
t[1][0]	t[1][1]	t[1][2]	…	t[1][n-1]
…	…	…	…	…
t[n-1][0]	t[n-1][1]	t[n-1][2]	…	t[n-1][n-1]

Figura 1.9

Într-un tablou pătratic se pot defini grupările de elemente numite ***diagonalele structurii*** – diagonala principală şi diagonala secundară, în aceeaşi manieră ca şi la matematică.

Diagonala principală grupează elementele de la colţul NV către colţul SE ale structurii pătratice. În notaţiile folosite până acum, gruparea înseamnă de la elementul **t[0][0]** la elementul **t[n-1][n-1]** (fig. 1.10).

Se observă că elementele diagonalei principale au indicele de linie egal cu indicele de coloană. Proprietatea este cunoscută din matematică şi poate fi exprimată în notaţiile de mai sus astfel:

t[i][i],

fără a mai fi necesară o altă variabilă pentru indicele de coloană.

t[0][0]	t[0][1]	…	t[0][k]	…	t[0][n-1]
t[1][0]	t[1][1]	…	t[1][k]	…	t[1][n-1]
…	…	…	…	…	…
t[k][0]	t[k][1]		t[k][k]	…	t[k][n-1]
…	…	…	…	…	…
t[n-1][0]	t[n-1][1]	…	t[n-1][k]	…	t[n-1][n-1]

Figura 1.10

Diagonala secundară grupează elementele de la colţul NE către colţul SV ale structurii pătratice. În notaţiile folosite până acum, gruparea înseamnă de la elementul indicat prin **t[0][n-1]** la elementul **t[n-1][0]** (fig.1.11).

Se observă că elementele diagonalei secundare au indicele de linie egal cu simetricul indicelui de coloană în cadrul şirului de valori ale indicilor: 0, 1, 2, …, $n-1$. Proprietatea poate fi exprimată în notaţiile de mai sus astfel:

t[0][0]	…	t[0][k]	…	t[0][n-2]	t[0][n-1]
t[1][0]	…	t[1][k]	…	t[1][n-2]	t[1][n-1]
…	…	…	…	…	…
t[k][0]		t[k][k]	…	t[k][n-2]	t[k][n-1]
…	…	…	…	…	…
t[n-1][0]	…	t[n-1][k]	…	t[n-1][n-2]	t[n-1][n-1]

Figura 1.11

t[i][n-1-i] ,

fără a mai fi necesară o variabilă pentru indicele de coloană.

Figura 1.12

În matricele pătratice diagonalele delimitează anumite triunghiuri de elemente:
– triunghiul inferior diagonalei principale, respectiv superior;
– triunghiul inferior diagonalei secundare, respectiv superior;
– triunghiurile de elemente delimitate de ambele diagonale: nord, est, sud şi vest (fig. 1.12).

Caracteristicile tabloului bidimensional

I. Este o structură **omogenă**, compusă din date organizate pe două niveluri ierarhice.
II. Echivalentul de structură din punct de vedere matematic este **matricea.**
III. Componentele tabloului se pot localiza prin **doi indici**, `a[i][j]`, unde *i* este indicele de linie şi *j* este indicele de coloană.
IV. Un element se poate localiza şi prin **rangul** său, care este numărul lui de ordine în cadrul tabloului.
V. Un tablou bidimensional poate conţine drept elemente **o dată structurată** (tablou, şir de caractere, structură etc.).
VI. **Alocarea** zonei de memorie pentru o variabilă tablou bidimensional se face în zone de octeţi succesive, conform numărului de elemente din cadrul liniilor.
VII. **Declararea** unei variabile tablou bidimensional se face în modul următor:
tip_element denumire_tablou[constanta nr_linii][constanta nr_coloane]; sau
typedef *tip_element denumire_linie[nr_coloane]; denumire_linie denumire_grup_linii[nr_linii];*
VIII. Cu elementele unui tablou bidimensional se pot realiza toate **operaţiile permise** tipului acelor elemente.
IX. Tabloul **pătratic** conţine vectori de tip **diagonale**, diagonala **I** cu elementele $\mathbf{a}_{i,i}$ şi diagonala a **II**-a cu elementele $\mathbf{a}_{i,\,n-i-1}$.

Testul 1

(Câte un punct pentru fiecare subiect, două puncte din oficiu)

1. Determinaţi corectitudinea următoarelor declarări de tablouri bidimensionale:
a) `int t[25][-3];` **b)** `float a[10][7];` **c)** `int M[n][m];`
d) `unsigned p[14][2.5];`

2. Care dintre următoarele variante reprezintă declarări de tablouri de maximum 30 de elemente reale?
a) `typedef int x[20]; x t[10];` **b)** `float a[30][1];`
c) `int a[float][30];` **d)** `float[5][6];` **e)** `float a[10][3];`
f) `typedef float m[6]; m n[5];` **g)** `float m[6][5];`

3. Fie declaraţia următoare: `int T[12][2];` Pentru a indica elementul de pe linia a patra şi coloana a doua se va scrie:
a) `T[4][2];` **b)** `T[2][4];` **c)** `T[3][1];` **d)** `T[1][3].`

4. Fie declaraţia de tablou de la testul 3. Pentru a indica elementul al 5-lea din tablou se va scrie:
a) `T[2][5];` **b)** `T[2][0];` **c)** `T[3][1];` **d)** `T[1][5].`

5. Fie declaraţia de tablou de la testul 3. Elementele `T[0][0]` şi `T[11][1]` desemnează capetele:
a) diagonalei I; **b)** diagonalei a II-a; **c)** liniei I; **d)** coloanei I;
e) nici un răspuns anterior nu e corect.

6. Într-o matrice pătratică, de **n** linii şi **n** coloane, triunghiul superior diagonalei **I** se referă la elementele din:
a) triunghiurile **N** şi **V** formate de diagonale; **b)** triunghiurile **N** şi **E** formate de diagonale;
c) triunghiul **N** sau **E** format de diagonale; **d)** toate elementele **a[i][j]** cu **i=0,1,..n** şi **j=i+1...n;**
e) toate elementele **a[i][j]** cu **i=1,..n-1** şi **j=i+1...n;**

7. Fie declaraţia următoare: `int T[12][2];` Stabiliţi al câtelea element din tablou se indică prin **T[5][1]:**
a) al 5-lea; **b)** al 9-lea; **c)** al 6-lea; **d)** al 10-lea; **e)** al 12-lea.

8. Fie declaraţia următoare: `typedef float m[21]; m n[10].` Specificaţi numărul de elmente ale tabloului **n**:
a) 200; **b)** 10; **c)** 210; **d)** 21.

1.4. Referirea elementelor tabloului cu ajutorul adreselor


exemplu

Fie declaraţia de tablou bidimensional: **int** t [3][2];
Amplasarea în memoria internă a elementelor, conform celor două coordonate, este dată în figura 1.13, unde este prezentat şi un exemplu numeric al conţinutului zonelor.

23	15	–7	45	–605	123
t[0][0]	t[0][1]	t[1][0]	t[1][1]	t[2][0]	t[2][1]

Figura 1.13

Adresa simbolică a tabloului **t** este **şi** adresa primului său element, adică a lui **t[0][0]**, unde este stocată valoarea 23. Ştiind că alocarea tabloului se face într-o zonă compactă, linie după linie şi că o matrice este un vector de linii, atunci *fiecare linie* poate fi desemnată prin indicele ei în vectorul de linii: **t[0]**, **t[1]** şi **t[2]**, pentru exemplul din figura 1.13.
Un element dintr-o linie, de exemplu valoarea 123, amplasat în **t[2][1]**, va putea fi accesat prin adresarea ***(t[2] +1)**. Înlocuind mai departe pe **t[2]** cu exprimarea lui prin adresă, ***(t+2)**, se poate scrie în final adresa valorii 123 astfel: ***(*(t+2)+1)**.
Deoarece o linie are două elemente, calculul de adresă pe care îl face sistemul pentru exprimarea ***(*(t+2) +1)** este: **t+2 x 2 +1**. Se ajunge astfel, în calculul intern de adresă, ca valoarea 123 să fie reperată prin adresa **t+5**.

În general, pentru un tablou **t** declarat, valoarea unui element de coordonate **i** şi **j** este localizată prin exprimări de adresă de forma:

t[i][j], sau ***(t[i]+j)**, sau ***(*(t+i)+j)**, sau **(*(t+i))[j]**.

Dacă **n** este numărul de coloane ale tabloului bidimensional (matricea) **t**, atunci un element **t[i][j]** va avea raportarea faţă de începutul **t** calculată prin **t+i*n+j**.

Programul de mai jos prezintă două moduri de acces la un element al matricei prin calculul de adresă:

```
#include<iostream.h>
void main()
{int a[4][3],i,j,k=0,*p;
  for (i=0;i<4;i++)
 for (j=0;j<3;j++) a[i][i]=++k;
p=a[0];// se foloseste adresa de lucru pentru ca a e constant
cout<<*(*(a+2)+2)<<endl; //prima modalitate
cout<<*(p+2*3+2)<<endl; //a doua modalitate
}
```

1.5. Prelucrări elementare ale tablourilor bidimensionale

a) Încărcarea cu valori a elementelor tabloului

1º ***La declararea tabloului – declarare cu iniţializare***
Odată cu declararea tabloului se poate face şi iniţializarea lui cu valori, respectând următorul ***model general***:

```
tip_element  nume_tablou [nr_linii][nr_coloane]={lista de valori};
```


observă

a) Dacă se doreşte iniţializarea cu zero a *tuturor* elementelor, atunci se va folosi modelul:

```
tip_element  nume_tablou [nr_linii][nr_coloane]={0};
```

b) Dacă se doreşte precizarea explicită a elementelor fiecărei linii, sau tabloul bidimensional este definit prin forma – vector de vectori – atunci se poate folosi modelul:

```
tip_element  nume_tablou [nr_linii][nr_coloane]=
{{lista din linia 0},{lista din linia 1},...,{lista din linia m-1}};
```

1. `typedef float tabl[3];`
   `tabl a[2] ={{14.5,25.3,100},{2.9,3,56.7}};`
2. `int b[2][2]={ 1, 2, -4, 0);` sau `int b[2][2]={ {1, 2},{ -4, 0});`
3. `unsigned este[2][4]={0};`

2º *Prin operaţia de atribuire – expresie de calcul sau copiere din alt tablou*

Ca şi la tabloul unidimensional, acest lucru se face prin calcul individual element cu element:

`a[i][j]= expresie;`

1. Se consideră un tablou pătratic de **n** linii şi **n** coloane, unde $\mathbf{n}_{max}$=**20**. Se doreşte completarea elementelor tabloului cu numerele naturale de la **1** la $\mathbf{n}^2$.

Fie **n**=4. Conţinutul tabloului va fi cel din careul alăturat (fig. 1.14).
Rezolvare. Se va considera o variabilă, **k**, în care se vor genera, pe rând, numerele şirului natural 1, 2, …, n^2. Valoarea generată în **k** va fi înregistrată în elementul curent, de coordonate **i** şi **j**, din cadrul tabloului. Pentru a nu se depăşi spaţiul alocat tabloului, programul repetă citirea valorii pentru **n** până când valoarea este **1 < n < 21**.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Figura 1.14

```
#include<iostream.h>
void main()
{ unsigned  v[20][20],k=1,n,i,j;
 do
   {cout<<"Dati numarul de linii ";
    cin>>n;
   }while(n<2||n>20);

for(i=0;i<n;i++)
   for(j=0;j<n;j++)
      v[i][j]=k++;
}
```

2. Se generează în variabila **x** numere aleatoare naturale, $\mathbf{x}_{max}$=**20**. Se doreşte înscrierea acestora într-un tablou **t**, cu **m** linii şi **n** coloane. Apoi, într-un alt tablou **u**, se vor copia toate numerele din **t** care sunt mai mici decât 11, iar pentru cele mai mari de 10 se va înregistra valoarea zero.

Dacă m=2, n=3 şi numerele generate sunt 3, 7, 19, 2, 3, 10, atunci tablourile **t** şi **u** vor avea următorul conţinut fiecare (fig. 1.15):
Rezolvare. Se vor considera două variabile de tip indici, **L** şi **C**, care vor fi folosite pentru ambele tablouri simultan.
Astfel, un element **t[L][C]** <11 va fi copiat în **u[L][C]**, iar un element **t[L][C]**>10 va produce înregistrarea valorii zero în **u[L][C]**.

matricea **t**			
	3	7	19
	2	3	10

matricea **u**			
	3	7	0
	2	3	0

Figura 1.15

```
#include<iostream.h>
#include<stdlib.h>
void main()
{unsigned
t[10][10],u[10][10]={0},m,n,L,C;
randomize();
do
  {cout<<"Dati numarul de linii ";
   cin>>m;
   cout<<"Dati numarul de coloane ";
   cin>>n;
}while(n<2||n>10||m<2||m>10);
//tabloul t se genereaza aleatoriu
 for(L=0;L<m;L++)
      for(C=0;C<n;C++)
            t[L][C]=random(20);
for(L=0;L<m;L++)
  for(C=0;C<n;C++)
    if(t[L][C]<11)u[L][C]=t[L][C];
  }
```

3º *Prin citire din mediul extern*

Încărcarea valorilor din mediul extern se face folosind fluxul de intrare de date:
– standard (din fişierul text creat de la tastatură);
– desemnat de utilizator, prin fişierul text propriu.

• *Intrarea de la tastatură*

Fişierul standard de intrare este gestionat de fluxul **cin**. În primul rând trebuie ca programului să i se dea numărul de linii şi numărul de coloane pentru care va primi apoi datele. Aceste numere, fie **m**, respectiv **n**, trebuie să fie numere naturale, cu valori mai mari sau egale cu 2 (pentru a exista structura bidimensională) şi mai mici sau egale cu valorile constantelor date ca **limite** la declararea tabloului.

Deoarece pentru un tablou cu **m** linii şi **n** coloane trebuie introduse **m** × **n** date, este necesar ca utilizatorul să fie anunţat care element al matricei este la rând pentru a primi valoare.

exemplu

Presupunem că s-au alocat maximum 20 de linii şi maximum 30 de coloane prin declararea

```
int V[20][30];
```

Atunci ***secvenţa pentru citirea valorilor*** elementelor se scrie astfel:

```
do
  {cout<<"Tastati numarul de linii ale matricei, minimum 2 si maximum 20: ";
   cin>>m;
  }while (m<2|| m>20);
do
  {cout<<"Tastati numarul de coloane, minimum 2 si maximum 30: ";
   cin>>n;
  }while(n<2||n>30) ;
for(i=0;i<m;i++)
  for(j=0;j<n;j++)
     {cout<<"V["<<i+1<<"]["<<j+1<<"]=";
      cin>>V[i][j];
     }
```

Mesajul:

```
cout<<"V["<<i+1<<"]["<<j+1<<"]=";
```

se mai poate formula şi astfel:

```
cout<<"Dati elementul de linie "<<i+1<<" si coloana "<<j+1<<"= ";
```

• *Intrarea din fişierul utilizatorului*

De regulă, fişierul utilizatorului este alcătuit astfel:

– pe prima linie sunt valorile pentru **m** şi **n**, separate printr-un spaţiu;

– pe următoarele **m** linii sunt câte **n** valori, de asemenea separate printr-un spaţiu între ele.

În această situaţie se presupune că sunt corecte valorile pentru **m** şi **n** şi nu mai trebuie desfăşurat dialogul cu utilizatorul. ***Secvenţa de mai jos*** citeşte datele dintr-un fişier numit **matrice.in** în tabloul **V** declarat ca mai sus. Fişierul se află în directorul de lucru al programului.

```
ifstream f("matrice.in");
f>>m>>n;
for(i=0;i<m;i++)
  for(j=0;j<n;j++)
      f>>V[i][j];
```

b) Scrierea valorilor tabloului pe mediul extern

Scrierea valorilor în mediul extern se face folosind fluxul de ieşire de date:

– standard (în fişierul text creat pe ecran);

– desemnat de utilizator, prin fişierul text propriu.

• *Afişarea*

Fişierul standard de ieşire este gestionat de fluxul **cout**. Afişarea tabloului se face element cu element. Este de dorit ca afişarea să producă pe ecran aşezarea tabloului în forma dreptunghiulară. Acest lucru revine la a lista câte o linie din tablou pe câte o linie de ecran.

Secvenţa pentru afişare, păstrând notaţiile folosite la citire, este:

```
for(i=0;i<m;i++)
  {cout<<endl;
   for(j=0;j<n;j++)
        cout<<V[i][j]<<" ";
   }
```

• *Ieşirea în fişierul utilizatorului*

Fişierul utilizatorului va fi alcătuit astfel:

– pe prima linie se vor trece valorile pentru **m** şi **n**, separate printr-un spaţiu;

– pe următoarele **m** linii se vor scrie câte **n** valori, de asemenea separate printr-un spaţiu între ele.

Secvenţa de mai jos scrie valorile elementelor ***într-un fişier*** numit **matrice.out**. Fişierul se află în directorul de lucru al programului.

```
ofstream f("matrice.out");
f<<m<<" "<<n;
for(i=0;i<m;i++)
{
 for(j=0;j<n;j++)
     f<<V[i][j]<<" ";
f<<endl;
}
```

Exemplele de mai jos se vor exersa la laborator.

exemplu

1. Să se afişeze următorul tablou:

1	1/2	1/3	1/4	1/5	1/6	1/7	1/8	1/9	1/10
1/2	1/3	1/4	1/5	1/6	1/7	1/8	1/9	1/10	1/11
.....									
1/10	1/11	1/12	1/13	1/14	1/15	1/16	1/17	1/18	1/19

Rezolvare. Se observă că, pentru un **i** şi un **j** curente, care parcurg mulţimea de valori de la 0 la 9, numitorul corespunzător se obţine din expresia: **i + j + 1**.

```
//generarea matricei dupa regula data
#include<iostream.h>
#include<conio.h>
void main()
{unsigned a[10][10],i,j;
clrscr();
for(i=0;i<10;i++)
       for(j=0;j<10;j++)
              a[i][j]=i+j+1;
cout<<"Careul generat este: "<<endl;
```

```
//afisarea matricei numitorilor
for(i=0;i<10;i++)
  {
       cout<<endl;
       for(j=0;j<10;j++)
          cout<<"1/"<<a[i][j]<<" ";
   }
getch();
}
```

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	0
2	3	4	5	6	7	0	1
3	4	5	6	7	0	1	2
4	5	6	7	0	1	2	3
5	6	7	0	1	2	3	4
6	7	0	1	2	3	4	5
7	0	1	2	3	4	5	6

Figura 1.16

2. Să se afişeze tabla adunării în baza de numeraţie **8**.

Rezolvare. După cum se ştie, simbolurile bazei 8 sunt şi cifre ale bazei 10 şi anume cele care reprezintă resturile împărţirii la 8, adică mulţimea {0, 1, 2, 3, 4, 5, 6, 7}.

Pentru tabla adunării în baza 8 trebuie calculate unităţile sumei între toate perechile de resturi, transportul în ordinul următor fiind 0 sau cel mult 1. De exemplu, $7_{(8)} + 7_{(8)} \rightarrow 14_{(10)} \rightarrow 6_{(8)}$ (şi transport 1 – vezi figura 1.16).

```
//tabla adunarii in baza 8
#include<iostream.h>
#include<conio.h>
void main()
{unsigned t8[8][8],i,j;
clrscr();
for(i=0;i<=7;i++)
       for(j=0;j<=7;j++)
              t8[i][j]=(i+j) % 8;
cout<<"Tabla generata este: "<<endl;
for(i=0;i<=7;i++)
       {
       cout<<endl;
       for(j=0;j<=7;j++)
              cout<<t8[i][j]<<" ";
       }
getch();
}
```

Testul 2

(Fiecare dintre subiectele 1, 2, 4, 5 are 1,5 puncte, subiectul 3 are două puncte, din oficiu două puncte)

1. Se consideră declaraţia `typedef float m[6]; m n[6]={0};` Efectul secvenţei: `for(i=0;i<6;i++) for(j=0;j<6;j++) n[i][j]=i==j;` este:
a) creează o matrice nulă; **b)** creează o matrice cu valoarea `i` în toate elementele;
c) creează o matrice unitate; **d)** este o secvenţă incorectă.

2. Fie declaraţia:
`typedef float m[4]; m n[4]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};`.
Imaginea pe ecran a elementelor create prin secvenţa:
`for(i=0;i<4;i++){for(j=0;j<=i;j++) cout<< n[i][j]<<" ";cout<<endl;}` este:
a) un pătrat; **b)** un triunghi dreptunghic cu unghiul drept la nord-est;
c) un triunghi dreptunghic cu unghiul drept la sud-vest; **d)** un şir de linii cu câte un element fiecare.

3. Din mediul de intrare standard se citesc valori. Care dintre liniile matricei `int m[6][5];` vor fi completate cu valori prin secvenţa: `for(i=0;i<5;i++) for(j=0;j<5;j++) cin<< m[i][j];`
a) primele 3; **b)** primele 4; **c)** primele 5; **d)** ultimele 5.

4. Din mediul de intrare standard se citesc valori. Care dintre liniile matricei `int m[6][5];` vor fi completate cu valori prin secvenţa: `for(i=0;i<1;i++) for(j=0;j<5;j++) cin<< m[i][j];`
a) prima; **b)** primele 2; **c)** primele 3; **d)** ultima.

5. Care va fi conţinutul matricei **m**, declarată prin `int m[4][4];`, după executarea secvenţei:
`for(k=0;k<n;k++) for(i=k;i<4;i++) for(j=k;j<4;j++) m[i][j]=k;`

a)				b)				c)				d)			
0	0	0	0	1	2	3	4	0	0	0	0	0	1	2	3
1	1	2	3	0	1	2	2	0	1	1	1	0	1	2	3
2	3	3	3	0	1	1	1	0	1	2	2	0	1	2	3
3	3	3	3	0	0	0	0	0	1	2	3	0	1	2	3

c) Prelucrări care necesită reguli de parcurgere a elementelor

Ordinea de parcurgere a elementelor tabloului devine acum, la tabloul bidimensional, un lucru esenţial, de multe ori mai important decât calculele în care intră elementele lui.

Tipuri de parcurgere a elementelor unui tablou bidimensional dat

Necesitatea stabilirii unui tip de parcurgere este legată de prelucrarea în care sunt folosite elementele tabloului sau de modul în care sunt memorate în spaţiul afectat tabloului.

➢ ***Parcurgerea standard***, matematică, constă în trecerea prin elementele fiecărei linii în ordinea liniilor şi reprezintă parcurgerea matriceală.

Acest tip de parcurgere a fost întâlnit în operaţiile de citire şi scriere prezentate mai sus.

În loc de citiri sau scrieri, asupra elementelor se pot face prelucrări de tip calcule numerice sau logice.

Ca prime exemple în acest sens pot fi luate programele din paragraful ***b)***.

Exemplele de mai jos se vor exersa la laborator.

1. Se consideră două matrice $\mathbf{A}_{m\times n}$ şi $\mathbf{B}_{p\times q}$. Dacă operaţia este posibilă, se doreşte afişarea matricei sumă.

Rezolvare. Se va verifica întâi dacă dimensiunile matricelor sunt aceleaşi, adică **m=p** şi **n=q**, altfel însumarea nu este posibilă. Suma celor două matrice se va realiza în matricea $\mathbf{C}_{m\times n}$.

În programul de mai jos s-a introdus o facilitate în ceea ce priveşte dimensionarea unui tablou. Astfel, este util să se definească prin declaraţia **const** o constantă simbolică, **dim**, care să reprezinte dimensiunea ce va fi alocată, iar în restul programului să se lucreze cu această constantă simbolică. Acest lucru ajută programatorul ca, în cazul în care modifică dimensiunile de lucru, să fie nevoit să intervină numai în linia definirii constantei simbolice şi nu în toate locurile în care valoarea dimensiunii a fost folosită explicit.

```
#include<iostream.h>
#include<conio.h>
void main()
{ const dim=10;
 float
 A[dim][dim],B[dim][dim],C[dim][dim];
 unsigned i,j,m,n,p,q;
 clrscr();
 cout<<"Citirea primei matrice"<<endl;
 do
  {cout<<"Dati numarul de linii m=";
   cin>>m;
  }while(m<2||m>dim);
do
 {cout<<"Dati numarul de coloane n=";
  cin>>n;
}while(n<2||n>dim);
for(i=0;i<m;i++)
  for(j=0;j<n;j++)
   {cout<<"A["<<i+1<<","<<j+1<<"]=";
    cin>>A[i][j];
   }
cout<<"Citirea matricei a doua"<<endl;
do
  {cout<<"Dati numarul de linii p=";
   cin>>p;
  }while(p<2||p>dim);
do
 {cout<<"Dati numarul de coloane q=";
  cin>>q;
  }while(q<2||q>dim);
for(i=0;i<p;i++)
  for(j=0;j<q;j++)
  {cout<<"B["<<i+1<<","<<j+1<<"]=";
   cin>>B[i][j];
  }
if(m==p&&n==q)
 { cout<<"\nMatricea suma"<<endl;
   for(i=0;i<m;i++)
      for(j=0;j<n;j++)
            C[i][j]=A[i][j]+B[i][j];
   for(i=0;i<m;i++)
     {for(j=0;j<n;j++)
        cout<<C[i][j]<<" ";
      cout<<endl;
     }
  }
 else cout<<"Adunare imposibila";
getch();
}
```

2. Se consideră două matrice $\mathbf{A}_{m\times n}$ şi $\mathbf{B}_{p\times q}$. Dacă operaţia este posibilă, se doreşte afişarea matricei produs.

Rezolvare. Se va verifica întâi dacă dimensiunile matricelor îndeplinesc toate condiţiile ca produsul să fie posibil, adică **n=p**, altfel înmulţirea nu este posibilă. Produsul celor două matrice se va realiza în matricea $\mathbf{C}_{m\times n}$, un element calculat fiind: $c_{ij} = \sum_{k=1}^{p} a_{ik} \times b_{kj}$, pentru `i=1,m` şi `j=1,q`.

```
#include<iostream.h>
#include<conio.h>
void main()
{const dim=10;
 float
 A[dim][dim],B[dim][dim],C[dim][dim]={0};
 unsigned i,j,k,m,n,p,q;
 clrscr();
 cout<<"Citirea primei matrice"<<endl;
 do
  {cout<<"Dati numarul de linii m=";
   cin>>m;
  }while(m<2||m>dim);
do
 {cout<<"Dati numarul de coloane n=";
  cin>>n;
  }while(n<2||n>dim);
 for(i=0;i<m;i++)
   for(j=0;j<n;j++)
    {cout<<"A["<<i+1<<","<<j+1<<"]=";
     cin>>A[i][j];
     }
 cout<<"Citirea matricei a doua"<<endl;
 do
  {cout<<"Dati numarul de linii p=";
   cin>>p;
```

```
  }while(p<2||p>dim);
do
 {cout<<"Dati numarul de coloane q=";
  cin>>q;
 }while(q<2||q>dim);
 for(i=0;i<p;i++)
   for(j=0;j<q;j++)
    {cout<<"B["<<i+1<<","<<j+1<<"]=";
     cin>>B[i][j]; }
 if(n==p)
  { cout<<"\nMatricea produs"<<endl;
    for(i=0;i<m;i++)
      for(j=0;j<q;j++)
        for(k=0;k<p;k++)
            C[i][j]+=A[i][k]*B[k][j];
  for(i=0;i<m;i++)
    {for(j=0;j<q;j++)
       cout<<C[i][j]<<" ";
     cout<<endl;
    }
 }
  else cout<<"Inmultire imposibila";
getch();
}
```

- ***Parcurgerea transpusă*** constă în trecerea prin elementele fiecărei coloane, în ordinea coloanelor. Asociind numele parcurgerii cu proprietatea matematică de transpunere a matricelor, putem spune că în acest mod de trecere prin tablou se va obţine matricea directă, iar cea dată ca sursă de valori este matricea transpusă. Ca un exemplu pentru această modalitate de parcurgere se poate lua cazul trecerii prin matricea **B** în operaţia de înmulţire a două matrice din programul anterior.
- ***Parcurgerea geometrică*** se constituie într-un traseu ce imaginează un desen: trecerea pe pătrate concentrice, trecerea în spirală, trecerea pe diagonale şi paralele la diagonale etc.

Exemplele de mai jos se vor exersa la laborator

1. Să se afişeze suma elementelor situate pe diagonala principală şi a celor de pe diagonala secundară ale unui tablou bidimensional pătratic care conţine numere naturale. Dimensiunea maximă este 10 (maximum 10 de linii şi 10 de coloane).

Rezolvare. Pentru exemplificare s-a ales varianta de a genera în mod aleatoriu valorile tabloului cu numere între 0 şi dimensiunea citită. Sumele respective sunt calculate în variabilele **s1** şi **s2**.

```
//sumele elementelor de pe diagonale:
//s1 si s2
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
unsigned careu[10][10],s1=0,s2=0;
int loc,k,i,j,m;
clrscr();
randomize();
do
 {cout<<"Dimensiune careu: ";
  cin>>m;
 }while (m<2 || m>10);
for(i=0;i<m;i++)
  for(j=0;j<m;j++)
      careu[i][j]=random(m);
cout<<"\n Afisarea careului generat";
cout<<endl;
for(i=0;i<m;i++)
  {
  for(j=0;j<m;j++)
      cout<<careu[i][j]<<' ';
  cout<<endl;
  }
//calcul sume
for(i=0;i<m;i++)
 {
  s1+=careu[i][i];
  s2+=careu[i][m-i-1];
 }
cout<<"Suma elementelor diagonalei
principale: "<<s1<<endl;
cout<<"Suma elementelor diagonalei
secundare: "<<s2<<endl;
getch();
}
```

2. Se consideră un tablou pătratic, în care elementele sunt litere. Se cere afişarea elementelor situate în triunghiul superior diagonalei **I**.

Rezolvare. Fie **n** numărul de linii şi de coloane. Triunghiul superior diagonalei **I** este format din grupul de elemente care se află deasupra diagonalei **I**. În figura 1.17, pentru **n** = 5, s-au ales litere ca să imagineze un text cu sens, iar careurile din triunghiul cerut sunt colorate în gri.

a	e	s	t	e
a	c	u	n	m
l	b	e	i	e
c	u	n	l	l
e	g	r	u	a

Figura 1.17

Pentru parcurgerea acestei zone din tabel, se observă că pentru o linie **i** dată, elementele cerute se situează pe coloanele **i+1**, **i+2**,…,**n-1**.

```
#include<iostream.h>
#include<conio.h>
void main()
{ const dim=10;
 char T[dim][dim];
 unsigned i,j,n;
 clrscr();
 do
  {cout<<"Dati dimensiunea n=";
   cin>>n;
  }while(n<2||n>dim);
 for(i=0;i<n;i++)
  for(j=0;j<n;j++)
   {cout<<"T["<<i+1<<","<<j+1<<"]=";
    cin>>T[i][j];   }
 cout<<"\n Elementele trg. superior
                 diagonalei I"<<endl;
 for(i=0;i<n-1;i++)
   for(j=i+1;j<n;j++)
      cout<<T[i][j];
getch();
}
```

Temă de laborator: Să se modifice programul de mai sus pentru a se afişa: a) conţinutul triunghiurilor inferior diagonalei I; b) superior diagonalei a II-a; c) inferior diagonalei a II-a.

3. Dându-se un tablou bidimensional pătratic, de mărime maximă 10 (10 linii şi 10 coloane), conţinând elemente naturale, se cere să se afişeze, pe câte un rând de ecran, lista elementelor de pe conturul fiecărui "pătrat" concentric.

Rezolvare. Ca şi în problema precedentă, accentul s-a pus pe modul de parcurgere şi mai puţin pe modul de generare a elementelor tabloului (şi aici s-a recurs la generarea aleatorie de numere de la 0 la mărimea citită).

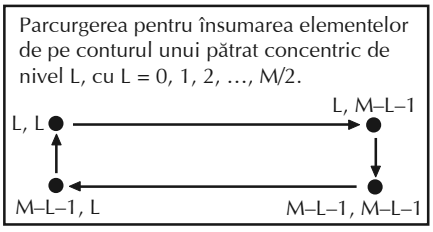


Figura 1.18

Fiecare contur de pătrat concentric începe cu un element al diagonalei principale, notat prin indicii (**L,L**). Cum tabloul are un număr **M** de linii şi de coloane, **L** ia valori de la **0** (linia şi coloana 0) la **M/2** (linia şi coloana pătratului central al tabloului – în indici interni **C/C++**).

În cazul în care **M** este impar, pătratul central se reduce la un element. Dacă **M** este par, pătratul central are patru elemente.

Pentru fiecare contur, se realizează 4 segmente de parcurgere, cum se vede şi în desenul din figura 1.18.

Oricare ar fi segmentele conturului, acesta nu cuprinde întreaga latură a pătratului curent – latura mai puţin ultimul element. În acest ultim element se va face schimbarea de direcţie, astfel că el va fi începutul laturii următoare. În final, conturul se încheie cu elementul de indici (**L+1,L**).

În matricea din figura 1.19 sunt puse în evidenţă aceste segmente pe un ***exemplu numeric***, cu **M**=5. Iniţial, nivelul **L** este 0, fiind vorba de primul pătrat concentric. Primul segment este ocupat de elementele primei linii, din coloanele de la 0 la 3, valorile 1, 2, 3, 4. Al doilea segment, pe ultima coloană (**M-0-1=4**), este format din valorile 5, 10, 15, 20. Al treilea segment, pe ultima linie, conţine valorile 25, 24, 23, 22. Ultimul segment care încheie primul pătrat cuprinde elementele primei coloane, 21, 16, 11, 6. După acest pătrat, **L** avansează la 1 şi desemnează ca origine pentru al doilea pătrat valoarea 7. Procesul se repetă, primul segment fiind format din valorile 7 şi 8. Al doilea segment conţine pe 9 şi 14 ş.a.m.d. Deoarece **M** este impar, ultimul pătrat este format dintr-un singur element, 13. Din acest motiv, în program se testează imparitatea lui **M** pentru a nu mai calcula inutil laturi vide. Elementul central este afişat independent.

1	2	3	4	5
6	7	8	9	10
11	12	**13**	14	15
16	17	18	19	20
21	22	23	24	25

Figura 1.19

```
//program suma_patrate_concentrice;
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
void main()
{ int careu[10][10],i,j,M,n,L;
  randomize();
  clrscr();
  do
   {cout<<"Dimensiune careu: ";
    cin>>M;
   }while(M<2 || M>10);
  for(i=0;i<M;i++)
   for(j=0;j<M;j++)
     careu[i][j]=random(M);
cout<<"Afisarea careului generat";
cout<<endl;
for(i=0;i<M;i++)
```

```
  {for(j=0;j<M;j++)
     cout<<careu[i][j]<<' ';
   cout<<endl;
  }
n=M/2;
cout<<"Afisarea contururilor"<<endl;
for(L=0;L<n;L++)
 {
  cout<<"Conturul "<<L+1<<endl;
  i=L;
  for(j=L;j<M-L-1;j++)
     cout<<careu[i][j]<<" ";
  for(i=L;i<M-L-1;i++)
     cout<<careu[i][j]<<" ";
  for(j=M-L-1;j>L;j--)
     cout<<careu[i][j]<<" ";
  for(i=M-L-1;i>L;i--)
     cout<<careu[i][j]<<" ";
  cout<<endl;
 }
if(M%2) cout<<"Ultimul nivel
                "<<careu[n][n];
getch();
}
```

Temă de laborator: Să se transforme programul de mai sus pentru parcurgerea în spirală a matricei.

4. Se consideră numerele naturale de la **1** la **m**, unde **m** este un pătrat perfect impar. Se cere afişarea unui pătrat magic, **P**, în care să fie dispuse aceste numere (un pătrat magic are proprietatea că, indiferent de modul de însumare a elementelor din punct de vedere al direcţiei de însumare – pe fiecare linie, pe fiecare coloană, pe fiecare diagonală – obţinem acelaşi total).

Rezolvare. Dacă **m** îndeplineşte condiţiile enunţului, atunci pe baza lui se stabileşte „latura" pătratului, în număr de elemente.

Pentru a obţine aceeaşi sumă, indiferent de linie, de coloană sau de diagonală, se porneşte din linia de mijloc, ultima coloană şi se continuă completarea elementelor situate pe paralele la diagonala principală. Astfel, dacă se pleacă dintr-o poziţie (**i,j**), se continuă pe linia şi coloana următoare (**i+1,j+1**). În cazul în care se ajunge într-o margine, următorul element se determină tot pe direcţie diagonală, în „cilindrul" imaginat prin lipirea ultimei coloane de prima, sau a ultimei linii de prima.

11	10	4	23	17
18	12	6	5	24
25	19	13	7	**1**
2	21	20	14	8
9	3	22	16	15

Figura 1.20

Există un caz particular, când numărul ce trebuie înregistrat în matrice este multiplu de latura pătratului. Atunci noul element se aşază în stânga elementului aşezat anterior, păstrându-se imaginea de „cilindru".

Exemplu: Pentru $n = 25$ rezultă „latura" de 5 şi aşezarea din figura 1.20.

Există şi alte trei variante de a porni completarea: elementul din mijloc din linia întâi elementul din mijloc din coloana întâi, sau elementul din mijloc din linia finală.

```
//program patrat_magic_impar;
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()
{unsigned P[11][11],n,k,i,j;
 clrscr();
cout<<"Patrat magic impar "<<endl;
do
  {cout<<"Dati numarul n, impar ";
      cin>>n;
  }while(n%2==0);i=(n+1)/2;
j=n;
for(k=1;k<=n*n;k++)
  {P[i][j]=k;
   if(k%n==0)
    {j=j-1; //k e multiplu de n
     if(j<1)j=n;//iese in stanga
    }
   else
    {j++;
   if(j>n) j=abs(n-j);//iese in dreapta
   i++;
   if(i>n) i=abs(n-i);//iese in jos
  }
}
for(i=1;i<=n;i++)
  {cout<<endl;
   for(j=1;j<=n;j++)
   cout<<P[i][j]<<" ";
  }
getch();
}
```

- ***Parcurgerea pe vecinătăţi*** constă în a pleca dintr-un element de coordonate date şi a trece numai la elementele vecine lui pe cele patru sau opt sensuri.

$a_{i-1\ j-1}$	$a_{i-1\ j+0}$	$a_{i-1\ j+1}$
$a_{i+0\ j-1}$	$a_{i\ j}$	$a_{i+0\ j+1}$
$a_{i+1\ j-1}$	$a_{i+1\ j+0}$	$a_{i+1\ j+1}$

Figura 1.21

În tabelul din figura 1.21 sunt puşi în evidenţă indicii elementelor vecine unui element a_{ij} dat.

Din analiza lor se pot determina următorii vectori ai deplasărilor pe fiecare direcţie: deplasarea indicelui **i** şi deplasarea indicelui **j** în jurul lui a_{ij}, începând din colţul **NV**, în ordinea acelor ceasului:

di=(-1,-1,-1,0,1,1,1,0) şi **dj** =(-1,0,1,1,1,0,-1,-1).

În acest mod, o inspectare a vecinilor elementului a_{ij} devine o operaţie repetată de opt ori, pentru fiecare vecinătate verificându-se existenţa ei (dacă nu iese din spaţiul tabloului), aplicându-se apoi prelucrarea cerută:

```
for(k=0;k<8;k++)
  if(i+di[k]<m && i+di[k]>=0 && j+dj[k]<n && j+dj[k]>=0) P;
```

unde prin **P** s-a notat prelucrarea care se aplică vecinului **a**[i+di[k]][j+dj[k]], iar **m** şi **n** sunt dimensiunile de lucru ale tabloului **a**.

Exemplele de mai jos se vor exersa la laborator

1. Se consideră un teren accidentat pe care geodezii l-au schiţat pe o hartă a altitudinilor de forma unui tablou bidimensional. Fiecare element al tabloului este un număr întreg şi reprezintă altitudinea suprafeţei de teren la care se referă acel element. Cineva aruncă o bilă pe acest teren, care cade în locul de coordonate (**i,j**). Să se determine dacă bila rămâne pe locul în care a căzut sau se va rostogoli în altă parte. Datele sunt citite dintr-un fişier text **teren.in**, în care, pe prima linie sunt dimensiunile terenului în număr de elemente, **m** linii şi **n** coloane, iar următoarele **m** linii conţin fiecare, cele **n** elemente ale liniei.

4 5

12	3	4	56	6
1	**5**	3	3	7
22	2	4	7	3
89	32	45	3	7

Figura 1.22

Rezolvare. Pentru mişcarea bilei din poziţia iniţială dată de perechea de coordonate **(i,j)** este nevoie să se analizeze, pe cele opt direcţii, existenţa celei mai mici înălţimi faţă de cota din locul **(i,j)**. Se va căuta minimul dintre cotele vecinilor în variabila **min** şi se vor înregistra linia minimului în **Lmin** şi coloana în **Cmin**. În variabila **mută** se reţine prin +1 faptul că bila se poate muta, prin 0 că nu se poate muta şi prin -1 situaţia în care bila iese din teren.

Un exemplu de conţinut al fişierului **teren.in** este dat în tabelul din figura 1.22. Dacă presupunem că bila se află la coordonatele (1,1), atunci ea se va rostogoli din această poziţie în locul (1,0)

```
#include<fstream.h>
#include<conio.h>
void main()
{const di[8]={-1,-1,0,1,1,1,0,-1};
 const dj[8]={0,1,1,1,0,-1,-1,-1};
 ifstream f("teren.in");
 int teren[10][10],i,j,m,n;
 int linie, col,min,x,y,muta=0;
 int Lmin,Cmin;
 f>>m>>n;
 for(i=0;i<m;i++)
  for(j=0;j<n;j++)
    f>>teren[i][j];
 do
{cout<<"Dati coordonatele bilei
                          "<<endl;
 cout<<"Linia intre 1 si "<<m<<" ";
 cin>>linie;
 cout<<"\nColoana intre 1 si "<<n<<" ";
 cin>>col;
}while(linie<1||linie>m||col<1||col>n);
 linie--; col--;
 min=teren[linie][col];
 for(int k=0;k<8;k++)
   {x=linie+di[k];
    y=col+dj[k];
    if(x<0||x>m-1||y<0||y>n-1) muta=-1;
      else
        if(teren[x][y]<min)
            {min=teren[x][y];
             Lmin=x;
             Cmin=y;
             muta=1;
            }
   }
if(muta<0)
 cout<<"Bila iese din teren  ";
 else
  if(muta==0)cout<<"Nu se muta";
   else
        cout<<"Bila se poate muta in
            "<<Lmin+1<<","<<Cmin+1;
}
```

2. Se consideră o matrice de **m** linii şi **n** coloane, completată cu valori 1 şi 0. Se cere determinarea locurilor valorilor 1 care au ca vecini numărul 1.

Rezolvare. Pentru a determina locurile elementelor 1 din matrice care au toţi cei opt vecini de valoare 1 este nevoie să se parcurgă matricea pe linii şi pentru fiecare element 1 se inspectează vecinii. Este evident că vor fi numărate astfel numai elemente interioare, fiindcă cele de margine nu au opt vecini. De exemplu, pentru conţinutul fişierului **matr.in** din tabelul din figura 1.23, vor fi indicate elementele de coordonate: (2,4), (2,5), (3,3), (4,2), (4,3).

5	6				
1	0	1	1	1	1
0	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	0	1
1	1	1	1	1	1

Figura 1.23

```
#include<fstream.h>
#include<conio.h>
void main()
{const di[8]={-1,-1,0,1,1,1,1,0,-1};
 const dj[8]={0,1,1,1,0,-1,-1,-1};
 const dim=10;
 int mat[dim][dim],i,j,m,n;
 int nr,x,y;
 clrscr();
 ifstream t("matr.in");
 t>>m>>n;
 for(i=0;i<m;i++)
  for(j=0;j<n;j++)
      t>>mat[i][j];
 t.close();
 cout<<"Matricea citita"<<endl;
 for(i=0;i<m;i++)
  {for(j=0;j<n;j++)
    cout<<mat[i][j]<<" ";
    cout<<endl;
  }
  for(i=0;i<m;i++)
   for(j=0;j<n;j++)
    if(mat[i][j])
      {nr=0;
       for(int k=0;k<8;k++)
        {x=i+di[k];
    y=j+dj[k];
    if(x>=0&&x<m&&y>=0&&y<n)
     if(mat[x][y])nr++;
    }
    if(nr==8)
    cout<<"("<<i+1<<","<<j+1<<"),";
   }
cout<<"\b ";//pt. stergere virgula
                          finala
getch();}
```

1. Completaţi programul de mai sus astfel încât să fie considerate şi elementele de margine dacă ele au toţi vecinii interiori de valoare 1. Pentru exemplul luat, se vor afişa şi locurile (1,4), (1,5), (1,6), (2,6), (4,1), (5,1), (5,2), (5,3).

2. Completaţi programul pentru deplasarea bilei astfel încât să se afişeze toate locurile vecine ei care au cota mai mică.

d) Obținerea unor elemente sau grupuri cu anumite proprietăți

Se pune problema localizării unor elemente sau grupuri de elemente cu anumite proprietăţi (valori maxime sau minime, linii sau coloane cu toate elementele nule etc.).

1. Se doreşte determinarea valorii maxime înregistrate într-o matrice de **m** linii şi **n** coloane (**m** şi **n** nu vor depăşi valoarea 10) ale cărei elemente sunt numere naturale citite de la tastatură. Se va afişa atât valoarea determinată cât şi coordonatele ei în matrice.

Rezolvare. Presupunând că matricea se numeşte **T**, pentru a determina numărul maxim se va iniţializa o variabilă **max** cu valoarea primului element al matricei, **T[0][0]**. Apoi, parcurgând tabloul linie cu linie, se va verifica apariţia unui element de valoare mai mare, caz în care se va actualiza valoarea din **max** şi se vor reţine coordonatele noii valori. Trebuie avut grijă ca la afişarea coordonatelor, acestea să fie exprimate în numerotarea familiară utilizatorului, adică începând cu 1, nu cu 0 – cum se realizează în calculele interne de adresă. Astfel, s-au scris expresiile **lin+1** şi **col+1** în comanda de afişare.

```
//maximul din matrice
#include<iostream.h>
#include<conio.h>
void main()
{
 const dim=10;
 unsigned T[dim][dim];
 unsigned i,j,m,n,lin,col,max;
```

```
 clrscr();
 do
  {cout<<"Dati numarul de linii m=";
   cin>>m;
  }while(m<2||m>dim);
do
  {cout<<"Dati numarul de coloane n=";
   cin>>n;
  }while(n<2||n>dim);
 for(i=0;i<m;i++)
 for(j=0;j<n;j++)
  {cout<<"T["<<i+1<<","<<j+1<<"]=";
   cin>>T[i][j];
   }
 cout<<"\n Elementul maxim"<<endl;
 max=T[0][0];lin=col=0;
 for(i=0;i<m;i++)
    for(j=0;j<n;j++)
       if(max<T[i][j])
         {max=T[i][j];
          lin=i;
          col=j;
         }
cout<<"Elementul maxim este "<<max;
cout<<" pe linia "<<lin+1<<" si coloana
                                "<<col+1;
getch();
}
```

Teme de laborator:

- aflaţi tate locurile în care apare maximul în cadrul matricei.
- aflaţi valoarea minimă din matrice şi de câte ori apare aceasta.

2. Se consideră un tablou bidimensional cu **m** linii şi **n** coloane citit din fişierul **matrice.in.** Se doreşte determinarea valorii maxime din fiecare linie a tabloului. Datele sunt înregistrate în fişier în modul următor: pe prima linie se găsesc valorile pentru **m** şi **n**, separate printr-un spaţiu, iar pe următoarele **m** linii se găsesc cele **n** valori din fiecare linie a tabloului.

Rezolvare. S-a ales varianta de a forma un vector în care fiecare element reţine valoarea maximă din linia corespunzătoare. Astfel se poate prezenta o soluţie pentru cazul în care maximele obţinute intră în alte prelucrări.

```
#include<fstream.h>
#include<conio.h>
void main()
{
 const dim=10;
 unsigned T[dim][dim],max[dim];
 unsigned i,j,m,n;
 clrscr();
 ifstream matr("matrice.in");
 matr>>m>>n;
 for(i=0;i<m;i++)
   for(j=0;j<n;j++)
     matr>>T[i][j];
 for(i=0;i<m;i++)
  {//initializarea maximului pentru
                          linia i
   max[i]=T[i][0];
   //cautarea maximului in linia i
   for(j=1;j<n;j++)
  if(max[i]<T[i][j])max[i]=T[i][j];
   }
cout<<"Elementele maxime pe linii
                         "<<endl;
for(i=0;i<m;i++)
  cout<<" pe linia "<<i+1<<"
                  "<<max[i]<<endl;
getch();
}
```

Teme de laborator:

- completaţi programul de mai sus astfel încât să se afişeze şi locul ocupat de maxim în fiecare linie;
- elaboraţi o variantă a programului, cu aceleaşi cerinţe ca mai sus, dar fără a folosi tabloul unidimensional al maximelor;
- realizaţi programul pentru afişarea minimului din fiecare coloană a tabloului şi notaţi pe caiet tipul de parcurgere a maricei folosit;
- realizaţi un program pentru a afişa numărul liniei care conţine maximul dintre maximele de linii; determinaţi un program mai eficient pentru această cerinţă şi anume fără a folosi tablouri unidimensionale;
- realizaţi un program care determină elementul din tablou care este maxim pe linia lui, dar minim pe coloana lui (aşa-numitul „punct şa").

3. Se consideră un tablou bidimensional cu **m** linii şi **n** coloane citit din fişierul **matrice.in**. Se doreşte determinarea liniilor care conţin numai elemente pare. Datele sunt înregistrate în fişier în modul următor: pe prima linie se găsesc valorile pentru **m** şi **n**, separate printr-un spaţiu, iar pe următoarele **m** linii ale fişierului se găsesc valorile din fiecare linie a tabloului.

Rezolvare. S-a ales ca matricea sursă să aibă $m = 10$ şi $n = 20$. Fiecare linie va fi cercetată pentru a stabili dacă toate elementele ei sunt pare. Dacă se confirmă proprietatea, $k = 1$, se afişează numărul liniei.

```
#include<fstream.h>
#include<conio.h>
void main()
{int matr[10][15],aux,k;
 int i,j,m,n;
 ifstream f("matrice.txt");
 f>>m>>n;
 for(i=0;i<m;i++)
   for(j=0;j<n;j++)
      f>>matr[i][j];
cout<<"\nContinutul dupa citire"<<endl;
for(i=0;i<m;i++)
 {for(j=0;j<n;j++)
    cout<<matr[i][j]<<' ';
  cout<<endl;
 }
for(i=0;i<m;i++)
{ k=1;//comutator de stare pt. linie
  for(j=0;j<n&&k;j++)
       if(matr[i][j]%2)k=0;
 if(k)
  cout<<"linia "<<i+1<<"are toate el.
                      pare"<<endl;
 }
getch();
}
```

Teme de laborator:

– realizaţi un program care să afişeze câte numere prime sunt înregistrate într-un tablou bidimensional cu **m** linii şi **n** coloane, citit din fişierul **matrice.in;**

– elaboraţi un program prin care se determină dacă matricea pătratică, de mărime **n**, citită din fişierul **tablou.txt** este o matrice simetrică. Valoarea lui **n** este citită de pe prima linie din fişier, iar următoarele **n** linii ale fişierului cuprinde cele **n** linii ale matricii;

Indicaţie: se va testa dacă – faţă de diagonala **I** – pentru orice pereche de coordonate **(i,j)** este adevărată relaţia $\mathbf{a_{ij}=a_{ji}}$;

– realizaţi un program care verifică dacă în fişierul **unitate.in** este înregistrată o matrice unitate de mărime **n**; valoarea lui **n** este citită de pe prima linie din fişier, iar următoarele **n** linii ale fişierului cuprind cele **n** linii ale matricii.

4. Dându-se un text cifrat într-un careu de caractere şi o grilă de aceleaşi dimensiuni, conţinând numai elemente binare (**0** sau **1**), să se afişeze mesajul decriptat prin culegerea caracterelor care corespund zerourilor din grilă când aceasta se suprapune peste textul cifrat.

a	x	2	*	c
%	e	t	s	z
t	y	w	t	;
e	z	z	x	t

0	1	1	1	**0**
1	**0**	1	**0**	1
0	1	1	**0**	1
0	1	1	**0**	**0**

De exemplu, conţinutul careului de caractere şi al grilei este cel dat în tabelele de mai sus.

În acest caz, prin suprapunerea grilei peste careul de caractere se va obţine mesajul: „acest text".

```
#include<fstream.h>
void main()
{char cript[10][10];
int  grila[10][10],i,j,m,n;
ifstream h("mesaj.txt");
h>>m>>n;
for(i=0;i<m;i++)
 for(j=0;j<n;j++)
  h>>cript[i][j];
cout<<"Afisarea careului citit"<<endl;
for(i=0;i<m;i++)
 {for(j=0;j<n;j++)
cout<<cript[i][j]<<' ';
  cout<<endl;
 }
for(i=0;i<m;i++)
 for(j=0;j<n;j++)
  h>>grila[i][j];
cout<<"Afisarea grilei
generate"<<endl;
for(i=0;i<m;i++)
 {for(j=0;j<n;j++)
      cout<<grila[i][j]<<' ';
  cout<<endl;
 }
cout<<"Textul decriptat:"<<endl;
for(i=0;i<m;i++)
 for(j=0;j<n;j++)
  if(grila[i][j]==0)
cout<<cript[i][j];
}
```

e) Prelucrări de tip ordonare

Se pune problema pregătirii tabloului bidimensional, în vederea unor prelucrări ulterioare, prin acţiuni de sortare după anumite criterii[1] . Ne-am obişnuit să înţelegem prin ordonare *sortarea* în ordine crescătoare sau descrescătoare a valorilor.

Fie un tablou bidimensional dat, pentru care se cunosc numărul de linii, **m**, şi numărul de coloane, **n**. Să se interschimbe liniile între ele astfel ca în final prima coloană să conţină elementele iniţiale, dar în ordine crescătoare.

Rezolvare. S-a recurs la generarea aleatorie de valori în tablou. Important este modul în care se interschimbă liniile.

De exemplu, pentru **m** = 3 şi **n** = 4, un conţinut al tabloului ar fi :

9	4	2	7
3	12	4	30
7	78	4	20

După ordonarea cerută, conţinutul tabloului se prezintă modificat astfel:

3	12	4	30
7	78	4	20
9	4	2	7

Conform cerinţei problemei ordonarea elementelor primei coloane impune interschimbarea în întregime a liniilor din care fac parte elementele reaşezate. Această mutare în bloc a elementelor se impune prin natura matricei, care este o structură de date în care fiecare element aparţine grupului de elemente din linia şi coloana în care este aşezat iniţial. Astfel, un element nu poate fi luat individual şi mutat în alt grup.

Luând un alt exemplu, dacă ar fi vorba de un tabel în care fiecare linie ar reprezenta punctajele obţinute de fiecare candidat la un concurs, iar coloanele ar fi probele concursului, deci s-ar recurge doar la interschimbarea valorilor din prima coloană, ar însemna modificarea punctajelor între candidaţi: primul candidat ar primi punctajul altui candidat la prima probă ş.a.m.d.

```
#include<fstream.h>
#include<conio.h>
#include<stdlib.h>
void main()
{int matr[10][5],aux,k;
int i,j,m,n;
randomize();
do{   cout<<"m si n :";
      cin>>m>>n;
  }while(m<2 ||m>5 || n<2 ||n>10);
for(i=0;i<m;i++)
 for(j=0;j<n;j++)
      matr[i][j]=random(m*n)+1;
cout<<"\nContinutul dupa generare"<<endl;
for(i=0;i<m;i++)
 {for(j=0;j<n;j++)
      cout<<matr[i][j]<<' ';
  cout<<endl;
 }
```

```
do
 {k=1;
  for(j=0;j<m-1;j++)
      if(matr[j+1][0]<matr[j][0])
        {k=0;
         for(i=0;i<n;i++)
          {aux=matr[j][i];
           matr[j][i]=matr[j+1][i];
           matr[j+1][i]=aux;
          }
        }
 }while(!k);
cout<<"\nContinutul dupa ordonarea
            primei coloane"<<endl;
for(i=0;i<m;i++)
 {for(j=0;j<n;j++)
      cout<<matr[i][j]<<' ';
  cout<<endl;
 }
}
```

1. Modificaţi programul de mai sus pentru a ordona crescător elementele primei linii.

2. Modificaţi programul de mai sus pentru a ordona crescător elementele primei diagonale.
Indicaţie: pentru a interschimba elementul **a[i][i]** cu elementul **a[i+1][i+1]** va fi nevoie să se interschimbe întâi linia **i** cu linia **i+1** şi apoi coloana **i** cu coloana **i+1**.

[1] Criteriile, sau regulile de ordonare, pot fi diverse, în funcţie de prelucrarea ulterioară. De exemplu, o aşezare de tip alternanţă a elementelor cu proprietatea P1 (de exemplu, paritate) cu elemente cu proprietatea P2 (de exemplu, imparitate), este tot o ordonare.

PROBLEME PROPUSE PENTRU TABLOURI BIDIMENSIONALE

Problemele se vor distribui ca teme de portofoliu.
Pentru problemele de mai jos, datele se citesc din fişiere text adecvate, asemănător exemplelor de mai sus.

1) Dându-se un tablou bidimensional cu elemente reale, ale cărui linii şi coloane nu depăşesc numărul 20, să se afişeze, pe rânduri separate pe ecran, conţinutul acelor linii care nu au elemente strict negative.

2) Dându-se un tablou cu maximum 100 de componente reale nenule, organizate în maximum 10 linii a câte maximum 10 coloane, să se afişeze suma, produsul, media aritmetică şi media geometrică a elementelor din fiecare linie în parte.

3) Dându-se un tablou bidimensional cu maximum 100 de componente naturale, să se afişeze cel mai mare divizor comun al elementelor lui de pe fiecare coloană.

4) Dându-se un tablou bidimensional pătratic, cu elemente reale, ale cărui linii şi coloane nu depăşesc numărul 20, să se afişeze suma elementelor de pe diagonala I şi suma elementelor de pe diagonala a II-a, calculate simultan (la o singură trecere prin tablou).

5) Dându-se un tablou bidimensional pătratic, cu elemente reale, ale cărui linii şi coloane nu depăşesc numărul 20, să se afişeze sumele obţinute, pe rând, din elementele dispuse pe paralelele la diagonala I.

6) Dându-se un tablou bidimensional cu elemente reale, ale cărui linii şi coloane nu depăşesc numărul 20, să se afişeze tabloul după ce fiecare element a fost înlocuit cu raportul dintre valoarea acestuia şi valoarea minimă din tablou.

7) Dându-se un tablou bidimensional pătratic, cu elemente reale, ale cărui linii şi coloane nu depăşesc numărul 20, să se afişeze valoarea maximă găsită între elementele de deasupra diagonalei I şi valoarea minimă a celor de sub diagonala I (nu se prelucrează elementele diagonalei, iar procesul se va desfăşura într-o singură trecere prin tablou).

8) La o sesiune ştiinţifică participă n persoane, numerotate de la **1** la **n**. Unele persoane cunosc alte persoane din sală, altele nu. Calculatorul care monitorizează reuniunea trebuie să comunice în final care este cea mai cunoscută persoană (adică este cunoscută de către cele mai multe persoane din sală). Pentru aceasta se introduc ca date în program perechi de forma **(i,j)** care semnifică faptul că persoana **i** cunoaşte persoana **j** (nu şi invers). Utilizând un tablou pătratic de mărime **n**, cu elemente din mulţimea **{0,1}**, să se afişeze numărul persoanei căutate.

9) Dându-se un tablou bidimensional pătratic, cu elemente reale, ale cărui linii şi coloane nu depăşesc numărul 20, să se afişeze noua aşezare a elementelor lui după ce acesta a fost „rotit" cu 90° la dreapta în jurul centrului.

10) Se consideră un tablou care imaginează tabla de şah. Se citesc un număr de linie şi un număr de coloană care vor reprezenta poziţia unui cal pe tablă. Să se afişeze coordonatele elementelor tablei pe care poate sări calul.

11) Dându-se un tablou bidimensional pătratic, cu elemente reale, ale cărui linii şi coloane nu depăşesc numărul 20, să se afişeze un mesaj corespunzător dacă există sau nu o simetrie privind valorile elementelor faţă de diagonala I (adică **tabl[i,j] = tabl [j,i]** se respectă pentru toate perechile de coordonate **(i,j)**).

12) Se consideră un tablou bidimensional de elemente naturale, maximum 20 de linii şi de coloane. Să se afişeze media armonică a elementelor fiecărei linii. Reamintim că media armonică este un număr real obţinut ca raport între numărul de elemente şi suma inverselor acestora.

13) Fie **n** puncte în plan, maximum 20, pentru fiecare punct **i** cunoscându-se coordonatele lui **(x_i,y_i)**. Datele sunt citite într-o matrice de două linii şi maximum 20 de coloane. În prima linie sunt coordonatele **x_i** şi în a doua linie sunt coordonatele **y_i**. Să se afişeze coordonatele capetelor şi mărimea celui mai lung segment care se poate construi cu punctele citite.

14) Într-un tablou de maximum 200 de caractere aşezate în maximum 10 linii de maximum 20 de coloane se poate afla şi caracterul **c**, citit. Interesează să se afişeze poziţiile pe care acest caracter s-ar găsi în tablou.

15) Se consideră un tablou cu maximum 100 de valori reale aşezate în maximum 10 linii de maximum 10 coloane. În fiecare linie sunt valori reprezentând înălţimile date în centimetri ale unor persoane din grupul

dat de numărul liniei +1. Să se afişeze dacă o anumită înălţime, *a)* citită, nu există în tablou sau toate sunt de valoarea *a* sau o parte sunt de valoarea *a* şi în ce procent. Să se stabilească grupul cu cele mai înalte persoane.

16) Să se verifice dacă elementele unui tablou de maximum 100 de elemente din mulţimea {0,1}, aşezate în maximum 10 linii de maximum 10 elemente fiecare, sunt în ordinea: 0,1,0,1.... pe fiecare linie a tabloului.

17) Se citeşte un tablou de maximum 100 de elemente din mulţimea {0,1,2,…,9}, aşezate în maximum 10 linii a câte maximum 10 elemente fiecare Să se verifice dacă fiecare linie este un palindrom (este acelaşi şir de cifre indiferent de ordinea de citire pe linie).

18) Un număr de **m** (**m<20**) candidaţi trebuie să susţină maximum **n** (**n<20**) probe. La acelea la care un candidat nu se prezintă se consideră punctaj nul. Este nevoie, pentru o prelucrare de tip clasament în aşa fel încât punctajele să apară ordonate descrescător.

19) Se consideră un tablou bidimensional de elemente naturale, având maximum 20 de linii şi de coloane. Să se elimine liniile care au elemente nule.

20) Se consideră un tablou bidimensional de elemente naturale, maximum 9 linii şi 20 de coloane. Se citeşte un număr natural **n** cu maximum 5 cifre. Să se completeze fiecare linie a tabloului cu cifrele reprezentării numărului **n** în baza **b**, unde **b** este **numarul liniei+2**.

21) Se consideră un tablou bidimensional ale cărui elemente sunt cifre. Tabloul are maximum 50 de coloane şi maximum 10 linii. Să se afişeze suma numerelor formate din cifrele fiecărei linii.

22)

1	2	3	2	2
6	8	12	10	3
7	11	23	12	8

Figura 1.24

Se consideră un tablou bidimensional de elemente naturale, maximum 20 de linii şi de coloane. Să se verifice dacă tabloul are aspect de relief „coamă", adică este ordonat crescător până la coloana **k** şi descrescător până la ultima coloană (de exemplu, tabloul din figura 1.24). Numărul **k** este citit de la tastatură (vezi tabloul din figura 1.23).

23) Pătrat în pătrat: se consideră două tablouri pătratice cu componente reale (maximum 10 x 10, al doilea pătrat fiind mai mic decât primul). Se citeşte un număr natural **p**. Să se înlocuiască tabloul al doilea în primul tablou, începând cu poziţia (**p,p**), dacă acest lucru este posibil.

Capitolul 2 Şiruri de caractere

În acest capitol veţi învăţa despre:

- Modul de structurare a datelor care reprezintă informaţii de tip text
- Proiectarea şi prelucrarea unui şir de caractere

2.1. Noţiunea de şir de caractere ca structură de date

Până acum, şirurile de caractere au intervenit sub formă de constante, utilizate mai ales în compunerea mesajelor către utilizator.

Să presupunem că am avea nevoie de prelucrarea unui set de intrare cu conţinut text, în care s-ar dori numărarea apariţiilor unui anume caracter.
Cu cele învăţate până acum, am construi un tablou unidimensional având ca elemente caracterele din text, în variabila **sir** şi apoi am căuta caracterul cerut, dat în variabila **ch**.
În maniera de lucru anunţată, este evident că utilizatorul trebuie să dea numărul de caractere din şir, în variabila **n**, înainte de citirea şirului.
Acest lucru este necesar pentru a se şti câte poziţii sunt ocupate în tabloul **sir** din totalul de, spre exemplu, 20 alocate prin declararea tabloului.

```
#include<iostream.h>
#include<conio.h>
void main()
{ char sir[20],ch;
  int i,nr=0,n;  clrscr();
  cout<<"Dati numarul de caractere de
                citit ";
  cin>>n;
  cout<<"\n Dati caracterele ";
for(i=0;i<n;i++)
      cin>>sir[i];
cout<<"\n Dati caracterul de cautat ";
cin>>ch;
for(i=0;i<n;i++)
     if(sir[i]==ch) nr++;
cout<<"\n S-au gasit "<<nr<<" aparitii";
getch();
}
```

Limbajul **C/C++ nu** dispune de un *tip de dată predefinit* pentru structura *şir de caractere*, dar oferă facilităţi în lucrul cu ea.

Variabilele de tip şir de caractere sunt văzute tot ca tablouri unidimensionale, de caractere, dar în care ultimul caracter este urmat de caracterul cu codul **ASCII** zero, pentru a se recunoaşte sfârşitul şirului.

Prin urmare, declararea unui şir de caractere se scrie:

char nume_sir [*constanta_de_lungime*];

unde constanta_de lungime, în cazul şirurilor de caractere, este de maximum 256, şi ea trebuie să numere şi caracterul ocupat de codul de sfârşit de şir.

Declararea **char** `s[6]` va putea conţine cinci caractere, în elementele: `s[0]`, `s[1]`, `s[2]`, `s[3]` şi `s[4]`, pentru că `s[5]` trebuie ocupat de zero. În figura 2.1, este prezentată repartizarea în zona **s** a cuvântului *ora20*.
În scriere **C/C++**, codul de sfârşit de şir se poate transmite prin caracterul escape '\0', sau prin numele **NULL** al constantei de valoare zero *(Atenţie! Codul* **ASCII** *zero, nu codul* **ASCII** *al cifrei 0*).

	'o'	'r'	'a'	'2'	'0'	**NULL** sau '\0'
coduri →	111	114	97	50	48	**0**
indici →	0	1	2	3	4	5

Figura 2.1

Caracteristicile tipului de date – *şir de caractere*

I. Şirul de caractere este o **structură liniară** cu elemente de tip *char,* asemănătoare tabloului unidimensional.
II. Fiecare element al structurii este reprezentat intern prin codul ASCII al caracterului instalat pe acel loc;
III. **Alocarea** în memorie a unei date *şir de caractere* se face în octeţi succesivi, numerotaţi începând cu 0, ca orice indice de tablou unidimensional.
IV. În *ultimul octet* al şirului trebuie să existe **valoarea zero** (primul cod **ASCII**) care are destinaţia specială de **sfârşit de şir**.
V. **Lungimea maximă alocată** structurii *şir de caractere* este de 256 de caractere: 255 de caractere utile plus codul de final de şir (codul de valoare zero).
VI. Sintaxa oferită de limbajul **C/C++** pentru declararea unei variabile de tip *şir de caractere* este cea utilizată pentru definirea unui tablou unidimensional, cu observaţia că se specifică *un număr de elemente* **cu o unitate în plus**.
VII. Elementele şirului pot fi **desemnate prin indicele** corespunzător sau prin **adresa simbolică** a elementului, ca în cazul tablourilor.

2.2. Atribuirea de valori și afișarea unui șir de caractere

Exită cinci posibilităţi ca o variabilă de tip şir de caractere să capete valori.

1º ***Iniţializare la declarare***. Odată cu declararea de tablou de caractere se face şi atribuirea unei valori iniţiale. Poziţiile neocupate din zona alocată se vor completa automat cu caracterul de sfârşit de şir (codul **ASCII** zero).

Este singurul caz permis de a se folosi ***expresia de atribuire*** pentru un şir de caractere.


exemplu

```
char s[10]="Un sir";// operatie realizata de catre compilator
char s[4]=""; // initializarea cu sirul vid
```

Dacă scriem însă:

```
char s[10];
s= "Un sir"; // operatie care ar trebui realizata la executia programului,
             // de catre microprocesor
```

atunci *iniţializarea este greşită*, deoarece atribuirea ar cere aici executarea unei copieri a constantei „Un sir", pentru care nu este definit spaţiu în memorie, deci nu are adresă pe care microprocesorul să o încarce într-un registru şi să comande apoi copierea din acea zonă.


observă

Pentru iniţializarea unui şir cu o constantă, nu este nevoie de precizarea lungimii zonei între parantezele drepte.
De exemplu, este corectă şi declararea:

```
char s[]="Un sir";
```

În acest fel, ***însă***, compilatorul face automat calculul de lungime şi alocă numărul de octeţi necesari constantei, iar o altă valoare, mai mare, va fi înregistrată cu sacrificarea caracterului de final de şir!

2º ***Citire***, din mediul de intrare (fişierul standard de intrare, sau un alt fişier, al utilizatorului). În procesul de transfer din mediul de intrare, întâlnirea unui caracter alb (spaţiu, salt prin tabulare, sfârşit de linie) produce încheierea transferului şi înregistrarea codului **NULL** în caracterul următor din zona de memorie a şirului.

3º ***Atribuire element cu element***, în maniera întâlnită la tabloul unidimensional, caz în care caracterul de final al şirului trebuie înregistrat în octetul respectiv prin instrucţiuni în mod explicit.


exemplu

În programul următor, se citeşte un şir de caractere în variabila **t** şi în şirul **s** se copiază cifrele din **t**.

```
#include<iostream.h>
#include<conio.h>
void main()
{
  clrscr();
  char s[10],t[10];
  int k=-1;
  cout<<"Dati textul, maximum 9 caractere\n";
  cin>>t;//citirea sirului de la tastatura
  for(int i=0;t[i];i++)
    if(t[i]>='0'&&t[i]<='9')
      s[++k]=t[i]; //atribuire caracter cu
                   //caracter
  s[k+1]=NULL;
  cout<<endl<<"Sirul cifrelor gasite in text\n";
  cout<<s<<endl;//afisarea unui sir
  getch();
 }
```

Figura 2.2

Programul pune în evidenţă, la o a doua rulare, modul de aşezare în memorie a celor două variabile, şi anume, întâi **t** şi apoi **s** (fig. 2.2). Astfel apare importanţa respectării spaţiului alocat. La a doua rulare în **t** se introduc10 caractere, „123bnmd123", care ocupă inclusiv octetul rezervat sfârşitului de şir. Din acest motiv, repetiţia **for** care trebuie să se oprească în momentul în care **t[i]** devine **NULL**, continuă să treacă cifre în **s**. Se iese din zona lui **t** mai departe, în memorie, se intră peste zona **s** şi cifrele găsite acolo sunt trecute în **s**, mai departe. Prelucrarea nu se mai opreşte.

Presupunem un alt exemplu, în care variabila char *sir[6]* primeşte şirul „acasa". Dacă se doreşte însă conţinutul „probabilitate", acesta nu va fi stocat complet în *sir*, decât sub forma „probab", iar octetul *sir[5]* va conţine codul **ASCII** al lui 'b' şi nu caracterul de final, 0.

4º Definirea drept ***constantă simbolică*** tip şir de caractere.

De exemplu: `const char sc[]="sir constant";`

5º ***Copierea în bloc*** a valorii din alt şir de caractere definit anterior. Această modalitate este realizată cu ajutorul funcţiei **strcpy**, din bioblioteca **string** şi va fi prezentată în paragraful *Prelucrări specifice şirurilor de caractere*.

observă

Pentru un şir declarat `char s[2]`, în care se doreşte drept conţinut litera **D**, atribuirea corectă, caracter cu caracter, este: `s[0]='D';s[1]=0;` şi nu `s[0]="D";`

Afişarea unui şir de caractere se poate face în mod global, de exemplu **cout<<s**, sau element cu element, ca la tablouri, până la ultimul caracter din şir (în programul precedent s-a folosit afişarea în bloc a caracterelor din şirul **s**).

exemplu

Considerăm un şir **s** care poate conţine maximum 18 caractere. La declarare, şirul este iniţializat cu textul „cascadorie". Exemplul va arăta cum restul spaţiului alocat este ocupat, în fiecare poziţie rămasă, de codul de final de şir. Pentru vizualizare (fig. 2.3), s-a ales afişarea caracter cu caracter a conţinutului din spaţiul de memorie alocat, afişându-se caracterul # pentru elementele care conţin codul **0** (cod neimprimabil). Apoi, s-a citit cuvântul *NOU* ca nou conţinut, mai scurt, al şirului **s** şi s-a pus în evidenţă, tot prin afişare caracter cu caracter, noua ocupare a zonei. La sfârşit, s-a afişat conţinutul variabilei **s**, ceea ce a făcut să apară pe ecran numai cuvântul *NOU*.

```
#include<iostream.h>
#include<conio.h>
void main()
{clrscr();
 char s[19]="cascadorie";
 cout<<"Prima scriere: ca valoare din
```

```
                                sir\n";
cout<<s<<endl;
cout<<"\n A doua scriere: ca zona de
                              memorie\n";
for(int i=0;i<19;i++)
  if(s[i]) cout<<s[i];
     else cout<<'#';
cout<<endl<<endl;
cout<<"Citirea noii valori ";
cin>>s;
 cout<<"\n\n  A treia scriere: ca
                   zona de memorie\n";
for(i=0;i<19;i++)
  if(s[i]) cout<<s[i];
     else cout<<'#';
cout<<endl<<endl;
cout<<"A patra scriere: ca valoare din
                        sir\n";
 cout<<s;
 getch();
}
```

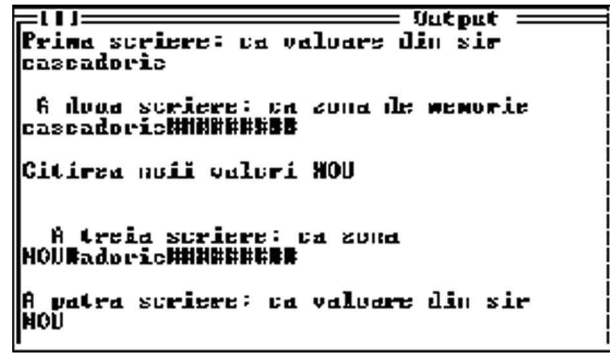


Figura 2.3 – Conţinutul ecranului după rulare.

1. Alegeţi declaraţia corectă pentru o variabilă tip şir de caractere:
a) `char s[0];` **b)** `char s[n];` **c)** `char s[200];` **d)** `char s[12][5];`
e) `char s[];`

2. Alegeţi varianta corectă pentru o variabilă tip şir de caractere iniţializată la declarare:
a) `char s[0]="";` **b)** `char s[5]="acasa";` **c)** `char s[7]="ora 20";`
d) `char s[]="2 siruri corecte";`

3. Stabiliţi care sunt secvenţele eronate din lista de mai jos:
a) `char s[10];s="###";` **b)** `char s[5];cin>>s;`
c) `char s[7],t[2]; s[0]=t[0]="0";`
d) `char s[]="siruri corecte";s[0]='S';`

4. Care dintre declaraţiile de mai jos definesc un vector cu trei elemente de tip şir de caractere care conţin cuvintele: *ieri, azi, maine*?
a) `char s[3]={"ieri","azi","maine"};`
b) `char s[3][5]={"ieri","azi","maine"};`
c) `char s[3][6]=("ieri","azi","maine");`
d) `char s[][6]={"ieri","azi","maine"};`
e) `char[4][6]={"ieri","azi","maine"};`

5. Alegeţi declaraţia corectă pentru o variabilă tip şir de caractere:
a) `char [];` **b)** `char s;` **c)** `char s[300];` **d)** `char s[12];` **e)** `char *s;`

2.3. Prelucrarea șirurilor care conțin caractere albe

Comunicarea între utilizator şi sistemul de operare se face prin mesaje.

Mesajul este un *şir de caractere* care intră-iese sub formă de flux (**stream**), în special prin dispozitivele standard de intrare-ieşire, adică prin consola sistemului (ansamblul de echipamente: tastatură şi monitor).

Deoarece mesajele sunt de lungimi diferite şi compuse din cuvinte adresate sistemului, există anumite caractere (coduri **ASCII**) definite de sistemul de operare cu rol de separatori, delimitatori între cuvinte sau între mesaje, numite **caractere albe**.

Această mulţime este formată din: caracterul *spaţiu*, caracterul de salt în rând – *tab*, caracterele de *sfârşit de linie* – codurile **ASCII** pentru **LF** şi **CR**[1] (caractere create de **ENTER** sau de constantele **C/C++**: eoln şi '\n').

[1] LF – Line Feed (trecere la linia următoare); CR – Carriage Return (retur de car – salt la început de rând).

Implicit, **citirea** unui text în care există **caractere albe** se opreşte la primul caracter alb întâlnit.

Pentru şirul declarat şi citit prin:

```
char sir[10];
cin>>sir;
```

din fluxul de intrare: "*un sir cu separatori ai sistemului*", va fi citit în variabila **sir** doar conţinutul "*un*".

Pentru a citi întreg şirul, va trebui să se recurgă la o specificare suplimentară pentru transfer, care diferă în funcţie de mediul de intrare.

a) Pentru un flux de intrare din ***fişierul standard*** creat de la tastatură[1], se foloseşte funcţia **get** a obiectului **cin**, care are forma:

```
in.get(sir,nr,ch);
```

unde:

- **sir** este variabila şir de caractere care va primi conţinutul de la tastatură, inclusiv caractere albe;
- **nr** este o variabilă sau o constantă întreagă care precizează că se doreşte transferul a **nr** – 1 caractere;
- **ch** este un caracter care precizează că citirea se realizează până la întâlnirea lui ca delimitator; el nu este transferat în **sir**; dacă **ch** a fost întâlnit mai devreme de a se citi cele **nr** – 1 caractere, citirea se încheie; dacă nu este trecută explicit o valoare în **ch**, el lipseşte din paranteză şi citirea se încheie la întâlnirea codului de linie nouă ('\n'), când acesta apare mai devreme de **nr** – 1 caractere.

b) Pentru un flux dintr-un ***fişier text al utilizatorului***, se foloseşte funcţia **getline** a obiectului fişier:

```
variabilă_fişier.getline(sir, nr, ch);
```

unde **sir**, **nr** şi **ch** au semnificaţiile de mai sus.

De exemplu, fie fişierul şi şirul de caractere declarate astfel:

```
ifstream f("fraze.in");
char sir[255];
```

Atunci citirea se poate scrie

```
f.getline(sir,255,'\n');
```

observă

1. Pentru a opri citirea şi la alte caractere, în **ch** se poate folosi şi alt caracter decât cele albe.
2. Dacă citirea este repetată, pentru o altă linie de intrare care se încheie cu acelaşi caracter **ch**, atunci, înainte de a se relua secvenţa de citire, *trebuie golită zona* în care s-a făcut citirea anterioară şi în care a rămas caracterul delimitator, **ch**. *Golirea* se va face prin scrierea aceleiaşi comenzi de citire, dar fără a mai trece ceva între paranteze, **cin.get()** – în cazul citirii de la consolă – sau **f.getline(sir,1)** – în cazul citirii dintr-un fişier al utilizatorului.

exemplu

Să presupunem că se citesc **n** fraze din fişierul standard, de la tastatură, fiecare frază conţinând mai multe cuvinte separate prin spaţii şi terminându-se cu punct. Atunci, secvenţa citirilor va fi cea din coloana din stânga de mai jos. Dacă aceleaşi **n** fraze se citesc dintr-un fişier nestandard, de exemplu **fraze.in**, atunci secvenţa citirilor va fi cea din coloana din dreapta de mai jos.

```
for(int i=1; i<=n;i++)
  {
    cin.get(linie,255,'.'); cin.get() ;
            ……}//prelucrarea liniei
  }
```

```
ifstream f("fraze.in");
for(int i=0 ;i<=n ;i++)
{f.getline(sir,255,'.');
 f.getline(sir,1);
 ……}//prelucrare sir}
```

rezolvă

1. Se consideră şirul «un exemplu» ca valoare de intrare pentru variabila char s[20]. Stabiliţi care este citirea corectă a valorii:
a) `cin<<s;` **b)** `getline(s);` **c)** `cin.get(s,30,'0');` **d)** `get<<s;`
e) `cin.get(s,12,'\n');`

[1] Fişierul standard de intrare este recunoscut de către sistemul de operare sub numele **stdin**, iar cel de ieşire, **stdout**.

2. Se consideră şirul «un exemplu» ca valoare de intrare pentru variabila char s[20]. Stabiliţi ce conţine variabila s după fiecare din citirile demai jos:
a) `cin<<s;` **b)** `cin.get(s,3,' ');` **c)** `cin.get(s,8,'e');`
d) `cin.get(s,12,'\n');`

2.4. Prelucrări specifice şirurilor de caractere

Deoarece şirurile de caractere sunt elementul principal al operaţiilor de transfer efectuate de către **SI/I**[1], fiind forma primară sub care ajunge informaţia în buffer-e[2], limbajele de programare dispun de biblioteci de subprograme ce realizează prelucrări specifice acestui tip de date.

Astfel, limbajul **C/C++** are o bibliotecă de funcţii pregătite pentru prelucrările generale la care sunt supuse şirurile de caractere. Biblioteca conţine funcţiile declarate în fişierul **`string.h`**.

Principalele prelucrări generale aplicate şirurilor de caractere:

1. Obţinerea lungimii unui şir – *strlen(sir)* (numele provine din ***str***ing ***len***gth);
2. Copierea conţinutului unui şir *sursă* peste un şir *destinaţie*, înlocuindu-l pe acesta – *strcpy(destinaţie,sursă)* (numele provine din ***str***ing ***c***o***py***);
3. Compararea a două şiruri *a* şi *b* – *strcmp(a,b)* (numele provine din ***str***ing ***c***o***mp***are);
4. Concatenarea a doua şiruri, *a* şi *b*, prin lipirea şirului *b* la sfârşitul şirului *a* – *strcat(a,b)* (numele provine din ***str***ing con***cat***enation);
5. Iniţializarea unui şir de caractere cu acelaşi caracter – *strset(sir, ch)*;
6. Inversarea conţinutului unui şir de caractere – *strrev(sir)* (numele provine din ***str***ing ***rev***erse);
7. Transformarea literelor mari în litere mici sau invers – *strlwr(sir)* şi *strupr(sir)* (numele provine din ***str***ing ***l***o***w***e***r*** şi ***str***ing ***u***p***p***e***r***);
8. Căutarea unui caracter într-un şir dat – *strchr(şir, ch)* (numele provine din ***str***ing ***ch***a***r***acter);
9. Căutarea unui subşir într-un şir dat – *strstr(şir,subşir)* (numele provine din ***str***ing sub***str***ing);

Mecanismul de lucru al principalelor prelucrări asupra şirurilor de caractere
Exerciţiile se vor pune în execuţie în orele de laborator.

Fiecare exerciţiu de mai jos este realizat prin două programe aşezate în paralel, pentru a fi comparate. În primul program (coloana stângă) se programează rezolvarea utilizând, cu mici excepţii, numai *cunoştinţele de bază* căpătate până în acest moment. În programul al doilea (a doua coloană) sunt utilizate *funcţiile corespunzătoare* din biblioteca **`string`**. În ambele programe sunt puse în evidenţă, prin *casete albe,* prelucrările care sunt echivalente.

1) Se citesc două şiruri de caractere, **`a`** şi **`b`**, din fişierul **`siruri.txt`**. Se cere afişarea şirului mai lung dintre ele.

Rezolvare. Primul program va realiza calculul lungimilor celor două şiruri, în variabilele **`m`** şi **`n`**, prin contorizarea caracterelor diferite de codul **`NULL`**.

Celălalt program va utiliza funcţia de calcul al lungimii unui şir de caractere, **`strlen`**, pusă la dispoziţie de biblioteca **`string`**, care realizează acelaşi lucru, fără însă a mai fi nevoie de variabilele **`m`** şi **`n`**.

```
//program comparare lungimi fara
//functii din biblioteca string;
#include<fstream.h>
#include<conio.h>
void main()
{
```

```
ifstream f("siruri.txt");
char a[20],b[10];
int m=0,n=0;
f>>a>>b;
while(a[m])m++;
while(b[n])n++;
```

[1] SI/I = Sistemul de Intrare–Ieşire al calculatorului.
[2] buffer = zonă de manevră din memoria internă în/din care se transmit informaţiile ce vor fi pregătite pentru intrare/ieşire în/din variabilele programului.

```
   if(m>n)
      cout<<"Sirul mai lung este="<<a;
     else
      cout<<"Sirul mai lung este="<<b;
      getch();
   f.close();
}
//program comparare lungimi cu functia
//din biblioteca string


#include<fstream.h>
#include<conio.h>
```

```
void main()
{
   ifstream f("siruri.txt");
   char a[20],b[10];
   f>>a>>b;
   if(strlen(a)>strlen(b))
      cout<<"Sirul mai lung este="<<a;
     else
      cout<<"Sirul mai lung este="<<b;
   getch();
  f.close();
}
```

2) Se citesc două şiruri de caractere, **a** şi **b**, din fişierul **siruri.txt**. Se cere afişarea şirurilor în ordinea inversă faţă de ordinea în care au fost citite. De exemplu, şirurile citite sunt „Ion" şi „Ioana", iar la afişare vor apărea în ordinea „Ioana" „Ion".

Rezolvare. Primul program va realiza interschimbarea şirurilor în mod explicit, caracter cu caracter. Celălalt program va utiliza funcţia de copiere şiruri, **strcpy**, aflată în biblioteca **string**.

```
//program interschimb fara
//functii din biblioteca string;
#include<fstream.h>
#include<conio.h>
void main()
{     ifstream f("siruri.txt");
      char a[10],b[10],aux[10];
      int m=0,n=0,i=0;
      f>>a>>b;
      while(a[m]) aux[m]=a[m++];
      aux[m]=0;
      while(b[n]) a[n]=b[n++];
      a[n]=0;
      while(aux[i]) b[i]=aux[i++];
      b[i]=0;
      f.close(); clrscr();
      cout<<a<<" "<<b;
 getch();
 }
```

```
//program interschimb cu funcþia
//din biblioteca string
#include<fstream.h>
#include<conio.h>
#include<string.h>
void main()
{
 clrscr();
      ifstream f("siruri.txt");
      char a[10],b[10],aux[10];
      f>>a>>b;
      strcpy(aux,a);
      strcpy(a,b);
      strcpy(b,aux);
      f.close();
      cout<<a<<" "<<b;
 getch();
 }
```

3) Se citesc două şiruri de caractere, **a** şi **b**, din fişierul **siruri.txt**. Se cere afişarea şirurilor în ordinea lor lexicografică. De exemplu, şirurile citite sunt: „Ion" şi „Ioana", iar la afişare vor apărea în ordinea „Ioana" „Ion". Dacă la citire au fost şirurile „Andrei" şi „Ion", atunci la afişare vor apărea în aceeaşi ordine.

Rezolvare. Primul program va realiza compararea şirurilor în mod explicit, caracter cu caracter. Compararea se termină cu o concluzie în variabila **i**, care este iniţializată cu zero. Dacă şirurile sunt egale, **i** va rămâne 0. Dacă primul şir este lexicografic înaintea celui de-al doilea, **i** va primi valoarea –1, iar pentru a treia situaţie, **i** va primi valoarea 1. Celălalt program va utiliza funcţia de comparare şiruri, **strcmp**, aflată în biblioteca **string**. Funcţia din bibliotecă realizează compararea prin scăderea codurilor **ASCII** ale caracterelor de pe aceeaşi poziţie din ambele şiruri. Scăderea provoacă un rezultat negativ în momentul în care în primul şir se întâlneşte un caracter cu codul **ASCII** mai mic decât codul caracterului corespunzător din al doilea şir şi un rezultat pozitiv pentru situaţia inversă. Atâta timp cât în ambele şiruri sunt aceleaşi caractere, rezultatul scăderii va fi 0. Dacă se menţine 0 până la sfârşitul celor două şiruri, rezultă că şirurile au un conţinut identic.

```
//program comparare siruri fara
//functii din biblioteca string;
#include<fstream.h>
```

```
#include<conio.h>
void main()
{ ifstream f("siruri.txt");
```

```
  char a[10],b[10],aux[10];
  int m=0,n=0,i=0;
  f>>a>>b;   f.close();
  while(a[m]&&b[n]&&a[m]==b[n])
    {m++;n++;}
  if(a[m]&&b[n]&&a[m]>b[n])i=1;
       else if(a[m])i=-1;
  if(i==-1)
       {cout<<"Ordinea sirurilor este ";
        cout<<a<<' '<<b; }
     else if(i==0) cout<<"siruri egale";
      else
       {cout<<"Ordinea sirurilor este ";
       cout<<b<<' '<<a;  }
 getch();
 }

//program comparare siruri cu functia
//din biblioteca string
#include<fstream.h>
```

```
#include<conio.h>
#include<string.h>
void main()
{
  ifstream f("siruri.txt");
  char a[10],b[10],aux[10];
  f>>a>>b;
  f.close();
  if(strcmp(a,b)<0)
      {cout<<"Ordinea sirurilor este ";
       cout<<a<<' '<<b; }
   else
    if(strcmp(a,b)==0) cout<<"siruri
                              egale";
      else
      {cout<<"Ordinea sirurilor este ";
       cout<<b<<' '<<a;  }
 getch();
 }
```

4) Dându-se două şiruri de caractere, să se afişeze rezultatul concatenării lor.

Rezolvare. Primul program va realiza concatenarea prin adăugarea, caracter cu caracter, la sfârşitul primului şir, **a**, pe cel de-al doilea, **b**. Celălalt program va utiliza funcţia de concatenare, **strcat**, din biblioteca **string**, care realizează acelaşi lucru, fără însă a mai fi nevoie să fie calculate explicit lungimile celor două şiruri, în variabilele **m** şi **n**.

Ambele programe citesc datele din fişierul **siruri.txt**, iar şirul **a** are o lungime dublă faţă de **b**, pentru a putea primi şi caracterele din **b.**

```
//program concatenare fara
//functii din biblioteca string;
#include<fstream.h>
#include<conio.h>
void main()
{      ifstream f("siruri.txt");
       char a[20],b[10];
       int m=0,n=0;//lungimi siruri
       f>>a>>b;
       while(a[m])m++;
       while(b[n])
             a[m+n]=b[n++];
       a[m+n]=0;//atasare NULL
       cout<<"Sirul concatenat este=";
       cout<<a;
       getch();
}
```

```
//program concatenare siruri cu functia
//din biblioteca string

#include<fstream.h>
#include<conio.h>
#include<string.h>
void main()
{
       ifstream f("siruri.txt");
       char a[20],b[10];
       f>>a>>b;
       clrscr();
       cout<<"Sirul concatenat este=";
       cout<<strcat(a,b);
       getch();
       f.close();
}
```

5) Se cere formarea pe ecran a unui tabel care să conţină sinusurile unghiurilor de la 1^o la 10^o. Tabelul va trebui să aibă două coloane: pentru unghi şi pentru sinusul acelui unghi. Tabelul are un cap de tabel conţinând textul *UNGHI u* şi *SIN(u)*, subliniate apoi de un rând de 40 de liniuţe.

Rezolvare. Primul program va realiza tabelarea prin generarea sublinierilor pentru capul de tabel caracter cu caracter. Celălalt program va construi un şir, **a**, cu 39 de caractere în care iniţial va fi conţinutul capului de tabel. Apoi şirul **a** va fi completat cu 39 de liniuţe duble folosind funcţia **strset** din biblioteca **string**. Funcţia realizează înlocuirea caracterelor din şirul dat cu un acelaşi caracter, până la întâlnirea caracterului **NULL**.

În programele de mai jos sunt folosite facilităţile obiectului **cout** privind fixarea lungimii zonei de afişare şi a numărului de zecimale cu care să fie afişat un număr real. Astfel construcţia **cout.width(valoare)** va fixa

lungimea zonei de afişare la **valoarea** dată în paranteză. Pentru fixarea numărului de zecimale, se foloseşte setarea afişării prin **cout.precision(zecimale)**. În program, setările de mai sus s-au făcut pentru: **cout.width(10)** şi **cout.precision(6)**.

```
//program multiplicare caracter în sir
//fara functii din biblioteca string;
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()
{  char a[40];
   int u;
   clrscr();
   for(u=0;u<39;u++)
      a[u]='=';
   a[39]=0;
   cout<<"    UNGHI u       |";
   cout<<"      SIN(u) "<<endl;
   cout<<a<<endl;
   for(u=1;u<=10;u++)
      {cout.width(10);
      cout<<u<<"        |     ";
      cout.width(10);
      cout.precision(6);
      cout<<sin(u*M_PI/180)<<endl;
   }
   getch();
}
```

```
//program multiplicare caracter în sir
// cu functia din biblioteca string
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include<string.h>
void main()
{  char a[40];
   int u;
   clrscr();
   strcpy(a,"    UNGHI u       |
SIN(u)        ");//39 pozitii
   cout<<a<<endl;
   strset(a,'=');
   cout<<a<<endl;
   for(u=1;u<=10;u++)
      {cout.width(10);
      cout<<u<<"        |     ";
      cout.width(10);
      cout.precision(6);
      cout<<sin(u*M_PI/180)<<endl;
      }
   getch();
}
```

6) Se citeşte un şir de caractrere de la tastatură şi se doreşte ca el să fie apoi afişat inversat. De exemplu, dacă se citeşte şirul „Note de zece", atunci se va afişa „ecez ed etoN".

Rezolvare. Primul program va realiza schimbarea în oglindă a şirului în mod explicit, caracter cu caracter, până la jumătatea acestuia. Celălalt program va utiliza funcţia de inversare, **strrev** din biblioteca **string**.

```
//program inversare sir fara functii
//din biblioteca string;
#include<iostream.h>
#include<string.h>
#include<conio.h>
void main()
{
 char sir[100],aux;
  unsigned n;   clrscr();
  cout<<"Dati sirul de inversat ";
  cin.get(sir,100);
  n=strlen(sir);

  for(int i=0;i<n/2;i++)
   {aux=sir[i]; sir[i]=sir[n-i-1];
    sir[n-i-1]=aux; }
  cout<<"Sirul inversat este= "<<sir;
 getch();
}
```

```
//program multiplicare caracter în sir
// cu functia din biblioteca string
#include<fstream.h>
#include<conio.h>
#include<string.h>
void main()
{  char sir[100];
  clrscr();
  cout<<"Dati sirul de inversat ";
  cin.get(sir,100);
  strrev(sir);
  cout<<"Sirul inversat este= "<<sir;
getch();
}
```

7) Se citeşte un şir de caractrere de la tastatură şi se doreşte ca el să fie afişat apoi astfel:
a) cu toate literele schimbate în litere mici;
b) cu toate literele schimbate în litere mari.

Rezolvare. Primul program va realiza schimbarea în mod explicit, caracter cu caracter, pentru orice literă găsită în şir. Celălalt program va utiliza funcţia de schimbare, **strlwr** respectiv, **strupr**, din biblioteca **string**. Se va folosi faptul că distanţa între setul de litere mari şi cel de litere mici în lista codurilor **ASCII** este de 32.

```
//program modificare caractere in sir
//fara functii din biblioteca string;
a)
#include<iostream.h>
#include<conio.h>
void main()
{
  char a[20];
  clrscr();
  cout<<"dati sirul de transformat ";
  cin>>a;
  for(int i=0;a[i];i++)
   if(a[i]>='A'&&a[i]<='Z')a[i]+=32;
      cout<<"Sirul cu literele mici
                  este="<<a;
  getch();
}
```

```
b)
#include<iostream.h>
#include<conio.h>
void main()
{
   char a[20];
   clrscr();
   cout<<"dati sirul de transformat ";
   cin>>a;
   for(int i=0;a[i];i++)
    if(a[i]>='a'&&a[i]<='z')a[i]-=32;

   cout<<"Sirul cu literele mici
                        este="<<a;
   getch();
}
```

```
//program modificare caractere in sir
// cu functia din biblioteca string
a)
#include<iostream.h>
#include<string.h>
#include<conio.h>
void main()
{  char a[20];
   clrscr();
   cout<<"dati sirul de transformat ";
  cin>>a;
  cout<<"Sirul cu literele mici este=";
  cout<<strlwr(a);
  getch();
}
```

```
b)
#include<iostream.h>
#include<string.h>
#include<conio.h>
void main()
{  char a[20];
   clrscr();
   cout<<"dati sirul de transformat ";
 cin>>a;
 cout<<"Sirul cu literele mari este=" ;
 cout<<strupr(a);
getch();
}
```

8) Se consideră citit de la tastatură un şir de caractere. Să se afişeze de câte ori apare în şir vocala 'a'.

Rezolvare. Primul program va realiza căutarea vocalei şi numărarea apariţiilor în mod explicit, caracter cu caracter. Celălalt program va utiliza funcţia de localizare a unui caracter dat într-un şir, **strchr**, din biblioteca **string**.

Este importantă compararea celor două variante din punct de vedere al scrierii algoritmului. Enunţul a restricţionat prelucrarea la determinarea apariţiilor caracterului 'a', nu şi ale lui 'A'. Acest lucru se poate adăuga foarte uşor în prima variantă, modificând instrucţiunea **if**, care devine **if**(sir[i]=='a'||sir[i]=='A'); în a doua variantă trebuie rescrisă secvenţa de căutare şi numărare de la început, acum pentru un alt caracter, 'A'.

De asemenea, în varianta a doua, este pusă în evidenţă folosirea unei *variabile de tip adresă* către un conţinut **char** (pointerul **p**). Acest lucru este necesar, deoarece funcţia **strchr** realizează localizarea caracterului căutat şi returnează ca rezultat al căutării adresa în şir unde este caracterul sau **NULL** dacă nu există acel caracter. O următoare căutare a caracterului, pentru numărarea apariţiilor, trebuie să se facă începând cu caracterul de la adresa **p+1** din şirul dat.

```
//program localizare vocala a in sir
//fara functii din biblioteca string;
#include<iostream.h>
#include<conio.h>
```

```
void main()
{
  char sir[100];
  unsigned nr=0;
```

```
  clrscr();
  cout<<"Dati sirul de inspectat ";
  cin.get(sir,100);
  for(int i=0;sir[i];i++)
      if(sir[i]=='a')nr++;
  cout<<"Sirul contine vocala a de ";
  cout<<nr<<" ori";
  getch();

}

//program localizare vocala a in sir
// cu functia din biblioteca string
#include<conio.h>
#include<iostream.h>
#include<string.h>
void main()
{
  char sir[100],*p;
  unsigned nr=0;
  clrscr();
  cout<<"Dati sirul de inspectat ";
  cin.get(sir,100);
  p=strchr(sir,'a');
  while(p)
      {nr++;
       p=strchr(p+1,'a');
      }
  cout<<"Sirul contine vocala a de ";
  cout<<nr<<" ori";
  getch();
}
```

9) Se consideră citit de la tastatură un şir de caractere în variabila **sir**. Să se afişeze de câte ori apare în acest şir subşirul citit în variabila **ss**.

Rezolvare. Primul program va realiza în mod explicit căutarea subşirului şi numărarea apariţiilor lui, prin compararea caracterelor corespunzătoare poziţiilor lor în cele două şiruri. Celălalt program va utiliza funcţia de localizare într-un şir a unui subşir dat: **strstr** din biblioteca **string**. De exemplu, pentru `s="aabaaab"` şi `ss="ab"` se vor găsi *nr* =2 apariţii.

```
//program localizare subsir ss în sir
//fara functii din biblioteca string;
#include<iostream.h>
#include<string.h>
#include<conio.h>
void main()
{  char sir[100],ss[10];
   unsigned ds,dss,nr=0,ok;
   clrscr();
   cout<<"Dati sirul ";cin.get(sir,100);
   ds=strlen(sir);
   cout<<"Dati subsirul "; cin>>ss;
   dss=strlen(ss);
   for(int i=0;i<ds-dss;i++)
     if(sir[i]==ss[0])
      { ok=1;
        for(int k=1;k<dss;k++)
            if(sir[i+k]!=ss[k])ok=0;
        if(ok) nr++;
      }
    cout<<"Sirul "<<ss<<" apare de ";
    cout<<nr<<" ori in "<<sir;
 getch();
}
```

```
//program localizare subsir în sir
// cu functia din biblioteca string
#include<fstream.h>
#include<conio.h>
#include<string.h>
void main()
{
  char sir[100],ss[10],*p;
  unsigned nr=0,ok;
  clrscr();
  cout<<"Dati sirul ";
  cin.get(sir,100);
  cout<<"Dati subsirul ";
  cin>>ss;
  p=strstr(sir,ss);
  while(p)
   {nr++;
    p=strstr(p+1,ss);
    }
  cout<<"Sirul "<<ss<<" apare de ";
  cout<<nr<<" ori in "<<sir;
  getch();
}
```

1. Stabiliţi ce va afişa următoarea secvenţă:

```
char sir[10]="galben", aux[7],*p; int n=strlen(sir);
p=& sir[n/2];strcpy(aux,p);*p=0; strcat(aux,sir);cout<<aux;
```

2. Stabiliţi ce va afişa următoarea secvenţă:

```
char sir[10]="1234"; int n=sir[3]-'0'+sir[0]-'0'; cout<<n;
```

3. Ce valoare afişează următoarea secvenţă pentru variabila **c**, dacă se citesc următoarele 8 cuvinte:

Foaie Verde FOAIE Lata Foaie lunga foaie scurta

```
char a[10][7], a1[10]; int c=1; cin>>a[0];
for(int i =1;i<8;i++){cin>>a[i]; strcpy(a1,a[i]); if(!strcmp(a[0],a1))
c++; }
cout<<c;
```

a) 0; **b)** 4; **c)** 2; **d)** 3.

4. Precizaţi ce va afişa secvenţa următoare:

```
char a[25]="acesta este un exemplu", b[15]="un exemplu";
strcpy(a,b);cout<<a;
```

a) un exemplue un exemplu; **b)** acesta este un exemplu; **c)** un exemplu;
d) acesta este un exemplu un.

5. Stabiliţi ce va afişa următoarea secvenţă:

```
char sir[10]="clasa11B"; cout<<strstr(sir,"11");
```

6. Stabiliţi ce va afişa următoarea secvenţă:

```
char sir[10]="clasa11B"; cout<<strupr(sir);
```

7. Stabiliţi ce va afişa următoarea secvenţă:

```
char sir[10]="12345"; cout<<strrev(sir);
```

8. Stabiliţi ce se va încărca în variabila definită prin **char** `sir [11]` prin citirile succesive şi ce se va afişa:

```
cin.get(sir,10,'.');cout<<sir<<endl;
cin.get();cin.get(sir,10,'.');cout<<sir<<endl;
```

dacă la intrare se tastează *12.02.2006*?

9. Stabiliţi ce va afişa secvenţa:

```
char sir1[10]="foaie", sir2[10]="VERDE"; if(sir1<sir2)cout<<sir1; else
cout<<sir2;
```

10. Stabiliţi ce va afişa secvenţa:

```
char sir[12]="Informatica", cout<<strstr(sir,"ma");
```

11. Poziţia pe care se găseşte caracterul **c** în şirul **s** este dată de funcţia:

a) `strlen(s,c)` **b)** `strpos(s,c)` **c)** `strchr(s,c)` **d)** `strcpy(s,c)`.

12. Două şiruri de caractere notate cu **s** şi **t** sunt identice dacă:

a) `s[0]=t[0]` **b)** `strlen(s)=strlen(t)`
c) `strcmp(s,t!=0)` **d)** `strcmp(s,t)==0`.

2.5. Tablouri de șiruri de caractere

Se introduce ideea de ***compunere a structurilor***: şirurile de caractere se pot constitui ca elemente ale unui tablou unidimensional.

În zona de memorie internă, un astfel de tablou va fi alocat ca un tablou bidimensional de caractere. Acest lucru înseamnă că fiecare linie este un şir de caractere de aceeşi lungime, iar fiecare element este un un caracter.

De exemplu, dacă pentru 5 copii se doreşte înregistrarea numelor lor, de maximum 9 litere, eventual pentru ordonarea lor, atunci *repartizarea în memorie* a tabloului de nume, **`N[5][10]`**, va fi ca în figura 2.4.

coloana / *linia*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
0	I	o	n	e	l	**NULL**	**NULL**	**NULL**	**NULL**	**NULL**
1	A	n	a	**NULL**	**NULL**	**NULL**	**NULL**	**NULL**	**NULL**	**NULL**
2	V	l	a	d	i	m	i	r	**NULL**	**NULL**
3	M	a	g	d	a	l	e	n	a	**NULL**
4	G	i	g	e	l	**NULL**	**NULL**	**NULL**	**NULL**	**NULL**

Figura 2.4

Acest tablou nu trebuie, însă, parcurs caracter cu caracter pentru a fi prelucrat. El beneficiază de calitatea de şir a fiecărei linii, astfel că prelucrarea lui se va face global pentru fiecare linie în parte, ca pentru cinci şiruri de caractere. De exemplu, afişarea datelor din acest tablou va fi scrisă:

```
for(int i=0;i<5;i++) cout<<N[i]<<" ";
```

Mecanismul de lucru cu tablouri de şiruri de caractere
Exerciţiile se vor pune în execuţie în orele de laborator.

1) Se citesc valori naturale între 1 şi 12 ce reprezintă numere de ordine a lunilor în an. Se cere afişarea numelui lunii corespunzătoare fiecărui număr citit.

Rezolvare. Programul repetă intrarea de număr natural până când utilizatorul apasă tasta *n*, moment în care prelucrarea se opreşte. Se va utiliza un tablou unidimensional ale cărui elemente sunt de tip *şir de caractere*, şi anume, numele lunilor. Indicele de tablou al fiecărei luni va fi folosit ca reper numeric pentru identificarea numelui lunii. Numele lunilor au fost generate într-un tablou de constante de tip şir de caractere, **a**:

```
const char a[12][11]={"ianuarie","februarie","martie","aprilie","mai","iunie",
      "iulie", "august","septembrie","octombrie","noiembrie","decembrie"};
```

Astfel, dacă se citeşte numărul 4, atunci se va repera locul 3 din tablou şi astfel va fi localizat numele „aprilie”.

```
#include<iostream.h>
#include<conio.h>
void main()
{
const char
a[12][11]={"ianuarie","februarie","martie",
"aprilie","mai","iunie","iulie","august",
"septembrie","octombrie","noiembrie",
"decembrie"};
unsigned nr;char c;
do
 { clrscr();
do
{cout<<"Numar de doua cifre,intre 1 si
12: ";
  cin>>nr;
 }while (nr<1 || nr>12);
 cout<<" Este  luna "<<a[nr-1]<<endl;
 cout<<"Mai introduceti numere de luni?
";
 cout<<"Pt. oprire tastati N, altfel orice
            tasta ";
 cin>>c;
}while(c!='N'&& c!='n');
getch();
}
```

2) Dându-se lista numelor celor maximum 30 de elevi ai unei clase, să se obţină lista numelor lor ordonată alfabetic.

Rezolvare. Se au în vedere operaţiile de comparare a şirurilor de caractere, numai că fiecare şir reprezintă un element din tabloul numelor elevilor, **elevi**. Ordinea se stabileşte lexicografic (pe principiul dicţionarului).

Datele se citesc dintr-un fişier, **clasa.in**. De exemplu, fişierul conţine numele a 7 elevi, cum se vede în figura 2.5.

clasa.in
```
7
Ionel
Gigel
Ana
Mariana
Bianca
Vladimir
Razvan
```

Figura 2.5

```
#include<fstream.h>
#include<string.h>
#include<conio.h>
void main()
{char elevi[30][15],aux[15];
unsigned i,n,ok,m;
ifstream cl("clasa.in");
cl>>n;
m=n;
for(i=0;i<n;i++)
      cl>>elevi[i];
do //ordonare prin metoda bulelor
 {ok=1;
  for(i=0;i<n-1;i++)
if(strcmp(elevi[i],elevi[i+1])>0)
       {ok=0;
        strcpy(aux,elevi[i]);
        strcpy(elevi[i],elevi[i+1]);
        strcpy(elevi[i+1],aux);
       }
  n--;
 }while(!ok);
cout<<"Lista numelor ordonate
          alfabetic:"<<endl;
for(i=0;i<m;i++)
  cout<<elevi[i]<<endl;
getch();
}
```

3) Se consideră un tablou bidimensional cu **m** x **n** elemente, numit **matr**. Fiecare element este şir de caractere. Să se realizeze un program care identifică cel mai lung şir de pe fiecare linie din tablou.

Rezolvare. Se au în vedere operaţii de comparare a lungimilor şirurilor de caractere, numai că fiecare şir reprezintă un element din tabloul bidimensional **matr** care are **m** linii şi **n** coloane.

```
#include<iostream.h>
#include<string.h>
#include<conio.h>
void main()
{char matr[20][20][20],max[20];
 unsigned m,n,i,j,lmax,len;
 clrscr();
 do
  {cout<<"Dati dimensiunile matricei ";
   cin>>m>>n;
  }while(n>20||m>20);

 for(i=0;i<m;i++)
  {lmax=0;strcpy(max,"");
   for(j=0;j<n;j++)
    {cout<<"matr["<<i+1<<","<<j+1<<"]=";
    cin>>matr[i][j];
    len=strlen(matr[i][j]);
    if(len>lmax)
      {lmax=len;
       strcpy(max,matr[i][j]);
      }
    }
cout<<"Pe linia "<<i+1 ;
cout<<"cuvantul de lung. maxima este: "
cout<<max<<endl;
}
getch();
}
```

4) Se consideră un şir de **n** cuvinte reprezentând numele disciplinelor din orarul zilei curente unui elev. Se cere afişarea listei materiilor pe coloane, în ordine lexicografică.

Dacă **n**=5 şi disciplinele sunt: „*chimia*", „muzica","engleza", „*matematica*", „*informatica*", atunci afişarea va fi similară celei din figura 2.6.

Rezolvare. Numele celor cinci materii vor fi aranjate într-un tablou de cinci şiruri de caractere. Acest lucru revine, de fapt, la alocarea unui spaţiu de cinci linii şi 12 coloane, deoarece cea mai lungă denumire are 11 litere. Rezultă că se va aloca un spaţiu bidimensional. Aşezarea în memorie

c	e	i	m	m
h	n	n	a	u
i	g	f	t	z
m	l	o	e	i
i	e	r	m	c
a	z	m	a	a
	a	a	t	
		t	i	
		i	c	
		c	a	
		a		

Figura 2.6

Lmax

c	*h*	*i*	*m*	*i*	*a*	0	0	0	0	0	0
e	*n*	*g*	*l*	*e*	*z*	*a*	0	0	0	0	0
i	*n*	*f*	*o*	*r*	*m*	*a*	*t*	*i*	*c*	*a*	0
m	*a*	*t*	*e*	*m*	*a*	*t*	*i*	*c*	*a*	0	0
m	*u*	*z*	*i*	*c*	*a*	0	0	0	0	0	0

Figura 2.7

a tabloului şi conţinutul lui după ordonarea lexicografică, va fi cea din figura 2.7. Din acest aranjament se observă că fiecare rând afişat pe ecran reprezintă conţinutul unei coloane din tabel. Astfel, se va parcurge tabelul pe coloane, până la coloana **lmax** şi se va afişa conţinutul fiecărei coloane pe câte un rând de ecran.

Temă de laborator: Completaţi rândul de puncte din programul de mai jos cu secvenţa de ordonare a şirurilor denumirilor materiilor.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()
{ char materii[16][12],aux[12];
  int n,i,len,Lmax=0;
  cout<<"dati numarul de materii ";
  cin>>n;
  for(i=0;i<n;i++)
     {cin>>materii[i];
 len=strlen(materii[i]);
 if(len>Lmax)     Lmax=len;
}
 ..................
for(int j=0;j<Lmax;j++)
  { for(i=0;i<n;i++)
       cout<<materii[i][j]<<' ';
    cout<<endl;  } }
```

2.6. Operații de conversie a unui *șir* de *caractere – cifre* într-o valoare *numerică* și invers.

Există situaţii în care o înşiruire de caractere de tip cifră conţine o informaţie numerică ce trebuie prelucrată prin operaţii matematice.

Expresia numerică: ***51+7-2*9***, figura 2.8, în care 51, 7, 2 şi 9 sunt constante întregi, trebuie evaluată pentru a se crea valoarea –6 cu care ar continua prelucrările mai departe în algoritm. Dacă expresia respectivă este citită ca dată comunicată de utilizator, forma în care ea se va regăsi în datele de intrare va fi forma şirului de caractere corespunzătoare cifrelor şi semnelor de operaţii pe care le conţine.

caractere	'5'	'1'	'+'	'7'	'-'	'2'	'*'	'9'
cod **ASCII**	53	49	43	55	45	50	42	57

Figura 2.8

Pentru a fi calculată expresia, însă, codurile **ASCII** ale cifrelor 5 şi 1 vor trebui să participe la crearea numărului întreg 51, iar din codul **ASCII** al cifrelor 7, 2 şi 9 trebuie determinate numerele întregi 7, 2 şi 9, valori care vor fi reprezentate în memorie în baza 2 şi cu acestea se vor face calculele matematice cerute.

În alte situaţii, cum este cea a afişării unor valori numerice interne, trebuie ca pe ecran să se prezinte şirul de caractere corespunzător numerelor. Această situaţie este întâlnită atât în mod implicit, în funcţionarea fluxului de ieşire, `cout`, pentru valori numerice, cât şi în mod explicit, în situaţia scrierii grafice într-un desen, prin funcţiile `outtext` şi `outtextxy`.

'1'	'2'	**NULL**
49	50	0

Figura 2.9

De exemplu, valoarea internă 00001100_2, care reprezintă numărul natural 12_{10}, trebuie scrisă pe ecran în mod grafic la coordonatele (200,150) prin intermediul funcţiei `outtextxy(200,150,sir)`. Variabila `sir` va trebui pregătită cu conţinutul **ASCII** al valorii interne 12_2 , cum se prezintă în figura 2.9.

Transformarea din format ***şir de caractere*** în format ***numeric*** intern a unei date *şi invers* este o operaţie de conversie a tipului de reprezentare.

Ea este o conversie mai complicată decât cele realizate de operatorul **cast**, pus la dispoziţie de limbaj. Astfel, dacă se poate scrie:

```
float a= (float)1/2;
```

pentru a se realiza conversia valorii 1 din tipul întreg în tipul real şi astfel calculul să dea 0.5 nu 0, **nu** la fel se poate scrie:

```
int a= char (12);
```

Biblioteca **stdlib**, pune la dispoziţie funcţii pentru conversii de acest tip. Modul de transformare este foarte uşor de recunoscut după numele funcţiei care este un acronim rezultat din sarcina acesteia. În tabelul din figura 2.10 sunt date câteva funcţii uzuale.

SENS CONVERSIE	NUME FUNCŢIE	EFECT	Explicaţia acronimului
şir → număr	`atoi(sir)`	Transformă *şirul de cifre* în valoare numerică întreagă.	**A**SCII **to** **i**nt
	`atol(sir)`	Transformă *şirul de cifre* în valoare numerică long.	**A**SCII **to** **l**ong
	`atof(sir)`	Transformă *şirul de cifre* în valoare numerică reală dublă precizie. Dacă şirul conţine mai multe puncte zecimale, conversia se opreşte la al doilea. Funcţia întoarce 0 dacă nu poate converti şirul în număr.	**A**SCII **to** **f**loat
număr → şir	`itoa(n,sir,b)`	Transformă numărul **n** din baza **b** în şir de caractere.	**i**nt **to** **A**SCII
	`ltoa(n,sir,b)`	Transformă numărul **n** din baza **b** în şir de caractere.	**l**ong **to** **A**SCII

Figura 2.10

Sarcină de laborator: Să se inspecteze funcţiile din fişierul antet **stdlib.h** cu ajutorul programului de asistare – **help** – al mediului de programare.

1. Scrieţi secvenţa de prelucrări prin care se realizează sarcina funcţiei **atoi** pentru şirul citit în variabila **s**, considerând că nu există funcţia respectivă în biblioteca limbajului.
2. Scrieţi secvenţa de prelucrări prin care se realizează sarcina funcţiei **atof** pentru şirul citit în variabila **s**, considerând că nu există funcţia respectivă în biblioteca limbajului.
3. Scrieţi secvenţa de prelucrări prin care se realizează sarcina funcţiei **itoa** pentru şirul citit în variabila **s**, considerând că nu există funcţia respectivă în biblioteca limbajului.

2.7. Validarea sintactică a datelor de intrare

Prelucrarea eşuează dacă datele de intrare sunt introduse cu erori. Acest lucru se întâmplă dacă utilizatorul unui program este o persoană oarecare, fără cunoştinţe de informatică sau având cunoştinţe precare. Chiar şi un utilizator experimentat poate greşi la introducerea valorilor, din grabă sau din neatenţie. Sarcina expresă a oricărei prelucrări a datelor este să asigure validarea datelor de intrare (testele de corectitudine). Validarea presupune două forme.

Formele de validare:
– validarea **sintactică** stabileşte corectitudinea formei în care intră o valoare;
– validarea **logică** stabileşte corectitudinea conţinutului datei.

Validarea sintactică se bazează pe **forma de şir de caractere sub care intră iniţial valoarea**. Şirul este verificat din punct de vedere sintactic şi apoi conţinutul şirului este convertit în tipul de reprezentare pe care trebuie să-l aibă acea valoare în prelucrările ulterioare.

Altfel, valoarea greşită ar intra direct în variabila căreia îi este destinată şi ar apărea o contradicţie de tip de reprezentare, finalizată cu întreruperea, cu eroare, a programului.

De asemenea, unele **validări logice** se servesc de forma intermediară de tip şir de caractere pentru situaţiile în care, deşi scrise corect, valorile nu ar intra în limitele necesare prelucrării. De exemplu, o variabilă de tip **int** ar primi valoarea 70 000, ceea ce ar produce o eroare logică, deşi 70 000 este corect scris ca număr în sine. Astfel, testat ca şir de caractere, 70 000 va fi remarcat în afara domeniului de reprezentare pentru întregi, fără ca programul să ruleze în continuare cu valoarea greşită.

Exerciţii rezolvate – validare sintactică
Exerciţiile se vor pune în execuţie ca teme de laborator.

1) Se consideră situaţia în care se citeşte o fracţie zecimală. Deoarece se poate greşi la tastarea valorii, atunci se face o verificare sintactică a datei introduse, înainte de a fi prelucrată ca număr real (**float** sau **double**). Greşelile sintactice pot consta în scrierea de caractere diferite de cifră sau în scrierea virgulei în loc de punctul zecimal.

Programul de mai jos preia informaţia tastată în şirul de caractere **sir** şi îi aplică funcţia **atof** pentru a încerca transformarea în număr real. Dacă funcţia reuşeşte transformarea, atunci va furniza fracţia zecimală corespunzătoare, ca reprezentare internă în virgulă mobilă, iar dacă transformarea eşuează, atunci funcţia va întoarce valoarea zero.

```
//validarea sintactica a intrarii unei
//fractii zecimale
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
void main()
{     char sir[20];
      double f;
clrscr();
cout<<"Dati fractia zecimala ";
cin>>sir;
f=atof(sir);
if(f)cout<<"Fractia este corecta "<<f;
 else cout<<"Fractia nu este corecta
";
getch();}
```

Cum funcţia **atof** nu sesizează situaţia în care se introduce chiar numărul 0.0 şi întoarce tot zero, ca pentru eroare, trebuie ca transformarea să se complice puţin, după cum este prezentată în programul de mai jos, în care se face testarea explicită a caracterelor din şirul **sir**.

```
//validarea sintactica explicita
//a unei fractii zecimale
#include<iostream.h>
#include<string.h>
#include<conio.h>
#include<stdlib.h>
void main()
{char sir[20],*p;
 double f; int ok=1;
 clrscr(); cout<<"Dati fractia zecimala ";
 cin>>sir;
 p=strchr(sir,'.');
 if(p)
  {p=strchr(p+1,'.');
   if(p) *p=NULL; }//oprire la primul •
if((sir[0]<'0'||sir[0]>'9')&&sir[0]!='-')
      ok=0;
 else
  for(int i=1;sir[i]&&ok;i++)
   if((sir[i]<'0'||sir[i]>'9'||sir[i]==',')
     && sir[i]!='.')
      ok=0;
 if(ok)
  {f=atof(sir);
   cout<<"Fractia este corecta "<<f;
}
  else cout<<"Fractia nu este corecta ";
      getch();
}
```

2) Se consideră situaţia în care se citeşte o notă şcolară. La citirea valorii, pot apărea erori de tastare, în sensul că se poate introduce alt caracter decât cifră sau se pot introduce valori care nu sunt note, adică sunt în afara limitelor 1 şi 10. Programul de mai jos prezintă o modalitate de a verifica dacă valoarea introdusă de la tastatură este o notă sau nu. S-a folosit funcţia **atoi**.

```
//validarea unei note scolare
#include<iostream.h>
#include<string.h>
#include<conio.h>
#include<stdlib.h>
void main()
{ char a[3];
  unsigned nota;
  clrscr();
  //citire nota
cout<<"Dati nota scolara ";
cin>>a;
//conversie sir in nr. natural
nota=atoi(a);
if(nota<1 || nota>10)
      cout<<"Nota este eronata ";
 else
      cout<<"Nota este corecta: "<<nota;
      getch();
}
```

Sarcini de laborator. Rulaţi programul de mai sus pentru situaţia în care şirul de intrare este "1a" în loc de "10". Notaţi ce se afişează şi rulaţi apoi pas cu pas pentru a vedea modificarea valorilor în variabilele programului.

3) Se citeşte o succesiune de maximum 8 caractere, din mulţimea {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'}, care reprezintă un număr scris în baza 16. Primul caracter nu trebuie să fie '0'. Să se afişeze valoarea în baza 10 a numărului hexazecimal.

Rezolvare. Se consideră scrierea polinomială a unui număr într-o bază oarecare, *b*:

$$a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \ldots + a_1 \cdot b^1 + a_0 \cdot b^0$$

unde cu a_i (cu *i* de la **n** la 0) s-au notat cifrele numărului în baza **b**. Evaluarea polinomului va consta în calculul progresiv, înlocuind coeficienţii cu valoarea cifrei respective în baza 16, iar pe **b** cu 16, pe baza grupării Horner:

$$(\ldots((a_n +0)\times b + a_{n-1})\times b + \ldots)\times b + a_1)\times b^1 + a_0 \times b^0.$$

```
//transformare din baza 16 in baza 10
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()
{
long nz=0;char h[8];int rest,i,x,n;
clrscr();
cout<<"Dati nr. hexa ";
cin>>h;
strupr(h);//transf. in litere mari
n=strlen(h);
x=1;
for(i=0;i<n;i++)
  if(!((h[i]>='0'&& h[i]<='9')||
       (h[i]>='A'&& h[i]<='F'))) x=0;
if(x)
  for(i=0;i<n;i++)
   {if(h[i]>='0'&&h[i]<='9')
            rest=h[i]-'0';
      else
            rest=h[i]-'A'+10;
    nz=nz*16+rest;
   }
cout<<"Nr.in baza zece : "<<nz;
getch();}
```

Sarcini de laborator. Adăugaţi în program testul pentru zerourile nesemnificative (ignorarea primelor caractere de tip cifră zero până la întâlnirea primului caracter hexa de tip cifră semnificativă).

4) Validarea sintactică a unei date calendaristice introdusă în formatul numeric: ***zz/ll/aaaa***, unde ***zz*** este numărul zilei, ***ll*** este numărul lunii şi ***aaaa*** este numărul anului.

Rezolvare. Programul verifică dacă s-a folosit separatorul '/' şi dacă în zonele datei sunt cifre. De asemenea, se verifică (ceea ce nu era obligatoriu pentru validarea sintactică), dacă cifrele care alcătuiesc ziua şi luna sunt între limitele de scriere ale unei zile şi, respectiv, ale unei luni din calendar. Urmează ca în alt program să fie făcută validarea logică a datei (corectitudinea valorică), după ce grupele numerice ale datei au fost transformate în numere. A fost organizată constanta **mes** pentru a furniza textul erorii pe care o va afişa programul. Vectorul **este**, cu patru elemente întregi, va înregistra valoarea 1 în componenta de indice corespunzător indicelui textului erorii din vectorul **mes**: de exemplu, **este[0]←1**, dacă s-a întâlnit un separator greşit (adică eroarea **mes[0]**).

```
//program validare_sintactica
//data calendaristica;
#include<iostream.h>
#include<string.h>
#include<conio.h>
#include<stdlib.h>
void main()
{const char
  mes[4][17]={"Separator gresit","Zi
   gresita","Luna gresita","an gresit"};
char a[11];
int  este[4]={0},i;
cout<<"Introduceti data in format
  numeric: zi/luna/an ";
cin>>a;
if(a[2]!='/'||a[5]!='/')este[0]=1;
if(a[0]<'0'||a[0]>'3'||a[1]<'0'||a[1]>'9')
             este[1]=1;
if(a[3]<'0'||a[3]>'1'||a[4]<'0'||a[4]>'2')
             este[2]=1;
for(i=0;i<3;i++)
 if(a[i+6]<'0'||a[i+6]>'9') este[3]=1;
for(i=0;i<4;i++)
 if(este[i]) cout<<mes[i]<<endl;
getch();
}
```

Sarcini de laborator:

1. Realizaţi un program în care să se valideze logic o dată calendaristică scrisă în format numeric zz/ll/aaaa.
2. Transformaţi programul de mai sus, astfel încât să folosiţi funcţii din biblioteca **string**.

1. Care dintre cele două secvenţe de mai jos afişează prima literă mică din şirul **a**?

```
i=0; while(a[i]&& (a[i]<'a'||a[i]>'z'))
i++; cout<<a[i];
i=0;do i++;
while(a[i]&&!(a[i]>='a'&&a[i]<='z'));
cout<<a[i];
```

2. Care dintre operaţiile de scriere din secvenţa de mai jos afişează textul *aa*?

```
char a[4][2]={"xa","ax","ya","ay"}; cout<<a[3][0]<<a[1][0];
cout<<a[[1][0]<<a[2][1]; cout<<a[1][1]<<a[2][0];
```

3. Stabiliţi valoarea de adevăr a afirmaţiei: pentru trei variabile de tip şir de caractere, declarate: **char** `s1[10],s2[10],s3[10];` instrucţiunea **if**`(strcmp(s1,s2)&&strcmp(s2,s3))` `cout<<1;` va afişa 1 dacă toate cele trei şiruri sunt egale.

4. Pentru declarările de variabile: **char** `s[4]="123",t[4];` **int** `x=123,y;` care dintre expresiile următoare au valoarea zero?

a) `atoi(s)!=x;` **b)** `itoa(x,t,10)==s;` **c)**`!strcmp(itoa(x,t,10),s);`
d) `x==(atoi(itoa(x,t,10));`

5. Care dintre următoarele prelucrări sunt scrise greşit, ştiind că se dau declarările: **char** `*p, s[10]`?

a) `p=strchr('m',s);` **b)** `p=strstr(s,"ma")-s;` **c)** `cout<<strlen("sir");`
d) `cout<<strcat("con","cat");`

6. Ce se va afişa în urma executării secvenţei de mai jos ?

```
char s[10]="Amalia"; for(int i=1;i<4;i++) strcpy(s+1,s+2); cout<<s;
```

7. Ce se va afişa în urma executării secvenţei de mai jos?

```
char s[10]="Amalia",*p; s[0]=s[0]+32;
p=strchr(s,'i');cout<<s[0]<<p[0]<<s[strlen(s)-1];
```

PROBLEME PROPUSE

Pentru problemele de mai jos se vor realiza programele corespunzătoare în cazurile generale.

1) Să se afişeze câte caractere albe sunt într-un şir de caractere citit.
2) Se dă un şir de caractere; să se întoarcă şirul invers, în aceeaşi zonă de memorie, şi să se afişeze rezultatul.
3) Se citeşte dintr-un fişier un text terminat cu caracterul punct. Să se stabilească numărul de litere, numărul de separatori şi numărul de cuvinte din text. Cuvintele sunt separate prin spaţii albe, punct şi virgulă sau două puncte.
4) Se dă un şir de caractere; să se afişeze un mesaj corespunzător dacă şirul este palindrom sau nu.
5) Se dă un şir de caractere **S**; să se afişeze de câte ori se regăseşte în el subşirul **s** (citit) încadrat de câte un spaţiu. Dacă şirul **s** apare chiar la începutul şirului **S**, el trebuie să fie urmat de un spaţiu.
6) Se citeşte un şir de caractere care reprezintă scrierea unei fracţii zecimale oarecare. Dacă fracţia zecimală este corectă, să se transforme în fracţie ordinară şi să se afişeze numărătorul şi numitorul obţinute.
7) Se citesc maximum 20 de cuvinte reprezentând nume de culori. Să se afişeze câte seturi de culori ale spectrului au intrat, prin set înţelegând o succesiune secvenţială de nume de culori cu proprietatea cerută.
8) Dându-se data curentă şi numele din săptămână al zilei în care a fost 1 ianuarie al anului curent, să se afişeze numele zilei desemnat de data curentă.
9) Se citeşte un număr în baza 10. Să se afişeze valoarea respectivă scrisă în baza 16.
10) Se citesc **n** şiruri de caractere care ar reprezenta mediile generale ale unor elevi. Să se calculeze media generală a grupului, dacă datele de intrare sunt corecte.
11) În fişierul **rebus.txt**, pe prima linie se află o valoare naturală care se va citi în variabila **n**. Apoi se citesc **n** linii ce reprezintă conţinutul orizontalelor dintr-un careu de rebus. Punctele negre sunt reprezentate prin caracterul *****. Să se afişeze **n** linii care reprezintă conţinutul verticalelor careului.
12) Pentru aceleaşi condiţii ca în problema 10, să se determine numărul de cuvinte din careu, ştiind că un cuvânt are minimum două litere.
13) Se citeşte un text de la tastatură. Din acest text să se extragă constantele numerice şi să se adune. Programul va afişa suma obţinută. Exemplu: dacă se citeşte şirul *ab12x!345* se va afişa suma **357** obţinută din 12 + 345.
14) Scrieţi un program prin care se realizează prelucrarea pe care o face funcţia **strstr** dar determină un întreg ce reprezintă poziţia în şir unde începe subşirul căutat, sau –1 în cazul în care subşirul nu este găsit.
15) ***Temă pentru portofoliu***. Să se elaboreze un program care citeşte din fişierul **text.in** câte un cuvânt de pe fiecare linie şi afişează acel cuvânt despărţit în silabe. Se utilizează următoarele reguli de despărţire în silabe, fără a se trata excepţiile:
– o consoană aflată între două vocale trece în silaba a doua;
– pentru două sau mai multe consoane aflate între două vocale, prima rămâne în silaba întâi, iar celelalte trec în silaba a doua.

Capitolul 3 Tipul de date înregistrate

În acest capitol veţi învăţa despre:

- Datele eterogene, cum apar în realitate şi modul lor de structurare
- Tipul de date înregistrare (articol) şi proiectarea înregistrărilor
- Declararea tipului şi definirea variabilelor de acest tip
- Organizarea prelucrărilor înregistrărilor
- Tablouri de înregistrări

3.1. Noţiunea de înregistrare

O *metodă generală de compunere* a datelor într-o structură unitară o oferă regula de structură numită ***înregistrare sau articol***.

Acest tip de structură apare necesar când utilizatorul doreşte prelucrarea informaţiilor care se referă la entităţi complexe şi care sunt descrise prin elemente ***eterogene***. Eterogenitatea provine din faptul că elementele pot fi de tipuri diferite de date.

Elementele structurii se numesc **câmpuri**.

exemplu

Fie un lot de **n** candidaţi care se prezintă la o testare, fiecare pe rând înregistrându-şi datele personale formate din *nume* şi *data naşterii*. Trebuie selectate şi listate persoanele care au vârsta de cel puţin 21 de ani.

Prelucrarea cerută mai sus are nevoie de o structură nouă de date, numită, de exemplu, *persoana* în care să se regăsească datele unei persoane care este verificată din punctul de vedere al vârstei. Totodată, informaţiile legate de datele calendaristice necesare în prelucrare (data curentă şi data naşterii), sunt şi ele date compuse din elementele: zi, lună, an.

Pentru datele referitoare la o persoană, rezultă gruparea de informaţii prezentată în tabelul din figura 3.1, alcătuită din tipurile de date menţionate sub ultimul rând:

NUME	DATA NAŞTERII		
	ZI	LUNA	AN
şir de caractere	nr. natural	nr. natural	nr. natural

Figura 3.1

Această schemă de structură este numită ***macheta înregistrării*** *(sau şablonul de structură)****.***

În cadrul **machetei înregistrării** sunt puse în evidenţă:
– ***câmpurile*** (elementele) înregistrării;
– ***tipul de date*** asociat fiecărui câmp;
– modul de ***grupare şi succesiune*** a câmpurilor.

Odată configurată macheta înregistrării, se pot desemna identificatori pentru câmpuri.

În exemplul de mai sus, pentru DATA NAŞTERII se poate crea identificatorul **`datan`**. În rest, celelalte denumiri pot deveni identificatori, eventual scrişi cu litere mici.

Proiectarea unei înregistrări

a) În faza iniţială, **se fixează entitatea** care va fi descrisă prin informaţiile viitoarei înregistrări. În mod concret, entitatea poate fi un obiect fizic, o fiinţă, un fenomen, un concept etc.
b) Se **stabileşte prelucrarea** în care va fi folosită acea entitate.
c) Se stabileşte apoi lista de aspecte, **atribute ale entităţii**, care trebuie luate în considerare pentru prelucrarea fixată.
d) Se configurează **macheta înregistrării**, pentru fiecare câmp stabilindu-se tipul – **mulţimea de valori**.
e) Se atribuie **identificatori** câmpurilor.

exemplu

1. Fie entitatea *elev*. Această entitate poate fi descrisă cu foarte multe atribute: nume, prenume, data naşterii, clasa, adresa, telefonul, numele şi prenumele părinţilor, locul de muncă al fiecărui părinte, boli avute, înălţime, greutate, rezistenţă la efort, rezultate la concursuri, medie generală pe semestre sau an etc. Se observă astfel că este foarte important să se stabilească mai întâi prelucrarea în care va intra entitatea *elev* şi apoi alegerea atributelor corespunzătoare.

Dacă se alege *prelucrarea statistică a rezultatelor lui şcolare* la finele semestrului I, atunci vor fi necesare *atributele*: nume, prenume, mediile la obiectele de studiu, media la purtare şi media generală. Prelucrarea va consta în determinarea calităţii de *promovat* al semestrului şi calculul mediei semestriale sau de *nepromovat*, fără a se mai calcula media semestrială.

Macheta proiectată pentru prelucrarea statistică va fi cea din figura 3.2:

Nume	Prenume	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	Medie generală
15 ch	15 ch	2n	2n	2n	2n	2n	2n	2n	2n	2n	2n	2n	2n	2n	2n	2n	2n	5.2

Figura 3.2

În rândul al doilea al tabelului din figura 3.2 s-au trecut, prescurtat, tipul valorilor corspunzătoare fiecărui câmp: **`ch`** pentru şir de caractere, **`2n`** pentru număr natural de 2 cifre, **`5.2`** pentru număr real cu două zecimale şi două cifre la partea întreagă (5 reprezintă lungimea numărului real: doi întregi, virgula, două zecimale).

După stabilirea identificatorilor, macheta va fi cea din figura 3.3.

N	P	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	MG
15 ch	15 ch	2n	2n	2n	2n	2n	2n	2n	2n	2n	2n	2n	2n	2n	2n	2n	2n	5.2

Figura 3.3

Se observă că grupul omogen al mediilor pe obiecte poate deveni un vector de 16 elemente naturale.

2. Fie entitatea *carte*. Obiectul *carte* poate fi descris prin *atributele*: titlu, autor, preţ, an apariţie, editură.

Dacă se stabileşte, însă, ca *prelucrare*, evidenţa stocului de carte dintr-o bibliotecă şcolară, atunci mai apar necesare atributele: cod carte şi număr de exemplare. *Macheta*, în forma finală, după stabilirea identificatorilor, va fi cea din figura 3.4.

Cod	Titlu	Autor	Preţ	An_apar	Editura	Nr_ex
6n	30 ch	20 ch	7.2	4n	15 ch	2n

Figura 3.4

Astfel, vor putea fi făcute calcule privind: valoarea cărţilor din tot stocul de carte (prin însumarea produselor **`Pret x Nr_ex`** pentru toate cărţile existente), clasificarea cărţilor după anul de apariţie, ordonarea cărţilor după autor etc.

rezolvă

Proiectaţi şi desenaţi pe caiet înregistrările necesare pentru următoarele cazuri:

1. Fie entitatea *vânzări* ale unor produse. Acest la fenomen se desfăşoară pentru fiecare produs în parte din stocul unui magazin, zilnic, în cantităţi şi preţuri date. Prelucrările vor fi: stabilirea valorii totale a vânzărilor zilnice şi listarea produselor celor mai solicitate.
2. Fie entitatea *număr complex*. Pentru această noţiune abstractă, prelucrările vor fi: realizarea operaţiilor de adunare, înmulţire şi împărţire a două numere complexe.
3. Fie entitatea *punct*. Prelucrările vor fi: stabilirea tuturor distanţelor între mai multe puncte date şi care puncte sunt coliniare.
4. Fie entitatea *străzi* din cadrul unui oraş. Prelucrările vor fi: listarea străzilor cu sens unic şi calcularea totalului distanţelor în kilometri pentru străzile cu dublu sens.
5. Fie entitatea *filme* care rulează în oraş. Prelucrările vor fi: listarea filmelor care încep la o oră dată şi listarea sălilor care au cele mai multe reluări ale filmului în zi.

3.2. Specificațiile limbajului de programare pentru înregistrări

Declararea machetei unei înregistrări utilizează cuvântul rezervat al limbajului: **struct**.

Prin această declarare programatorul ***creează un tip nou de dată*** față de cele standard, cunoscute de compilator. Regula generală de sintaxă cere ca declararea câmpurilor împreună cu tipurile lor să se scrie între acolade. Ca orice declarare de date, formularea se încheie cu semnul punct și virgulă:

struct *nume_structură* {*tip*$_1$ *listă câmpuri; tip*$_2$ *listă câmpuri; … ; tip*$_n$ *listă câmpuri*}*;*

Noul tip de dată creat de programator este un tip structurat și are denumirea pe care acesta o dă în `nume_structura`.

Pentru prelucrarea informațiilor se ***definesc variabile de acest tip*** nou introdus, numite ***variabile_înregistrare***.

Definirea variabilelor_înregistrare se poate face în două moduri:

a) după declararea machetei, prin:

```
nume_structura nume_variabila;
```

b) odată cu declararea machetei, prin:

```
struct nume_structura{tip1 camp1; tip2 camp2;…} nume_variabila1, nume_variabila2...;
```

Pentru forma **b)** a declarării variabilelor_înregistrare se poate scrie declararea machetei fără a mai menționa `nume_structura`. Este cazul definirii unei ***structuri anonime***, importante fiind doar declararea machetei și a variabilelor:

```
struct {tip1 camp1; tip2 camp2;…} nume_variabila1, nume_variabila2...;
```

Calculul lungimii în octeți a zonei ocupate de o variabilă de tipul structurii declarate se face însumând numărul de octeți corespunzători tipurilor câmpurilor.

1. Fie o grupare de informații care descriu situația școlară a unui elev după macheta din figura 3.5.

Nume	Medie
25 ch	**float**

Figura 3.5

Declararea structurii va fi:

```
struct elev{char nume[25]; float medie;};
```

Structura conține un cîmp, **nume**, care, la rândul său, este o structură de date, vector de caractere.

Pentru structura **elev** se pot lua variabilele **e1** și **e2**, de exemplu, declarate ca fiind de tipul **elev**:

```
elev  e1, e2;
```

Calculul de lungime arată că atât pentru **e1**, cât și pentru **e2**, sunt repartizați câte 25+4 =29 octeți.

2. Pentru exemplul dat mai sus, privind persoana care se prezintă la o testare, structura va cuprinde două câmpuri: **nume** (pentru numele și prenumele persoanei) și **datan** (pentru informația *data nașterii*).

Pentru că informația **datan** este, la rândul ei, o *dată compusă* din trei elemente, ea va fi declarată ca structură **data**, având câmpurile: **zi**, **luna**, **an**. Declararea machetei pentru informațiile unei persoane va fi scrisă prin două declarări de tip **struct**, declararea structurii **data** fiind necesar a fi făcută înainte de structura **persoana**:

```
struct data {unsigned zi,luna,an;};
struct persoana{char nume[30]; data datan;};
```

Pentru această descriere se poate declara variabila_înregistrare **p** care este de tipul **persoana**:

```
persoana  p;
```

Calculul lungimii ocupate de **p** este: 30 +3 x 2=36 octeți (câte un octet pentru cele 30 de caractere și câte doi octeți pentru tipul de date **unsigned**).

Conform observației de mai sus, este valabilă și următoarea descriere a structurii **persoana**:

```
struct persoana{char nume[30]; struct{unsigned zi,luna,an;} datan;};
```

În această situație se pierde definirea noului tip de dată pentru datele calendaristice, necesar poate și pentru alte informații, de exemplu, pentru data curentă.

Acest exemplu prezintă cazul în care un câmp al structurii este el însuși o structură de date-înregistrare la rândul său (data calendaristică).

3. Pentru exemplul care foloseşte entitatea **elev** aşa cum apare înregistrarea din catalog, descrierea în limbajul de programare se modifică în:

```
struct elev_c{char N[15],P[15]; unsigned M[16]; float MG;};
```

În acest exemplu unele elemente ale structurii sunt, la rândul lor, structuri omogene de date (şirurile de caractere ale numelui şi prenumelui, vectorul mediilor). Tipul de date **elev_c** ocupă 66 de octeţi.

4. Pentru entitatea **carte** descrierea tipului de date va ocupa 77 de octeţi şi va fi făcută astfel:

```
struct carte {long cod; char titlu[30], autor[20]; float pret; unsigned an_apar;
              char editura[15]; unsigned nr_ex;};
```

5. Pentru entitatea **vânzari** descrierea tipului de date va ocupa 18 octeţi şi va fi făcută astfel:

```
struct vanzare{long cod_prod, cant; float pret; data  data_v;};
```

6. Pentru entitatea **numar complex** descrierea tipului de date va ocupa 8 octeţi şi va fi făcută astfel;

```
struct complx{ float re,im;);
```

7. Pentru entitatea **punct** descrierea tipului de date va ocupa 8 octeţi şi va fi făcută astfel:

```
struct punct{ float x,y;);
```

8. Pentru entitatea **strazi** descrierea tipului de date va ocupa 25 de octeţi şi va fi făcută astfel:

```
struct strazi {char denumire[20];float lungime; char sens;};
```

unde sens va avea valorile 'U' sau 'D'.

9. Pentru entitatea **filme** descrierea tipului de date va ocupa 56 de octeţi şi va fi făcută astfel:

```
struct filme {char titlu[30] ; float ora _inceperii[6] ; unsigned rep;};
```

1. Utilizaţi modelele de mai sus pentru a defini un tip de date **elev_p** în care să fie structurate informaţiile personale ale unui elev, aşa cum sunt acestea trecute în catalog.
2. Utilizaţi modelele de mai sus pentru a defini un tip de date **autor** în care să fie structurate informaţiile referitoare la autori, necesare într-o bibliotecă.
3. Utilizaţi tipul de date **punct** pentru a defini tipul de date **segment**.
4. Pe baza modelului **vanzari** definiţi un tip de date **produs** pentru informaţiile referitoare la produsele dintr-un magazin.
5. Definiţi un tip de date **dreapta** pentru coeficienţii unei drepte, pornind de la expresia standard a unei drepte ($ax + by + c$).
6. Definiţi un tip de date **dreptunghi** folosind tipul de date **punct**.
7. Definiţi un tip de date **triunghi** pentru care se dau coordonatele din plan ale vârfurilor, folosind tipul de date **punct** şi tipul de date **segment**.

Referirea câmpurilor dintr-o înregistrare – operatorul • (punct)

Accesul la oricare dintre câmpurile structurii din cadrul variabilei declarate ca înregistrare se face utilizându-se operatorul • **(punct)**. Operatorul • se numeşte ***operator de referinţă a unui câmp*** din cadrul variabilei şi se scrie între variabila_înregistrare şi numele câmpului referit:

```
variabila_inregistrare•nume_camp
```

O dată referit astfel un cîmp, el intră în prelucrări la fel ca orice variabilă simplă.

Pentru exemplul tipului de dată `elev`, luat mai sus, se poate testa media elevului **e1** dacă este mai mare decât media elevului **e2** prin:

```
if(e1.medie>e2.medie) ...
```

Exemplificarea folosirii tipului înregistrare în programe

Exerciţiile se vor pune în evidenţă ca teme de laborator.

1. Se consideră **e1** şi **e2**, două variabile de tip înregistrare în care sunt citite datele a doi elevi conform machetei **elev** descrisă mai sus. Se doreşte afişarea numelor celor doi elevi în ordinea descrescătoare a mediilor lor.

Rezolvare. Macheta înregistrărilor **e1** şi **e2** este descrisă de structura **elev**. Programul de mai jos prezintă modul de utilizare a acesteia.

```
#include<iostream.h>
#include<conio.h>
void main()
{
struct elev {char nume[25];
             float medie;};
elev e1,e2;
clrscr();
cout<<"Dati numele primului elev ";
cin>>e1.nume;
cout<<"Dati media primuui elev ";
cin>>e1.medie;
cout<<"Dati numele elevului urmator ";
cin>>e2.nume;
cout<<"Dati media lui ";
cin>>e2.medie;
if(e1.medie>e2.medie)
      cout<<e1.nume<<" "<<e2.nume;
  else
      cout<<e2.nume<<" "<<e1.nume;
getch();
}
```

2. În programul următor sunt puse în evidenţă elementele noi, discutate mai sus, care sunt necesare definirii şi utilizării noului tip de structură. Programul rezolvă problema enunţată pentru persoanele care se înscriu la testare şi care trebuie să aibă vârsta de minimum 21 de ani.

Referinţa pentru *luna* din data naşterii unei persoane apare sub forma *persoana.datan.luna* , utilizându-se de două ori operatorul de referinţă, deoarece câmpul *luna* este dublu subordonat: direct câmpului *datan* (care este o structură) şi indirect structurii *persoana*.

```
#include<iostream.h>
#include<conio.h>
void main()
{int n, varsta,z,l,a;
struct data{unsigned zi,luna,an;};
struct persoana
            {char nume[25];
             data datan;};
persoana p;data azi;
//variabila p este de tipul persoana
cout<<" Dati numarul de persoane " ;
cin>>n ;for(int i=1;i<=n;i++)
{cout<<"Dati numele persoanei ";
 cin>>p.nume;
cout<<"Dati data nasterii ";
 cin>>p.datan.zi;
 cin>>p.datan.luna;
 cin>>p.datan.an;
 cout<<"Dati data curenta ";
 cin>>azi.zi>>azi.luna>>azi.an;

 varsta=azi.an-p.datan.an;
 if(azi.luna>p.datan.luna)varsta++;
  else if(azi.luna==p.datan.luna
                  && azi.zi>p.datan.zi)
          varsta++;
 if(varsta>=21)
  cout<<p.nume<<"varsta="<<varsta<<endl;
 }
getch();
}
```

Clasificare

Din punctul de vedere al numărului câmpurilor necesare descrierii unei entităţi definite de către programator, structura înregistrare are două forme:

– *forma fixă*, în care se utilizează acelaşi număr de câmpuri pentru descrierea oricărui exemplar din entitatea definită de utilizator (de ex. *persoana* din programul de mai sus);

– *forma cu variante*, în care definirea câmpurilor, ca număr şi tip, depinde de specificul exemplarelor entităţii definite de utilizatorul programului.

Caracteristicile tipului înregistrare

I. Înregistrarea este o structură **neomogenă**, putând compune date de tipuri diferite.
II. Componentele înregistrării se numesc **câmpuri.**
III. O înregistrare poate conţine drept **câmp o dată structurată** (tablou, şir de caractere, structură etc.).
IV. **Alocarea** zonei de memorie pentru o variabilă de tip **struct** se face în octeţi succesivi, conform lungimii fiecărui câmp.

V. **Declararea** unui tip de dată înregistrare se face în sintaxa următoare:

struct *denumire{tip*$_1$ *câmp*$_1$*; tip*$_2$ *câmp*$_2$*;… ; tip*$_n$ *câmp*$_n$*; }.*

VI. Dacă mai multe **câmpuri succesive sunt de același tip**, se poate practica scrierea tipului respectiv pentru toată lista acelor câmpuri.

VII. Datorită eterogenității, localizarea unui câmp nu se poate face printr-un indice. Din acest motiv apare necesar un operator nou şi anume **operatorul de referință_câmp** desemnat prin *caracterul punct* (•). Referinţa va respecta sintaxa

nume înregistrare . nume câmp

VIII. Cu componentele unei variabile de tip înregistrare se pot realiza toate **operațiile permise** tipului acelor componente.

IX. Un câmp poate avea **aceeași denumire ca o altă variabilă** din program sau ca un câmp din altă înregistrare fără a se crea confuzii, deoarece numele câmpului este legat şi precedat în scriere de numele înregistrării din care face parte.

3.3. Prelucrări de bază asupra variabilelor_ înregistrare

a) Atribuirea de valori

Câmpurile unei variabile_înregistrare pot primi valoare prin:

- atribuirea valorii unei ***expresii***, de acelaşi tip ca tipul câmpului;
- atribuirea ***globală***, prin atribuirea unei variabile_înregistrare altei variabile_înregistrare de acelaşi tip;
- iniţializare prin ***constante*** sau prin constantă simbolică;
- ***citirea***, la nivel elementar, a valorilor, câmp cu câmp, de la tastatură sau dintr-un fişier text al utilizatorului. În programele date ca exemple, valorile câmpurilor s-au citit din fişierul standard de intrare.

Exerciţiile se vor pune în execuţie ca teme de laborator.

1. Se consideră tipul de date **complx** şi trei variabile de acest tip: **z1**,**z2** şi **z3**. În **z3** se va reţine suma **z1+z2**. Secvenţa următoare de program atribuie valori câmpurilor variabilelor astfel: iniţializare prin constante, pentru **z1** şi **z2** şi expresii de calcul pentru a se obţine în **z3** suma dintre **z1** şi **z2**.

```
...
struct complx{ float re,im;};
complx z1={2,5},z2={1,-6},z3;
```

```
z3.re=z1.re+z2.re;
z3.im=z1.im+z2.im;
...
```

1. Din exemplul luat va rezulta `z3=3-i`.

2. Fie tipul de date **elev**. Se consideră elevii **e1** şi **e2** şi se doreşte să se interschimbe datele lor pentru a obţine în **e1** datele elevului cu media generală mai mare. Se va face o atribuire globală între variabilele **aux**, **e1** şi **e2**.

```
void main()
{struct elev{char nume[20];float
mg;};
elev e1,e2,aux;
cin>>e1.nume>>e1.mg;
cin>>e2.nume>>e2.mg;
if(e1.mg<e2.mg)
```

```
      {aux=e1;
      e1=e2;
      e2=aux;
      }
cout<<e1.nume<<" "<<e2.nume;
}
```

b) Afişarea

Conţinutul unei înregistrări ***se afişează la nivel elementar, câmp cu câmp***. Această modalitate a fost exemplificată în programele de mai sus.

test

Câte două puncte pentru exerciţiile 1, 2, 3, câte un punct pentru exerciţiile 4 şi 5, două puncte din oficiu.

1. Ştiind că variabila **p** este folosită pentru a memora şi utiliza în calcule coordonatele reale ale unui punct în plan, stabiliţi care dintre următoarele declarări corespunde scopului propus:
a) `struct p{float x,y;};` **b)** `struct {float x;float y;}p;`
c) `p struct{float x;float y};` **d)** `struct {float x,y;}p;`

2. Variabila **n** este utilizată pentru a memora numele şi prenumele unui elev. Precizaţi care dintre declarările următoare nu este corectă:
a) `struct {char nume[15],prenume[15];}n;` **b)** `char n[50];`
c) `char n[15[15];` **d)** `char n[2][15];`

3. Folosind tipul de structură **complx**, definit mai sus, precizaţi ce se realizează în secvenţa de mai jos:
```
struct polar {float mod, arg;);
polar w; cmplx z;
w.mod=sqrt(z.re*z.re+z.im*z.im);
w.arg=atan2(z.im,z.re);
cout<<w.mod<<" "<<w.arg;
```
a) se calculează modulul lui `z`; **b)** se converteşte `z` în coordonate polare; **c)** se converteşte `w` din coordonare polare în exprimare de număr complex.

4. Fiind definite structurile de date **punct** şi **cerc** şi variabilele **p** şi **c**, să se precizeze ce afişează secvenţa de mai jos, în care funcţia **pow(a,b)**, din biblioteca **math**, realizează calculul a^b:

```
struct punct {float x, float y;}p;
struct cerc {punct centru; float raza;};
cerc c;
float dist;
dist=sqrt(pow(p.x-c.centru.x,2)+pow(p.y-
c.centru.y,2));
cout<<dist-pow(c.raza,2);
```

a) distanţa dintre un punct **p** şi centrul cercului;
b) distanţa dintre două puncte;
c) puterea punctului **p** faţă de cercul **c**;
d) aria cercului **c**.

5. Fie definirea unui punct material în spaţiu şi trei astfel de puncte: **p, q** şi **w.** Să se determine ce se afişează prin secvenţa de mai jos:

```
struct punct_m{float x,y,z,masa;} p,q,w;
w.masa=p.masa+q.masa;
w.x=(p.x*p.masa+q.x*q.masa)/w.masa;
w.y=(p.y*p.masa+q.y*q.masa)/w.masa;
w.x=(p.z*p.masa+q.z*q.masa)/w.masa;
cout<<w.x<<" "<<w.y<<" "<<w.z;
cout<<w.masa;
```

a) ponderea distanţei între puncte;
b) masa punctului `w`;
c) centrul de greutate al celor două puncte.

c) Tablouri de înregistrări

Foarte frecvent este nevoie să se reţină în memoria internă, pe parcursul programului, toate valorile pe care le ia o variabilă_înregistrare. Acest lucru apare când aceste valori participă în prelucrări în care unele depind de altele.

exemplu

1. Fie situaţia şcolară a elevilor unei clase pentru care se doreşte obţinerea unei liste în care aceştia să fie ordonaţi descrescător, după media generală. Lista va fi folosită în scopul premierii elevilor.
Rezolvare. După cum se ştie, premierea este condiţionată şi de media 10 la purtare, nu numai de media generală (**MG**) a elevului. Pentru acest scop este necesară întâi organizarea unei machete care să descrie un **elev** (fig. 3.6).

Este nevoie ca datele din catalog să fie introduse în memoria internă, unde se va forma un tabel al elevilor clasei. Apoi, acest tabel va fi ordonat descrescător după datele din coloana **MG**. Rezultă un grup de înregistrări de acelaşi tip, **elev**, grup care va alcătui clasa din care fac parte elevii. Fiind un grup omogen, se poate proiecta un vector **clasa**:

Nume	Prenume	Purtare	MG
15 ch	20 ch	**unsigned**	**float**

Figura 3.6

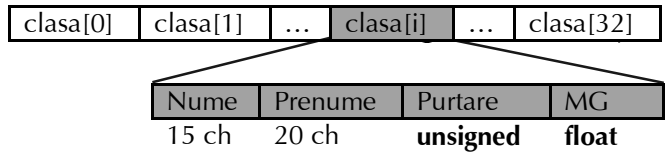


Figura 3.7

Declararea structurii **elev** şi apoi definirea variabilei **clasa** se scriu astfel:

```
struct elev { char nume[15],prenume[20]; unsigned purtare;float MG;};
elev clasa[32];
```

Referirea la fiecare elev din **clasa** pentru citirea numelui, prenumelui, a mediei la purtare şi a mediei sale generale se face sub forma indexării cunoscute de la tabloul unidimensional: **clasa[i].**

În procesul ordonării, valoarea câmpului **MG** al elevului de ordin **i** se va compara cu valoarea din câmpul **MG** al elevului de ordin **i+1** astfel:

```
if(clasa[i].MG<clasa[i+1].MG)...//interschimbul inregistrarilor clasa[i]
                               //cu clasa[i+1]
```

Pentru realizarea programului, datele de intrare se citesc din fişietul text **clasa.in**, iar rezultatul ordonării se transferă în fişierul text **clasa.out**.

```
//program situatie scolara;
#include<fstream.h>
void main()
{
struct elev{char nume[15],prenume[20];
      unsigned purtare; float MG;};
elev clasa[32], aux;
unsigned n,I,ok;
ifstream in("clasa.in");
ofstream out("clasa.out");
in>>n;
for(i=0;i<n;i++)
 {in>>clasa[i].nume;
 in>>clasa[i].prenume>>clasa[i].purtare;
 in>>clasa[i].MG;
  }
//ordonarea descrescatoare
do
 {ok=1;
  for(i=0;i<n-1;i++)
       if(clasa[i].MG<clasa[i+1].MG)
            {aux=clasa[i];
             clasa[i]=clasa[i+1];
             clasa[i+1]=aux;
             ok=0;
            }
 }while(!ok);
//transferare in fisierul out
for(i=0;i<n;i++)
  {out<<clasa[i].nume<<" ";
      out<<clasa[i].prenume<<" ";
      out<<clasa[i].purtare;
      out<<clasa[i].MG<<endl;
  }
}
```

1. Să se adauge în program secvenţa necesară afişării elevilor care vor fi premiaţi (media generală ≥ 8.50 şi media 10 la purtare).

2. Să se reproiecteze programul de mai sus, astfel încât pentru fiecare elev să fie introduse din fişier datele primare din catalog: *numele, prenumele, media anuală pentru fiecare obiect, media la purtare,* iar programul să calculeze media generală a fiecărui elev, să ordoneze elevii descrescător după această medie, să treacă datele complete în fişierul de ieşire şi să afişeze lista elevilor selectaţi pentru premiere.

2. Se cere realizarea unui program care să poată afişa valoarea unui polinom, într-o nedeterminată reală, cu coeficienţi reali, pentru un număr real citit de la tastatură şi să se specifice dacă numărul citit este rădăcină a ecuaţiei polinomiale corespunzătoare.

Rezolvare. Polinomul $P(X)=a_nX^n+a_{n-1}X^{n-1}+\ldots+a_1X+a_0$ este furnizat programului prin gradul şi coeficientul fiecărui monom. Deoarece un polinom dat programului de către utilizator poate să nu conţină toate monoamele, atunci este convenabilă numai memorarea datelor pentru monoamele prezente. Pentru aceasta se va defini tipul

grad	coeficient
unsigned	float

Figura 3.8

de date înregistrare **monom** care are macheta din figura 3.8. Pentru a memora întregul polinom se va declara o variabilă de tip vector de monoame, **poli**.

```
struct monom{unsigned grad; float coeficient;};
monom poli[21];
```

Calculul numărului de elemente de tip **monom** din cadrul vectorului **poli** trebuie să ţină cont de faptul că, pentru un polinom de gradul **n** sunt prezente maximum **n+1** monoame. Datele se vor citi din fişierul **poli.in**, pe prima linie fiind numărul de monoame, **m**, iar pe următoarele **m** linii, câte o pereche de numere pentru *gradul* şi *coeficientul* monomului curent. Datele din fişier trebuie să fie ordonate descrescător în funcţie de gradul monomului. De exemplu, pentru conţinutul fişierului `poli.in` dat în figura 3.9, polinomul este $P(X) = X^2 - 2X + 1$.

```
3
2   1
1   -2
0   1
```

Figura 3.9

```
//program evaluare polinom
#include<fstream.h>
#include<math.h>
void main()
{struct monom{unsigned grad;
              float coeficient;};
monom poli[21];
unsigned m,i;
//m=numarul de monoame existente
float x0,P=0;//x0=punctul de evaluare
ifstream in("poli.in");
in>>m;
for(i=0;i<m;i++)
 {in>>poli[i].grad;
  in>>poli[i].coeficient;
  }
cout<<"Dati punctul de evaluare ";
cin>>x0;
//evaluare
for(i=0;i<m;i++)
      P=P+pow(x0,poli[i].grad)*
                  poli[i].coeficient;
cout<<"Valoarea="<<P;
}
```

Adăugaţi în programul de mai sus secvenţa prin care se determină dacă valoarea **x0** este sau nu rădăcina ecuaţiei polinomiale.

3. Se cere realizarea unui program care să poată afişa valoarea sumei a două polinoame, $P(X)$ şi $Q(X)$, fiecare polinom fiind într-o nedeterminată reală, cu coeficienţi reali,

Rezolvare. Se vor folosi declarările şi definirile de variabile ca în programul anterior. De asemenea, se va folosi fişierul **poli.in** în care, după liniile cu datele primului polinom vor urma liniile cu datele celui de-al doilea. Spre exemplu, pentru a aduna $P(X)=X^2 - 2X + 1$ cu $Q(X) = 2X^3 + 4X$, fişierul **poli.in** are conţinutul tabelului din figura 3.10. Fişierul trebuie să conţină datele fiecărui polinom ordonate descrescător după gradul monomului. Adunarea în sine constă în realizarea polinomului $S(X)$, prin interclasarea vectorilor monoamelor celor două polinoame date, în caz de grade egale însumându-se coeficienţii.

```
3
2   1
1   -2
0   1
2
3   2
1   4
```

Figura 3.10

Pentru exemplul luat, $S(X) = 2X^3 + X^2 + (4 - 2)X^1 + 2X^0$.

```
//program suma 2 polinoame
#include<fstream.h>
#include<math.h>
void main()
{struct monom{unsigned grad;
              float coeficient;};
monom P[21],Q[21],S[21];
int m,n,i,j,k;
//m=numarul de monoame existente in P
//n=numarul de monoame existente in Q
ifstream in("poli.in");
in>>m;
for(i=0;i<m;i++)
 {in>>P[i].grad;
  in>>P[i].coeficient;
  }
in>>n;
for(i=0;i<n;i++)
 {in>>Q[i].grad;
  in>>Q[i].coeficient;
  }
i=0;j=0;k=-1;
while(i<m && j<n)
 if(P[i].grad>Q[j].grad)
      S[++k]=P[i++];
  else
   if(P[i].grad<Q[j].grad)
            S[++k]=Q[j++];
```

```
   else
   {S[++k].coeficient=
    P[i++].coeficient+Q[j++].coeficient;
    S[k].grad=P[i-1].grad;
   }
if(i==m)
      while(j<n) S[++k]=Q[j++];
 else
      while(i<m)S[++k]=P[i++];
//afisarea suma
cout<<"\nPolinomul suma="<<endl;
for(i=0;i<=k;i++)
      {cout<<S[i].coeficient<<"X^";
      cout<<S[i].grad<<"+";}
cout<<"\b ";//sterge ultimul +
}
```

4. Dându-se un număr de *n* zile ale unei perioade, pentru fiecare zi înregistrându-se *data* şi *temperatura*, să se afişeze ziua în care s-a înregistrat temperatura maximă. Datele se citesc dintr-un fişier text **grade_zi.txt** În fişier, pe prima linie este memorat numărul de zile, *nz*, iar pe următoarele *nz* linii sunt înregistrate data şi temperatura zilei respective. De exemplu, pentru conţinutul fişierului dat în figura 3.11, se citesc datele a 5 zile, afişându-se valoarea 19.5.

```
5
12  4  2006  8
15  4  2006  12.5
16  4  2006  10
20  4  2006  19.5
22  4  2006  18
```

Figura 3.11

Rezolvare. S-a definit un tip generic de înregistrare, **data**, pentru data calendaristică. Datele fiecărei zile înregistrate sunt citite pe rând în variabila_înregistrare **t**, pentru care s-a definit structura de date **termica_zilei**. Variabila **data_max** este o dată structurată de tip **data** în care se reţine data în care s-a înregistrat temperatura maximă. Pentru început, în această variabilă se transferă data primei zile din fişier. Variabila **t_max** reţine temperatura maximă care se va determina. Variabila se iniţializează cu temperatura primei zile din fişier.

```
#include<fstream.h>
#include<conio.h>
void main()
{
clrscr();
typedef struct data
            {unsigned zi,luna,an;};
struct termica_zilei
            {data zi_lu;
             float temp;};
float t_max;
data data_max;
termica_zilei t;
unsigned este,nz,i,an;
t_max=0;
ifstream T("grade_zi.txt");
T>>nz;
T>>data_max.zi>>data_max.luna;
T>>data_max.an;
T>>t_max;
for(i=2;i<=nz;i++)
  {
     T>>t.zi_lu.zi>>t.zi_lu.luna;
     T>>t.zi_lu.an;
     T>>t.temp;
     if(t_max<t.temp)
        {t_max=t.temp;
         data_max=t.zi_lu;
        }
  }
cout<<"Data cu temperatura maxima: ";
cout<<"Zi="<<data_max.zi;
cout<<" Luna="<<data_max.luna;
cout.width(5);cout.precision(2);
cout<<"\nTemperatura maxima=";
cout<<t_max<<" grade";
getch();
}
```

1. Transformaţi programul de mai sus, astfel încât el să poată afişa *toate zilele* din fişier în care s-a înregistrat *temperatura maximă*.
2. Transformaţi programul de mai sus pentru a afişa *temperatura maximă* înregistrată *în fiecare lună* din care fac parte zilele din fişier, dacă perioada urmărită cuprinde zile din luni diferite.

5. Se doreşte crearea unui program prin care elevii dintr-o clasă mai mică să fie verificaţi la chimie din capitolul *Hidrocarburi*. Datele testului sunt înregistrate în fişierul **HC_test.txt,** în modul următor: pe prima linie se găseşte numărul de întrebări, *ni*, iar pe următoarele *ni* linii sunt perechi de numere naturale în care primul reprezintă numărul de atomi de carbon şi al doilea numărul de atomi de hidrogen. Pe ecran vor apărea cele două numere, iar elevul va trebui să tasteze numele hidrocarburii care are formula afişată. Punctajul se obţine prin

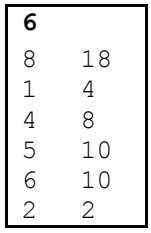


Figura 3.12

contorizarea răspunsurilor corecte. Pentru cele 6 hidrocarburi din fişierul dat ca exemplu în figura 3.12, răspunsurile sunt: *octan, metan, butenă, pentenă, hexină, etină*.

Rezolvare. În programarea acestui test a fost nevoie să se utilizeze *două tipuri de structuri de date*: şiruri de caractere şi înregistrări. Şirurile de caractere au fost folosite pentru a crea constantele de tip rădacina şi sufixul denumirii hidrocarburii, **rad** şi **sufixe** şi variabila **nume** în care se compune denumirea substanţei prin concatenarea radăcinii cu sufixul corespunzător. De asemenea, în variabila **rasp** se citeşte textul introdus de elev ca răspuns pentru a se compara apoi cu **nume**. Pentru formula hidrocarburii este proiectat tipul de date **formula** ca structură cu două câmpuri: **carbon** şi **hidrogen**. Variabila **HC** de tip **formula** va reţine, pe rând, câte o pereche de numere naturale corespunzătoare unei formule atomice citite din fişier. Pentru o anume formulă atomică din **HC** se copiază în **nume** rădăcina denumirii corespunzătoare numărului de atomi de **carbon -1** (deoarece indicii în vectorul **rad** încep cu valoarea 0). Apoi se stabileşte grupa de hidrocarburi după relaţia cu numărul de atomi de hidrogen: C_nH_{2n+2} pentru alcani, C_nH_{2n} pentru alchene şi C_nH_{2n-2} pentru alchine. În funcţie de grupa stabilită se adaugă, prin concatenare, sufixul corespunzător în variabila **nume**. După citirea răspunsului în variabila **rasp**, se transformă toate caracterele în litere mici pentru a evita situaţiile în care elevul tastează şi majuscule.

```
#include<fstream.h>
#include<conio.h>
#include<string.h>
void main()
{ clrscr();

const char rad[10][6]={"met","et",
        "prop","but","pent","hex",
        "hept","oct","non","dec"};
const char
     sufixe[3][4]={"an","ena","ina"};
struct formula
 {unsigned carbon,hidrogen;};
unsigned p=0,ni,i;
ifstream T("HC_test.txt");
char nume[10],rasp[10];
formula HC;
T>>ni;//numarul de intrebari
for(i=1;i<=ni;i++)
 {
  T>>HC.carbon>>HC.hidrogen;
  strcpy(nume,rad[HC.carbon-1]);
  if(HC.hidrogen==2*HC.carbon+2)
            strcat(nume,sufixe[0]);
      else
  if(HC.hidrogen==2*HC.carbon)
            strcat(nume,sufixe[1]);
            else
              strcat(nume,sufixe[2]);
  cout<<"Carbon="<<HC.carbon;
  cout<<"  Hidrogen="<<HC.hidrogen;
  cout<<"\nNumele hidrocarburii=";
  cin>>rasp;cout<<endl;
  //transf.raspunsul in litere mici
  strlwr(rasp);
  //comparare raspuns si contorizare
  if(strcmp(nume,rasp)==0)p++;
 }
cout<<"Ati raspuns corect la ";
cout<<p<<" intrebari";
cout<<"din cele "<<ni;
getch();
}
```

1. Transformaţi programul de mai sus pentru a afişa, la sfârşitul evaluării, rezolvarea testului. Pentru aceasta programul va prezenta pe ecran, grupate pe tipuri, hidrocarburile la care se referă testul: formula şi denumirea.
2. Construiţi un program asemănător pentru testarea reciprocă: se citesc din fişier şi se afişează pe ecran 9 denumiri de hidrocarburi; la sfârşitul afişării, elevul va trebui să introducă numărul de atomi de carbon şi numărul de atomi de hidrogen corespunzători fiecărei substanţe.

PROBLEME PROPUSE

1) Să se realizeze programul pentru prelucrările de tip situaţie statistică la sfârşitul semestrului necesare pentru o clasă de maximum 32 de elevi, fiecare elev fiind descris prin structura dată în exemplul 1 din paragraful 3.1.

2) Un fişier **personal.txt** conţine date pentru un număr de maximum 800 de subiecţi. Descrierea fiecărei persoane conţine date despre: *nume*, *data naşterii* (în format numeric) şi *profesie* dintre variantele {constructor, inginer, profesor, mecanic, pilot}. Se doreşte crearea unui fişier **vârste.out** în care se trec numele persoanelor înregistrate grupate pe categorii de vârste astfel: între 18 şi 25 de ani, între 26 şi 40 de ani, între 41 şi 57 de ani, peste 58 de ani. Apoi, se va crea fişierul **profesii.out** în care se vor trece numele persoanelor şi profesiile, grupate pe profesii.

3) Asupra unui lot de maximum 100 de subiecţi se aplică o serie de teste (maximum 20) care au ca variante de răspuns 'DA', 'NU' şi 'INDIFERENT'. Fiecare subiect este înregistrat cu numele şi numărul de răspunsuri date la fiecare variantă pentru toate testele. De exemplu, Popescu – 5 de DA, 6 de NU şi 2 de INDIFERENT. Să se afişeze în ordine descrescătoare statistica răspunsurilor (numărul de DA, de NU şi de INDIFERENT) pe tot lotul de subiecţi şi persoana care deţine numărul maxim de DA.

4) Un set de *n* cuburi, $n \leq 200$, descrise prin *latură* şi *culoare*, se află înregistrate în fişierul **cuburi.in**. Se doreşte trecerea lor în fişierul **cuburi.out** în ordinea în care, dacă ar fi aşezate unul peste altul, s-ar crea un turn stabil. Pe ecran se va afişa numărul de cuburi vecine în turn care au aceeaşi culoare.

5) Pentru un campionat internaţional de tenis s-a creat un fişier **tenis.in** cu următoarele date despre concurenţi: *ţara*, *numele*, *vârsta*. Să se afişeze o listă a ţărilor participante în ordine alfabetică şi numele şi ţara celui mai tânăr concurent.

6) Se consideră două dreptunghiuri în plan, paralele cu axele de coordonate. Datele despre dreptunghiuri se referă la coordonatele colţurilor stînga-sus şi dreapta-jos. Să se afişeze un mesaj corespunzător privind relaţia dintre ele.

7) Să se construiască un program care testează un elev la limba modernă I astfel: dintr-un fişier **cuvinte.txt** se citesc cuvinte ale limbii respective împreună cu semnificaţia lor în limba română. Pe ecran se afişează numai cuvîntul în limba străină, aşteptându-se ca elevul să tasteze traducerea. La sfârşit se comunică punctajul şi rezolvarea testului.

Capitolul 4 Utilizări ale tehnicii structurării datelor – liste

În acest capitol veţi învăţa despre:

- Utilizarea structurilor de date pentru organizarea listelor de informaţii
- Organizarea elementelor şi prelucrărilor listelor alocate static
- Proiectarea listelor cu caracter particular: stive şi cozi

Reamintim:

Structurile de date sunt colecţii de date pentru care s-au **precizat:**

- **tipul** elementelor;
- proprietăţile de **organizare** ale elementelor, relaţiile între elemente;
- regulile de **acces** la elemente;
- **operaţiile** specifice.

Astfel, în declaraţiile de mai jos:

```
typedef   int tab[5];
typedef   tab tab2[26];
typedef struct mon{unsigned grad;float coef;};
int a; tab b; char c;
tab2 d; tab g[2];mon poli[21];
```

variabilele *a* şi *c* sunt date elementare, iar *b, d, g* şi *poli* sunt date structurate descrise prin posibilităţile limbajului de programare **C/C++**.

Clasificarea structurilor de date

1. Suportul ocupat determină: structuri *interne* (alocate în memoria internă) şi structuri pe suport *extern*.

2. Regula de organizare a elementelor determină: structură *liniară, ierarhizată* şi *reţea*.

În general, o structură este *liniară, ierarhizată* sau *reţea,* după relaţiile care sunt stabilite între elementele structurii, relaţii din care rezultă şi modul de acces la acele elemente. Apar astfel unele *informaţii secundare necesare accesului* şi *prelucrări* specifice la care sunt supuse acestea pentru a localiza un element din structură.

3. Alocarea de spaţiu în memorie determină: *structuri fixe, alocate static* şi *structuri alocate dinamic*.

Structura fixă de date este alocată static, deoarece repartizarea zonei de memorie necesară se face **înaintea execuţiei programului**, în *segmentul de date sau de stivă.* La declararea structurii se defineşte o zonă compactă de octeţi, pe care aceasta o va ocupa în memorie şi care nu se poate modifica pe parcursul execuţiei programului.

Structura alocată dinamic ocupă o zonă specială din memoria internă (zona *heap* – de adrese libere, vezi figura 1.1. din Capitolul 1) şi se determină **în timpul execuţiei programului**, prin instrucţiuni ale programului.

exemplu

Tabloul **t**, a cărui declarare este

```
int t [3][2];
```

va avea alocarea în memorie de tip static, fixată ca în figura 4.1,

t[0][0]	t[0][1]	t[1][0]	t[1][1]	t[2][0]	t[2][1]

Figura 4.1

chiar dacă va fi folosit, la un moment dat al rulării programului, spre exemplu, numai pentru primele 4 elemente.

Nu acelaşi lucru, adică alocarea unui spaţiu maxim care poate rămâne parţial nefolosit, se întâmplă cu datele repartizate pe suport extern, în afara memoriei de lucru, adică în structura de tip fişier.

4.1. Listele – structuri liniare de date

Una dintre cele mai complexe *metode de compunere* a datelor într-o structură **unitară** este **lista**.

exemplu

1. Organizarea spaţiului pe discul magnetic, practicată de către sistemul de operare constă în alocarea de zone-disc pentru înregistrările fişierelor sistemului sau ale utilizatorului, *astfel încât*, oricât de mari ar fi fişierele, viteza de transfer să fie minimă (ştiind că transferul extern consumă timp cu operaţii mecanice, electronice, de conversii de date).

Soluţia adoptată de către sistemul de operare constă în alocarea câte unei zone-disc fiecărei înregistrări nou-venite în locul liber găsit imediat după ultima înregistrare scrisă pe disc. Acest lucru se face indiferent de fişierul căruia îi aparţine ultima înregistrare scrisă. Cum acest loc liber nu urmează întotdeauna ultimei înregistrări din fişierul căruia îi aparţine noua înregistrare, sistemul asigură o legătură pentru succesiunea logică între înregistrări prin adrese-disc[1]. Astfel, ultimei înregistrări scrise dintr-un fişier îi ataşează adresa-disc a locului unde va fi scrisă înregistrarea lui următoare ş.a.m.d. pentru întreg fişierul.

Fie trei fişiere, **`F1`**, **`F2`** şi **`F3`**, ale căror înregistrări se scriu pe disc **în momente diferite** şi notaţia adr_n pentru adresa-disc alocată în caseta[2] a **`n`**-a; atunci acest lucru se poate reprezenta ca în figura 4.2.

F1 → adr2	F1 → adr10	F2 → adr4	F2 → adr5	F2 → adr9	F3 → adr7	F3 → adr8
adr1	adr2	adr3	adr4	adr5	adr6	adr7
F3 **eof**	F2 → adr12	F1 → adr11	F1 → adr13	F2 **eof**	F1 **eof**	
adr8	adr9	adr10	adr11	adr12	adr13	adr14

Figura 4.2

Se observă că primele două înregisrări ale lui **`F1`** au fost efectuate în cursul aceleiaşi prelucrări, în zone-disc succesive. Apoi **`F1`** este pus în aşteptare, pentru că pe suport se scriu primele trei înregistrări din fişierul **`F2`**, în zone succesive. Mai târziu, se trece pe disc **`F3`**, care este compus din trei înregistrări. În continuare, apare o nouă înregistrare de adăugat la **`F2`**, care nu a avut loc liber după ultima lui înregistrare de la adresa *adr5*. În acest moment, la *adr5* se notează că se va continua la adresa *adr9* pentru fişierul **`F2`**. Când prelucrarea cere memorarea unei noi înregistrări în **`F1`**, găseşte adresa *adr3* ocupată, drept pentru care în ultima înregistrare de pe disc a fişierului **`F1`** (adică în zona de adresă adr2) se înscrie *adr10* ca adresă de continuare etc.

Acest mod de alocare se numeşte **alocarea secvenţial-înlănţuită**, localizarea fizică pe suport a unei înregistrări fiind făcută în funcţie de adresa-disc ataşată înregistrării anterioare acesteia din punct de vedere logic.[3]

În concluzie, exemplul de mai sus oferă imaginea organizării a trei liste distincte de înregistrări, **`F1`**, **`F2`** şi **`F3`**, pentru care succesiunea logică a elementelor listelor este diferită de succesiunea lor fizică.

În realitatea înconjurătoare, majoritatea acţiunilor conţin prelucrări de informaţii în a căror organizare ordinea logică nu coincide cu ordinea lor fizică, drept pentru care trebuie să existe definite repere de identificare a succesiunii logice.

2. La biblioteca şcolii cărţile sunt aşezate în rafturi, grupate pe domenii, iar în cadrul domeniilor, sunt grupate pe autori. Astfel că, fiecare carte, pe lângă codul care îi este ataşat, are şi un reper de tip număr_raft. Dacă bibliotecarul se referă la domeniul *Beletristică* din care alege numai grupa *Scriitori români*, el va putea selecta o grupă compactă de rafturi în care sunt aşezate cărţile pe care le are în inventarul bibliotecii din această categorie. Dacă, însă, doreşte să alcătuiască o colecţie numai din cărţile din *Dramaturgia română*, atunci va organiza o listă în care se vor regăsi numai anumite cărţi din lista iniţială, cărţile fiind de această dată aşezate în rafturi diferite, în funcţie de autor, nu una după alta.

3. La aceeaşi bibliotecă a şcolii, în situaţia în care este „casată" o carte (se scoate din inventar, deoarece este uzată sau depăşită), bibliotecarul o ia din raft şi reface raftul sau adaugă alta în loc.

---

[1] *Adresa-disc* reprezintă construcţia alcătuită din: nr. cilindru, nr. pistă şi nr. sector.

[2] Caseta reprezintă un număr de cluster-e pe care le alocă sistemul de operare pentru fiecare înregistrare.

[3] Acest mod de lucru poate conduce, după ştergeri repetate de fişiere, la o suprafaţă fragmentată a discului, astfel încât trecerea din înregistrare în înregistrare pentru fişierele rămase devine consumatoare de timp pentru localizările respective. Această deficienţă se rezolvă prin defragmentarea periodică a spaţiului de pe disc, operaţie în care sistemul de operare realizează aranjarea contiguă a fişierelor.

4. La o firmă de turism există o listă alfabetică a localităţilor care pot fi alese pentru a alcătui anumite voiaje. Un turist doreşte lista voiajelor asigurate de către acea firmă. Un anume voiaj conţine o listă a unora dintre localităţi, alese într-o ordine care diferă de aşezarea lor în lista alfabetică.

5. În pauză, la bufetul şcolii se formează o coadă de elevi care aşteaptă să cumpere ceva de mâncare. Elevii sunt serviţi în ordinea în care au sosit şi s-au ataşat cozii, fiecare elev sosit aşezându-se după ultimul.

În fiecare din exemplele de mai sus apar liste de informaţii pentru care se regăsesc aceleaşi caracteristici generale.

Lista este o structură de date *liniară,* care are două extremităţi*, început şi sfârşit,* în care fiecărui element i se asociază informaţia privind *locul elementului următor* lui din punct de vedere logic, accesul fiind posibil prin ambele capete.

Clasificarea structurilor de date

I. Înregistrările pe disc, cărţile din bibliotecă, elevii de la coada la bufet sau oraşele vizitate prin firma de turism alcătuiesc **colecţii de informaţii** necesare în prelucrările descrise.
II. Colecţiile sunt omogene, informaţiile definesc **elemente de acelaşi tip.**
III. Între elementele colecţiei există o **ordine logică de succesiune**.
IV. În fiecare colecţie, elementele **se succed liniar**, conform unei liste de informaţii.
V. La **crearea** listei, ordinea logică coincide cu ordinea fizică de aşezare a elementelor.
VI. Fiecare element din listă are un element **precedent** şi unul **următor**, mai puţin elementele capete (primul şi ultimul).
VII. Asupra listelor se fac **operaţii de parcurgere şi actualizare** (modificare de informaţii, ştergere şi adăugare de elemente), respectându-se ordinea logică a elementelor.
VIII. Operaţiile în care intră elementele colecţiilor pot modifica relaţia dintre **ordinea fizică** şi **ordinea logică a elementelor**.

Din cele învăţate până acum s-a văzut că structura de *tip* **tablou unidimensional** *este o structură liniară.* Ea poate fi considerată drept **cazul elementar de listă**, deoarece:
– elementele sunt de *acelaşi tip*;
– între elemente există *ordinea logică primară,* adică ordinea fizică a locurilor ocupate de ele;
– la fiecare element se poate avea *acces* direct, printr-un indice (sau mai mulţi, în funcţie de ierarhia structurii), dar, în realitate, sistemul de operare calculează locul elementului precizat de indice printr-o operaţie secvenţială specifică unei progresii aritmetice. Se cunoaşte adresa primului element, care este şi numele vectorului, se cunoaşte raţia – lungimea în octeţi a tipului de date declarat pentru elemente – şi se cunoaşte rangul (indicele) elementului unde se cere localizarea. Astfel se face calculul de adresă: `nume_vector+rang*ratie`.
– cu ajutorul indicelui se poate identifica pentru fiecare element, *precedentul şi succesorul* său, ori dacă este primul sau ultimul, fără a mai fi nevoie de o informaţie suplimentară pentru localizarea lui;
– operaţia de parcurgere a elementelor se poate realiza foarte uşor;
– operaţiile impuse de *actualizarea valorilor* din vector vor consuma însă **timp de prelucrare**, deoarece vor necesita deplasări ale elementelor pentru a insera un element nou sau pentru a şterge un anumit element;
– operaţiile de *inserare* sau de *ştergere* vor *modifica lungimea tabloului*, astfel că este nevoie să se estimeze, încă de la declararea lui, spaţiul maxim necesar;
– există *situaţia limită* de **listă plină**, când spaţiul alocat este complet folosit.

4.2. Dispunerea elementelor listei

În general, dispunerea elementelor în listă se prezintă în funcţie de ordinea logică stabilită între ele. În figura 4.3 este schematizată forma generală de dispunere.

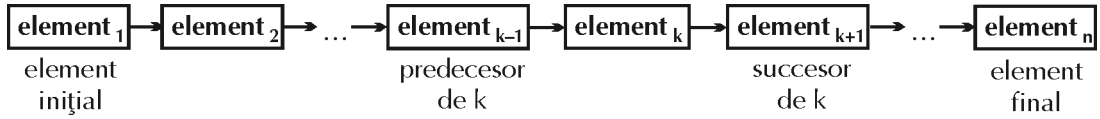


Figura 4.3

Ordinea fizică a elementelor **nu** se identifică *întotdeauna* cu ordinea lor logică.

Dispunerea elementelor în listă este foarte importantă din punct de vedere *al spaţiului de memorie utilizat* şi al *vitezei de prelucrare a elementelor*.

Metode de dispunere

a) Secvențială, în zonă compactă de octeţi (zonă contiguă), posibilă numai în alocarea statică. Elementele listei sunt stocate în ordinea în care au venit valorile, în zone de memorie succesive de aceeaşi lungime în octeţi. Modelul general al acestei dispuneri este cel din figura 4.3.

Structura de tip tablou unidimensional este utilizată pentru a stoca şi a prelucra, în mod secvenţial, liste foarte răspândite în practică, dar pentru care limbajele de programare nu dispun de tipuri şi de prelucrări predefinite destinate acestora.

b) Înlănțuită, posibilă atât în alocare statică, cât şi în alocare dinamică. Elementele listei nu mai sunt obligatoriu stocate în zone de memorie succesive.

În lista simplu-înlănţuită un element (o componentă) se declară ca o dată structurată de tip articol (înregistrare), formată din două câmpuri: *informaţia propriu-zisă* şi *informaţia de legătură*. Informaţia propriu-zisă poate fi, la rândul ei, structurată sau nu.

Alocarea înlănţuită a componentelor structurii impune un **mecanism** prin care o **nouă componentă** apărută este **legată în succesiune logică de corpul structurii** format până atunci.

Modul de înlănţuire prin adrese a componentelor se poate face:
– într-un singur sens, generându-se *lista simplu-înlănţuită*;
– în ambele sensuri, generându-se *lista dublu-înlănţuită*.

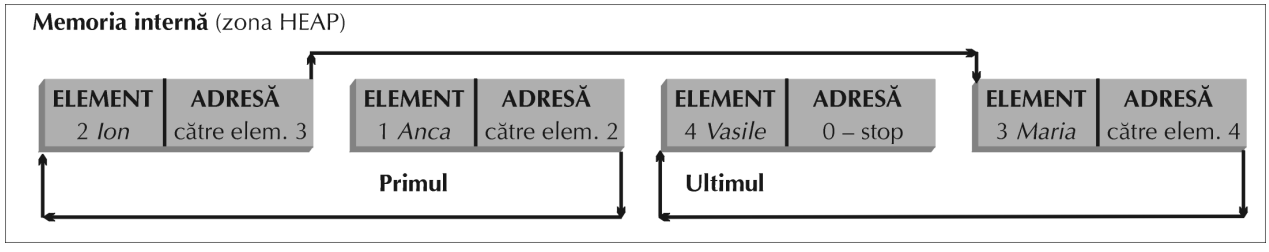
Rezultă că fiecare componentă, pe lângă informaţia propriu-zisă pe care o deţine, conţine şi o informaţie legată de componenta care îi succede logic.

Informaţia de legătură va fi **adresa componentei** spre care se realizează succesiunea logică, iar mecanismul se numeşte **alocare înlănţuită după adrese**.

- Pentru a*locarea înlănţuită dinamică*, fiecare element ocupă o zonă în memoria de adrese libere (**HEAP**) şi este identificat printr-o **variabilă de tip adresă**, pe care predecesorul său o are înregistrată în structura lui (precum şi a succesorului său, în cazul listei dublu-înlănţuite).

În figura 4.4 este organizată o listă simplu-înlănţuită formată din patru elemente, nume de persoane, pentru care alocarea memoriei s-a făcut în zone dispersate, după cum a fost loc liber în momentul în care a fost înregistrat fiecare element.

Figura 4.4



- *Alocarea înlănţuită statică* va folosi tot structura de date tablou unidimensional, dar fiecare element va fi însoţit şi de informaţia de indice a elementului care îi succede logic.

Presupunem aceleaşi patru elemente din exemplul anterior, în care am considerat că informaţia din fiecare element este numele unei persoane. Soluţia este organizarea unui vector pentru memorarea listei lor care trebuie să aibă drept element (componentă) o structură de tipul:

```
struct persoana {char nume[15]; int indice_urmator;};
persoana  grup[4];
```

Configuraţia spaţiului alocat celor patru elemente de tip **persoana**, după ce au fost citite numele, va fi cea din figura 4.5. Se observă că, la crearea listei, ordinea logică a elementelor coincide cu ordinea lor fizică. Deoarece persoana *Maria* este ultima intrată în listă, ea va avea ca indice de continuare valoarea –1, ceea ce înseamnă că nu îi urmează nici o altă persoană (în **C/C++** nu există indicele –1, deoarece nici un element al vreunui tablou nu are –1 elemente în faţa sa). Indicele de început al listei este 0, adică indicele primului element înregistrat (**primul** = 0) .

câmpuri ale structurii **persoana**	nume	indice următor	nume	indice următor	nume	indice următor	nume	indice următor
valorile câmpurilor	*Ion*	1	*Anca*	2	*Vasile*	3	*Maria*	–1
indicii în variabila **grup**	**primul** = 0		1		2		**ultimul** = 3	

Figura 4.5

Cum, de obicei, o listă de persoane este cerută în ordine alfabetică, înseamnă că, după ordonarea numelor celor patru persoane, ordinea logică a elementelor va fi alta decât ordinea lor fizică pe care o aveau la crearea listei. Acum **primul** =1 şi se referă la Anca, iar ultimul este Vasile, adică elementul de indice 2 (fig. 4.6).

câmpuri ale structurii **persoana**	nume	indice următor	nume	indice următor	nume	indice următor	nume	indice următor
valorile câmpurilor	*Ion*	3	*Anca*	0	*Vasile*	–1	*Maria*	2
indicii în variabila **grup**	0		**primul** =1		**ultimul** = 2		3	

Figura 4.6

Trebuie remarcat faptul că valorile, informaţia din fiecare element, nu-şi schimbă locurile în listă, ci are loc o reorganizare a înlănţuirii lor logice.

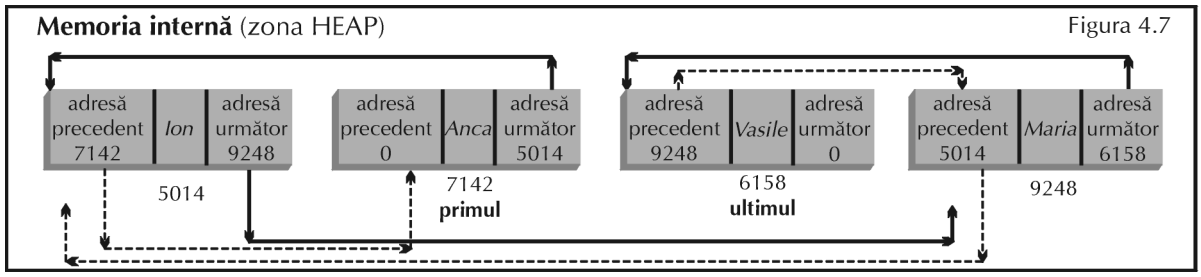
rezolvă

1. Stabiliţi tipul informaţiei dintr-un element al unei liste alocate *static, secvenţial*, care să conţină notele unui elev la un obiect.
2. Scrieţi declararea pentru spaţiul listei formată din elementele de tipul definit în exerciţiul anterior.
3. Desenaţi pe caiet lista formată în condiţiile din exerciţiile 1 şi 2 şi completaţi elementele vectorului cu notele obţinute la *limba română*, în ordinea în care au fost obţinute.
4. Scrieţi o secvenţă în limbajul de programare prin care să se ordoneze crescător notele înregistrate în lista creată până acum.
5. Reluaţi exerciţiile 1–3 pentru a descrie lista notelor în forma *secvenţial-înlănţuită, alocată static*.
6. Redesenaţi lista secvenţial-înlănţuită creată, modificând câmpul de indice **următor**, astfel încât notele să fie parcurse în ordine crescătoare. Marcaţi elementele **prim** şi **ultim** în desen.
7. Realizaţi exerciţiile 1–6 pentru cazul în care lista trebuie să conţină datele termice al fiecărei zile dintr-o săptămână. Informaţia dintr-un element al listei va specifica numele zilei şi temperatura înregistrată.

În lista dublu-înlănţuită adresele asigură între elemente o ordine logică în ambele sensuri. Astfel, lista dublu-înlănţuită are avantajul de a oferi o parcurgere a elementelor în ambele sensuri: de la primul la ultimul şi de la ultimul către primul.

exemplu

Considerăm aceleaşi patru persoane din exemplul anterior, numai că acum numele lor vor compune o listă dublu-înlănţuită (fig. 4.7).
În acest caz, fiecare element are câte două repere de tip adresă: adresa elementului precedent şi adresa elementului următor.



- Pentru *alocarea dinamică*, în figura 4.7 s-au considerat patru adrese de memorie în zona **HEAP**, în ordinea în care au fost ele furnizate ca libere pentru înregistrarea fiecărui nume de persoană: 7142, 5014, 9248 şi 6158. ***Notaţia 0*** pentru adresa elementului precedent al numelui *Anca* desemnează faptul că nu există un element precedent, nu indică adresa octetului 0 al memoriei interne[1]. De asemenea, valoarea 0 ca adresă de element următor numelui *Vasile* arată că nu există element următor, lista încheindu-se aici. Prin săgeţi pline este pus în evidenţă traseul de tip ***succesori***, adică parcurgerea de la *primul* element către *ultimul*. Prin săgeţi punctate este pus în evidenţă traseul de tip ***precedenţi***, adică o parcurgere a listei de la *ultimul* element către *primul*.
- În cazul *alocării statice*, lista va ocupa un vector de elemente de tipul:

```
struct persoana {char nume[15]; int indice_precedent,indice_urmator;};
persoana  grup[4];
```

Pentru exemplul celor patru nume de persoane, conţinutul vectorului de înregistrări, conform descrierii structurii **persoana**, este cel din figura 4.8.

nume	indice precedent	indice următor	nume	indice precedent	indice următor	nume	indice precedent	indice următor	nume	indice precedent	indice următor
Ion	1	3	*Anca*	–1	0	*Vasile*	3	–1	*Maria*	0	2
0			**primul** = 1			**ultimul** = 2			3		

Figura 4.8

Prin indicele precedent = –1 se desemnează faptul că *Anca* este primul nume în lista alfabetică. Ei îi urmează elementul de pe locul 0, adică *Ion*, care are ca precedent elementul de pe locul 1, adică pe *Anca*, iar ca următor pe *Maria*, elementul de pe locul 3. *Vasile* nu mai are înregistrat un element următor, de accea acest indice este –1, dar are elementul de pe locul 3 ca precedent, adică pe *Maria*.

Dacă informaţiile care se înregistrează în listă sunt tot de tipul **int**, ca şi valorile de indice necesare înlănţuirii, atunci pentru o listă simplu-înlănţuită se poate organiza o matrice cu două linii şi un număr de coloane, în funcţie de maximul de cazuri necesare a fi înregistrate. În prima linie se vor trece valorile de tip informaţie, iar în a doua linie se vor trece indicii pentru legăturile de ordine logică între elemente. În consecinţă, pentru lista dublu-înlănţuită se va organiza o matrice cu trei linii.

Alocarea dinamică este tehnica potrivită organizării informaţiilor în liste, deoarece permite ocuparea spaţiului în memoria internă, atât cât este nevoie şi în momentul în care este necesar. De asemenea, ea prezintă o flexibilitate maximă în ceea ce priveşte actualizarea conţinutului listelor, oferind un câştig substanţial de timp. Deoarece, însă, depăşeşte cadrul manualului, ***în continuare se va studia numai cazul listelor alocate static***.

1. Reluaţi exerciţiile date mai sus, la lista simplu-înlănţuită şi transformaţi-le pentru înregistrarea informaţiilor într-o listă dublu-înlănţuită alocată static.
2. Scrieţi lista indicilor elementelor în ordinea în care acestea sunt parcurse pentru a oferi ordinea descrescătoare a temperaturilor înregistrate în săptămâna aleasă.

[1] Primul segment de memorie, în general ocupat de nucleul sistemului de operare, este considerat zonă rezervată şi nici un program nu are acces prin comenzi proprii în această zonă. Astfel, nu poate fi accesată adresa 0.

3. Desenaţi o listă dublu-înlănţuită alocată static în care informaţia fiecărui element conţine titlul şi autorul unei cărţi.
4. Completaţi lista, pe desen, cu cărţile dintr-un raft al bibliotecii personale. Reţineţi decizia luată în situaţia în care desenul iniţial nu poate cuprinde toate cărţile din raftul ales.
5. Refaceţi indicii de înlănţuire pentru a desena situaţia în care se introduce în raft o carte nouă între a patra şi a cincea existente.
6. Refaceţi indicii de înlănţuire pentru a parcurge lista cărţilor în ordinea alfabetică a titlurilor.

4.3. Prelucrările principale la nivelul listei înlănțuite alocate static

4.3.1. Liste simplu-înlănțuite

a) Definiri de date necesare

• Reluăm definirea listei ca vector de elemente de tipul înregistrare şi declarăm modelul general astfel:

```
struct element {tip informatie; int indice_urmator;};
element  lista[n_max];
```

unde **n_max** este constanta prin care se specifică numărul maxim posibil de elemente din listă.

• Notaţii pentru variabilele necesare prelucrării listei:
– **prim**, **ultim**, **curent** – vor specifica indicii cu rol de a desemna locul primului, ultimului şi, respectiv, elementului curent tratat;
– **valoare** va reprezenta variabila din care se copiază conţinutul în câmpul de informaţie al unui element din listă;
– **n** va fi variabila naturală care arată numărul de elemente aşezate în listă la un moment dat;
– **L** va fi un vector care are lungimea maximă ca şi vectorul listei, constanta **n_max**, şi va gestiona ocuparea şi eliberarea locurilor în listă. Un element al lui L va conţine valoarea 0, dacă elementul de acelaşi indice din listă nu este liber, şi 1, în cazul în care elementul este liber, nealocat unei informaţii.

b) Iniţializarea listei

Prelucrarea iniţială pentru o listă este definirea listei vide. Astfel, înainte de a se înregistra primul element în spaţiul alocat listei, se vor face operaţiile:
– atribuirea valorilor de indici inexistenţi pentru *prim* şi *ultim*: **prim=-1; ultim=-1;**
– completarea vectorului **L** cu valoarea 1 în toate cele **n_max** elemente;
– iniţializarea cu 0 a variabilei **n**, numărul de elemente existente în listă.

Presupunem exemplul precedent al persoanelor ale căror nume se înregistrează într-o listă simplu-înlănţuită. De asemenea, presupunem declararea structurii **persoana** ca element al listei şi a variabilei **grup** ca fiind lista propriu-zisă, dar pentru n_max=10. Secvenţa operaţiilor de declarare şi iniţializare va fi:

```
struct persoana {char nume[15]; int urm;};
element  grup[10];int prim, ultim, n,L[10], curent;
prim=ultim=-1; n=0; for(curent=0; curent<n_max; curent++) L[curent]=1;
```

c) Crearea listei prin adăugarea unui element nou

Crearea listei înseamnă preluarea informaţiilor unui element (prin citire sau prin calcul) şi înregistrarea lor în primul loc liber existent în listă. Se poate întâmpla să nu existe loc liber. În acest caz, se va semnala utilizatorului că lista este plină şi nu mai poate fi adăugat noul element. Operaţia de creare se repetă până ce utilizatorul anunţă că nu mai sunt elemente de introdus în listă sau până la situaţia de listă plină.

Presupunem că, pentru exemplul luat, se introduc numele în ordinea dată de utilizator până ce acesta scrie cuvântul „stop”. Numele se citesc în variabila **den** (denumire) şi, dacă nu s-a citit cuvântul „stop” şi mai este loc în listă, atunci noul nume este instalat în primul loc liber găsit în **L** de variabila **curent**. Cum la acest moment numele citit este ultimul aşezat în listă, el nu va avea un element următor, aşa că **grup[curent].urm=-1** şi **ultim=curent.**

Secvenţa pentru această prelucrare este:

```
char den[15]; int plina=0,i;
cout<<"Dati primul nume ";cin>>den;
while(! plina && strcmp(strlwr(den)!="stop")
  {  n++; curent=-1;
     for(i=0;i<n_max && curent!'-1; i++)
      if(L[i]) curent=i;      //cauta loc liber
     if (curent)
      {L[curent]=0; strcpy(grup[curent].nume,den); grup[curent].urm=-1;
        if(prim==-1}prim=curent; else grup[ultim].urm=curent;
        ultim=curent,
       cout<<"Dati numele urmator "; cin>>den;
      }
    else plina=1;
  }
```

d) Parcurgerea listei

Parcurgerea listei se poate face pentru mai multe scopuri. Cea mai frecventă operaţie este afişarea listei.

Variabila **curent** va trece pe rând prin listă începând cu indicele **prim**. Secvenţa asociată exemplului de până acum este:

```
curent=prim;
while(curent>=0)
{cout<<grup[curent].nume<<" "; curent=grup[curent].urm; }
```

e) Căutarea unui element de o proprietate dată

Proprietatea după care se face căutarea poate fi legată de:

– ***valoarea*** informaţiei din acel element;

– ***poziţia*** elementului în listă.

Căutarea se încheie în momentul *găsirii elementului*, furnizând locul (indicele) lui în listă, sau în momentul *terminării listei*, furnizând valoarea –1 ca adresă (loc) în listă pentru un element inexistent.

Pentru rezultatul căutării se va folosi variabila **gasit**, în care se va înregistra valoarea adresei (locului) determinat şi variabila **ant**, pentru locul elementului anterior. În scrierea secvenţei generale a prelucrării, notarea proprietăţii după care se face căutarea se va face prin **Prop**, urmând ca în cazul concret al enunţului problemei să fie înlocuită cu expresia logică a condiţiei concrete de căutare.

```
curent=prim;
while(curent!=-1 && !Prop) curent=grup[curent].urm;
gasit=curent;
```

1. În grupul de persoane luat ca exemplu se doreşte verificare existenţei persoanei cu numele «Bianca». Secvenţa se va particulariza astfel:

```
curent=prim;
while(curent!=-1 && strcmp(grup[curent],"Bianca")!=0)
                    curent=grup[curent].urm;
gasit=curent;
```

Printre cele patru nume înregistrate în exemplul dat, numele «Bianca» nu există, aşa că rezultatul va fi `curent=-1`.

2. Tot în grupul de persoane de mai sus se doreşte să se determine ce nume este înregistrat pe poziţia a treia în listă. Secvenţa de identificare va trebui să numere persoanele peste care se trece şi va folosi în acest sens variabila **nr**.

```
curent=prim;nr=0;
while(curent!=-1 && ++nr!=3) curent=grup[curent].urm;
gasit=curent;
```

Pentru prelucrările prezentate până acum, programul de mai jos exemplifică asamblarea secvenţelor.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()
{typedef struct persoana
      {char nume[15];int urm;};
int prim,ultim, curent,L[4],i,plina=0,n;
persoana grup[4];
prim=ultim=-1;n=0;
char den[15];
for(i=0;i<4;i++)  L[i]=1;
clrscr();
cout<<"Dati numele primei persoane ";
cin>>den;
while(!plina &&
        strcmp(strlwr(den),"stop")!=0)
 {curent=-1;
  for(i=0;i<4&&curent==-1;i++)
      if(L[i]) {L[i]=0;curent=i;}
  if(curent==-1)plina=1;
    else
      {strcpy(grup[curent].nume,den);
    grup[curent].urm=-1;
    if(prim==-1)prim=curent;
      else grup[ultim].urm=curent;
    ultim=curent;
     n++;}
  cout<<"Dati numele urmator sau stop ";
  cin>>den;
 }// sf. while
if(plina) cout<<"\nLista e plina"<<endl;
//caut al treilea
 int nr=0;
 curent=prim;
 while(curent!=-1 &&++nr!=3)
          curent=grup[curent].urm;
 if(curent!=-1)
      cout<<grup[curent].nume;
 else
  cout<<"Nu sunt trei pers. in lista ";
getch();
}
```

În figura 4.9 se prezintă momentul în care s-a identificat al treilea nume înregistrat în grupul celor 4 persoane. Se observă, în fereastra **watch**, **nr**=3, iar pe ultimul rînd în fereastra **output**, s-a afişat numele găsit pe poziţia a treia (iniţiala este literă mică în urma aplicării **strlwr(den)**). De asemenea, în fereastra **watch** se observă în variabila **grup** ataşarea indicilor de legătură, în ordinea în care au venit la înregistrare cele patru nume, iar variabila **prim=0** arată către începutul lustei (aici: primul element din vectorul **grup**).

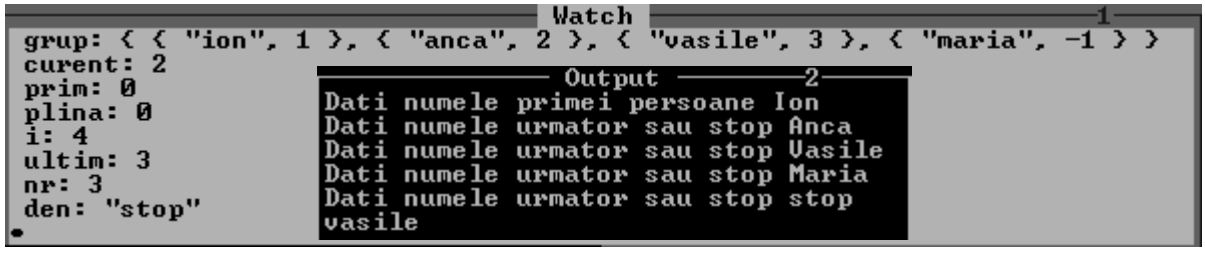


Figura 4.9

1. Modificaţi programul de mai sus, astfel încât să se generalizeze prelucrările pentru maximum **20** de persoane şi pentru căutarea persoanei de ordinul **k** – citit de la utilizator.
2. Adăugaţi programului creat la *tema 1* o variabilă, **ant**, şi secvenţa corespunzătoare pentru a se reţine în **ant** adresa (indicele) elementului ***anterior*** celui găsit că îndeplineşte proprietatea cerută.

f) Actualizarea conţinutului listei – inserarea, ştergerea sau modificarea unui element

Inserarea unui element

Ca şi la crearea prin adăugare, inserarea unui nou element trebuie să se realizeze numai dacă există loc liber în spaţiul alocat listei. Spre deosebire de adăugare, care presupune ataşarea unui nou element la sfârşitul de până atunci al listei (append), în cazul inserării se poate ivi una dintre următoarele trei situaţii: inserarea la ***începutul*** listei, la ***sfârşitul*** ei sau în ***interiorul*** acesteia.

Inserarea la începutul listei presupune că *elementul nou venit va deveni primul în listă*.

Pentru notaţiile din exemplul ales:

• variabila **curent** va primi indicele unui loc liber în listă, dacă acesta există;

• la locul indicat de **curent** se va instala **noul nume** (informaţia nou venită - **valoarea**);

• adresa lui de legătură va fi încărcată cu valoarea din *prim*: **grup[curent].urm=prim;**

• variabila **prim** preia noul loc, **prim=curent;**

În figura 4.10 este schiţată operaţia de inserare înaintea primului elelement din listă. Aşezarea fizică a elementelor nu contează, important este aspectul legat de realizarea corectă a înlănţuirii prin adrese (indici).

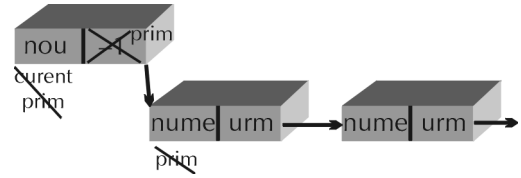


Figura 4.10

Inserarea la sfârşitul listei este echivalentă cu adăugarea unui nou element, acesta ataşându-se de ultimul element al listei de până atunci.

Pentru notaţiile din exemplul ales:

• variabila **curent** va primi indicele unui loc liber în listă, dacă acesta există;

• la locul indicat de **curent** se va instala **noul nume** (informaţia nou venită);

• adresa lui de legătură va fi încărcată cu valoarea –1, deoarece este ultimul: **grup[curent].urm=-1;**

• elementul desemnat de **ultim** îşi va schimba acum legătura de indice din **-1** în ce indică variabila **curent**:

grup[ultim].urm=curent;

• variabila **ultim** preia noul loc, **ultim=curent;**

În figura 4.11 este schiţată operaţia de inserare după ultimul element din listă. Aşezarea fizică a elementelor nu contează, important este aspectul legat de realizarea corectă a înlănţuirii prin adrese (indici).

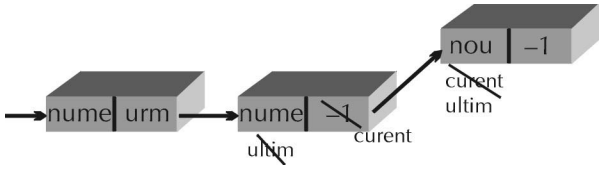


Figura 4.11

Inserarea în interiorul listei se poate face ***înainte*** sau ***după*** un anume element care îndeplineşte o proprietate **propr** impusă. Inserarea se face *după ce s-a realizat căutarea* elementului având proprietatea respectivă.

O variabilă **ant** va reţine întotdeauna adresa elementului anterior. Pentru notaţiile din exemplul ales:

• variabila **curent** va primi indicele unui loc liber în listă, dacă acesta există;

• la locul indicat de **curent** se va instala **noul nume** (informaţia nou venită);

• adresa lui de legătură va fi încărcată cu valoarea de indice din **ant: grup[curent].urm=grup[ant].urm;**

• adresa de legătură a elementului anterior se modifică în **curent: grup[ant].urm=curent.**

Analizând figura 4.12, se poate observa un *aspect comun inserărilor înainte sau după* elementul de proprietatea **propr**. Plecând de la faptul că ***operaţia de căutare furnizează indicii*** (adresele) elementului anterior şi ai celui care are proprietatea cerută, în variabilele **ant** şi, respectiv, **gasit**, atunci:

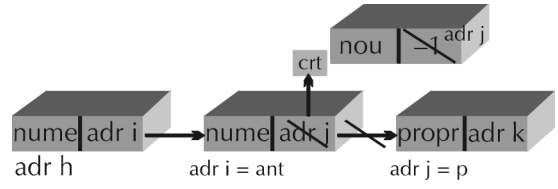


Figura 4.12

– inserarea înaintea elementului de proprietate **propr** va face inserţia în ordinea logică între elementele de locuri **ant** şi **gasit**;

– inserarea după elementul care are proprietatea cerută va face inserţia în ordinea logică între elementele de locuri **gasit** şi următorul lui **gasit**, adică, conform notaţiilor din exemplu, **grup[gasit].urm**.

În programul de mai jos sunt figurate operaţiile inserării, pentru cele trei cazuri – înaintea primului element, la sfârşitul listei şi în interiorul listei, înaintea elementului care conţine un nume mai mare faţă de ordinea lexicografică (mai mare alfabetic).

Programul foloseşte vectorul **grup** care se creează acum prin operaţia de inserare, spre deosebire de varianta anterioară, de creare prin adăugare. Fiecare nou nume citit este inserat din punct de vedere lexicografic înaintea celui care îi urmează alfabetic.

Fizic, elementele ocupă aceleaşi zone în vectorul **grup** ca în varianta de creare prin adăugare. Logic, însă, indicii de înlănţuire din câmpurile **urm** sunt diferiţi faţă de prima variantă. De asemenea, şi valoarea din variabila **prim**. Parcurgerea listei create acum va produce o afişare în ordine alfabetică a numelor celor patru persoane luate ca exemplu. În figura 4.13 se observă aceste lucruri în ferestrele **watch** şi **output**.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()
{typedef struct persoana
      {char nume[15];int urm;};
int prim,ultim, curent,ant,gasit,L[4];
int i,plina=0,n=0;
persoana grup[4];
prim=ultim=-1;
char den[15];
for(i=0;i<4;i++)  L[i]=1;
clrscr();
cout<<"Dati numele primei persoane ";
cin>>den;
while(!plina &&
         strcmp(strlwr(den),"stop")!=0)
 {if(prim==-1)
  //ataseaza primul element  ***
   { L[0]=0;prim=0;ultim=0;
     strcpy(grup[prim].nume,den);
     grup[prim].urm=-1;n++;
   }
  else
  { curent=-1;
    for(i=0;i<4&&curent==-1;i++)
      if(L[i]) {L[i]=0;curent=i;}
    if(curent==-1)plina=1;
     else
     {strcpy(grup[curent].nume,den);
      //cauta locul in ord. alfabetica
                          pt. inlant.
      ant=-1;
      gasit=prim;
      n++;
   while(gasit!=-1&&
      strcmp(den,grup[gasit].nume)>=0)
       {ant=gasit;gasit=grup[gasit].urm;}
    if(ant==-1)
       //inserare inainte de primul
       { grup[curent].urm=prim;
         prim=curent;
       }
     else
        if(gasit==-1)
        //inserare la sfarsit
        { grup[curent].urm=-1;
          grup[ultim].urm=curent;
          ultim=curent;
        }
        else
        //inserare intre ant si gasit, in
                    interiorul listei
   {grup[curent].urm=grup[ant].urm;
          grup[ant].urm=curent;
       //sau:
       // grup[curent].urm=gasit; ;
       //grup[ant].urm=curent;
          }
    }//sf. else  ordine alfabetica
  } //sf. else lista nu e plina
  cout<<"Dati numele urmator sau stop ";
  cin>>den;
}//sf. while citire nume
if(plina) cout<<"\nLista e plina"<<endl;
//afisarea listei
curent=prim;
while(curent!=-1)
 {cout<<grup[curent].nume<<endl;
  curent=grup[curent].urm;
 }
getch();}
```

Figura 4.13

Se consideră un vector care conţine o listă cu numerele naturale din tabelul de mai jos, primul rînd. În al doilea rând sunt trecuţi indicii zonelor.

8		21		34		13		5			
0		1		2		3		4		5	

1. Completaţi zonele gri cu adresele – indicii care se cer pentru ca numerele date să fie organizate în ordine crescătoare.
2. Completaţi valorile pentru variabilele **prim** şi **ultim**.
3. Figuraţi inserarea numărului 10, astfel încât să nu fie încălcată regula de ordine crescătoare.
4. Figuraţi inserarea numărului 3 în structura dată iniţial, astfel încât să nu fie încălcată regula de ordine crescătoare.
5. Figuraţi inserarea numărului 50 în structura dată iniţial, astfel încât să nu fie încălcată regula de ordine crescătoare.

Teme

1. Modificaţi programul anterior, astfel încât să se insereze şi primul element în listă în cadrul secvenţei opţiunilor de inserare şi nu separat, la început (marcajul cu ***).

2. Realizaţi un program general, pentru **n** persoane, **n≤20**, care să conţină o singură secvenţă în care să fie cuprinse toate cele trei cazuri de inserare.

3. Realizaţi o secvenţă pentru inserarea în listă a unui nume de persoană, **x** citit, după un anumit nume existent, **y** citit, astfel: căutaţi un loc liber în **L** şi furnizaţi-l în **curent**; stabiliţi în variabila **p** locul în listă al numelui din variabila **y**; încărcaţi în câmpul **urm** de la adresa **curent** indicele de legătură din câmpul **urm** al adresei **p**; încărcaţi în câmpul **urm** de la adresa **p** valoarea variabilei **curent**; încărcaţi în câmpul **nume** al adresei **curent** numele de la adresa **p**; încărcaţi valoarea din **x** în câmpul **nume** de la adresa **p**.

Ştergerea unui element

Operaţia de ştergere a unui element din listă implică o serie de etape care vor fi listate mai jos utilizând notaţiile de până acum. În figurile 4.14 – 4.16 sunt schiţate cazurile de ştergere a unui element dintr-o listă pentru care sunt figurate trei elemente. Etapele de parcurs sunt:

– localizarea lui în listă după o proprietate **propr** dată;
– furnizarea adresei lui în variabila **curent**;
– furnizarea adresei elementului anterior lui în variabila **ant**;
– analiza cazului de ştergere:

• Elementul care trebuie şters este localizat ca fiind primul, adică: **ant==-1** şi **curent==prim**.
Operaţiile care trebuie realizate sunt : **L[curent]=1; prim=grup[prim].urm;**

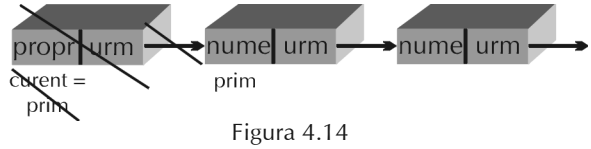


Figura 4.14

• Elementul care trebuie şters este localizat ca fiind ultimul, adică: **curent==ultim**.
Operaţiile care trebuie realizate sunt : **L[curent]=1; grup[ant].urm=-1; ultim=ant;**

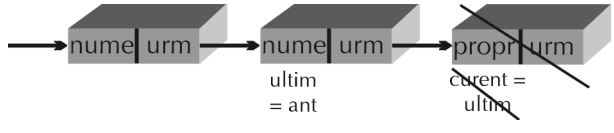
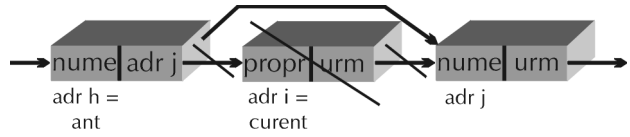


Figura 4.15

• Elementul care trebuie şters este în interiorul listei.
Operaţiile care trebuie realizate sunt : **`L[curent]=1; grup[ant].urm=grup[curent].urm;`**

Figura 4.16



rezolvă

Se consideră un vector **`V`**, care conţine o listă cu numerele naturale din tabelul de mai jos, primul rînd. În al doilea rând sunt trecuţi indicii zonelor.

8	5	21	2	34	–1	13	1	5	0	10	3
0		1		ultim=2		3		prim=4		5	

1. Desenaţi un tabel care să imagineze conţinutul vectorului `L`.
2. Scrieţi declararea de structură pentru elementul unei astfel de liste de numere naturale şi definirile de variabile necesare.
3. Stabiliţi ce număr este indicat prin `V[V[prim].urm].urm`.
4. Refaceţi indicii din zonele colorate care sunt afectaţi de eliminarea numărului 21, astfel ca ordinea logică în listă să se păstreze;
5. În starea listei lăsată de cerinţa anterioară, faceţi modificările necesare care apar în urma eliminării numărului 5.
6. Scrieţi noul conţinut al tabelului care imaginează vectorul `L`.

Modificarea unui element

Prelucrarea este uşor de realizat după ce s-au înţeles operaţiile de parcurgere a listei şi de căutare a unui element care îndeplineşte o anumită proprietate. Problema modificării unui element din listă presupune modificarea informaţiei conţinute de acel element şi constă în următoarele etape:

– parcurgerea listei pentru a localiza elementul care îndeplineşte proprietatea cerută (de exemplu: o anumită valoare, o anumită relaţie cu o valoare fixată, o anumită calitate a valorii, un anumit loc în listă);

– dacă elementul a fost găsit, se aplică modificarea (înlocuirea informaţiei din element cu o altă **`valoare`** dată din exterior sau din calcul).

Modificarea unui element nu afectează informaţiile legate de adrese.

Teme

1. Realizaţi un program pe baza aceluiaşi exemplu cu numele a **`n`** persoane, care să citească un nume de la tastatură în variabila **`x`** şi să şteargă acel nume din lista **`grup`**. Programul va afişa un mesaj corespunzător, în funcţie de situaţie: a şters numele din listă sau numele nu a fost găsit în listă. La sfârşit, programul va afişa noul conţinut al listei.

2. Realizaţi un program care să prelucreze o listă de numere naturale, de tipul celei date în exerciţii, după cum urmează: *crearea* listei prin inserarea fiecărui element nou, astfel încât să se obţină o ordine crescătoare a valorilor, citirea unui număr care trebuie *inserat* apoi, fără a modifica ordinea crescătoare, citirea unei valori care, dacă există în listă, va fi *şteasă* şi vor fi refăcute legăturile ordinii logice, *afişarea* listei finale.

3. Realizaţi un program prin care se citeşte un nume de la tastatură, în variabila **`x`**. Apoi, în lista **`grup`** a persoanelor studiată până acum, se caută acest nume şi se înlocuieşte cu un altul citit în variabila **`y`**. La sfârşit, se afişează lista sau mesajul corespunzător privind inexistenţa numelui căutat.

Aplicaţie 1

Fie o problemă a unei firme de turism care se enunţă astfel: dintr-o listă de **`n`** localităţi, **`n`**$\leq$**`20`**, aşezate în ordine alfabetică, se vor alcătui anumite voiaje. Costul zilnic de cazare în fiecare localitate este cunoscut dintr-un fişier de date **`locuri.txt`**. Un turist doreşte să afle costul total al unui voiaj pe care îl alcătuieşte din lista de localităţi prezentată de firmă. Nu se iau în considerare cheltuielile de transport.

De exemplu, fie lista localităţilor însoţite de costul de cazare pe zi:

ADAMCLISI	AGAPIA	BUCUREŞTI	HOREZU	MONEASA	SAPÂNŢA	VORONEŢ
20	10	30	15	17	7	10

şi se doreşte afişarea unui itinerariu care porneşte din Bucureşti. Acesta poate fi cel din figura 4.17, care pleacă din Bucureşti şi se încheie la Horezu.

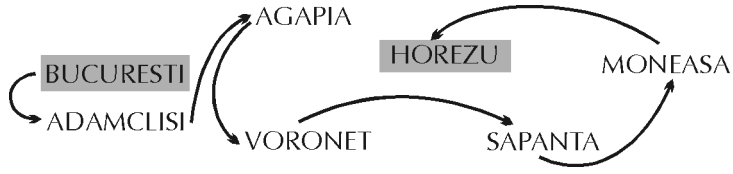


Figura 4.17

Costul total va fi de 109 lei noi.

Rezolvare. Pentru a rezolva această problemă este nevoie, în primul rând, de organizarea datelor, astfel ca localităţile să fie reprezentate şi la nivelul programului. Se va alege o listă simplu-înlănţuită, **lista**. Apoi, pe baza listei localităţilor şi a propunerilor făcute de client, se alcătuieşte un voiaj; acesta şi costul total se afişează costul total.

```
#include<fstream.h>
#include<conio.h>
#include<string.h>
void main()
{typedef struct loc
  {char nume[15];float pret;int urm;};
int prim,ultim,curent,ant,p,L[20],i,
         v[20],plina=0;
loc lista[20];
prim=ultim=-1;
char den[15]; float cost, cost_total;
ifstream f("locuri.txt");
for(i=0;i<20;i++) L[i]=1;
clrscr();
while(!f.eof()&& !plina)//creare lista
 {f>>den;if(f.eof())break;
  f>>cost;
  curent=-1;
  for(i=0;i<20&&curent==-1;i++)
    if(L[i])  {L[i]=0;curent=i;}
  if(curent==-1)plina=1;
    else
    {strcpy(lista[curent].nume,den);
     lista[curent].pret=cost;
     lista[curent].urm=-1;
     ant=-1;
     p=prim;
     while(p!=-1 &&
          strcmp(den,lista[p].nume)>=0)
      {ant=p;p=lista[p].urm;}
       if(ant==-1)
        { lista[curent].urm=prim;
          prim=curent;
          if(ultim==-1)ultim=curent;
        }
        else
         if(p==-1)
          { lista[ultim].urm=curent;
            ultim=curent;
          }
        else
   { lista[ant].urm=curent;
     lista[curent].urm=p;
    }
  }
 }
if(plina) cout<<"\nLista e plina"<<endl;
f.close();
//Afisare lista
curent=prim;
while(curent!=-1)
      { cout<<lista[curent].nume<<" ";
        curent=lista[curent].urm;
      }
//construire voiaj
cout<<endl;
cost_total=0;i=-1;
cout<<"Dati localitatea de pornire ";
cin>>den;
do
  {curent=prim;
   while(curent!=-1 &&
     strcmp(lista[curent].nume,den)!=0)
     curent=lista[curent].urm;
   if(curent==-1)
      cout<<"Nu este o localitate din
                lista"<<endl;
    else
     {v[++i]=curent;
      cost_total+=lista[curent].pret;
     }
   cout<<"Dati localitatea urmatoare ";
    cin>>den;
 }while(strcmp(den,"stop")!=0);
```

```
cout<<"\n Ati ales voiajul "<<endl;
for(int k=0;k<=i;k++)
      cout<<lista[v[k]].nume<<endl;
cout<<"Cu costul total "<<cost_total;
getch();
}
```

În program este folosit vectorul **v** în care sunt înregistraţi indicii localităţilor alese de client din **lista**.

Modificaţi programul aplicaţiei de mai sus pentru a extinde dialogul cu clientul, astfel încât acesta să poată propune mai mult de un voiaj; pentru fiecare nouă propunere se afişează lista localităţilor pe un rând-ecran, iar pe rândul următor, costul acelui voiaj.

Aplicaţie 2

Un caz interesant de listă este ***lista circulară***, adică lanţul închis. Pentru acest tip de listă extremitatea de început coincide cu cea de sfârşit. Simularea unei astfel de liste cu ajutorul structurilor fixe va folosi tot un tablou unidimensional. În prelucrările informaţiilor din acest tip de listă se va avea grijă ca elementul cu indicele de sfârşit al listei, **ultim**, să aibă ca succesor elementul de indice iniţial al listei, **prim**. În cazul eliminărilor unor elemente dintr-o listă circulară apare necesară determinarea situaţiei de listă cu un singur element, adică momentul în care **prim=ultim**, sau, altfel exprimat, **lista[prim].urm=prim**.

Ca aplicaţie, se va considera lista **grup** utilizată în exemplele paragrafului. Numele persoanelor din lista **grup** vor fi ale unor copii care sunt aşezaţi în cerc în ordinea venirii. Copiii pregătesc un joc prin alegerea celui care să pornească jocul. Pentru aceasta, începând cu primul aşezat în cerc, este eliminat fiecare al **n**-lea copil, pînă ce mai rămâne un singur copil care va începe jocul. Se cere organizarea acestor prelucrări într-un program care să afişeze numele copilului care este ales în acest mod, numele copiilor citindu-se dintr-un fişier, **copii.txt**, iar valoarea lui **n** – de la tastatură.

```
#include<fstream.h>
#include<conio.h>
#include<string.h>
void main()
{typedef struct persoana
      {char nume[15];int urm;};
int prim,ultim,ant,curent,L[10],i,
              plina=0,n;
persoana grup[10]; char den[15];
prim=ultim=-1;
for(i=0;i<10;i++)  L[i]=1;
clrscr();
ifstream f("copii.txt");
while(!f.eof()&&!plina)//creare lista
{ f>>den;
  if(f.eof()) break;
  curent=-1;
  for(i=0;i<10&&curent==-1;i++)
    if(L[i]) {L[i]=0;curent=i;}
  if(curent==-1)plina=1;
   else
    {strcpy(grup[curent].nume,den);
     grup[curent].urm=-1;
     if(prim==-1)  prim=curent;
         else grup[ultim].urm=curent;
     ultim=curent;
      }
}if(plina) cout<<"\nLista e plina"<<endl;
f.close();
//afisarea listei
cout<<"lista creata este"<<endl;
curent=prim;
while(curent!=-1)
      {cout<<grup[curent].nume<<" ";
       curent=grup[curent].urm;
       }

//inchiderea circulara
grup[ultim].urm=prim;

//eliminare cu pasul n
cout<<endl;
cout<<"Dati pasul de elmiminare ";
cin>>n;
curent=prim;

while(grup[curent].urm!=curent)
  {for(i=1;i<n;i++)
        {ant=curent;
            curent=grup[curent].urm;
        }
      grup[ant].urm=grup[curent].urm;
      L[curent]=1;
      curent=grup[ant].urm;
  }

cout<<endl;
cout<<"A ramas copilul ";
cout<<grup[curent].nume;
getch();
}
```

Pentru fiecare subiect se acordă câte 1 punct; două puncte sunt din oficiu.

1. Se consideră o listă simplu-înlănţuită, **grup**, pentru care **p** reprezintă adresa primului element iar **d**, adresa elementului după care trebuie inserat un element nou care a fost instalat la adresa **c**. Stabiliţi care secvenţe din lista de mai jos realizează corect această inserare:
a) `grup[d].urm=c; grup[c].urm=d;` **b)** `d=grup[c].urm; grup[c].urm=grup[d].urm;`
c) `grup[c].urm=grup[d].urm; grup[d].urm=c;` **d)** `d=c;grup[c].urm=grup[d].urm;`

2. Fie **a**, **b** şi **c** trei variabile care conţin indici (adrese) într-o listă simplu-înlănţuită de numere naturale, numită **nat** şi descrisă prin: `struct {unsigned nr; int urm;}nat[20];`
Să se precizeze ce va afişa secvenţa:
`nat[a].nr=1; c=a; nat[b].nr=2; b=c; cout<<nat[b].nr<<",”<<nat[c].nr;`
a) 2,2; **b)** 1,1; **c)** 1,2; **d)** 2,1.

3. Se consideră lista simplu-înlănţuită de numere naturale numită **nat** şi descrisă prin:
`struct {unsigned nr; int urm;}nat[20];`
pentru care **prim** conţine indicele primului element, iar **curent** conţine un indice oarecare din listă. Ştiind că lista conţine numerele 1, 2, 3, 4, în această ordine, să se precizeze ce afişează secvenţa:
`curent=prim;`
`while(nat[curent].urm!=-1) {curent=nat[curent].urm; cout<<nat[curent].nr<<",";}`
a) 1,2,3,4,5; **b)** 2,3,4,5; **c)** 1,2,3,4; **d)** 2,3,4.

4. Fie **a**, **b** şi **c** trei variabile care conţin indici (adrese) într-o listă simplu-înlănţuită de numere naturale, numită **nat** şi descrisă prin:
`struct {unsigned nr; int urm;}nat[20];`
Să se precizeze ce va afişa secvenţa:
```
nat[a].nr=3; nat[b].nr=7; nat[a].urm=b;nat[b].urm=a; c=a;
for(i=1;i<=3;i++)c=nat[c].urm; cout<<nat[c].nr;
```
a) 7; **b)** 3 3 3; **c)** 7; **d)** 7 7 7.

5. Se consideră o listă simplu-înlănţuită, **grup**, conţinând nume de persoane. Pentru această listă, **p** reprezintă adresa primului element, iar **d**, adresa elementului înaintea căruia trebuie inserat un element nou ce a fost instalat la adresa **c**. Stabiliţi care secvenţe din lista de mai jos realizează corect această inserare:
a) `grup[d].urm=c; grup[c].urm=d;` **b)** `d=grup[c].urm; grup[c].urm=grup[d].urm;`
c) `strcpy(aux,grup[d].nume;) strcpy(grup[d].nume,grup[c].nume);`
`strcpy(grup[c].nume,aux); grup[c].urm=grup[d].urm; grup[d].urm=c;`
d) `d=c;grup[c].urm=grup[d].urm;`

6. Se consideră o listă simplu-înlănţuită, **grup**, conţinând nume de persoane. În această listă, **d** reprezintă adresa elementului care trebuie şters din listă. Stabiliţi care secvenţe din lista de mai jos realizează corect această ştergere:
a) `grup[grup[d].urm].urm=grup[d].urm; strcpy(grup[grup[d].urm].nume, grup[d].nume);`
b) `grup[d].urm=grup[grup[d].urm].nume; grup[d].nume=grup[grup[d].urm].urm;`
c) `grup[d].urm=grup[grup[d].urm].urm;`
d) `strcpy(grup[d].nume,grup[grup[d].urm].nume); grup[d].urm=grup[grup[d] .urm].urm;`

7. Pentru lista **nat** folosită mai sus, se realizează secvenţa: `c=prim; do {cout<<nat[c].nr<<" "; c=nat[c].urm;}while(c!=prim);` Secvenţa descrie:
a) afişarea listei; **b)** o listă circulară; **c)** afişarea unei liste circulare;
d) afişarea capetelor listei.

8. Se consideră lista simplu-înlănţuită **nat** – descrisă în subiectul 2 – şi un element care nu este nici primul, nici ultimul. Adresa (indicele) elementului se află în variabila **s**. Nu se cunoaşte adresa primului element din listă. Scrieţi o secvenţă prin care elementul de loc **s** să fie şters din listă.

4.3.1.1. Stiva

Toată lumea cunoaşte noţiunea de stivă[1] din practică, de la banala stivă de farfurii. Astfel, dacă cineva doreşte să ia o anumită farfurie din stivă, atunci va fi obligat să le ia, pe rând, pe toate celelalte de deasupra, să le pună într-o nouă stivă de farfurii şi, în sfârşit, să o ia pe cea pe care o dorea, după care să le pună la loc pe celelalte. De aici apare şi definiţia acestei structuri de date.

Stiva este o listă de informaţii care are două extremităţi, numite *bază* şi *vârf*, şi la care accesul este permis numai pe la vârf.

Datorită modului de operare asupra elementelor stivei, şi anume, asupra locului din **`vârf`**, stivei i se mai spune pe scurt şi structură **LIFO** (*Last In First Out* – ultimul intrat în structură este primul care va ieşi la prelucrare).

O stivă fără nici un element se numeşte *stivă vidă*. Într-o stivă vidă, **`virful`** coincide cu **`baza`**. În cazul unei stive nevide, pentru a ajunge ca **`vârful`** să coincidă cu **`baza`** trebuie extrase toate elementele de deasupra **`bazei.`**

Simularea structurii de stivă cu ajutorul tabloului unidimensional, care are o alocare statică de dimensiune fixă, duce, cum se întâmplă, în general, în cazul listelor, la apariţia unei alte noţiuni legate de starea stivei, şi anume ***stivă plină***, când toate poziţiile alocate ei au fost ocupate (stivele din realitate sunt însă în condiţii ideale, infinite).

Prelucrări specifice stivei

a) Adăugarea unui element în vârf sau extragerea elementului din vârf.
b) Crearea unei stive, crearea fiind o adăugare de element nou în vârful stivei, repetată pentru mai multe elemente.
c) Consultarea unei stive, pentru vizualizarea elementelor ei.
d) Concatenarea a două stive.
e) Dispersia unei stive în mai multe stive, conform unui criteriu dat.
f) Simularea operaţiilor de inserare sau de ştergere (eliminarea) a unui element oarecare, din interiorul stivei.

Vor fi date exemple pentru alocarea statică secvenţială.


exemplu

Să se creeze două stive, *a* şi *b*, apoi să se obţină stiva *c* prin concatenarea celor două. Elementele stivelor sunt caractere.

Rezolvare. În programul de mai jos este tratată o prelucrare complexă în care se regăsesc prelucrările de bază: *crearea unei stive, listarea unei stive, adăugarea unui element, extragerea element cu element până la golirea stivei*.

Pentru stivele *a* şi *b* s-a alocat un spaţiu de 20 de elemente (**`nmax`**=20), într-un tablou unidimensional de caractere, iar pentru stiva *c*, un spaţiu de **2×`nmax`** (40 de elemente). La început stivele sunt vide, astfel că *vârfurile lor, vf1 şi vf2*, sunt egale cu indicii bazelor (adică valoarea –1 prin care se desemnează neocuparea nici unui loc în vector).

– La ***crearea*** fiecăreia dintre stivele *a* sau *b*, ***înregistrarea unui caracter*** citit se face numai dacă acel caracter este diferit de caracterele de *sfârşit de linie* sau dacă stiva nu este *plină* (nu s-au ocupat încă cele 20 de locuri rezervate elementelor ei).

– Apoi, pentru a muta stiva *a* în *c* în vederea alipirii ulterioare a stivei *b*, se face o prelucrare ce pare curioasă pentru accesul la elementele unui tablou (care poate fi accesat din orice capăt dorim), dar este absolut necesară pentru a reflecta principiul de acces în stivă: elementele din *a* se mută, prin ***extargere***, într-o stivă auxiliară, *aux*, după care se mută în *c*.

– Mutarea elementelor din stiva *a* în stiva *c* înseamnă, de fapt, ***eliminarea*** elementelor din stiva *a* şi ***golirea*** ei.

– Stiva *b* va fi şi ea golită, fiind mutată în *aux*, iar apoi, din *aux* în stiva *c*.

– La sfârşit se va ***lista stiva*** nou obţinută, stiva c.

De exemplu, fie **`nmax`**=4. Dacă avem **`a`**=('1','2','3','4'), atunci '4' este ultimul element intrat, deci primul care trebuie scos. Mutată direct **`a`** în **`c`**, ca în orice operaţie de mutare între vectori, se va produce **`c`**=('4','3','2','1') ceea ce nu mai reprezintă ordinea din stiva **`a`**. De aceea **`a`**=('1','2','3','4') → **`aux`**=('4','3','2','1') → **`c`**=('1','2','3','4')

De exemplu, dacă **`b`**=('1', '2', '3', '4', '5') atunci, în final, în vârful stivei **`c`** se va găsi valoarea 5, deci **`c`**=('1', '2', '3', '4', '1', '2', '3', '4', '5') , astfel încât, la listarea stivei, primul care va apărea pe ecran va fi 5, apoi 4 ş.a.m.d., 1, apoi 4, 3, 2, 1.

[1] stack [stșk] – stivă (de lemne – engl.)

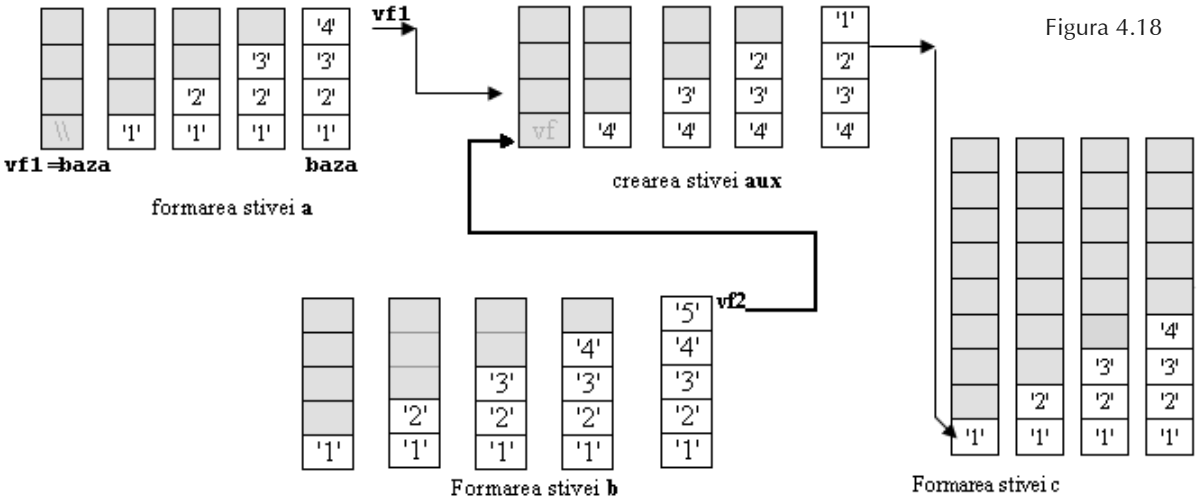


Figura 4.18

```
#include <iostream.h>
#include<conio.h>
const nmax=20;
typedef char stiva[nmax];
stiva a,b,aux;char ch[2];
int vf1,vf2,vf3,i;char c[2*nmax];
void main()
{  vf1=-1;
// vf1 este varful primei stive,
// la inceput coincide cu baza
clrscr();
cout<<"Dati continutul primei stive ";
ch[1]=getche();
while(ch[1]!=13&&ch[1]!=10&&vf1<nmax-1)
 {
   vf1++; a[vf1]=ch[1];ch[1]=getche();
 }
cin.get(ch,1,'\n');//golire buffer-e
// vf2 este varful stivei a doua
// la inceput coincide cu baza
vf2=-1;
cout<<"\nDati continutul stivei a
doua";
ch[1]=getche();
while(ch[1]!=13&&ch[1]!=10&&vf2<nmax-1)
  {
  vf2++; b[vf2]=ch[1];ch[1]=getche();
  }
cin.get(ch,1,'\n');//golire buffer-e
for (i=0; vf1>=0;vf1--,i++)
       aux[i]=a[vf1]; //golire stiva a
i--;vf1=-1;
for (vf3=0;i>= 0 ;i--,vf3++)
              c[vf3]=aux[i]; //stiva c
for (i=0;vf2>=0;vf2--,i++)
       aux[i]=b[vf2];//golire stiva b
i--;vf2=-1;
for (;i>=0;i--,vf3++) //adaug in stiva c
             c[vf3]=aux[i];
cout<<"\nStiva rezultata:";
for (i=vf3-1;i>=0;i--) //afisare stiva c
            cout<<c[i]<<" ";
getch();
}
```

Pentru a se executa programul de mai sus, utilizatorul trebuie să introducă caracterele fiecărei liste *unul după altul, în continuare*, iar abia la epuizarea unei liste să tasteze **ENTER**.
Programul este dat şi cu scopul de a prezenta un model de citire în ecou a unui caracter (getche()) şi pentru recapitularea modului de folosire a caracterelor albe.

1. Dându-se o stivă, să se elemine un element de valoare dată. *Indicaţie*: Pentru eliminarea unui caracter din interiorul stivei, acesta se cere utilizatorului şi se verifică, prin consultare, dacă există în stivă. Dacă nu există, se afişează un mesaj corespunzător. Dacă elementul există pe un loc **k**, atunci se descarcă stiva într-o stivă auxiliară, **aux**, inclusiv elementul de pe locul **k**. Vârful stivei originale va fi la indicele **k-1**. Începând de aici, se va urca înapoi în stiva iniţială cu elementele salvate în stiva auxiliară, mai puţin ultimul intrat în stiva auxiliară, care este elementul ce trebuie eliminat.

rezolvă

2. Realizaţi operaţiile de creare, adăugare şi ştergere pentru stiva alocată acum înlănţuit, urmărind eliberarea locurilor cu ajutorul vectorului L, utilizând cele învăţate la liste.

4.3.1.2. Coada

Exemplul cel mai întâlnit de coadă[1] este şirul de aşteptare creat, de exemplu, la casa unui mare magazin. Primul sosit la coadă este şi primul servit.

Coada este o listă ale cărei extremităţi se numesc *cap* şi *sfârşit* şi pentru care o informaţie nu poate intra decât pe la extremitatea ***sfârşit*** şi nu poate ieşi decât pe la extremitatea ***cap***.

Din modul în care este definit accesul structurii de tip coadă, acestui tip de listă i se mai spune, pe scurt, **FIFO** (*First In First Out* – primul element intrat este şi primul care iese la prelucrare).

O coadă fără nici un element se numeşte *coadă vidă*.

Simularea structurii de coadă cu ajutorul tabloului unidimensional duce, ca şi pentru stivă, la apariţia noţiunii legate de starea cozii, şi anume *coadă plină*, când toate poziţiile alocate ei au fost ocupate (cozile din realitate însă sunt, în condiţii ideale, infinite).

Prelucrări specifice

a) Adăugarea unui element în coadă sau extragerea primului element.
b) Crearea unei cozi (operaţie care presupune o adăugare repetată).
c) Consultarea sau traversarea unei cozi, în scopul vizualizării elementelor.
d) Concatenarea a două cozi.
e) Dispersia unei cozi în mai multe cozi, conform unui criteriu dat.
f) Simularea operaţiei de inserare sau ştergere (eliminare) a unui element oarecare.

Problemă rezolvată

1) Să se creeze două liste de tip coadă, *a* şi *b*, conţinând caractere, iar prin concatenarea lor, o a treia listă de tip coadă, *c*.

Rezolvare. În programul de mai jos este tratată o prelucrare complexă, în care se regăsesc prelucrările de bază: *crearea unei cozi, listarea unei cozi, adăugarea unui element, extragerea element cu element până la golirea cozii*.

Pentru listele **a** şi **b** s-a alocat un spaţiu de **nmax**=20 elemente într-un tablou unidimensional de caractere, iar pentru coada **c,** s-a alocat un spaţiu de 2***nmax**=40 elemente. Variabilele de tip adrese pentru aceste liste vor fi:

- **cap1=0**, **cap2=0** şi, respectiv, **cap3=0**, pentru oricare dintre cele trei liste care conţine cel puţin un element;
- **sf1**, **sf2**, **sf3**, care vor conţine adresele (indicii) ocupate de ultimele elemente existente în listă.

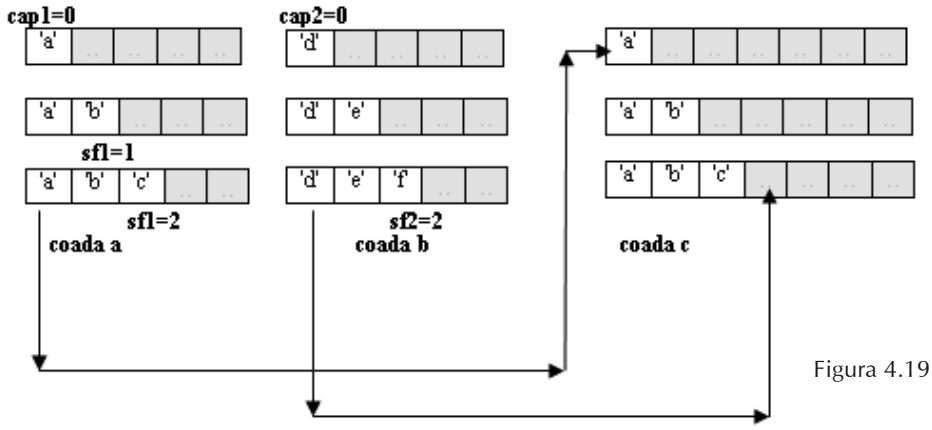


Figura 4.19

[1] queue [kju:] – coadă (engl.)

Operaţiile:

– ***Iniţial*** cozile sunt *vide*: cap1=–1, cap2=–1, cap3=–1.

– La ***crearea*** fiecăreia dintre cozile **a** sau **b**, înregistrarea unui caracter citit se face numai dacă acel caracter este *diferit* de caracterele de *sfârşit de linie* sau dacă acea coadă nu *este plină* (nu s-au ocupat încă cele 20 de locuri rezervate elementelor ei).

– Pentru a ***muta*** coada **a** în **c**, în vederea alipirii ulterioare a cozii **b**, se face o prelucrare simplă de ***copiere*** a elementelor din **a** în **c**, aşa cum s-a practicat în mod obişnuit la tablouri până acum (a[1] →c[1], a[2]→c[2], etc.). Pe măsură ce copierea se produce, indicele cap1 scade până ajunge la valoarea –1, simulând procesul de ***extragere*** de elemente din **a**, până la starea de coadă vidă pentru lista **a**. În continuare, pe acelaşi principiu, se mută coada **b**).

Exemplu:

```
// concat_coada;
#include <iostream.h>
#include<conio.h>
const nmax=20;

typedef char coada[nmax];
coada a,b;
int cap1,cap2,cap3=-1,sf1,sf2,sf3=-
1,i;
char c[2*nmax],ch[2];
void main ()
{cap1=-1; sf1=-1;
clrscr();
cout<<"Dati continutul primei cozi ";
ch[1]=getche();
while(ch[1]!=13&&ch[1]!=10&&sf1<nmax-1)
      {
        a[++sf1]=ch[1];ch[1]=getche();
      }
if(sf1!=-1)cap1=0;
cap2=-1; sf2=-1;
cin.get(ch,1,'\n');
//golirea buffer-ului de caractere albe
cout<<"\nDati continutul cozii a doua";
ch[1]=getche();
while(ch[1]!=13&&ch[1]!=10&&sf2<nmax-1)
      {
        b[++sf2]=ch[1];ch[1]=getche();
      }
cin.get(ch,1,'\n');
if(sf2!=-1)cap2=0;
if(sf1>-1) {for (;cap1<=sf1;cap1++)
                c[cap1]=a[cap1];
            cap3=0;sf3=sf1;cap1=-1;
           }
if(sf2>-1){for (;cap2<=sf2;cap2++)
                c[sf3+cap2+1]=b[cap2];
           sf3+=sf2+1;sf2=-1;cap2=-1;
          }
cout<<"\nCoada rezultata:"<<endl;
for (i=0;i<=sf3;i++)
       cout<<c[i]<<" ";
getch();
}
```

observă

Pentru a se executa programul de mai jos, utilizatorul trebuie să introducă caracterele fiecărei liste *unul după altul, în continuare*, iar abia la epuizarea unei liste să tasteze **ENTER**.

rezolvă

1. Dându-se o coadă, *a*, să se elimine un element cerut de către utilizator.
   Indicaţie: Pentru eliminarea unui element dintr-o listă de tip coadă, acesta se cere utilizatorului şi se verifică, prin consultarea listei, Dacă nu există, se afişează un mesaj corespunzător. Dacă elementul există pe un loc **k**, atunci se descarcă restul elementelor din coadă într-o listă auxiliară, **aux**, inclusiv elementul de pe locul **k**. Indicele cap al listei originale nu se modifică, în schimb indicele de sfârşit va avea valoarea **k-1**. Se vor ataşa înapoi la coada iniţială elementele salvate în coada auxiliară, mai puţin primul intrat în **aux**, care este elementul ce trebuie eliminat.
2. Realizaţi operaţiile de creare, adăugare şi ştergere pentru coada alocată acum înlănţuit, urmărind eliberarea locurilor cu ajutorul vectorului **L**, utilizând cele învăţate la liste.

Exerciții și probleme propuse

1) Se presupune lista simplu-înlănţuită **grup**, pentru care variabila **p** reţine indicele primului element. Dacă la sfârşitul executării secvenţei
```
r=grup[p].urm; while(r!=p && r!=-1) r=grup[r].urm;
```
valoarea variabilei **r** este –1 atunci lista:
a) este incorect construită; **b)** este vidă; **c)** are cel puţin două elemente; **d)** nu este circulară.

2) Se consideră lista **grup** ca fiind de tip coadă, pentru care variabilele **p** şi **u** reţin indicii de început, respectiv de sfârşit. Variabila **r** desemnează un indice oarecare din listă. Care dintre următoarele operaţii reprezintă o adăugare corectă în listă:
a) `r=grup[u].urm;u=r;` **b)** `grup[u].urm=r; u=r;`
c) `grup[r].urm=p;p=r;` **d)** `grup[r].urm=u; u=r;`

3) Se consideră lista **grup** ca fiind de tip stivă, pentru care variabila **p** reţine indicele vârfului. Variabila **r** desemnează un indice de lucru. Care dintre următoarele operaţii reprezintă o extragere corectă din listă:
a) `L[p]=1;p=grup[p].urm;` **b)** `r=p;p=grup[p].urm;L[r]=1;`
c) `r=grup[p].urm;p=r;L[p]=1;` **d)** `r=grup[p].urm; p=r;L[r]=1;`

4) Ştiind că lista simplu-înlănţuită **grup** are patru elemente şi adresa (indicele) primului element este memorată în variabila **p**, atunci adresa penultimului element este localizată prin:
a) `grup[grup[p].urm].urm;` **b)** `grup[grup[grup[p].urm].urm].urm;`
c) `grup[p].urm.urm;` **d)** `r=grup[p].urm;r=grup[r].urm;`

5) În lista **grup**, elementele aflate la adresele **q** şi **r** sunt consecutive în listă dacă şi numai dacă:
a) `grup[q].urm==grup[r].urm;` **b)** `grup[r].urm==q && grup[q].urm===r;`
c) `grup[q].urm==r||grup[r].urm==q;` **d)** `r==q;`

6) Ştiind că în lista simplu-înlănţuită **grup** variabila **p** reţine indicele (adresa) primului element şi că expresia **grup[grup[grup[p].urm].urm].urm** are valoarea –1, stabiliţi numărul de componente din care este formată lista.

Probleme propuse pentru portofoliu

1) Se consideră o cutie în care sunt aşezate cărţi, una peste alta, în ordinea alfabetică a titlului. Cutia poate cuprinde maximum 20 de cărţi. O persoană doreşte să trimită un colet cu aceste cărţi, dar a uitat să mai pună o carte. Ce operaţii va realiza persoana pentru a pune cartea în cutie fără a deranja ordinea alfabetică a cărţilor? Realizaţi un program pentru aceste operaţii.

2) La un depou s-a organizat o garnitură de tren cu vagoanele numerotate începând cu 1, de la locomotivă. În ultimul moment, când locomotiva a pornit deja pe linia de ieşire, s-a observat că lipseşte vagonul *k*. Ce operaţii trebuie făcute pentru a se organiza trenul complet, fără a manevra locomotiva? Construiţi un program care codifică datele şi operaţiile respective.

3) La o firmă de turism care organizează excursii pe anumite trasee s-a primit informaţia că un anume oraş din lista traseelor este în carantină deci nu poate fi vizitat. Ce decizie ia firma respectivă? Realizaţi un program care să modifice un traseu dat eliminând oraşul în carantină, dacă respectivul traseu îl conţine.

4) La un depou există o cale ferată de forma alăturată. Pe linia de intrare se află *n* vagoane, numerotate de la 1 la *n*, aşezate în dezordine. Să se construiască un program care mută vagoanele de pe linia de intrare pe linia de ieşire, ajutându-se de linia *S*, astfel încât la ieşire vagoanele să fie în ordinea naturală, de la 1 la *n*. Operaţiile permise sunt: mutarea unui vagon de pe linia de intrare pe linia *S*; mutarea unui vagon de pe linia *S* pe linia de ieşire. Dacă aranjamentul vagoanelor pe linia de intrare nu permite formarea garniturii la ieşire, se va afişa un mesaj corespunzător.

5) Să se organizeze două liste de tip coadă, *P* şi *Q*, în care în fiecare element se înregistrează datele câte unui monom al fiecăruia dintre polinoamele *P*(*X*), respectiv *Q*(*X*). Gradul maxim al polinoamelor este

$n = 20$. Să se utilizeze listele create pentru a scrie un program care efectuează suma celor două polinoame, pe baza algoritmului descris în Capitolul 3 şi să se afişeze polinomul sumă.

6) Pentru aceleaşi date descrise în problema 5, scrieţi un program prin care se construieşte lista de tip coadă a polinomului produs între $P(X)$ şi $Q(X)$.

7) Construiţi o listă simplu-înlănţuită, *M*, în care să se înregistreze elementele unei matrice rare. O matrice rară este o matrice de dimensiune mare care conţine zerouri în proporţie de peste 70% din totalul elementelor. Din acest motiv, nu este economic ca ea să fie înregistrată în memorie ca structură de matrice. Lista propusă, *M*, va conţine informaţiile despre elementele diferite de zero: linia, coloana şi valoarea elementului. Pe acelaşi principiu, construiţi o listă *N*, pentru o a doua matrice şi realizaţi programul care adună cele două matrice rare memorate în listele *M* şi *N*.

Capitolul 5 Elemente de teoria grafurilor

În acest capitol veţi învăţa despre:

- Graful ca model matematic care să reprezinte reţeaua din realitate
- Tipuri de grafuri şi proprietăţile asociate acestora
- Metode de memorare a informaţiilor unui graf
- Aplicaţii practice în care intervin grafuri

5.1. Scurt istoric al teoriei grafurilor

Jocurile şi amuzamentele matematice au fost punctul de plecare în ceea ce astăzi numim „teoria grafurilor". Dezvoltându-se la început paralel cu algebra, această ramură a ştiinţei a căpătat în timp formă şi conţinut propriu, devenind un tot unitar bine conturat şi bine fundamentat teoretic, cu largă aplicare practică.

Printre primii care s-au ocupat de acest domeniu au fost **König** şi **Berge**. Aceştia au stabilit primele noţiuni de limbaj specific domeniului.

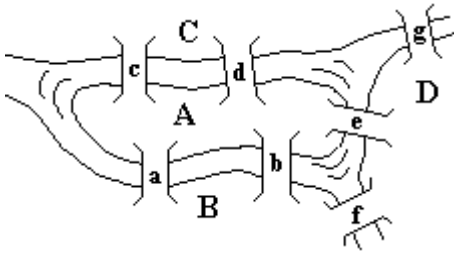


Figura 5.1

„Data naşterii" teoriei grafurilor poate fi considerată anul 1736, când matematicianul elveţian **Leonhard Euler** a publicat în revista *Comentarii Academiae Scientiarum Imperialis Petropolitanae* un articol în limba latină (*Solutio problematis ad geometriam situs pertinentis* – Soluţia unei probleme legate de geometria poziţiei) în care a clarificat „problema celor şapte poduri", stabilind astfel o metodă pentru rezolvarea unei întregi clase de probleme.[1]

Concret, problema este următoarea: râul Pregel împarte oraşul Köenigsberg (astăzi Kaliningrad), prin care trece, ca în figura 5.1.

Porţiunile de uscat, notate A, B, C şi D sunt unite între ele prin şapte poduri notate a, b, c, d, e, f şi g. Întrebarea pe care şi-a pus-o Euler a fost dacă este posibil ca, plecând dintr-un punct, să se poată trece pe toate podurile, câte o singură dată, revenindu-se în final în punctul de plecare.

Dacă se figurează porţiunile de uscat prin cerculeţe şi podurile ca legături între acestea, se obţine schiţa din figura 5.2, care este, de fapt, un graf.

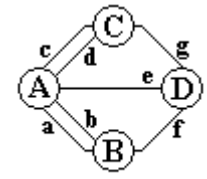


Figura 5.2

Ca izvoare ale teoriei grafurilor mai pot fi considerate: *fizica* – studiul reţelelor electrice –, *geografia* – problema colorării hărţilor cu cel mult patru culori –, *chimia* – principalul iniţiator fiind Cayley etc.

Bazându-se pe noţiuni care astăzi fac parte din domeniul teoriei grafurilor, fizicianul **Kirchoff** a studiat reţelele electrice contribuind în mod decisiv la dezvoltarea teoriei electricităţii (în 1845 a formulat legile care guvernează circulaţia curentului într-o reţea electrică, iar în 1847 a arătat cum poate fi construită într-un graf o mulţime fundamentală de cicluri).

Teoria grafurilor[2] este o ramură destul de nouă a teoriei mulţimilor, care s-a dovedit foarte utilă şi cu aplicaţii în domenii variate: economie, chimie organică, organizare, psihologie, anumite domenii ale artei etc. Grafurile

[1] Articolul lui Euler a fost tradus şi publicat în revista *Nouvelles Annales de Mathematiques* în anul 1851, făcând astfel posibilă dezvoltartea acestui domeniu. Fără a avea cunoştinţă de această lucrare, matematicianul **Carl Hierholzer** a demonstrat în 1873 unele rezultate pe care Euler le considerase evidente.

[2] În 1936, la Leipzig, matematicianul maghiar **Dénes König** publică prima carte de teoria grafurilor în care, în semn de preţuire faţă de contribuţia lui Euler, autorul denumeşte unele noţiuni legate de grafuri cu numele acestuia: graf eulerian, lanţ (ciclu) eulerian ş.a.

Derivând din termenul „notaţie grafică" din chimie, în 1978, apare pentru prima dată termenul de *graf* în sensul său actual, într-un articol publicat în primul număr al revistei *American Journal of Mathematics* de către matematicianul **J. Sylvester** (prieten al lui Cayley).

oferă cele mai potrivite metode de a exprima relaţii între obiecte, de aceea aria lor de utilizare practică este foarte vastă, de la economie la psihologie socială.

Relaţia graf–reţea

În scopul familiarizării cu diversele domenii de aplicare, prezentăm câteva probleme „clasice" a căror rezolvare implică noţini legate de teoria grafurilor. Din analiza acestora, se constată că noţiunea de **graf** utilizată în stabilirea aspectelor teoretice este ceea ce se desemnează în practică prin noţiunea de **reţea**.

Reţea rutieră

Se doreşte construirea unei şosele între două localităţi notate cu **0** şi, respectiv **7**, care ar putea să treacă prin localităţile **1**, **2**, ..., **6**. Cunoscând costul lucrării pentru fiecare dintre tronsoanele ce leagă două localităţi, trebuie să se determine traseul şoselei între localităţi, astfel încât costul general al lucrării să fie cât mai mic. Dacă localităţile se figurează prin cercuri, iar porţiunile de şosea prin linii, se obţine figura 5.3, care este, de asemenea, un graf.

Ieşirea din labirint

Labirinturile sunt construcţii străvechi, primele referiri la ele întâlnindu-se în mitologie. Mai târziu s-au construit labirinturile din garduri vii, în grădinile şi parcurile curţilor imperiale, pentru amuzament, dar şi pentru frumuseţea lor. Indiferent din ce materiale au fost construite, acestea presupun existenţa unui număr de culoare separate între ele, care se întâlnesc în anumite puncte. Studiul lor, început la sfârşitul secolului al XIX-lea, a demonstrat că unui labirint i se poate asocia un graf, în care vârfurile corespund intersecţiilor, iar muchiile, culoarelor labirintului. O problemă imediată ar fi aceea de a găsi ieşirea din labirint, pornind dintr-un punct al său şi parcurgând un traseu care să fie, eventual, cât mai scurt.

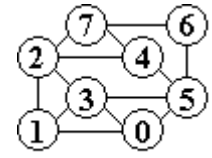


Figura 5.3

Competenţă

Trei muncitori trebuie repartizaţi să lucreze pe trei maşini. Se cunoaşte randamentul fiecărui muncitor pe fiecare maşină în parte şi se doreşte stabilirea unei repartizări a muncitorilor pe fiecare maşină, astfel încât să se obţină un maximum de randament. Notând cu **1**, **2** şi **3** cei trei muncitori, cu **a**, **b** şi **c** cele trei maşini şi cu linii drepte posibilităţile de asociere dintre fiecare muncitor şi fiecare maşină, se obţine reprezentarea din figura 5.4.

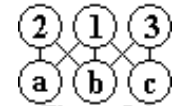


Figura 5.4

Dacă muncitorul nu are calificarea necesară pentru a putea lucra pe maşina respectivă, legătura nu este trasată, (de exemplu, legăturile 2–c şi 3–a lipsesc).

Problema protocolului

La un dineu oficial participă $2n$ persoane, fiecare dintre acestea având printre invitaţi cel mult $n-1$ persoane cu care nu este în relaţii de prietenie. Pentru reuşita întâlnirii, organizatorii trebuie să aşeze la masă fiecare persoană, astfel încât aceasta să aibă ca vecini numai persoane cu care se află în relaţii bune.

Problema datoriilor

Într-un grup sunt mai multe persoane care au, unele faţă de altele, diverse datorii. Pentru achitarea datoriilor, fiecare persoană poate plăti sumele datorate, urmând să primească, la rândul său, sumele cuvenite de la datornici. Procesul se poate bloca dacă există persoane care nu dispun de întreaga sumă pe care o datorează altora, deşi suma pe care o posedă ar acoperi diferenţa între suma datorată şi cea cuvenită. În cazul în care sumele disponibile sunt suficiente, ar trebui să se stabilească subgrupurile de persoane care au datorii reciproce, pentru reglementarea datoriilor în cadrul subgrupului.

În exemplele date, prin folosirea unor puncte şi a unor legături între acestea, figurate prin segmente, problemele din practică s-au transformat în probleme ce ţin de teoria grafurilor.

5.2. Definiţie şi clasificare

a) Graf

Se consideră mulţimile finite $\boldsymbol{X} = \{x_1, x_2, \ldots, x_n\}$ şi mulţimea $U = X \times X$ (produsul cartezian al mulţimii X cu ea însăşi).

Se numeşte **graf** *o pereche ordonată de mulţimi* (X,U),
unde:
– X *este o mulţime finită şi nevidă de elemente numite* **vârfuri** *sau* **noduri;**
– U *este o mulţime de perechi (submulţimi cu două elemente) din mulţimea* X, *numite* **muchii** *sau* **arce**.

Mulţimea X *se numeşte* **mulţimea vârfurilor** *sau* ***nodurilor*** *grafului, iar mulţimea* U, **mulţimea muchiilor** *sau* **arcelor**.

Un graf va fi notat cu $G = (X,U)$; pentru reprezentarea sa, vârfurile (nodurile) sunt desemnate prin numere sau litere, muchiile prin linii neorientate şi arcele prin linii orientate.
Această caracteristică a grafurilor, de reprezentare intuitivă a noţiunilor, face ca problemele de teoria grafurilor să fie studiate cu plăcere de către elevi.

b) Clasificare

Mulţimea U *are proprietatea de* **simetrie**, *dacă şi numai dacă având* $[x_i,x_k] \in U$ *rezultă şi* $[x_k,x_i] \in U$.

Dacă mulţimea U *are proprietatea de simetrie, graful* $G = (X,U)$ *este* ***graf*** **neorientat**.

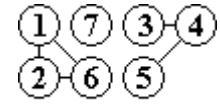


Figura 5.5

Pentru $G = (X,U)$ graf *neorientat*, o muchie **u** se notează $[x_i,x_k]$ şi reprezintă o pereche *neordonată* de vârfuri distincte din U (adică $x_i, x_k \in X$ şi $x_i \neq x_k$).


exemplu

Fie: $X = \{1, 2, 3, 4, 5, 6, 7\}$ şi $U = \{[1, 2], [1, 6], [2, 6], [3, 4], [4, 5]\}$.
Graful $G = (X,U)$, reprezentat în figura 5.5, este un graf neorientat cu şapte vârfuri şi cinci muchii.

Dacă mulţimea U *nu are proprietatea de simetrie, se spune că graful* $G = (X,U)$ *este* ***graf*** **orientat** *sau* **direcţionat** *sau* **digraf**.

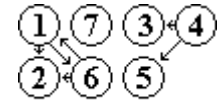


Figura 5.6

Dacă $G = (X,U)$ este un graf *orientat*, muchiile se numesc ***arce***; un arc **u** se notează cu (x_i,x_k) şi este o pereche *ordonată* de vârfuri distincte din U (adică $x_i, x_k \in X$ şi $x_i \neq x_k$).


exemplu

Fie $X = \{1, 2, 3, 4, 5, 6, 7\}$ şi $U = \{(1, 2), (1, 6), (4, 3), (4, 5), (6, 1), (6, 2)\}$.
Graful $G = (X,U)$, reprezentat în figura 5.6, este un graf orientat, cu şapte vârfuri şi şase arce.


observă

Se poate spune că *un graf orientat este un caz particular al grafului neorientat*, fiecare muchie având un sens precizat sau că un graf neorientat este un graf în care fiecare muchie între cele două noduri considerate condensează două muchii de sens opus.

Dacă $U = \emptyset$ *(mulţimea vidă), atunci graful* $G = (X,U)$ se *numeşte* **graf nul** *şi reprezentarea lui în plan se reduce la figurarea unor puncte izolate, care reprezintă nodurile.*

Este evident caracterul particular a unui astfel de graf, studiul său neprezentând interes.

5.3. Grafuri neorientate

5.3.1. Noţiuni de bază

a) Adiacenţa, incidenţa

Vârfurile x_i şi x_k se numesc ***extremităţile*** muchiei **u** şi se spune că sunt **adiacente**.
Dacă un vârf nu este extremitatea nici unei muchii, atunci el se numeşte vârf **izolat**.
Considerând muchia **u**, notată $[x_i,x_k]$, se spune că vârfurile x_i şi x_k sunt **incidente** cu muchia $[x_i,x_k]$.
Muchiile care au o extremitate comună se spune că sunt **incidente**.
Uneori, prin abuz de limbaj, se mai foloseşte notaţia $[x_i,x_k] \in U$.

b) Gradul vârfului

Se defineşte **gradul** *unui vârf* **x** *al unui graf* G = (*X,U*) *ca fiind numărul muchiilor incidente cu* **x** *şi se notează cu d(x)* (în limba franceză *dégre* = *grad*).

Dacă un vârf are gradul 0 (*nu există nicio muchie incidentă cu el*), *atunci el se numeşte vârf* **izolat**, *iar dacă are gradul* 1 (*este incident cu o singură muchie*) *se numeşte vârf* **terminal**.

În graful din figura 5.7, b) vârfurile au, respectiv, gradele:
– vârful 1, gradul 2;
– vârful 2, gradul 2;
– vârful 3, gradul 0 (vârf izolat);
– vârful 4, gradul 3;
– vârful 5, gradul 2;
– vârful 6, gradul 1 (vârf terminal).

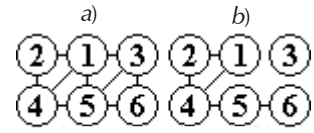


Figura 5.7

Scrieţi tabelul gradelor vărfurilor pentru graful din figura 5.7, *a*).

Proprietate

Fie graful G = (*X,U*) *cu n vârfuri* $x_1, x_2, ..., x_n$ *şi m muchii.*

Atunci suma gradelor nodurilor grafului este 2m, adică: $\sum_{k=1}^{n} d\left(x_k\right) = 2m$.

Demonstraţie. Este evident, fiecare muchie contribuind cu o unitate la gradul unui vârf x, deci cu două unităţi la suma totală.

Corolar. *Fiecare graf are un număr par de vârfuri cu grad impar.*

Demonstraţie

Notând cu S_1 şi S_2 suma tuturor gradelor impare, respectiv pare ale vârfurilor grafului G şi observând că S_2 şi suma totală sunt pare, rezultă că diferenţa dintre ele, adică S_1, este un număr par. Stiind că S_1 este o sumă de numere impare, rezultă că are un număr par de termeni.

c) Relaţia între două grafuri – graf parţial, graf complementar şi subgraf

Fie graful G = (*X,U*) *şi* $V \subseteq U$.

Graful G_p = (*X,V*) *se numeşte* **graf parţial** *al grafului* G = (*X,U*).

Un graf parţial se poate obţine păstrând toate nodurile, dar eliminând o parte din muchiile grafului iniţial. (Dacă *V* = *U*, atunci G_p coincide cu G).

Fie graful cu *X* = {1, 2, 3, 4, 5, 6} şi *U* = {[1, 2], [1, 3], [1, 4], [1, 5], [2, 4], [3, 5], [3, 6], [4, 5], [5, 6]} reprezentat în figura 5.7, *a*). Eliminând muchiile [1, 3], [1, 5], [3, 5] şi [3, 6] se obţine G_p = (*X,V*) – graf parţial al grafului dat – cu aceeaşi mulţime de vârfuri *X* = {1, 2, 3, 4, 5, 6} şi cu muchiile *V* = {[1, 2], [1, 4], [2, 4], [4, 5], [5, 6]} (fig. 5.7, *b*).

Fie graful G = (*X,U*) *şi* $Y \subseteq X$. *Fie* $V \subseteq U$, *unde V conţine toate muchiile din U care au ambele extremităţi în Y. Graful* H = (*Y,V*) *se numeşte* **subgraf** *al grafului* G = (*X,U*).

Se spune că subgraful H *este* ***indus*** *sau* ***generat*** *de mulţimea de vârfuri Y.*

Folosind graful din figura 5.7, *a*), considerând *Y* = {1, 2, 4, 5} şi *U* = {[1, 2], [1, 4], [1, 5], [2, 4], [4, 5]}, se obţine subgraful *H* = (Y,V), din figura 5.8.

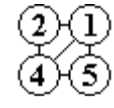


Figura 5.8

Se poate spune că un subgraf se obţine prin suprimarea într-un graf a anumitor vârfuri şi a tuturor muchiilor adiacente acestora.

d) Graf complet

Se consideră $U = \{[x_i,x_j] \mid x_i, x_j \in X, \forall i, \forall j, i \neq j\}$. *Atunci graful* $G = (X,U)$ *se numeşte* **graf complet** *şi se notează* $\mathbf{K_n}$ (*n fiind numărul de vârfuri ale grafului*).

exemplu

Graful din figura 5.9 este un graf complet. Se notează K_5.

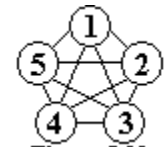


Figura 5.9

observă

Un graf complet este un graf care are proprietatea că oricare două noduri diferite sunt adiacente. Pentru graful neorientat, graful complet este unic.

Proprietate. *Pentru un graf neorientat cu n vârfuri, graful* K_n *are* C_n^2 *muchii.*

Demonstraţie. Considerîndu-se faptul că între oricare două noduri există muchie, atunci sunt atâtea muchii câte combinări de *n* noduri luate câte două există.

e) Graf bipartit

Graful $G = (X,U)$ *se numeşte* **bipartit** *dacă există două mulţimi nevide A şi B cu* $X = A \cup B$, $A \cap B = \emptyset$ *şi orice muchie* ***u*** *a sa are o extremitate în A şi cealaltă în B.*

exemplu

Graful din figura 5.10 este un graf bipartit cu $A = \{1, 2, 3\}$ şi $B = \{4, 5\}$.

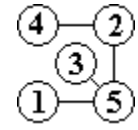


Figura 5.10

Graful $G = (X,U)$ *bipartit se numeşte* **bipartit complet** *dacă între orice vârf* x_i *din A şi orice vârf* x_k *din B există muchia* $[x_i,x_k]$.

Dacă *A* are *p* elemente, iar *B* are *q* elemente, atunci numărul total de muchii ale grafului este evident $p \cdot q$. Acest graf bipartit complet se va nota $\mathbf{K_{p,\,q}}$.

exemplu

1. În figura 5.11, este desenat un graf $\mathbf{K_{3,\,2}}$ cu $A = \{1, 2, 3\}$ şi $B = \{4, 5\}$.
2. Având două mulţimi nevide, *A* şi *B*, cu *n*, respectiv *m* elemente, numărul de funcţii care se pot defini pe *A* cu valori în *B* este m^n.

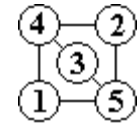


Figura 5.11

f) Costul unui graf, funcţia cost

În practică se întâlnesc foarte des probleme de tipul următor: se doreşte conectarea mai multor consumatori la o sursă de energie electrică astfel încât costul branşării să fie minim.

Transpunând problema în termenii teoriei grafurilor, se va organiza un graf neorientat în care fiecare muchie va avea o valoare reprezentând costul investiţiei.

Suma costurilor muchiilor unui graf se numeşte **costul** *grafului.*

Dacă se defineşte funcţia: $c: U \to R_+$ care asociază fiecărei muchii un număr real numit **cost**, costul grafului este: $c(G) = \sum_{u \in U} c(u)$. *Funcţia c se numeşte* **funcţia cost**.

exemplu

Fie graful din figura 5.12, cu nouă vârfuri şi paisprezece muchii. Lista muchiilor grafului şi a costurilor asociate, este:

Muchia	[1,2]	[1,5]	[1,6]	[2,3]	[2,6]	[3,4]	[3,6]	[3,8]	[4,8]	[4,9]	[5,6]	[6,7]	[7,8]	[8,9]
Cost	4	5	1	2	5	3	3	2	6	7	3	1	4	5

Costul acestui graf se determină prin însumarea costurilor fiecărei muchii. Se obţine $c(G) = 51$.

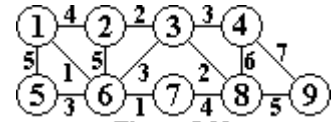


Figura 5.12

Exerciţii rezolvate

1. Fiind dată mulţimea de vârfuri $X = \{x_1, x_2, ..., x_n\}$, să se determine numărul total de grafuri neorientate ce se pot forma cu aceste vârfuri.

Rezolvare. Reamintim un rezultat stabilit la matematică: având două mulţimi A şi B cu n şi, respectiv m elemente, între A şi B se pot defini m^n funcţii.

Având graful $G = (X,U)$ cu n vârfuri pentru fiecare submulţime$\{x_i,x_j\}$, se poate defini funcţia:

$$f : U \to \{0,1\} \text{ astfel: } f(\{x_i,x_j\}) = \begin{cases} 1, & \text{pentru } (x_i, x_j) \in U \\ 0, & \text{pentru } (x_i, x_j) \notin U \end{cases}$$

Aşadar, ştiind că U are $\frac{n(n-1)}{2}$ elemente, iar mulţimea $\{1, 2\}$ are două elemente, se obţine că numărul total de funcţii este $2^{\frac{n(n-1)}{2}}$. Cum fiecărei muchii i s-a ataşat o funcţie, şi reciproc, se poate spune că numărul total de grafuri neorientate ce se pot forma cu n vârfuri este $2^{\frac{n(n-1)}{2}}$.

2. Se consideră o tablă de şah pe care se deplasează un cal, sărind în L, astfel încât să ocupe o singură dată fiecare dintre cele n^2 poziţii. Să se cerceteze dacă este posibil ca, după parcurgerea tablei, calul să se poată întoarce în poziţia din care a pornit.

Rezolvare. Se poate considera că mutările calului reprezintă un graf unde două vârfuri sunt adiacente dacă există o săritură de la unul la celălalt. Deoarece la fiecare „săritură" calul trece de pe o poziţie de o culoare pe o poziţie de cealaltă culoare, rezultă că graful este bipartit.

Dacă n este **par** cele două subulţimi vor avea număr egal de elemente, deci problema poate avea soluţie; în caz contrar, când n este impar, nu există soluţie.

3. Toate cele n clase ale unei şcoli îşi trimit câte un reprezentant în Consiliul elevilor. Să se calculeze cât timp este necesar pentru ca fiecare participant să-i salute pe toţi ceilalţi, ştiind că pentru a se saluta două persoane este nevoie de două minute.

Indicaţie. Persoanele se pot reprezenta prin vârfurile grafului complet K_n, iar saluturile simultane prin muchii care au două câte două extremităţi comune.

Caracteristicile grafurilor neorientate

I. Într-un **graf orientat** adiacenţele se numesc **muchii**.
II. În graful orientat $G=(X,U)$, U **are** are proprietatea de **simetrie**.
III. Dacă $U=\emptyset$ (mulţimea vidă), atunci graful $G=(X,U)$ se numeşte **graf nul** şi reprezentarea lui în plan se reduce la figurarea unor puncte izolate.
IV. **Gradul** unui vârf **x** reprezintă numărul muchiilor incidente cu **x.**
V. În matricea de adiacenţă **semisuma** elementelor unei **linii** reprezintă gradul **exterior.**
VI. **Suma gradelor** nodurilor unui graf cu n vârfuri şi m muchii este $2m$.
VII. Un **graf parţial** se poate obţine păstrând toate nodurile, dar eliminând o parte din muchii.
VIII. Un **subgraf** se obţine prin suprimarea într-un graf a anumitor vârfuri şi a tuturor muchiilor adiacente acestora.
IX. Dacă oricare două vârfuri ale unui graf sunt adiacente, graful se numeşte **complet**.
X. Fie graful $G = (X,U)$. Dacă există două mulţimi nevide A şi B cu $X = A \cup B$, $A \cap B = \emptyset$ şi orice muchie u a sa are o extremitate în A şi cealaltă în B, atunci graful se numeşte **bipartit**.
XI. Asociind fiecărei muchii un număr real ce reprezintă „costul" acesteia, se poate calcula **costul** grafului ca sumă a costurilor muchiilor.

rezolvă

1. Care este numărul muchiilor grafului bipartit complet $K_{p,q}$, pentru $p = 16$ şi $q = 14$:
a) 244; **b)** 124; **c)** 412; **d)** 224.

2. Stabiliţi corespondenţa corectă între fiecare element din coloana A şi proprietatea corespunzătoare din coloana B.

A	B
a) nod izolat; b) graf nul; c) adiacenţă; d) incidenţă; e) grad; f) subgraf; g) nod terminal; h) graf parţial.	a) numărul muchiilor incidente într-un nod; b) este incident cu o singură muchie; c) se obţine prin eliminarea unor muchii; d) nu există muchii incidente cu ele; e) se obţine prin eliminarea tuturor nodurilor; f) o mulţime de noduri izolate; g) nodurile extreme şi muchia corespunzătoare; h) există muchie între două noduri; i) se obţine prin eliminarea unor noduri şi a muchiilor adiacente.

3. Fie un graf G = (*X*,*U*), cu *n* noduri şi *m* muchii; suma gradelor nodurilor sale este:
a) $m + n$; **b)** $2n$; **c)** $m*n$; **d)** $2m$.

4. Un graf complet cu n noduri are proprietatea că:

a) nu există noduri izolate; **b)** are $\frac{n(n-1)}{2}$ muchii; **c)** are $n*n$ muchii;

d) oricare două noduri sunt adiacente; **e)** nu există noduri terminale.

5. Se consideră un graf cu $n = 5$ noduri şi $U = \{[1,2], [1,3], [2,3], [2,4], [3,4], [4,5]\}$. Eliminând muchiile [1,2] şi [3,4], se obţine:
a) un subgraf; **b)** un graf parţial; **c)** un graf parţial ca subgraf al lui G.

6. Se consideră graful neorientat pentru care $X = 6$ şi $U = \{[1,6], [2,3], [2,5], [3,4], [4,5], [4,6]\}$. Stabiliţi numărul minim de muchii care trebuie adăugate pentru ca graful să devină complet.

7. Stabiliţi câte subgrafuri pot rezulta dintr-un graf complet cu n = 4 noduri.

8. Stabiliţi câte grafuri parţiale pot rezulta dintr-un graf complet cu n = 4 noduri.

9. Identificaţi care dintre secvenţele următoare reprezintă şirul gradelor nodurilor unui graf complet:
a) 1 2 3 4; **b)** 5 5 5 5 5; **c)** 4 4 4 4 4; **d)** 1 2 1 2 1 2.

10. Care este numărul minim de muchii care pot fi plasate într-un graf neorientat cu 65 de noduri astfel încât să nu existe nici un nod izolat?
a) 32; **b)** 64; **c)** 33; **d)** 63.

11. Să se construiască programul pentru citirea valorilor variabilelor *n* – numărul de vârfuri, *m* – numărul de muchii, gr_1, gr_2, … , gr_{n-2} – gradele a $(n – 2)$ noduri şi calcularea gradelor celorlalte două noduri rămase, dintre care unul este nod terminal.

12. Fie G = (*X, U*) un graf neorientat dat cu *n* vârfuri. Care dintre următoarele afirmaţii sunt adevărate.

a) Numărul de muchii ale grafului este egal cu $\frac{n(n-1)}{2}$;

b) Este posibil ca vârfurile grafului să aibă toate gradele distincte două câte două (să nu există două vârfuri cu acelaşi grad);

c) Suma gradelor vârfurilor este egală cu dublul numărului de muchii;

d) Mulţimea vârfurilor grafului poate fi vidă.

rezolvă

13. Fie graful G = (*X*,*U*) cu *n* = 9 vârfuri şi zece muchii, definit astfel:
X = {1,2,3,4,5,6,7,8,9} şi *U* = {[1,2], [1,3], [2,3], [3,4], [3,9], [4,9], [5,6], [5,7], [5,8], [6,8]}.
a) Stabiliţi dacă graful este bipartit.
b) Stabiliţi suma gradelor vârfurilor.
c) Stabiliţi dacă graful este complet.

14. Gradul maxim pe care îl poate avea un nod într-un graf neorientat cu *n* noduri este:
a) *n*/2; **b)** *n* – 1; **c)** *n*; **d)** 2.

15. Fie G = (*X*,*U*) un graf neorientat dat, cu n vârfuri. Care dintre următoarele afirmaţii sunt adevărate?

16. Un graf neorientat are 80 de noduri şi 80 muchii. Numărul de noduri izolate este cel mult:
a) 90; **b)** 67; **c)** 10; **d)** 66.

17. Pentru un număr *n*, natural nenul şi diferit de 1, se cere să se deseneze un graf neorientat cu proprietatea că gradul oricărui vârf este par.

5.3.2. Metode de reprezentare a grafurilor neorientate

a) Cea mai facilă metodă de memorare a unui graf neorientat şi care reflectă mulţimea *U* din definiţia grafului este **matricea de adiacenţă** sau **matricea asociată** grafului G = (*X*,*U*) $A = (a_{ij})$, definită astfel: $a_{ij} = \begin{cases} 1, \text{ pentru} [x_i, x_j] \in U \\ 0, \text{ pentru} [x_i, x_j] \notin U \end{cases}$.

Pentru graful neorientat din figura 5.5, matricea de adiacenţă este prezentată alăturat.

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Reflectând informaţiile mulţimii *U*, se observă că matricea de adiacenţă este simetrică, deoarece, pentru o pereche $[x_i,x_j]$, în *U* există şi simetrica ei, $[x_j,x_i]$. Deoarece elementele matricei de adiacenţă pot fi doar valorile 1 sau 0, matricea se mai numeşte şi **booleană**.

Analizând o matricea booleană ataşată unui graf, se observă că linia *i* reprezintă toate muchiile care au extremitatea iniţială în vârful *i*, iar coloana *j* reprezintă muchiile care au extremitatea finală în vârful *j*.

rezolvă

Utilizând matricea de adiacenţă dată ca exemplu, stabiliţi gradul fiecărui vârf al grafului.
Reprezentarea prin matricea de adiacenţe permite un acces rapid la muchiile grafului, fiind astfel utilă în rezolvarea problemelor care implică testarea prezenţei unei anumite muchii şi prelucrarea informaţiei ataşate acesteia.

observă

Metoda este însă dezavantajoasă deoarece, pentru un graf cu *n* vârfuri, numărul de muchii este C_n^2, cu mult mai mic decât n^2, deci memoria necesară pentru a păstra matricea este ineficient utilizată.

b) Deoarece apariţia teoriei grafurilor a fost necesară pentru rezolvarea unor probleme practice, diversitatea şi particularităţile acestora determină necesitatea de memorare eficientă a grafurilor neorientate în modalităţi specifice. În continuare vor fi prezentate câteva dintre cele mai importante.

b.1) Lista de adiacenţe

Această metodă este indicată în cazul în care graful are un număr mare de vârfuri şi un număr mic de muchii.

Principiul acestei metode de memorare constă în:

– Pentru fiecare vârf *k* se alcătuieşte lista vecinilor săi, notată L_k.

– Pentru un graf cu *n* vârfuri şi *m* muchii, se construieşte apoi un vector care conţine adresele celor *n* liste ale vecinilor fiecărui vârf, plus cele *n* liste propriu-zise.

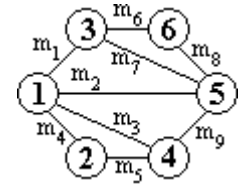


Figura 5.13

Pentru memorarea mai uşoară prin această metodă se va construi o matrice, *T* cu două linii şi *n* + 2*m* coloane. Numărul de coloane este stabilit ţinând seama de faptul că

pentru fiecare nod ale grafului este necesară o coloană (total n coloane) şi pentru fiecare muchie trebuie specificate cele două noduri adiacente cu ea, (încă $2m$ coloane).

Completarea tabloului se face astfel:

1. Prima linie, $T[1,j]$, conţine numere de vârfuri:
- dacă $1 \le j \le n$, atunci $T[1,j] = j$;
- dacă $n + 1 \le j \le n + 2m$, atunci $T[1,j]$ conţine elementele listelor L_k (vîrfuri adiacente).

2. A doua linie, $T[2,j]$, conţine indici ai coloanelor, adresele de legătură în care se continuă o listă:
- dacă $1 \le j \le n$, atunci $T[2,j] = p$, unde p este coloana unde se află primul element din lista L_j.
- dacă $n + 1 \le j \le n + 2m$, atunci:
  - dacă $T[1,j]$ nu este ultimul element al listei L_j, atunci $T[2,j] = p$, unde p este indicele coloanei unde se află elementul ce urmează vârfului $T[1,j]$ în lista lui de adiacenţă;
  - dacă $T[1,j]$ este ultimul element al listei, atunci $T[2,j] = 0$ (adresa finală).

Spre deosebire de reprezentarea prin matrice de adiacenţe, reprezentarea prin liste de adiacenţe foloseşte mai eficient memoria, dar căutarea efectivă a muchiilor este mult mai anevoioasă. Memoria necesară reprezentării este proporţională cu suma dintre numărul de vârfurişi numărul de muchii ale grafului, dar timpul de căutare al unui muchii este proporţional cu numărul de vârfuri (dacă graful are n vârfuri, atunci un vârf poate avea $n - 1$ vârfuri adiacente).

Fie graful din figura 5.14, cu $n = 5$ vârfuri şi $m = 6$ muchii. Pentru fiecare vârf k, lista L_k a vecinilor săi este:

Vârful k	Lista L_k a vârfurilor adiacente cu vârful k
1	2, 3
2	1, 3, 4
3	1, 2, 4, 5
4	2, 3
5	3

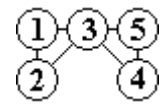


Figura 5.14

Pentru exemplul dat tabelului T (care are 2 linii şi 5 + 2 x 6 = 17 coloane) este:

j	1	2	3	4	5	L_1		L_2			L_3				L_4		L_5
						6	7	8	9	10	11	12	13	14	15	16	17
$T[1,j]$- **vârfuri**	1	2	3	4	5	2	3	1	3	4	1	2	4	5	2	3	3
$T[2,j]$- **adrese de legătură**	6	8	11	15	17	7	0	9	10	0	12	13	14	0	16	0	0

unde, de exemplu:
- $T[2,1]$ este indicele coloanei pe care se află primul element din L_1, deci $T[2,1] = 6$;
- $T[2,2]$ este indicele coloanei în care se află primul element din L_2, deci $T[2,2] = 8$;
- $T[2,6]$: pentru că elementul $T[1,6]$ nu este ultimul element din L_1, $T[2,6]$ este indicele coloanei în care se află elementul următor elementului $T[1,6]$ din L_1, deci $T[\ [2,6] = 7$;
- $T[2,7]$: pentru că elementul $T[1,7]$ este ultimul element din lista L_1, $T[2,7] = 0$;

Pentru a facilita înţelegerea s-au precizat atât indicii coloanelor, cât şi listele vârfurilor.

Parcurgerea listelor utilizând această matrice utilizează înlănţuirea adreselor de tip indice de legătură. De exemplu, pentru nodul 1 ($T[1,1]$), următorul vecin este conţinut în coloana 6 ($T[2,1]$, unde se găseşte nodul 2 ($T[1,6]$). Pentru nodul 2, adresa următorului vecin este coloana 7 ($T[2,6]$), unde se găseşte nodul 3 ($T[1,7]$). Deoarece $T[2,7] = 0$, rezultă că nodul 3 nu mai are vecini nevizitaţi. Se poate repeta raţionamentul pentru fiecare dintre cele cinci noduri ale grafului.

Pentru programul în **C++** se organizează o listă simplu înlănţuită alocată static, de tip coadă, definită astfel:

```
struct element{unsigned nod; int urm;};
element graf[17];
```

Temă de laborator

Construiţi un program care să simuleze metoda de memorare a grafului de mai sus, ultizând o listă simplu înlănţuită alocată static, notată *T*.

Varianta de reprezentare a unui graf G = (*X*,*U*) cu *n* vârfuri şi *m* muchii, creează o matrice *L*, numită ***legături***, cu două linii şi 2*m* coloane, astfel:

1. *L* [1,*j*] se completează cu toate elementele listelor L_j, (corespunzător fiecărui vârf *j* în parte);

2. *L* [2,*j*] se completează astfel:

– dacă *L*[1,*j*] nu este ultimul element al listei L_j, atunci *L*[2,*j*] = *p*, unde *p* este indicele coloanei unde se află elementul ce urmează vârfului *L*[1,*j*] în lista L_j (corespunzător fiecărui vârf *j* în parte);

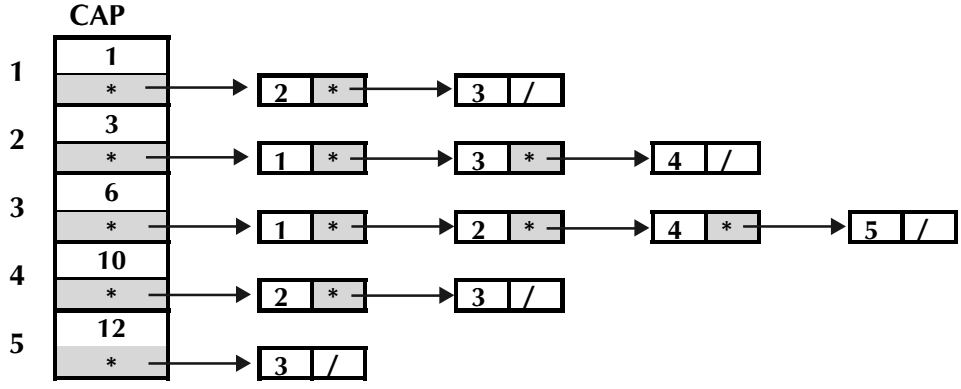
– dacă *L*[1,*j*] este ultimul element al listei lista L_j, atunci *L*[2,*j*] = 0 (corespunzător fiecărui vârf *j* în parte).

Indicele coloanei din *L* care conţine primul element din lista L_j se reţine în coloana *j* a unui tablou unidimensional **CAP**. Raţionamentul este analog cu cel descris pentru tabloul *T*. Aşadar, se obţine:

j	L_1		L_2			L_3				L_4		
	1	2	3	4	5	6	7	8	9	10	11	12
L[1,*j*]	2	3	1	3	4	1	2	4	5	2	3	3
L[2,*j*]	2	0	4	5	0	7	8	9	0	11	0	0

j	1	2	3	4	5
CAP	1	3	6	10	12

Întregul graf s-a reprezentat prin tabloul unidimensional CAP, indexat după noduri, fiecare element *j* al său fiind un indice (adresă de legătură) spre lista *L* a nodurilor adiacente nodului *j*.



Temă de laborator

Din fişierul **noduri.txt** se vor citi:

– de pe prima linie: numărul de noduri – *n***;**

– de pe fiecare dintre următoarele *n* linii, până la întâlnirea marcajului de sfârşit de linie, lista vecinilor fiecărui nod.

Din datele citite se va forma un ansamblu de liste secvenţial înlănţuite alocate static.

Pentru fiecare listă în parte, indicele primului element se memorează în vectorul CAP.

b.2) O altă metodă de reprezentare a unui graf cu *n* noduri şi *m* muchii este aceea care foloseşte *un tablou unidimensional* de înregistrări definit astfel:

```
struct muchie{unsigned x,y;};
muchie u[ct];//ct-constantã, precizeazã spaþiul maxim rezervat
```

Referirea la extremităţile **x** şi **y** ale muchiei **k** se face prin **u[k].x**, respectiv **u[k].y**.

Procedând în acest mod, se ajunge la o înglobare naturală în tipul de date *muchie* şi a altor informaţii asociate muchiei (cost, lungime etc). Este astfel posibilă prelucrarea succesivă a acestor informaţii, după o eventuală modificare a ordinii muchiilor în tabloul *u*.

Temă de laborator

Se citesc din fişierul **muchii.txt**:
– de pe prima linie: numărul de noduri – *n* – şi numărul de muchii **–** *m***;**
– de pe următoarea linie *m* perechi de forma (*i, j*) pentru fiecare muchie.
Să se alcătuiască un program pentru a forma din datele citite matricea de adiacenţă a grafului.

1. Se consideră garful $X = \{1,2,3,4\}$ şi $U = \{[1,2], [1,3], [1,4], [2,3], [3,4]\}$.
Care este matricea sa de adiacenţă?

a)

0	1	1	1
1	0	1	0
1	1	0	1
1	0	1	0

b)

0	1	0	1
1	0	1	0
0	1	0	1
1	0	1	0

c)

0	1	1	1
1	0	1	0
1	1	0	0
1	0	1	0

d)

0	1	1	1
1	1	1	0
1	1	0	1
1	0	1	0

2. Stabiliţi care dintre următoarele variante este matricea de adiacenţă a unui subgraf al grafului. Se consideră graful $X = \{1,2,3,4\}$ şi $U = \{[1,2], [1,3], [2,3]\}$.

a)

0	1	1	0
1	0	1	0
1	1	0	0
0	0	0	0

b)

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

c)

0	1	0	0
1	0	1	0
0	1	1	1
1	0	0	0

d)

0	1	1	1
1	0	1	1
1	0	0	0
1	0	1	0

3. Fiind dată matricea de adiacenţă a unui graf, să se determine:
a) gradul fiecărui vârf; **b)** numărul de muchii; **c)** vârfurile cu grad minim şi cele cu grad maxim.

4. Se consideră matricea de adiacenţă a unui graf neorientat alăturată:
Determinaţi gradele vârfurilor grafului şi stabiliţi care dintre următoarele afirmaţii este adevărată:
a) toate vârfurile au grad par;
b) gradele vârfurilor sunt diferite două câte două;
c) gradele vârfurilor sunt egale două câte două;
d) există un unic vârf cu grad maxim.

0	0	0	0	0
0	0	0	1	1
0	0	0	1	1
0	1	1	0	0
0	1	1	0	0

5. Determinaţi numărul total de grafuri neorientate distincte cu trei noduri (două grafuri se consideră distincte dacă au matrice de adiacenţă diferite).
a) 7; **b)** 8; **c)** 4; **d)** 64.

6. Pentru un graf neorientat, pentru care se cunosc numărul de noduri şi matricea de adiacenţă, să se construiască secvenţa de program pentru fiecare cerinţă de mai jos în parte:
a) verificarea existenţei de vârfuri izolate;
b) verificarea proprietăţii de completitudine;
c) stabilirea numărului de muchii;
d) stabilirea proprietăţii de matrice de adiacenţă a matricei A care s-a citit pentru graf.

7. Să se verifice dacă există un graf neorientat cu 8 noduri şi cu gradele vârfurilor egale cu 1, 7, 3, 3, 5, 6, 4, respectiv 4.

8. Cunoscându-se *n* – numărul de vârfuri – şi *a* – matricea de adiacenţă a unui graf –, alcătuiţi secvenţa de program care determină dacă nodul *x*, citit, are gradul **gr**, citit.

5.3.3. Parcurgerea grafurilor neorientate

Parcurgerea unui graf neorientat indică posibilitatea de a **vizita** o singură dată **fiecare vârf** al grafului, pornind de la un vârf dat x_k şi trecând pe muchii adiacente.

Această operaţiune poartă numele şi de **traversare** a vârfurilor grafului şi este efectuată cu scopul prelucrării informaţiei asociate vârfurilor.

Deoarece graful este o structură neliniară de organizare a datelor, prin parcurgerea sa în mod sistematic se realizează şi o aranjare lineară a vârfurilor sale, deci informaţiile stocate în vârfuri se pot regăsi şi prelucra mai uşor. Pentru a facilita scrierea, convenim ca în loc de $\{x_1, x_2, ..., x_n\}$ să se scrie $\{1, 2, ..., n\}$, fără ca valabilitatea rezultatelor să fie diminuată. Astfel, prin similitudine, se poate folosi drept relaţie de ordine între vârfurile grafului relaţia de ordine a numerelor naturale (notată cu „<").

Cele mai cunoscute metode de parcurgere a unui graf sunt parcurgerea **BF** (din limba engleză: ***B**readth **F**irst* – „în lăţime") şi parcurgerea **DF** (***D**epth **F**irst* – „în adâncime"). Deoarece este mai accesibilă, se va prezenta numai prima dintre ele.

a) Metoda de parcurgere BF

Principiul constă în vizitarea întâi a vârfului iniţial, apoi a vecinilor acestuia, apoi a vecinilor nevizitaţi ai acestora şi aşa mai departe.

Vizitarea unui vârf înseamnă, de fapt, o anumită prelucrare specificată asupra vârfului respectiv, însă în acest moment este importantă înţelegerea problemelor legate de parcurgerea unui graf prin metoda **BF**, astfel încât se va insista doar asupra acestui aspect. Derularea algoritmului presupune alegerea, la un moment dat, dintre toţi vecinii unui vârf, a aceluia care nu a fost încă vizitat.

Acest lucru este posibil prin folosirea unui vector **VIZITAT** de dimensiune n, ale cărui componente se definesc astfel:

$$\text{VIZITAT}[k] = \begin{cases} 1, & \text{dacă vârful } k \text{ a fost vizitat} \\ 0, & \text{în caz contrar} \end{cases}, \forall k \in \{1, 2, \cdots, n\}.$$

Se foloseşte o ***structură de tip coadă*** (simulată în alocare statică secvenţială printr-un vector `V`) în care prelucrarea unui vârf `k` memorat în vector (elementul curent al cozii), constă în adăugarea la sfârşitul cozii a tuturor vârfurilor `j` vecine cu `k`, nevizitate încă.

Iniţial, `k` este egal cu vârful indicat iniţial de utilizator, `i`, şi toate componentele lui **VIZITAT** sunt zero.

***Algoritmul constă în următorii paşi*:**

Pasul 1. Se prelucrează vârful iniţial *k*:
- se adaugă în coadă;
- se reţine ca vârf curent.

Pasul 2. Cât timp coada nu este vidă, pentru vârful curent, se execută:

Pasul 2.1. dacă vârful curent mai are vecini ce nu au fost adăugaţi în coadă:
- toţi aceştia se adaugă în coadă;

Pasul 2.2.
- este afişat vârful curent;
- este extras din coadă vârful curent.
- primul vârf din coadă devine vârf curent, în cazul în care coada nu este vidă.

Ordinea în care vârfurile au fost extrase din coadă reprezintă ordinea de parcurgere în adâncime a grafului dat.

exemplu

Se consideră graful din figura 5.15. Se doreşte parcurgerea sa începând cu vârful 1. În figură s-a marcat cu linii punctate ordinea de vizitare a vârfurilor. Ţinând seama de faptul că într-o coadă se prelucrează nodul cel mai „vechi", se vor trata vârfurile de la stânga spre dreapta.

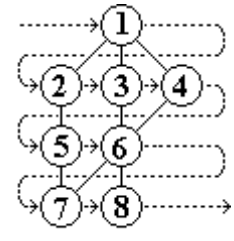


Figura 5.15

Pentru aceasta se execută paşii:

Pas	1	2.1	2.2	2.1	2.2	2.1	2.2	2.2	2.1	2.2	2.1	2.2	2.2	2.2
Vârf curent	1	1	2	2	3	3	4	5	5	6	6	7	7	8
Adăugat	1	2,3,4		5		6	6				7			
Extras şi afişat			1		2		3	4		5		6	7	8
Coada conţine	1	1, 2, 3, 4	2, 3, 4	2, 3, 4, 5	3, 4, 5	3, 4, 5, 6	4, 5, 6	5, 6	5, 6, 7	6, 7	6, 7, 8	7, 8	8	–

Ordinea în care au fost extrase vârfurile indică succesiunea de vârfuri rezultată din parcurerea acestui graf folosind metoda **BF.** Aceasta este 1, 2, 3, 4, 5, 6, 7, 8.

Programul **C++** corespunzător este:

```
//program parcurgere_BF;
#include<iostream.h>
#include<conio.h>
void main()
{ int VIZITAT[20] = {0};
 int A[20][20] = {0};
 int V[20];
//Numarul de varfuri ale graf. determina dimensiunea tablourilor
//A-matricea de adiacenta
//V-vectorul de memorare a varf. nevizitate, vecine cu varful k
 int n,m;      //n-numar varfuri; m-numar muchii
 int p;           //p-indica elementul curent al cozii
 int u;           //u-indica ultimul element al cozii
 int i,j;      //i,j-indici si contori
 int k;           //k-varful in lucru
 int x1,x2;  //x1 si x2-varfurile ce determina o muchie
clrscr();
 cout<<"\n"<<" Introduceti numarul de varfuri n: ";cin>>n;
 cout<<" Introduceti numarul de muchii m : ";cin>>m;
//completare matrice adiacenta
 for (i = 1;i< = m;i++)
  { cout<<" Muchia "<<i<<" este ";
  cin>>x1;cout<< " si ";cin>>x2;
  A[x1][x2] = 1;A[x2][x1] = 1;
  };
 cout<<" Varful de plecare: ";cin>>i;     // introducere varf de plecare
//prelucrarea primului varf
 V[1] = i;       //introducere varf i in coada C
 p = 1; u = 1;    //indica primul si ultimul element al cozii
 VIZITAT[i] = 1;          //se marcheaza varful i ca fiind vizitat
while (p< = u)            //cat timp coada nu este vida, executa:
 {k = V[p];         //se extrage primul varf din coada
  for (j = 1;j< = n;j++)
  //cautarea unui vecin nevizitat al lui k
  if (A[k][j] = = 1&&VIZITAT[j] = = 0)
    {//daca se gaseste un astfel de varf
     u = u+1;       //apare un nou element in coada
     V[u] = j;    //se adauga in coada varful gasit
```

```
      VIZITAT[j] = 1;  //se marcheaza varful ca fiind vizitat
     }
  p = p+1;
 }
//listarea succesiunii varfurilor obtinute dupa parcurgerea BF
 cout<<"  Plecand din varful "<<i<<", ";
 cout<<"\n"<<" parcurgand graful prin metoda BF,";
 cout<<"\n"<<" succesiunea varfurilor, este :";
 cout<<"  "<<i<<" ";
 for (j = 2;j< = u;j++) cout<<V[j]<<" ";
 cout<<"\n";getch();
}
```

Temă de laborator

Modificaţi programul pentru ca datele să fie citite dintr-un fişier text `graf.in`.

b) Metoda de parcurgere `DF` (Aprofundare)

Numele metodei provine, de asemenea, din limba engleză: ***D**epth **F**irst* – „întâi în adâncime".[1]

Pentru rezolvarea practică a acestei probleme se foloseşte o stivă (simulată în alocare statică printr-un vector *V*). Astfel există, în orice moment, posibilitatea de a ajunge de la vârful curent la primul dintre vecinii săi nevizitaţi încă, acesta din urmă plasându-se în vârful stivei. De la acest vârf se continuă parcurgerea în acelaşi mod.

Un vector **`URM`** determină, la fiecare pas, următorul vârf ce va fi vizitat după vârful *k*, (atunci când acesta există). Pentru a-l determina, se parcurge linia *k* din matricea de adiacenţă A asociată grafului, începând cu următorul element, până se găseşte un vecin *j* al lui *k*, nevizitat încă.

Dacă nu se poate determina un astfel de vârf, se coboară în stivă (*p* se micşorează cu 1), încercând să se aplice acelaşi procedeu următorului element al stivei.

Folosind o structură de tip stivă, prelucrarea unui vârf *k* aflat la un capăt al stivei (vârful stivei) constă în introducerea, în acelaşi capăt al ei, a tuturor vârfurilor *j* vecine cu *k* nevizitate încă.

Evident, pentru început, *k* este egal cu vârful indicat iniţial *i*.

Algoritmul constă în următorii paşi:

Pasul 1. Se prelucrează vârful *k*:
- se adaugă în stivă;
- se reţine ca vârf curent;
- se afişează.

Pasul 2. Cât timp stiva este nevidă, se analizează vârful curent:

Pasul 2.1. Dacă vârful curent mai are vecini nevizitaţi:
- se adaugă în stivă vârful *j*, primul dintre vecinii nevizitaţi ai vârfului curent, şi se afişează;
- vârful *j* devine vârf curent.

Pasul 2.2. Dacă nu există un astfel de vârf:
- se extrage din stivă vârful curent;
- vârful precedent celui curent devine vârf curent, dacă stiva nu este vidă.

Se consideră graful din figura 5.16.
Se doreşte parcurgerea sa în adâncime începând cu vârful 1.
În figură s-a marcat cu linii punctate ordinea de vizitare a vârfurilor.
Ţinând seama de faptul că într-o stivă se prelucrează nodul cel mai „nou", se vor trata vârfurile de la dreapta spre stânga.
Pentru aceasta se execută paşii:

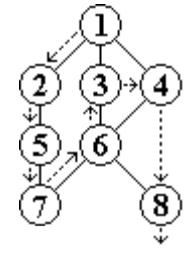


Figura 5.16

[1] Această metodă de parcurgere a unui graf a fost propusă de *Trémaux* în secolul al XIX-lea, ca o tehnică de rezolvare a amuzamentelor matematice legate de parcurgerea labirintelor.

Pas	1	2.1	2.1	2.1	2.1	2.1	2.2	2.1
Vârf curent	1	2	5	7	6	3	6	4
Adăugat şi afişat	**1**	**2**	**5**	**7**	**6**	**3**		**4**
Vârf extras							3	
Stiva conţine	**1**	**1, 2**	**1, 2, 5**	**1, 2, 5, 7**	**1, 2, 5, 7, 6**	**1, 2, 5, 7, 6, 3**	**1, 2, 5, 7, 6**	**1, 2, 5, 7, 6, 4**

Pas	2.2	2.1	2.2	2.2	2.2	2.2	2.2	2.2
Vârf curent	6	8	6	7	5	2	1	–
Adăugat şi afişat		**8**						
Vârf extras	4		8	6	7	5	2	1
Stiva conţine	**1, 2, 5, 7, 6**	**1, 2, 5, 7, 6, 8**	**1, 2, 5, 7, 6**	**1, 2, 5, 7**	**1, 2, 5**	**1, 2**	**1**	**–**

Succesiunea de vârfuri afişată (1, 2, 5, 7, 6, 3, 4, 8) indică succesiunea de vârfuri rezultată din parcurerea acestui graf folosind metoda **DF.**

Deoarece în cazul în care un vârf are mai mulţi vecini se poate alege oricare dintre aceştia, se pot obţine mai multe parcurgeri **DF**.
Pentru a elimina acest neajuns, se poate face convenţia ca vecinii vrfului curent să se adauge în stivă în ordinea crescătoare a notării lor.

5.3.4. Proprietatea de conexitate

a) Lanţ, ciclu

Fie graful G = (*X,U*). *Numim* **lanţ** *o succesiune de vârfuri* $L = [x_1,x_2,...,x_p]$ *cu proprietatea că oricare două vârfuri succesive sunt adiacente,*

$$\text{adică } [x_1,x_2]\in U,\ [x_2,x_3]\in U,\ \ldots,\ [x_{p-1},x_p]\in U.$$

Vârfurile x_1 şi x_p *se numesc* **extremităţile lanţului**, *iar numărul de muchii ce îl compun se numeşte* **lungimea lanţului.**
Dacă vârfurile $x_1,x_2,\ \ldots\ x_p$ *sunt distincte două câte două, lanţul se numeşte* **lanţ elementar**; *în caz contrar, lanţul se numeşte* **neelementar**.
Dacă, într-un lanţ, toate muchiile sunt diferite între ele, lanţul se numeşte **simplu**; *în caz contrar lanţul se numeşte* **compus.**

Exemplu. Fie graful din figura 5.17.
Succesiunea *L* = [1, 5, 3, 5, 2, 4] reprezintă un lanţ neelementar ale cărui extremităţi sunt vârfurile notate cu 1 şi 4, iar lanţul *L* = [1, 5, 2, 4] este un lanţ elementar.

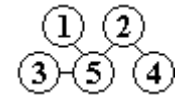


Figura 5.17

Dacă vârfurile x_1 *şi* x_p *coincid, lanţul se numeşte* **ciclu**.
Dacă un ciclu are o lungime pară, el se numeşte **par**; *dacă ciclul are o lungime impară, se numeşte* **impar**.
Dacă toate vârfurile unui ciclu sunt distincte, cu excepţie primului şi ultimului, atunci ciclul se numeşte **ciclu elementar**.

Ordinea enumerării vârfurilor într-un ciclu este lipsită de importanţă, deci succesiunea de vârfuri ce determină ciclul nu este unică.

exemplu

1. Fie graful din figura 5.18. Succesiunea C = [1, 5, 2, 4, 3, 5, 1] este un ciclu neelementar, iar C_1 = [1, 5, 2, 4, 1] şi C_2 = [2, 4, 3, 5, 2] sunt cicluri elementare.

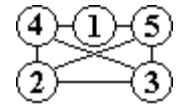


Figura 5.18

2. Fiind dată reţeaua 5.19, stabiliţi câte drumuri sunt între nodurile A şi B care nu parcurg de mai multe ori aceeaşi muchie.

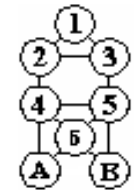


Figura 5.19

Rezolvare

Se pot determina 14 lanţuri. De exemplu, primele 3 sunt: A, 6, B; A, 4, 6, B; A, 6, 5, B.

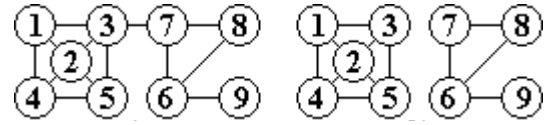
Temă: Continuaţi determinarea celorlalte soluţii.

b) Graf conex, **componentă conexă**, **multigraf**

Un graf G se numeşte **conex**, *dacă oricare ar fi două vârfuri x şi y, există un lanţ ce le leagă.*

exemplu

Se consideră grafurile din figura 5.20.
Primul exemplu (a) este un graf conex; al doilea (b) nu este conex, deoarece sunt perechi de vârfuri pentru care nu există niciun lanţ care să le unească (exemplu, vârfurile 3 şi 7 etc).



a) Figura 5.20 b)

Fie graful G = (*X*,*U*). *Un subgraf al său* C = (X_1,U_1), *conex, se numeşte* **componentă conexă** *a grafului* G = (*X*,*U*), *dacă are în plus proprietatea că nu există niciun lanţ în* G *care să lege un vârf din* X_1 *cu un nod din* $X - X_1$.

Aşadar, o componentă conexă a unui graf este o mulţime maximală de noduri ale grafului, cu proprietatea că între oricare două noduri există cel puţin un lanţ.

Noţiunea de **conexitate** este foarte importantă în teoria grafurilor şi, în special, în aplicaţiile sale.

exemplu

Să ne imaginăm următoarea situaţie: se doreşte arondarea la trei asociaţii de elevi cu preocupări ecologiste a trei zone turistice montane ce cuprind nouă cabane şi refugii. Între acestea se află cărări şi poteci, pentru întreţinere şi marcare.

Într-un tur de recunoaştere, fiecare echipă trebuie să treacă pe la fiecare punct de popas cel puţin o dată, pentru aprovizionare şi verificare.

Nici una dintre echipe nu poate primi o zonă în care se află o cabană sau un refugiu inaccesibil, sau la care se poate ajunge pornind din alte zone. Oricare ar fi două astfel de puncte, există totdeauna o succesiune de drumuri de la unul la altul.

Graful corespunzător acestei situaţii poate fi cel din figura 5.21.

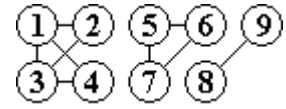


Figura 5.21

Se poate observa cu uşurinţă că acest graf are trei componente conexe: C_1 = {1, 2, 3, 4}, C_2 = {5, 6, 7} şi C_3 = {8, 9}.

observă

Având nodul x_i al grafului G = [*X*,*U*], componenta conexă X_i este alcătuită din x_i şi toate nodurile $x_j \in X$ pentru care există un lanţ care conţine x_i.

Teoremă. *Componentele conexe* X_i *ale unui graf* G = [*X*,*U*] *generează o* ***partiţie*** *a lui* *X*, *adică mulţimile* X_i *au proprietăţile*:

1. $X_i \neq \emptyset$, $\forall\, i \in N$;

2. $X_i \neq X_j \Rightarrow X_i \cap X_j = \emptyset$, $\forall\, i \in N$;

3. $\bigcup_{i \in N} X_i = X$;

Demonstraţie

1. Din modul de definire a mulţimii X_i rezultă $X_i \neq \emptyset$.

2. Fie $X_i \neq X_j$. Reducem la absurd. Presupunem că $X_i \cap X_j \neq \emptyset$, adică există $x_k \in X_i \cap X_j$. Pentru că $x_k \in X_i$, rezultă că există un lanţ care trece prin x_i şi x_k. Deoarece $x_k \in X_j$, rezultă că există un lanţ care trece prin x_j şi x_k. Aşadar, există un lanţ ce trece prin x_i şi x_j, adică $x_i \in X_j$, de unde $X_i = X_j$, ceea ce contrazice ipoteza.

3. Deoarece $X_i \subset X$ pentru orice $x_i \in X_i$, înseamnă că şi reuniunea mulţimilor X_i este inclusă în X, adică $\bigcup_{i \in N} X_i \subset X$. Pe de altă parte, orice $x_i \in X$ este conţinut în X_i, aşadar $\bigcup_{i \in N} X_i \supset X$.

Din cele două relaţii se deduce egalitatea căutată.

Teoremă. *Un graf este conex dacă şi numai dacă are o singură componentă conexă.*

Demonstraţie

Dacă graful ar avea două componente, X_i şi X_j, ar însemna că vîrfurile x_i şi x_j nu se află pe acelaşi lanţ, deci nu există un lanţ de la x_j la x_i, adică graful nu ar fi conex.

Dacă graful este conex, înseamnă că pentru oricare două noduri x_i şi x_j există un lanţ care le conţine, deci există o singură componenetă conexă.


exemplu

Formulele de structură ale substanţelor chimice sunt grafuri pentru care legăturile dintre vârfuri corespund legăturilor dintre grupările sau atomii care compun molecula. Astfel, în figura 5.22 sunt reprezentările chimice pentru apă (a), acetilenă (b) şi glucoză (c).

Dacă aceste formule de structură sunt reprezentate sub formă de grafuri pentru care nodurile sunt atomii (respectiv grupările de atomi) din moleculă, iar muchiile sunt legăturile lor chimice, se obţin reprezentările din figura 5.23.

Se observă că graful b) nu este graf în sensul definiţiei date, deoarece între anumite perechi de vârfuri există mai multe muchii.

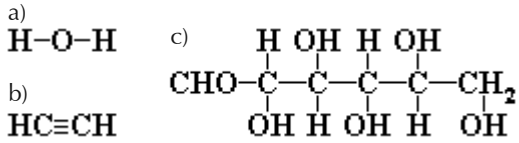


Figura 5.22

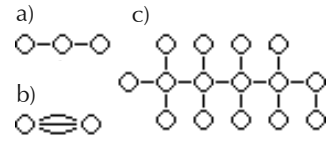


Figura 5.23

Fie graful G = (*X*,*U*). *Dacă există vârfuri între care există mai mult de o muchie, graful este numit* **multigraf**.


exemplu

Exemplu. Considerându-se schema căilor ferate din ţara noastră se obţine un graf care are drept noduri localităţile numite „noduri de cale ferată” (coincidenţă!), muchiile reprezentând legăturile directe pe calea ferată dintre două noduri. Cunoscându-se toate distanţele asociate fiecărei muchii, se poate determina cel mai scurt traseu între două localităţi. Acesta va fi, de fapt, un lanţ elementar care uneşte cele două localităţi şi care va avea lungime minimă.

O excursie în circuit care trece o singură dată prin anumite localităţi, întorcându-se în localitatea de pornire, va corespunde unui ciclu elementar în acest graf.

Algoritmul de verificare a proprietăţii de conexitate. Fie graful conex G = [*X*,*U*]. Conform definiţiei unui graf conex, pentru oricare două noduri x_i şi x_j există un lanţ care le conţine.

După parcurgerea grafului se poate ajunge în una dintre situaţiile:

– nu rămân noduri nevizitate – înseamnă că graful este conex;

– rămân noduri nevizitate – atunci acestora li se aplică metoda de parcurgere folosită, la fiecare parcurgere determinându-se câte o componentă conexă.

Pentru stabilirea conexităţii unui graf programul **C++** următor foloseşte metoda de parcurgere **BF** şi, în caz de neconexitate, afişează componentele conexe.

```
//program componente_conexe
#include<iostream.h>
#include<conio.h>
void main()
{ int VIZITAT[20] = {0};
 int A[20][20] = {0};
 int V[20];
 int valid;
 int n,m;
 int p;
```

```
 int u;
 int i,j;
 int k;
 int nc; //nc-retine componenta
         conexa
 int x1,x2;  clrscr();
 cout<<"\n"<<" Introduceti numarul de
     varfuri n: ";cin>>n;
 cout<<" Introduceti numarul de muchii
        m : ";cin>>m;
//completare matrice adiacenta
 for (i = 1;i< = m;i++)
  {cout<<" Muchia "<<i<<" este ";
  cin>>x1;cout<< " si ";cin>>x2;
  A[x1][x2] = A[x2][x1] = 1;
  };
 nc = 0;i = 1;
 cout<<" ";
 do
 { nc = nc+1;
cout<<"\n"<<"\nComponenta "<<nc<<" :";
 p = 1;u = 1;
 VIZITAT[i] = 1;
 cout<<"  "<<i<<" ";
 V[1] = i;
 while (p< = u)
  { k = V[p];
  for (j = 1;j< = n;j++)
            if(A[k][j] = = 1&&VIZITAT[j] = = 0)
            {
    u = u+1;
    V[u] = j;
    VIZITAT[j] = 1;
    cout<<j<<" ";
            }
  p = p+1;
  }
este = 0;
for (j = 1;j< = n&&!este;j++)
  {
  if (VIZITAT[j] = = 0)
            {i = j;este = 1;}
  }
 cout<<"\n";
 }while(este);
//determina un varf nevizitat al grafului
 }
```

1. Fie G = (*X*,*U*) un graf neorientat cu *n* = 8 vârfuri şi *U* = {[1,2], [1,3], [1,4], [2,5], [3,5], [4,6], [5,7], [6,7], [6,8]}.
Parcurgerea în lăţime, începând cu vârful 1, este:
a) 1, 2, 3, 5, 7, 6, 8; **b)** 1, 2, 3, 4, 5, 6, 7, 8; **c)** 1, 4, 3, 2, 5, 6, 8, 7; **d)** 1, 4, 3, 2, 6, 5, 8, 7.

2. Pentru acelaşi graf, să se precizeze care dintre succesiunile următoare reprezintă parcurgerea sa în adâncime, începând cu vârful 1:
a) 1, 2, 3, 5, 7, 6, 4, 8; **b)** 1, 3, 2, 5, 6, 4, 7, 8; **c)** 1, 2, 5, 3, 6, 4, 8, 7; **d)** 1, 4, 3, 2, 6, 5, 8, 7.

3. Precizaţi care dintre următoarele afirmaţii este adevărată:
a) un graf este conex dacă şi numai dacă are toate componentele conexe;
b) un graf este conex dacă şi numai dacă are o singură componentă conexă;
c) un graf este conex dacă între oricare două vârfuri există un lanţ care le leagă.

4. Pentru graful cu $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ $U = \{[1, 2], [1, 3], [2, 3], [3, 4], [3, 5], [4, 5], [6, 7], [6, 8], [7, 8], [8, 9]\}$, să se stabilească dacă este este conex sau complet.

5. Fie graful G = (*X,U*), citit, care nu este conex. Să se scrie o secvenţă de program prin care să se adauge numărul de muchii minim pentru ca graful să devină conex. Muchiile adăugate se vor lista.

6. Fie graful neorientat G = (*X,U*) cu $n = 8$ şi $U = \{[1, 3], [1, 8], [3, 8], [5, 6], [6, 7]\}$. Identificaţi care dintre mulţimile de muchii următoare adăugate grafului conduc la un graf conex:
a) {[2, 4], [2, 8], [3, 5]}; **b)** {[2, 5], [2, 3], [3, 4}; **c)** {[2, 4], [4, 7], [4, 8]}; **d)** {[2, 3], [2, 4], [2, 7]}.

7. Fie graful G = (*X,U*) cu $n = 7$ vârfuri şi nouă muchii, definit astfel:
X = {1, 2, 3, 4, 5, 6, 7, 8, 9} şi U = {[1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [2, 3], [4, 5], [6, 7]}.
Care dintre următoarele afirmaţii sunt adevărate?
a) G este complet; **b)** G este conex; **c)** G este bipartit; **d)** G este aciclic (nu are cicluri).

8. Fie graful G = (*X,U*) cu $n = 9$ vârfuri şi cincisprezece muchii, definit astfel: $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$şi $U = \{[1, 2], [1, 9], [2, 3], [2, 5], [3, 4], [3, 9], [4, 5], [4, 7], [5, 6], [5, 7], [6, 7], [6, 8], [6, 9], [7, 8], [7, 9], [8, 9]\}$. Stabiliţi valoarea de adevăr a fiecăreia dintre propoziţiile:
a) Graful este conex.
b) Nodurile 1 şi 5 sunt adiacente.
c) Graful este complet.
d) Graful este ciclic.
e) Gradul nodului 3 este 3.
f) Există cel mult două lanţuri care leagă nodul 2 cu nodul 5.
g) Matricea de adiacenţă asociată este nesimetrică.
i) Cel mai scurt lanţ între nodurile 1 şi 3 are lungimea 3.
j) Subgraful generat de nodurile {2,3,4} formează o componentă conexă.

9. Fie graful G = (*X,U*) cu $n = 9$ vârfuri şi zece muchii, definit astfel:
$X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ şi $U = \{[1, 2], [1, 3], [2, 3], [3, 4], [3, 9], [4, 9], [5, 6], [5, 7], [5, 8], [6, 8]\}$.
Să se determine, în cazul în care există:
a) toate ciclurile elementare;
b) toate ciclurile neelementare;
c) toate componentele conexe;
d) toate lanţurile care nu sunt elementare;
e) toate lanţurile elementare de la vârful 1 la vârful 9.

10. Să se testeze dacă un graf, dat prin lista muchiilor sale, conţine cicluri.

11. Stabiliţi corespondenţa corectă între fiecare element din coloana A şi proprietatea corespunzătoare din coloana B.

12. Să se arate că oricare două lanţuri elementare de lungime maximă ale unui graf conex au cel puţin un vârf comun.

A	B
a) repartiţia partidelor de şah într-un sistem de joc „fiecare cu fiecare”; b) traseul de vizitare a galeriilor unei expoziţii; c) mai multe calculatoare neinterconectate.	a) graf conex; b) graf nul; c) graf complet.

13. Fie G = (*X,U*) un graf neorientat dat, cu $n = 5$ vârfuri, $X = \{A, B, C, D, E\}$. Muchiile au fiecare un cost dat ca a treia componentă în descrierea fiecărei muchii $U = \{[A, B, 3], [A, D, 3], [A, E, 1], [B, C, 4], [B, E, 2], [C, D, 2], [C, E, 6], [D, E, 3]\}$. Stabiliţi valoarea de adevăr a fiecărei afirmaţii de mai jos:
a) G este conex;

rezolvă

b) G este aciclic;
c) există mai multe vârfuri cu grad par;
d) G conţine 4 subgrafuri complete de 3 vârfuri;
e) parcurgerea în lăţime pornind de la nodul B este: B, A, D, E, C;
f) cel mai ieftin ciclu este de valoarea 7.
g) ciclul A, E, B, C, D, A este cel care trece prin toate vârfurile o singură dată şi are costul minim = 12.

14. Să se testeze dacă un graf, dat prin lista muchiilor sale, conţine cicluri.

15. Fiind dat un graf G = (*X*, *U*), să se listeze toate perechile sale de vârfuri între care există un lanţ.

5.4. Arbori

5.4.1. Definiţii

Din punct de vedere structural, cele mai simple grafuri sunt cele numite **`arbori`**. Acestea sunt, de fapt, şi cele mai folosite în practică. De-a lungul timpului, de studiul arborilor s-au ocupat matematicienii şi fizicienii de primă mărime.[1]

Trebuie observat că organizarea de tip listă nu este totdeauna cea mai adecvată în unele aplicaţii. Astfel, dacă trebuie descrisă structura unui produs, aceasta nu se face descriindu-i componentele una câte una, ci se procedează la o descriere ierarhică a părţilor care îl compun, adică o structură asemănătoare unui arbore.

a) Nod, rădăcină, subarbore, ascendenţă – descendenţă, ordinul unui nod

Organizarea ierarhică este întâlnită în cele mai diverse domenii, de la organizarea administrativă a unei ţări, la planificarea meciurilor în cadrul unui turneu sportiv, de la structurarea unei cărţi până la stabilirea ordinii de execuţie a operaţiilor efectuate pentru determinarea valorii unei expresii aritmetice.

De exemplu, cataloagele în care sunt grupate fişierele de pe discurile fixe sau flexibile au o structură ierarhică. Această organizare este impusă, în principal, de raţiuni de gestionare cât mai comodă a fişierelor de diverse tipuri, aparţinând diverşilor utilizatori ai aceluiaşi sistem de calcul. Şi exemplele pot continua.

Pentru vârfurile unui arbore se va folosi termenul de **nod**.

Figurativ, o structură de *tip arbore* arată ca un arbore, în înţelesul general, doar că este răsturnat. Fiecare element din această structură poate fi privită ca o rădăcină de la care pornesc ramuri către rădăcinile altor arbori. În reprezentarea grafică a unui arbore, nodurile se desenează pe niveluri, astfel: rădăcina se află pe primul nivel, codificat cu 0, vârfurile adiacente cu rădăcina pe următorul nivel, codificat cu 1, şi aşa mai departe.

O primă definiţie, intuitivă, a structurii de arbore este următoarea:

Un arbore A este format fie dintr-un nod **rădăcină** *(R), fie dintr-un nod rădăcină căruia îi este ataşat un număr finit de arbori. Aceştia sunt denumiţi* **subarbori** *ai lui A, datorită relaţiei de „subordonare" faţă de rădăcină.*

Deci, într-un arbore, orice nod este rădăcina unui subarbore, iar orice arbore poate fi sau poate deveni subarbore. Între doi subarbori nu poate exista decât o relaţie de incluziune (unul este subarbore al celuilalt) sau de excluziune (cei doi subarbori nu au noduri comune, dar aparţin aceluiaşi arbore).

Mulţi termeni folosiţi în studiul arborilor sunt împrumutaţi din terminologia utilizată în cazul arborilor genealogici sau al celor din natură.

Astfel, pentru a desemna o relaţie directă între două noduri se folosesc termenii: **tată**, **fiu** şi **frate**, cu semnificaţia obişnuită. Pentru relaţiile indirecte, de tipul „fiul fiului... fiului", se folosesc termenii **descendent** sau ***urmaş*** şi, respectiv, **ascendent** sau ***strămoş***.

Nodurile fără descendenţi sunt numite ***noduri*** **terminale** *sau, prin analogie cu arborii din natură,* **frunze**.

[1] Astfel, **Cayley** a studiat arborii şi posibilitatea aplicării lor în chimia organică, iar **Kirchoff** a studiat grafurile bazându-se pe considerente din fizică, şi anume reţelele electrice.
Arborii au fost numiţi astfel de către *Cayley* în *1857*, datorită aspectului asemămător cu arborii din botanică.

exemplu

1. În figura 5.24 este un prezentat un arbore cu rădăcina în nodul 1, care are trei subarbori, şi anume 2, 3 şi 4. Nodul 1 este un ascendent pentru nodurile 2, 3 şi 4, care îi sunt *descendenţi*. Nodurile 2, 4, 5, 7, 8 şi 9 sunt noduri terminale, iar nodurile 3 şi 6 sunt la rândul lor, rădăcini pentru subarborii 5, 6 şi 7, respectiv 8 şi 9.
2. Figura 5.25 prezintă un arbore binar, fiecare nod având cel mult 2 subarbori.

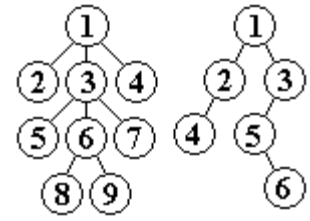


Figura 5.24

Figura 5.25

Accesul de la rădăcina unui (sub)arbore nevid la oricare alt nod presupune parcurgerea unei căi formate din a muchii ($a = 0$) pe care se găsesc q noduri ($q = a + 1$). Valoarea **q** reprezintă **nivelul** pe care se găseşte nodul faţă de rădăcină. Rădăcina este considerată, prin convenţie, pe nivelul 0.

Înălţimea unui arbore se poate defini ca *maximul dintre nivelurile nodurilor terminale*, sau:
***înălţimea** unui arbore este egală cu 1 + maximul dintre înălţimile subarborilor săi.*

Numărul de descendenţi direcţi ai unui nod reprezintă **ordinul** nodului. În cazul în care ordinul nodurilor nu este limitat, arborele este denumit **arbore multicăi**.

Teoremă. (de caracterizare a arborilor) *Pentru un graf* G = (*X, U*) *cu* $n \geq 2$ *noduri*, ***m** muchii, următoarele afirmaţii sunt echivalente şi caracterizează un arbore*:

1. G este conex şi fără cicluri;
2. G este fără cicluri şi $m = n - 1$;
3. G este conex şi $m = n - 1$;
4. G este un graf conex fără cicluri, maximal (dacă se adaugă o muchie între două noduri neadiacente, se formează un ciclu);
5. G este un graf conex, minimal (dacă se elimină o muchie oarecare se obţine un graf care nu mai este conex);
6. orice pereche de noduri este legată print-un lanţ şi numai unul.
Temă: Demonstraţi teorema de mai sus.

Caracteristicile arborilor

I. Un arbore este un graf **conex** şi **fără cicluri**
II. Vârfurile unui arbore se numesc **noduri**.
III. Nodul iniţial se numeşte **rădăcină**.
IV. Orice descendent al unui nod defineşte un **subarbore** al acelui nod.
V. Dacă un nod nu are descendenţi şi nu e nod rădăcină, atunci el este **nod terminal**.
VI. Un arbore cu ***n* noduri** are ***n* – 1 muchii**.
VII. Într-un arbore orice pereche de noduri este legată de **un lanț și numai unul**.
VIII. Un graf cu n noduri şi ***n* – 1 muchii** este arbore dacă **nu are cicluri**.
IX. Orice arbore H=(*X,V*) cu ***n*** ≥ **2** noduri conţine **cel puțin două noduri terminale**.
X. **Gradul maxim** al unui nod x al unui arbore cu n noduri este ***n* – 1.**
XI. Lungimea lanţului dintre un nod şi nodul rădăcină determină **nivelul** nodului respectiv.
XII. Lungimea lanţului maximal din arbore precizează **înălțimea** acestuia.
XIII. **Costul** unui arbore este suma costurilor muchiilor.

c) Arbore parţial

Fie un graf G. *Un graf parţial al său, care în plus este şi arbore, se numeşte* **arbore parțial**.

Corolar. *Un graf* G = (*X, U*) *conţine un arbore parţial dacă şi numai dacă* G *este conex.*

5.4.2. Reprezentarea arborilor

Arborii, fiind cazuri particulare de grafuri, beneficiază de aceleaşi metode de reprezentare. Din cauza faptului, însă, că au *n*-1 muchii, metoda cea mai potrivită reprezentării lor este prin vectorul „taţilor".

Un element din vectorul taţilor reprezintă eticheta nodului tată pe care îl are nodul cu indicele elementului respectiv.

Exerciţii rezolvate

1. Se consideră arborele din figura 5.26. Să se reprezinte acest arbore.

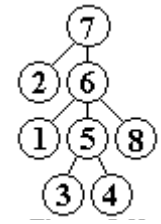


Figura 5.26

Rezolvare. Se alcătuieşte un tabel cu două linii şi coloane în număr egal cu numărul de noduri, urmărind regulile:

– pe prima linie se trec etichetele nodurilor;

– pe linia a doua linie (**vectorul taţilor**), pentru fiecare nod se trece „tatăl" nodului respectiv.

Nodul i	1	2	3	4	5	6	7	8
TATA[i]	6	7	5	5	6	7	0	6

Se obţine:

Nodul i	1	2	3	4	5	6	7	8
TATA[i]	8	1	6	5	0	5	1	5

2. Se dă următorul vector al „taţilor":

Se cere:

a) să se determine nivelul fiecărui nod;

b) să se determine înălţimea arborelui;

c) să se determine numărul de noduri de pe fiecare nivel al arborelui;

d) să se reconstruiască arborele pe care îl reprezintă;

Rezolvare

a) Pentru nodul cu eticheta *k*, consultând tabelul, se parcurg etapele:

– din coloana k se determină nodul *j* – ascendentul nodului *k*;

– nodul *j* devine nod *k*;

– se reia procedura până se ajunge la nodul rădăcină (care nu are ascendent, deci în linia a doua a tabelului este trecută valoarea 0). La fiecare pas nivelul creşte cu o unitate.

Nodul i	1	2	3	4	5	6	7	8
Nivel	2	3	2	1	0	1	3	1

Practic, pentru exemplul concret dat, se obţine:

b) Arborele are înălţimea 3 (maximul nivelurilor: max{0, 1, 2, 3} = 3).

c) Se aranjează nodurile în ordinea crescătoare a nivelurilor lor:

– nivel 0: nodul 5 (nodul rădăcină);

– nivel 1: nodurile: 4, 6, 8;

– nivel 2: nodurile: 1, 3;

– nivel 3: nodurile: 2, 7.

d) Se determimă nodul rădăcină, ştiind că acesta este singurul ce nu are „tată". Se deduce că rădăcina arborelui este nodul 7.

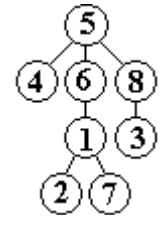


Figura 5.27

Pentru a detemina descendenţii acestui nod, pe linia a doua se determină coloanele unde apare eticheta rădăcinii. Fiecare indice de coloană este eticheta unui descendent.

Se reia procedeul pentru fiecare nod.

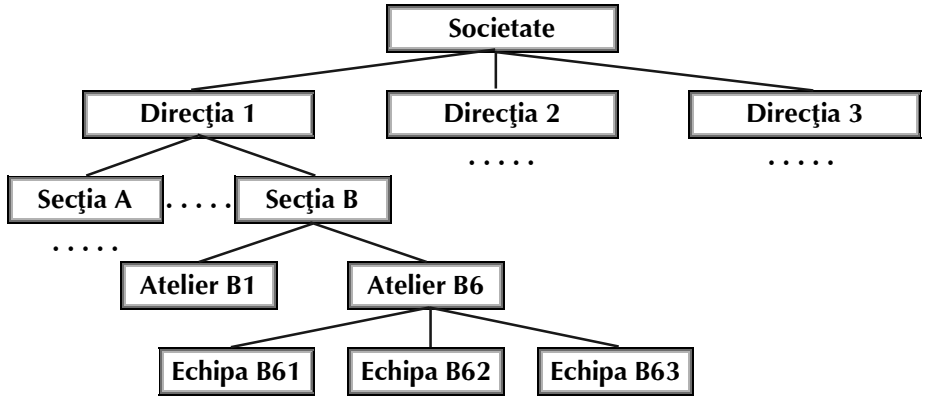
Se obţine arborele din figura 5.27.

5.4.3. Parcurgerea arborilor

Prelucrarea informaţiilor memorate într-o structură de tip arbore implică parcurgerea acestuia, adică inspectarea (vizitarea) fiecărui nod şi prelucrarea informaţiei specifice. Problema care se pune este aceea a ordinii în care se prelucrează nodurile arborelui, rădăcina şi nodurile din subarbori. De cele mai multe ori, acesta este impusă de specificul aplicaţiei.

Deoarece arborii sunt structură neliniară de organizare a datelor, rolul traversării este tocmai obţinerea unei aranjări liniare a nodurilor, pentru ca trecerea de la unul la altul să se realizeze cât mai simplu posibil.

Fie o structură multicăi în care sunt memorate informaţiile despre organizarea unei societăţi



comerciale.

Parcurgerea acestui arbore se poate face în mai multe feluri. Astfel, presupunând că se solicită lista personalului cu funcţii de conducere, gruparea persoanelor poate fi tipărită: pe niveluri ierarhice sau punându-se în evidenţă relaţiile de subordonare:

Corespunzător, se obţine:

1. Societate: Aldea Alin
Direcţia_1: Bazon Barbu
Direcţia_2: Copcea Călin
Direcţia_3: Dragu Dan
Secţia_A: Epuran Emilia
Secţia_B: Firică Fana
.....
Atelier_B1: Ganea Gheorghe
.....
Atelier_B6: Hrib Horia
Echipa_B61: Ionescu Ion
Echipa_B62: Jitaru Jean
Echipa_B63: Kelemen Karl

Direcţia_3: Dragu Dan

.....
2. Societate: Aldea Alin
Direcţia_1: Bazon Barbu
Secţia_A: Epuran Emilia
.....
Secţia_B: Firică Fana
Atelier_B1: Ganea Gheorghe
.....
Atelier_B6: Hrib Horia
Echipa_B61: Ionescu Ion
Echipa_B62: Jitaru Jean
Echipa_B63: Kelemen Karl
Direcţia_2: Copcea Călin
.....

Prima variantă de parcurgere se numeşte ***parcurgere în lăţime*** (pe niveluri), a doua ***parcurgere în adâncime*** (ierarhică), aşa cum s-a procedat şi la grafuri.

1) În cazul ***parcurgerii în lăţime***, se tipăreşte mai întâi informaţia din nodul rădăcină, după care sunt prelucrate, de la stânga la dreapta, nodurile aflate pe primul nivel, apoi pe cel de-al doilea şi aşa mai departe.

2) În cazul ***parcurgerii în adâncime***, fiii unui nod sunt vizitaţi tot de la stânga spre dreapta, dar trecerea de la nodul curent la fratele din dreapta se realizează numai după vizitarea tuturor descendenţilor nodului curent, deci a întregului subarbore dominat de acesta.

5.4.4. Arbore parțial de cost minim

Fie G = (*X*,*U*) un graf conex în care fiecare muchie are ataşat un cost.

Dacă din acesta se elimimă muchii astfel încât să se obţină un arbore parţial al cărui cost să fie minim, acesta se numeşte arbore **parţial de cost minim**.

Proprietate. *Pentru graful* G *conex, cu funcţia de cost* **c**, *există un graf parţial* H *conex şi de cost minim, care este şi arbore.*

Aplicaţie

O sursa de apă rebuie să alimenteze o localitate. Astfel, un nod devine sursa de apă notată *S*, iar figura 5.28 reprezinta reţeaua traseelor posibile pentru conducte. Ponderea fiecărei muchii reprezintă costul investiţiei pentru tronsonul respectiv.

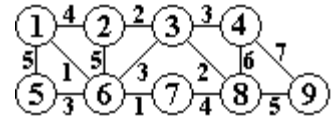


Figura 5.28

Se cere determinarea acelor tronsoane care realizează la un cost minim al investiţiei alimentarea consumatorilor. Problema revine la a determina un arbore parţial de cost minim **(APM).**

Algoritmul pentru determinarea unui arbore parţial de cost minim (APM) fost stabilit de *Kruskal* în anul 1956 şi de cele mai multe ori referirea la algoritm se face folosind numele autorului.

Fie un graf conex G = (*X*,*U*) cu o funcţie cost c:$U \to R_+$ cu *n* vârfuri şi *m* muchii.

Pentru determinarea unui arbore parţial de cost minim se procedează în felul următor:

– se consideră, iniţial, graful nul cu *n* vârfuri, deci cu *n* componente conexe;

– se alege şi se ataşează muchia *u* care are costul *c(u)* minim – unificarea componentelor conexe;

– procedeul se repetă pentru următoarea muchie de cost minim, *dintre muchiile nealese, astfel încât să nu formeze cicluri cu muchiile deja alese*.

– procesul se încheie când se obţine o mulţime de muchii $V \subseteq U$, deci un graf parţial H = (*X*,*V*) al lui G, cu proprietatea că oricare dintre muchiile lui G rămase formează un ciclu cu muchiile lui H.

Deci H este un graf fără cicluri, maximal, cu aceeaşi mulţime de vârfuri ca şi G. Conform teoremei de caracterizare, rezultă că H este un arbore parţial al lui G şi din modul de selectare a muchiilor, el este de cost minim.

♦ Memorarea grafului se face folosind varianta de reprezentare **b2)** a unui graf, adică lista muchiilor. Pentru a şti în fiecare moment care sunt nodurile care aparţin aceluiaşi subarbore parţial H_k, se asociază tuturor nodurilor lui H_k aceeaşi valoare. Pentru subarbori parţiali distincţi, nodurile acestora vor avea asociate valori distincte. Pentru memorarea arborilor parţiali în care se găsesc la un moment dat vârfurile grafului G se va folosi o listă, *L*, cu *n* poziţii, astfel încât poziţia *k* din listă, notată *L*(*k*), să indice numărul de ordine al arborelui parţial (componentei conexe) în care se găseşte vârful *k* al grafului.

Pentru a uşura căutarea muchiei de cost minim, lista muchiilor grafului se ordonează crescător după costuri. Algoritmul se va opri după ce vor fi selectate *n* – 1 muchii, deoarece, aşa cum s-a stabilit cu ajutorul teoremei, un arbore cu *n* noduri are exact *n* – 1 muchii.

Paşii algoritmului sunt daţi în continuare.

Pasul 1. Se iniţializează lista *L* cu *L*(*k*) = *k*, pentru $k \in \{1,2,...,n\}$, adică se presupune că fiecare vârf aparţine unei componente conexe distincte (graful nul);

Pasul 2. Se ordonează crescător lista muchiilor grafului, după costurile *c*;

Pasul 3. Fie [*p*,*q*] prima muchie din şirul muchiilor lui G;

Pasul 4. Dacă s-au selectat *n* – 1 muchii, STOP, deoarece s-a obţinul un arbore parţial minim. Dacă s-au selectat mai puţine muchii se trece la pasul 5.

Pasul 5. Se verifică dacă *L*(*p*) = *L*(*q*), adică dacă vârfurile *p* şi *q* aparţin aceleiaşi componente conexe:

a) dacă da, înseamnă că muchia face parte din acelaşi arbore parţial, ea nu va fi selectată (s-ar obţine un ciclu). Se repetă pasul 5, pentru următoarea muchie din listă.

b) dacă răspunsul este negativ, adică $L(p) \neq L(q)$, înseamnă că muchia face parte din arbori parţiali diferiţi. Se trece la pasul 6.

Pasul 6. Se va selecta muchia [*p*,*q*] ca o nouă muchie a arborelui parţial minim. Dacă *L*(*p*) < *L*(*q*), atunci se înlocuiesc toate elementele *L*(*i*) = *L*(*q*) cu valoarea *L*(*p*). Dacă *L*(*p*) > *L*(*q*), atunci se înlocuiesc toate elementele *L*(*i*) = *L*(*p*) cu valoarea *L*(*q*). Se merge la pasul 4.

Este evident că, procedând astfel, în final se va ajunge ca $L(i) = 1$, pentru toate valorile lui $i \in \{1,2,...,n\}$.

exemplu

Considerând graful din figura 5.26, lista muchiilor, ordonată după costurile asociate, este:

Muchia	[1,6]	[6,7]	[2,3]	[3,8]	[3,4]	[3,6]	[5,6]	[1,2]	[7,8]	[1,5]	[2,6]	[8,9]	[4,8]	[4,9]
Costul	1	1	2	2	3	3	3	4	4	5	5	5	6	7

Lista L este:

i	1	2	3	4	5	6	7	8	9
L[i]	1	2	3	4	5	6	7	8	9

Arborii sunt: $H_k = (k, \emptyset)$ $k \in \{1,2, ... ,9\}$.

Se consideră muchia [1, 6]. Deoarece $L(1) = 1 \neq 6 = L(6)$, se selectează muchia [1,6] şi $L(6) = L(1) = 1$. Lista L devine:

i	1	2	3	4	5	6	7	8	9
L[i]	1	2	3	4	5	1	7	8	9

Arborii sunt: $H_1 = (\{1,6\},\{[1,6]\})$ şi $H_k = (k, \emptyset)$ $k \in \{2,3,4,5,7,8,9\}$.

i	1	2	3	4	5	6	7	8	9
L[i]	1	1	1	1	1	1	1	1	1

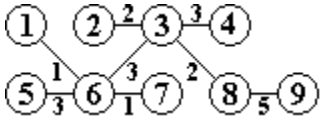


Figura 5.29

Considerând $V = \{[1, 6], [6, 7], [2, 3], [3, 8], [5, 6], [3, 6], [3, 4], [8, 9]\}$, costul pentru graful parţial H = (X,V), este:

$c(H) = c([1, 6]) + c([6, 7]) + c([2, 3]) + c([3, 8]) + c([5, 6]) + c([3, 6]) + c([3, 4]) + c([8, 9]) =$
$= 1 + 1 + 2 + 2 + 3 + 3 + 3 + + 5 = 20.$

În figura 5.29 s-au marcat muchiile grafului parţial **H** dedus anterior, care este arbore al cărui cost este minim, costul acestui arbore fiind 20.

observă

În general, dacă există mai multe muchii având acelaşi cost, pentru un graf conex G pot exista mai multe posibilităţi de alegere a unei muchii de cost minim care nu formează cicluri cu muchiile deja alese, deci pot exista mai mulţi arbori parţiali de cost minim. Aceşti arbori se vor deosebi prin muchiile ce vor fi luate în considerare şi nu prin costul asociat, deoarece acest cost va fi acelaşi pentru toţi arborii, şi anume toţi vor avea un cost minim.

Programul **C++** corespunzător este:

```
//program APM;
#include<iostream.h>
#include<conio.h>
 int n,m,i,j,k;
struct muchie {int x1,x2; float cost; } u[20], aux;  //u-lista muchiilor
 int L[20];
//L-lista varfurilor grafului (ordonata crescator dupa costuri int p;
int v,w;
//v si w- pastreaza extremitatile muchiei
float ct;          //ct- insumeaza costurile
//Toate celelalte variabile au aceeasi semnificatia stiuta
void main ()
{clrscr();
//Completare matrice costuri
cout<<"\n"<<" Introduceti numarul de varfuri n: ";cin>>n;
cout<<" Introduceti numarul de muchii m :
        "; cin>>m;
```

```
for (i = 1;i< = m;i++)
 {cout<<" Muchia "<<i<<" este ";
 cin>>u[i].x1;cout<<" si ";cin>>u[i].x2;
 cout<<" costul muchiei "<<i<<": ";
 cin>>u[i].cost;
 }
//Sorteaza crescator dupa costuri a listei muchiilor
for (i = 1;i< = n;i++) L[i] = i;
p = m;
while(p>1)
 { k = 0;
  for (i = 1;i<p;i++)
    if(u[i].cost>u[i+1].cost)
    { aux = u[i];
     u[i] = u[i+1];
     u[i+1] = aux;
     k = i;
    }
  p = k;
 }
//Determinare arbore partial de cost minim
cout<<"\n Arborele partial de cost minim";
cout<<"\n este compus din urmatoarele muchii: \n";
ct = 0; //itinializate cost total
k = 0;//initializare mumar total de muchii alese
i = 1;
 while (k<n-1) //se aleg cele n-1 muchii
             ale APM
 { if (L[u[i].x1]! = L[u[i].x2])
    { k = k+1;    //alege muchia i
     ct = ct+u[i].cost;
     cout<<"["<<u[i].x1<<";"<<u[i].x2<<"]; ";
     v = L[u[i].x2];//urmeaxa unificarea subarborilor
     w = L[u[i].x1];
     for (j = 1;j< = n;j++)
     if (L[j] = = v) L[j] = w;
 }
 i = i++;
//se trece la muchia urmatoare
 }
cout<<"\n * Costul total :"<<ct;
getch();
}}
```

5.4.5. Arbori binari

a) Proprietăţi

Subarborele stâng şi subarborele drept (definiţi deja) se obţin prin suprimarea rădăcinii şi a muchiilor adiacente cu aceasta. Oricare dintre ei poate fi vid. Dacă arborele binar este format dintr-un singur nod, atunci ambii subarbori sunt vizi.

Fie arborii binari din figura 5.29.
Ca *arbori* sunt identici, dar ca *arbori binari* sunt diferiţi între ei, deoarece primul are subarborele drept vid, iar al doilea are subarborele stâng vid.
Numerotând nivelurile nodurilor într-un arbore binar, rădăcina va fi pe nivelul 0, descendenţii săi pe nivelul 1, descendenţii acestora pe nivelul 2 ş.a.m.d.

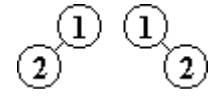


Figura 5.29

Dacă toate nodurile unui arbore, cu excepţia celor terminale, au exact doi descendenţi, arborele se numeşte **arbore binar complet**.

Proprietate. *Un arbore binar complet care are* **n** *noduri terminale, toate situate pe acelaşi nivel, are în total* $2n - 1$ *noduri.*

Corolar. *Un arbore binar complet are un număr impar de noduri.*
Demonstraţie. Evident, deoarece $2n - 1$ este număr impar.

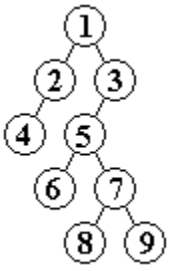


Figura 5.30

b) Reprezentarea arborilor binari

Există mai multe posibilităţi de reprezentare a arborilor binari. În continuare sunt descrise două dintre cele mai des folosite.

1. Reprezentarea standard se bazează pe următorul principiu: pentru fiecare nod în parte se precizează, dacă există, descendentul stâng şi descendentul drept. Dacă un nod este nod terminal, atunci acest lucru se semnalează punând 0 în locul descendenţilor săi. Pentru aceasta, se utilizează fie doi vectori numiţi, de exemplu, *S* – pentru descendenţii din stânga – şi *D* – pentru descendenţii din dreapta.

Dacă pentru reprezentarea unui arbore binar cu ***n*** noduri se folosesc vectorii *S* şi *D*, atunci pentru fiecare $i \in \{1,2,...,n\}$componenta *S*[*i*] conţine indicele descendentului stâng al nodului *i*, iar componenta *D*[*i*] conţine indicele descendentului drept al nodului *i*.

Aplicăm acest algoritm la un caz concret. Se consideră arborele binar din figura 5.30. În acest caz, cei doi vectori vor avea următorul conţinut:

Nodul i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
S[i]	2	4	5	0	6	0	8	0	0
D[i]	3	0	0	0	7	0	9	0	0

Se observă că nu este important să se precizeze rădăcina, deoarece ea nu este descendentul nici unui nod.

2. Se folosesc **doi vectori**: **`TATA`** şi **`DESC`**. Pentru pentru fiecare nod **`i`** **`TATA`**[`i`] precizează care nod îi este ascendent. **`DESC`**[`i`] poate lua două valori: **–1** dacă i este descendent stâng pentru **`TATA`**[`i`] şi **1** dacă **`i`** este descendent drept pentru acesta. Pentru nodul rădăcină, care nu are un nod părinte asociat, valoare corespunzătoare în vectorii **`TATA`** şi **`DESC`** este **0**.

Pentru arborele binar considerat, avem:

Nodul i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
TATA[i]	0	1	1	2	2	5	5	7	7
DESC[i]	0	-1	1	-1	1	-1	1	-1	1

3. Reprezentarea **cu paranteze** (**parantezată**). Pentru a obţine o reprezentare a arborelui folosind parantezele, se procedează în felul următor:

1) se scrie nodul rădăcină;
2) fiecare nod al arborelui va fi urmat de:
– paranteză_rotundă_deschisă;
– descendent_stâng;
– virgulă;
– descendent_drept;
– paranteză_rotundă_închisă.

Pentru arborele din figura 5.34, reprezentarea parantezată este 1(2(4,3(5,(6,7(8,9))))).

Pentru citirea datelor de structură a unui arbore, în programare se foloseşte scrierea numărului 0 pentru a marca absenţa unui descendent. Astfel, reprezentarea de mai sus se va scrie:

1(2(4(0,0),0),3(6,7(8,9)),0))

În situaţia în care fiecare nod al arborelui conţine o informaţie, se va folosi un vector suplimentar **INFO**, în care, **INFO**[i] conţine informaţia corespunzătoare nodului i.

c) Parcurgerea arborilor binari

Pentru parcurgerea unui arbore binar este nevoie de precizarea sensului acestei parcurgeri, adică de stabilirea ordinii de trecere prin nodurile arborelui. Sensurile de parcurgere pot fi:

• *în lăţime* – pe niveluri.
– se porneşte cu nodul rădăcină (aflat pe nivelul 0);
– se iau în consideraţie nodurile de pe nivelul 1;
– se continuă cu celelalte niveluri, până la ultimul nivel.

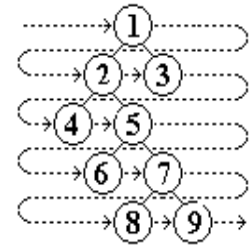


Figura 5.31

exemplu

Fie arborele binar din figura 5.30.
Parcurgerea în lăţime a acestui arbore (fig. 5.31) este: 1, 2, 3, 4, 5, 6, 7, 8, 9.

• *în adâncime* – pe subarbori.

Analizând situaţia unui arbore binar, se observă că, pentru a avea o parcurgere riguroasă, dar mai ales eficientă, trebuie precizate în fiecare moment poziţiile a trei componente, şi anume: rădăcina şi cei doi subarbori ai săi (subarborele stâng şi subarborele drept). Sunt deci 3! = 6 variante.

1) Sensuri cu **terminologie consacrată**, când se ţine seama de regula: este reprezentat subarborele stâng şi apoi subarborele drept. În raport de aceştia, se poziţionează rădăcina.

S-au obţinut astfel trei posibilităţi de reprezentare:
– stânga → radăcină → dreapta, sau sensul **SRD (inordine** – rădăcina este la mijloc**)**;
– rădăcină → stânga → dreapta, sau sensul **RSD (preordine** – rădăcina este prima**)**;
– stânga → dreapta → rădăcină, sau sensul **SDR (postordine** – rădăcina este ultima**)**.

observă

Denumirile metodelor semnifică momentul când se vizitează rădăcina în raport cu vizitarea subarborilor. De aceea, traversarea în preordine se mai numeşte RSD, traversarea în inordine – SRD, traversarea în postordine – SDR.

2) Sensuri **permutate**, când este reprezentat subarborele drept şi apoi subarborele stâng, rădăcina poziţionându-se în raport de aceştia. Se mai obţin, astfel, încă trei posibilităţi:
– dreapta → rădăcină → stânga, sensul **DRS**;
– dreapta → stânga → rădăcină, sensul **DSR**;
– rădăcină → dreapta → stânga, sensul **RDS**;

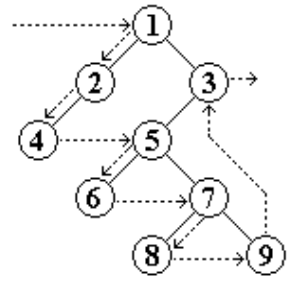


Figura 5.32

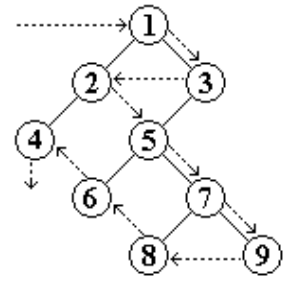


Figura 5.33

exemplu

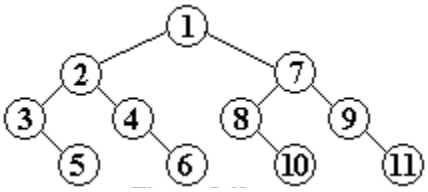
Fie arborele binar din figura 5.28.
Considerând nodul **1** ca rădăcină, ca rezultate ale parcurgerilor în diverse moduri, se obţin, de exemplu, pentru RSD: 1, 2, 4, 5, 6, 7, 8, 9, 6, 3 şi RDS: 1, 3, 2, 5, 7, 9, 8, 6, 4, situaţii evidenţiate în grafurile din figurile 5.32 şi, respectiv 5.33.

Temă de laborator

Continuaţi determinarea celorlalte soluţii, corespunzătoare diverselor procedee de parcurgere a grafului. De fiecare dată, trasaţi pe graf traseul parcurs. Remarcaţi asemănările şi deosebirile între rezultatele obţinute în cele 3! = 6 posibilităţi.

exemplu

Se consideră cuvântul COMUNICATIE. Atanjaţi literele lui în nodurile arborelui binar din figura 5.34 astfel încât prin parcurgerea SDR să se obţină cuvântul dat.



Rezolvare. Parcurgând arborele SDR se obţine:

5	3	6	4	2	10	8	11	9	7	1
C	O	M	U	N	I	C	A	T	I	E

Figura 5.34

d) Aplicaţie a arborilor binari – traducerea expresiilor de către compilator

Arborii binari au foarte multe aplicaţii practice, dintre care cea mai importantă este **forma poloneză** a expresiilor aritmetice (sau notaţia fără paranteze a expresiilor aritmetice). Metoda a primit şi denumirea de ***formă poloneză***, deoareace de studiul acestei probleme s-a ocupat matematicianul polonez *J. Lukasiewicz*.

Esenţa acestei scrieri constă în: într-o expresie aritmetică, un operator se aplică unui operand sau unei perechi de operanzi, de unde asocierea unei expresii aritmetice cu un arbore binar.

Se consideră E_1 şi E_2, două expresii aritmetice cărora li se aplică operandul notat „op". Se vor accepta doar operatorii binari[1] : adunarea (+), scăderea (–), înmulţirea (*), împărţirea (/) şi ridicarea la putere (^).

Reguli:

R1. Unei expresii aritmetice formată dintr-un singur operand i se asociază un arbore binar format doar din nodul rădăcină în care se scrie operandul respectiv;

R2. Dacă expresia aritmetică *E* este de forma E_1 *op* E_2, atunci arborele binar complet asociat are ca rădăcină operatorul *op*, ca subarbore stâng arborele binar asociat expresiei E_1, iar ca subarbore drept arborele binar asociat expresiei E_2.

R3. Dacă $E = (E_1)$, atunci arborele asociat lui *E* coincide cu arborele binar asociat lui E_1.

Fie expresiile cele mai simple care se pot forma cu operatorii binari cunoscuţi: $a + b$, $a – b$, $a*b$, a/b şi a^b.
Considerând *op* unul dintre operatorii binari +, –, *, /, ^, arborele binar asociat este cel din figura 5.35.

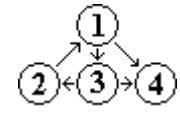


Figura 5.35

Se pot folosi şi operatorii unari + şi – , dar numai dacă se transformă în operatori binari printr-un artificiu aritmetic simplu, şi anume: se va scrie 0+a în loc de +a şi 0–a în loc de –a.
Deoarece nu toţi operatorii aritmetici sunt comutativi, asocierea dintre expresii aritmetice şi arbori este justificată, deoarece şi arborii binari sunt necomutativi.

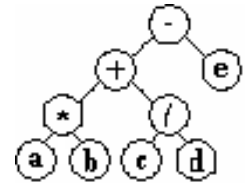


Figura 5.36

Fie expresia $E = a*b+c/d – e$. Să determinăm arborele binar asociat.
Etapele traducerii şi evaluarii acesteia de către compilator utilizând arborele binar asociat sunt:
1. Se construieşte arborele bazându-ne pe prioritatea operatorilor. Se observă că expresia iniţială se descompune în: $E_1 = a*b$, $E_2 = c/d$ şi $E_3 = E_1 + E_2 – e$.
Se obţine, corespunzător, arborele binar din figura 5.31, în care lui E_1 îi corespunde subarborele stânga al rădăcinii +, iar lui E_2 subarborele dreapt al rădăcinii +. Lui E_3 îi corespunde arborele ce are ca subarbore stâng subarborele ce are ca rădăcină operatorul +, iar subarborele drept este reperezentat de nodul terminal ce conţine valoarea *e*.

2. Se parcurge în manieră RSD arborele obtinut.
Acest arbore binar parcurs în maniera RSD conduce la forma poloneză a expresiei: RSD: – + * *a b* / *c d e*.

3. A treia etapă este tratarea formei poloneze astfel:
– se porneşte de la sfârşitul şirului de caractere către începutul acestuia;
– la prima pereche de operanzi precedaţi de un operator se face calculul, iar valoarea înlocuieşte subşirul alcătuit din cei doi operanzi şi operatorul folosit;

[1] Operator unar – acţionează asupra unui singur operand (cele care se referă la semnul unui operand: + şi –).
Operator binar – acţionează asupra a doi operanzi (operatori aritmetici: =, –, *, /).

– se reiau primii doi paşi până la epuizarea şirului.

Pentru exemplul considerat, vom avea: $- + * a\ b / c\ d\ e \rightarrow - + * a\ b\ r_1 e \rightarrow - + r_2 r_1\ e \rightarrow - r_3\ e \rightarrow E = r_4$

unde: $/ c\ d = r_1$
$* a\ b = r_2$
$+ r_2 r_1 = r_3$
$- r_3\ e = r_4$


exemplu

Parcurgând acest arbore în toate modurile posibile, se obţine:

În adâncime	$- + e * / a\ b\ c\ d$			
În lăţime	RSD:	$- + * a\ b / c\ d\ e$	RDS:	$- e + / d\ c * b\ a$
	SRD:	$a * b + c / d - e$	DRS:	$e - d / e c + b * a$
	SDR:	$a\ b * c\ d / + e -$	DSR:	$e\ d\ c / b\ a * + -$


observă

Analizând rezultatul obţinut la fiecare tip de parcurgere, se observă că parcurgerea în SRD (inordine) se apropie cel mai mult de aspectul iniţial al expresiei date.


rezolvă

1. Un arbore binar este memorat cu ajutorul vectorilor: tata = (0, 1, 1, 2, 3, 3, 5, 5) şi desc = (0, –1, 1, 1, –1, 1, –1, 1), unde prin –1 se consideră descendent stânga iar prin 1 se consideră descendent dreapta. Să se determina lista pentru parcurgerea arborelui în ordinea, pe rând: RSD, SRD şi SDR şi apoi să se determine lista nodurilor terminale.

2. Un arbore binar este memorat cu ajutorul vectorilor: tata = (2, 0, 1, 2, 1, 5, 4, 4, 5, 8) şi desc = (–1, 0, 1, 1, –1, –1, 1, –1, 1, 1), unde prin –1 se consideră descendent stânga iar prin 1 se consideră descendent dreapta. Să se determina lista pentru parcurgerea arborelui în ordinea, pe rând: DSR, DRS şi RDS şi apoi să se determine lista nodurilor terminale.

3. Fie G = (X, U) un arbore şi $G_1 = (X_1, U_1)$, $G_2 = (X_2, U_2)$ doi subarbori ai săi. Dacă $Y = X_1 \cap X_2$, să se arate că Y este mulţimea nodurilor unui subarbore al lui G.

4. Pentru un arbore binar dat, să se determine adâncimea sa (nivelul maxim).

5. Să se elaboreze un algoritm care să stabilească dacă un graf este sau nu arbore.

6. Se dă un arbore binar conţinând numere naturale nenule. Să se determine valoarea maximă memorată în nodurile sale.

7. Se numeşte ***diametru*** al unui arbore distanţa maximă dintre două noduri ale arborelui. Pentru un arbore dat, să se determine diametrul său.

8. Să se creeze arborele „genealogic" personal pe parcursul a trei sau patru generaţii, punându-se în nodul rădăcină prenumele propriu, iar ca descendenţi ai fiecărui nod, numele părinţilor. (Se va obţine un „arbore genealogic" asemănător cu cel în accepţiune clasică, deşi noţiunile de „strămoş" şi „urmaş" au fost inversate).

9. Folosind matricea de adiacenţă a unui graf, să se scrie un program care să decidă dacă graful este sau nu arbore.

10. Se consideră cuvântul INTELIGENTA şi un arbore precizat cu ajutorul vectorilor „taţi" dat mai jos. Aranjaţi literele lui în nodurile arborelui binar astfel încât prin parcurgerea SRD să se obţină cuvântul dat.

Nodul i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
TATA[i]	**0**	**1**	**2**	**2**	**4**	**1**	**6**	**7**	**6**	**7**	**9**

11. Se consideră un arbore pentru care vectorul de taţi este: TATA = (2, 0, 2, 5, 2, 5). Determinaţi numărul de niveluri al arborelui.

rezolvă

12. Să se deducă scrierea obişnuită a următoarelor expresii aritmetice, ştiind că toate constantele au o singură cifră:
a) $E_1 = - - * * * 5xyz * 4 * \wedge x5y * 3z$; b) $E_2 = \wedge x + a + * 2b * 3 \wedge c2$;
c) $E_3 = * + * + x * 3a - y12 + \wedge x25$.

13. Să se scrie fără paranteze următoarele expresii aritmetice:
a) $E_1 = (x + y)(y + z)(z + x)$; b) $E_2 = 3x^3z - 5x^2y^2 + 2xy^3$; c) $E_3 = \frac{x+y}{y+z} + \frac{y+z}{z+x} + \frac{z+x}{x+y}$.

5.5. Grafuri ORIENTATE

5.5.1. Noțiuni introductive

a) Adiacenţa, incidenţa, grad interior şi grad exterior

O foarte mare parte dintre noţiunile legate de grafurile orientate sunt analoage celor de la grafurile neorientate: *graf nul*, *graf parţial*, *subgraf*, *graf complet*.

Se numeşte **graf orientat** *o pereche ordonată de mulţimi* (X,U)*, unde* X *este o mulţime finită şi nevidă de elemente numite* **vârfuri**, *iar* U *este o mulţime de perechi ordonate (submulţimi cu două elemente) din mulţimea* X*, numite* **arce**.
Mulţimea X *se numeşte* **mulţimea vârfurilor** *grafului, iar mulţimea* U **mulţimea arcelor**.
Dacă mulţimea U *nu are proprietatea de simetrie, se spune că graful* $G = (X,U)$ *este* **graf orientat** *sau* **direcţionat** *sau* **digraf**.

Dacă $G = (X,U)$ este un graf *orientat*, muchiile se numesc **arce**, un arc **u** se notează cu (x_i,x_k) şi este o pereche *ordonată* de vârfuri distincte din U (adică $x_i,x_k \in U$ şi $x_i \neq x_k$).

Pentru un arc (x_i,x_k), x_i este numit **originea** arcului sau **extremitatea iniţială**, iar x_k este numit **vârful** arcului sau **extremitate finală**.

Se spune că x_k este **adiacent lui** x_i. Având arcul (x_i,x_k), se mai spune că acesta este **incident spre exterior cu** x_i şi **incident spre interior cu** x_k.

Dacă un vârf nu este nici extremitatea iniţială, nici extremitatea finală a vreunui arc, atunci el se numeşte **vârf izolat**.

observă

Ţinând seama de modul în care au fost definiţi arborii, se poate spune că *un arbore este un graf orientat*, datorită noţiunilor de ***rădăcină*** şi ***subarbore***.

Fie graful orientat $G = (X,U)$ *şi un vârf* $x \in X$*. Numărul arcelor de forma* (x,y) *se numeşte* **gradul exterior** *al vârfului* x *şi se notează* $d^+(x)$*, iar numărul arcelor de forma* (y, x) *se numeşte* **gradul interior** *al vârfului* x *şi se notează* $d^-(x)$.

Cu alte cuvinte, $d^+(x)$ reprezintă numărul arcelor ce „ies" din vârful x, iar $d^-(x)$ numărul arcelor ce „intră" în x.
Folosind aceste noţiuni, pentru un vârf al unui graf orientat putem defini:
– ***mulţimea succesorilor*** *lui* x: $U^+(x) = \{y \in X \mid (x,y) \in U\}$;
– ***mulţimea predecesorilor*** *lui* x: $U^-(x) = \{y \in X \mid (y,x) \in U\}$;
– ***mulţimea arcelor*** *ce ies din* x: $\omega^+(x) = \{u = (x,y) \mid u \in U\}$;
– ***mulţimea arcelor*** *ce intră în* x: $\omega^-(x) = \{u = (y,x) \mid u \in U\}$.

Dacă graful $G = (X,U)$ este orientat, există mai multe grafuri complete cu un număr dat de vârfuri, acestea deosebindu-se prin orientarea arcelor sau prin aceea că între două vârfuri oarecare există un arc sau două arce de sensuri contrare.

b) Lanţ, drum, drum elementar, circuit, lungime

Un **lanţ** *al unui graf orientat se defineşte ca un şir de arce* $L = [u_1, u_2, ..., u_p]$ *cu proprietatea că oricare arc* u_k *din acest şir are o extremitate comună cu* $u_{k=1}$ *şi cealaltă extremitate comună cu* u_{k+1}, *pentru orice* $k \in \{1, 2, ..., p-1\}$. *Numărul p se numeşte* **lungimea lanţului.**
Un lanţ al unui graf orientat $L = [u_1, u_2, ..., u_p]$ *se numeşte* **drum** *dacă toate arcele care îl compun au aceeaşi orientare, dată de sensul deplasării de la* x_0 *(extremitatea iniţială a lui* u_1*), la* x_p *(extremitatea finală a lui* u_p*). Vârfurile* x_0 *şi* x_p *se numesc* **extremităţile lanţului.**
Dacă vârfurile $x_0, x_1, ..., x_p$ *sunt distincte două câte două, drumul se numeşte* **elementar**. *În caz contrar, drumul se numeşte* **neelementar**.

Aşadar, un drum într-un graf orientat G = (*X,U*) este un şir de vârfuri notat $D = (x_0, x_1, ..., x_p)$, cu proprietatea că (x_0, x_1), (x_1, x_2), ..., $(x_{p-1}, x_p) \in U$, deci sunt arce ale grafului. Un drum se scrie punând în evidenţă fie succesiunea arcelor sale, fie succesiunea vârfurilor.

Prin **costul unui arc** *u se înţelege numărul nenegativ* $l(u) \geq 0$, *asociat arcului.*
Se numeşte **costul unui drum** *suma costurilor arcelor care îl compun.*
Un drum finit pentru care vârful său iniţial coincide cu vârful terminal, se numeşte **circuit**.
Un circuit se va numi **circuit simplu** *dacă foloseşte o singură dată fiecare arc şi se va numi* **elementar** *dacă trece o singură dată prin fiecare vârf.*

Noţiunea de conexitate asociată unui graf orientat este analoagă noţiunii similare de la grafuri neorientate, cu deosebirea că în aceste cazuri (grafuri orientate) conexitatea este legată de existenţa lanţurilor, şi nu a drumurilor.

c) Tare conexitate, componentă tare conexă

Un graf G = (*X,U*) *se numeşte* **tare conex** *dacă pentru orice pereche de vârfuri* $x, y \in X$ *există un* **drum** *de la x la y.*
O **componentă tare conexă** *a unui graf* G = (*X,U*) *este un subgraf* $G_1 = (X_1, Y_1)$ *al lui* G *tare conex şi maximal în raport cu această proprietate, adică oricare ar fi* $x \in X \setminus X_1$, *subgraful lui* G *generat de* $X1 \setminus \{x\}$ *nu mai este tare conex.*

Cel mai simplu exemplu de graf tare conex este graful asociat reţelei rutiere, unde se întâlnesc arce cu dublu sens şi arce cu sens unic.

Pentru a cerceta dacă un graf orientat este sau nu un graf tare conex şi, în caz afirmativ, pentru determinarea componentelor tare conexe, se foloseşte un procedeu analog cu cel utilizat la studierea conexităţii grafurilor neorientate. *Propunem aceasta ca exerciţiu.*

5.5.2. Metode de reprezentare

I. Matricea de adiacenţă sau **matricea asociată** grafului G = (*X,U*) A = (a_{ij}) este definită ţinând seama de sensul arcului. Astfel:

$$a_{ij} = \begin{cases} 1, \text{ pentru } (x_i, x_j) \in U \\ 0, \text{ pentru } (x_i, x_j) \notin U \end{cases}$$

Fie $X = \{1, 2, 3, 4, 5, 6, 7\}$ şi $U = \{(1,2), (1,6), (4,3), (4,5), (6,1), (6,2)\}$.
Graful G = (*X,U*), reprezentat în figura 5.37, este un graf orientat, cu şapte vârfuri şi şase arce.
Pentru graful *orientat* din figura 5.37, matricea de adiacenţă este:

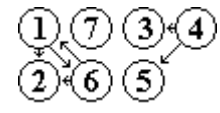


Figura 5.37

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Se observă că în cazul unui graf orientat ***matricea de adiacenţă nu este simetrică***. Analizând o matricea booleană ataşată unui graf, se observă că linia *i* reprezintă toate arcele care au extremitatea iniţială în vârful *i*, iar coloana *j* reprezintă arcele care au extremitatea finală în vârful *j*.
Reprezentarea prin matrice de adiacenţe permite un acces rapid la arcele grafului, fiind astfel utilă în rezolvarea problemelor ce implică testarea prezenţei unui anumit arc şi prelucrarea informaţiei ataşate acestuia. Metoda este însă dezavantajoasă deoarece, pentru un graf cu *n* vârfuri, numărul de arce este C_n^2, cu mult mai mic decât n^2, deci memoria necesară pentru a păstra matricea este ineficient utilizată.

II. Matricea vârfuri-arce sau **matricea de incidenţă** a grafului G este o matrice A = (a_{ij}) care se defineşte astfel:

$$a_{ij} = \begin{cases} 1, & \text{dacă } x_i \text{ este extremitatea initială a arcului } u_j \\ -1, & \text{dacă } x_i \text{ este extremitatea finală a arcului } u_j \\ 0, & \text{dacă } x_i \text{ nu este extremitatea a arcului } u_j \end{cases}$$

Fie graful orientat din figura 5.38. Matricea vârfuri-arce asociată acestui graf este:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 0 & -1 \end{pmatrix}$$

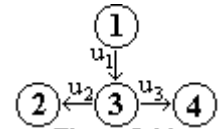


Figura 5.38

1. Pe fiecare coloană *j* există două elemente nenule: unul egal cu 1 (corespunzător extremităţii iniţiale a arcului u_j) şi unul egal cu –1 (ce corespunde extremităţii finale a arcului u_j);
2. Numărul de elemente 1 de pe linia *i* reprezintă gradul exterior al lui x_i, iar numărul de elemente –1 de pe linia *i* indică gradul interior al lui x_i; o linie cu toate elementele 0 corespunde unui vârf izolat.

III. Lista de adiacenţe. În cazul unui graf orientat se procedează identic, cu precizarea că în lista de adiacenţă L_k vor fi trecute doar acele vârfuri pentru care vârful **k** este extremitate iniţială, nu şi vârfurile pentru care este extremitate finală.

Fie graful din figura 5.39 cu *n* = 3 vârfuri (care sunt extremităţi iniţiale) şi *m* = 5 arce. Pentru fiecare vârf *k*, lista L_k a vecinilor săi este:

Vârful k	Lista L_k a vârfurilor adiacente cu vârful *k*
1	3,4
2	1
3	2,4
4	–

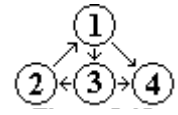


Figura 5.39

Pentru exemplul dat tabelului *T* (care are 2 linii şi 3 + 5 = 8 coloane) este:

j	**1**	**2**	**3**	**4**	L_1		L_2	L_3	
					5	**6**	**7**	**8**	**9**
T [1, *j*]- **vârfur**i	1	2	3	4	3	4	1	2	4
T [2, *j*]- **adrese de legătură**	5	7	8	-	6	0	0	9	0

IV. Matricea drumurilor grafului G este o matrice $B = (b_{ij})$ care se defineşte astfel:

$$b_{ij} = \begin{cases} 1, & \text{dacă există drum în G de la } x_i \text{ la } x_j \\ 0, & \text{în caz contrar} \end{cases}$$

exemplu

Fie graful din figura 5.39.

Matricea drumurilor asociată acestui graf este: $B = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$.

observă

Studiind matrice *B*, se constată că sunt două situaţii:
1. Dacă $b_{ii} = 1$ înseamnă că există un circuit care trece prin x_i.
2. Dacă linia *i* şi coloana *i* cuprind numai elemente zero, se deduce că x_i este un vârf izolat (nu există un drum care să plece din x_i sau să ajungă în x_i).

Un algoritm simplu de determinare a matricii drumurilor pornind de la matricea de adiacenţă este algoritmul Roy-Warshall, care constă în următoarea transformare: Un element $a[i,j] = 0$ al matricii devine 1 dacă există un vârf *k* astfel încât $a[i, k] = 1$ şi $a[k, j] = 1$, deci când există drum de la x_i la x_k şi drum de la x_k la x_j, cerinţă asigurată de condiţia: $a[i, j] := \min(a[i, k], a[k, j])$. Descrierea secvenţei este:

```
for k=1 to n
  for i=1 to n
    if i<>k
      then
        for j=1 to n
          if j<>k
            then
              if a[i,j]=0
                then
                  a[i,j]:=min(a[i,k],a[k,j])
              endif
          endif
        endfor
    endif
  endfor
endfor
```

Caracteristicile grafurilor orientate

I. Într-un **graf orientat** adiacenţele se numesc **arce**.
II. În graful orientat G=(*X*,*U*), *U* **nu** are proprietatea de **simetrie.**
III. Gradul **interior** al unui vârf este dat de numărul arcelor care **intră** în el.
IV. Gradul **exterior** al unui vârf este dat de numărul arcelor care **ies** din el.
V. În matricea de adiacenţă suma elementelor unei **linii** reprezintă gradul **exterior**.
VI. În matricea de adiacenţă suma elementelor unei **coloane** reprezintă gradul **interior**.
VII. **Drumul** între două noduri *x* şi *y* este alcătuit din lanţul orientat de la *x* la *y*.
VIII. Un graf orientat conex este **tare conex** dacă între oricare două vârfuri există un drum.

1. Fie graful G = (*X*,*U*) cu *n* = 9 vârfuri şi nouă arce definit astfel:
X = {1, 2, 3, 4, 5, 6, 7, 8, 9} şi *U* = {(2, 1), (1, 6), (2, 5), (2, 3), (3, 4), (4, 6), (5, 7), (4, 8), (8, 9)}.
Care sunt vârfurile legate de vârful 2 prin drumuri de lungime egală cu a drumului minim dintre vârfurile 2 şi 6 ?

2. Fie graful G = (*X*,*U*) cu *n* = 7 vârfuri şi nouă muchii, definit astfel:
X = {1, 2, 3, 4, 5, 6, 7} şi *U* = {(1, 2), (1, 3), (2, 3), (2, 6), (3, 5), (3, 6), (5, 4), (5, 1), (6, 7), (7, 3)}.
Identificaţi vârfurile care fac parte din circuite de lungimi pare.

3. Se consideră graful orientat G = (*X*,*U*), unde card *X* = 5 şi *U* = {(1, 2), (4, 1), (2, 3), (2, 5)}. Identificaţi numărul minim de arce care trebuie adăugate pentru ca orice vârf să aibă gradul interior egal cu gradul exterior.

4. Folosind reprezentarea prin liste de adiacenţă a unui graf orientat, să se determine gradul interior şi gradul exterior al fiecărui vârf.

5. Se consideră o mulţime formată din *n* persoane, în care fiecare persoană se cunoaşte pe sine şi, eventual, alte persoane. Stiind că o persoană aparţine unui singur grup şi că relaţia „*x* cunoaşte pe *y*" nu este în mod normal nici simetrică, nici tranzitivă, să se formeze grupuri în care fiecare persoană cunoaşte toate celelalte persoane din grup.

6. Să se calculeze:
a) numărul grafurilor orientate cu *n* vârfuri;
b) numărul grafurilor parţiale ale unui graf cu *m* arce;
c) numărul minim de arce ale unui graf orientat conex cu *n* vârfuri.

7. Numim **transpusul** unui graf G = (*X*,*U*), graful G' care are aceeaşi mulţime de vârfuri, arcele sale fiind arcele grafului G, dar având sens opus. Dându-se G prin matricea de adiacenţă sau prin liste de adiacenţă, să se determine în fiecare caz transpusul grafului dat.

8. Să se arate că dacă într-un graf există un drum între două vârfuri distincte *x* şi *y*, atunci există un drum elementar de la *x* la *y*.

9. Fie graful orientat G = (*X*,*U*) cu *m* arce. Atunci $\sum_{k=1}^{n} d^{-}\left(x_k\right) = \sum_{k=1}^{n} d^{+}\left(x_k\right) = m$.

PARTEA a II-a

Tehnici de structurare a prelucrărilor

Capitolul 6 Subprograme

În acest capitol veţi învăţa despre:

- Descompunerea unei probleme în subprobleme şi ierarhizarea acestora
- Tehnica modularităţii aplicată în rezolvarea problemelor
- Organizarea prelucrărilor datelor în subprograme
- Proiectarea subprogramelor

6.1. Probleme şi subprobleme

Rezolvarea oricărei probleme complexe se poate descompune în grupuri de acţiuni care rezolvă probleme cunoscute, elementare sau de complexitate mai mică.

Aceste grupuri de acţiuni vor deveni module independente care, asamblate apoi, construiesc rezolvarea problemei. Rezultă că modulele trebuie să fie subordonate unui modul *principal* care are rolul de coordonare a prelucrării subproblemelor.

Modulul principal conţine un minimum de comenzi necesare prelucrărilor de ansamblu, legate de comunicarea cu utilizatorul şi de activarea „subordonaţilor".

Fiecare modul subordonat poate interveni în rezolvare o dată sau de mai multe ori, sau poate fi folosit numai în funcţie de îndeplinirea anumitor condiţii.

Apoi, în faza de programare, modulul principal va deveni ***program principal***, iar modulele subordonate vor deveni ***subprograme***.


exemplu

1. Se doreşte crearea unui program care să lucreze cu **fracţii ordinare**. Spre exemplu, fie *n* fracţii ordinare pentru care se calculează suma ca fracţie ireductibilă. Deoarece compilatorul nu cunoaşte tipul de date *fracţie ordinară*, datele despre o fracţie oarecare se citesc ca pereche de numere întregi *(a,b)*, unde *a* este numărătorul, iar *b* este numitorul.

Este evident că rezolvarea nu va avea în vedere transformarea fiecărei fracţii ordinare în fracţie zecimală, însumarea lor şi apoi revenirea la fracţie ordinară, deoarece pot apărea erori de reprezentare la împărţirile în mulţimea numerelor reale.

Pentru rezolvarea problemei este necesar să elaborăm o *listă de prelucrări*:

– se va citi **`n`**, numărul de fracţii;

– pentru fiecare pereche **`(a,b)`** citită se vor realiza:

– adunarea ca fracţie ordinară cu fracţia sumă de până atunci;

– aducerea la forma ireductibilă a sumei nou obţinute;

Pentru a însuma două fracţii, $\frac{a}{b}+\frac{b}{c}$ sunt necesare calculele:

– **`c.m.m.d.c.(b,d)`**;

– **`c.m.m.m.c (b,d)`**, prin împărţirea produsului $b\times d$ la c.m.m.d.c. (*b*, *d*),

pentru ca apoi să efectuam suma lor astfel:

$$\frac{a\times\left(d:cmmdc\left(b,d\right)\right)+c\times\left(b:cmmdc\left(b,d\right)\right)}{cmmmc\left(b,d\right)}.$$

Valoarea iniţială a fracţiei sumă, $\frac{sa}{sb}$, va fi $\frac{0}{1}$, adică raportul dintre elementul neutru al adunării şi elementul neutru al înmulţirii.

Se conturează trei module: **cmmdc, cmmmc** şi modulul **principal**.

```
MODUL PRINCIPAL
Citeste valoarea lui n cat timp aceasta este mai mica decât 2
                        (nu are sens problema pentru una sau zero fractii).
Initializeaza fractia suma: sa←0 si sb←1
pentru fiecare din cele n fractii executa:
      citeste o pereche de numere întregi:a si b
      aux ← cmmmc(sb,b)
      sa ← sa *  (b/cmmdc(sb,b) + a* (sb/cmmdc(sb,b))
      sb ← aux
      aux ←  cmmdc(sa,sb)
      sa ← sa/ aux
      sb ← sb/ aux
reia
scrie „Fractia suma=", sa, „/",sb
stop
```

În rezolvarea acestei probleme, modulul **cmmdc** este folosit şi de către modulul **cmmmc** şi de către modulul **principal**.

2. Fie patru puncte în plan, ***A***, ***B***, ***C*** şi ***P***, date prin coordonatele lor, în primul cadran al axelor carteziene. Punctele ***A***, ***B*** şi ***C*** alcătuiesc un triunghi. Se cere să se stabilească poziţia punctului ***P*** faţă de triunghi şi anume: exterior, pe contur sau interior triunghiului.

În figura 6.1 s-au ales următoarele puncte:

A(1, 1), *B*(5, 2), *C*(3, 4) *şi P*(6, 4).

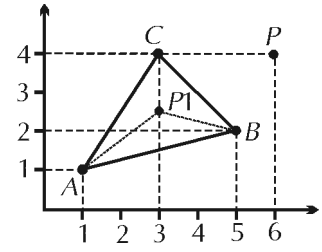


Figura 6.1

După cum se observă, punctul este exterior triunghiului.

Dacă alegem un alt punct, *P1*(3,2.5), acesta este interior triunghiului.

Cel mai simplu mod de a afla poziţia punctului faţă de triunghi este de a calcula ariile triunghiurilor formate de punct cu câte două vârfuri ale triunghiului şi de a compara apoi suma acestor trei arii cu aria triunghiului.

Dacă se obţine egalitate, punctul este interior triunghiului.

Dacă suma ariilor celor trei triunghiuri este mai mare decât aria triunghiului, punctul este exterior triunghiului *ABC,* iar dacă este egală, atunci punctul este interior sau pe contur.

Pentru a rezolva această problemă în cazul general, pentru oricare coordonate ale vârfurilor *A*, *B*, *C* ale triunghiului şi ale punctului *P*, dar în cadranul I, se observă că se vor folosi de mai multe ori unele *prelucrări*:

– verificarea apartenenţei unui punct la cadranul I: fiecăruia dintre cele patru puncte îi va fi aplicată o verificare a coordonatelor (*x* – abscisa; *y* – ordonata) – dacă sunt pozitive simultan;

– calculul lungimii unui segment când se cunosc coordonatele capetelor lui, notate, în general: $M(x_1,y_1)$ şi $N(x_2,y_2)$ prin formula dedusă din teorema lui Pitagora: $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$

– calculul ariei unui triunghi, pentru fiecare triunghi din cele formate de punctul *P* cu câte două vărfuri şi aria triunghului *ABC*, pe baza formulei de calcul a ariei când se cunosc laturile, *a*, *b*, *c* şi *p* – semiperimetrul triunghiului: $\sqrt{p(p-a)(p-b)(p-c)}$.

Aceste trei prelucrări vor fi acţionate la momentul potrivit din modulul principal, care va verifica apoi relaţia dintre arii pentru a trage concluzia.

Convenim să numim cele trei prelucrări menţionate mai sus astfel: **verif**, **latura** şi **aria**. Atunci, modulul principal va avea următoarele acţiuni:

```
MODUL PRINCIPAL
Citeste coordonatele: xa,ya,xb,yb,xc,yc,xp,yp;
Daca verif(xa,ya) si verif(xb,yb) si verif(xc,yc) si verif(xp,yp)
atunci
        a=latura(xb,yb,xc,yc)
        b=latura(xa,ya,xc,yc);
        c=latura(xa,ya,xb,yb);
        pa=latura(xp,yp,xa,ya);
        pb=latura(xp,yp, xb,yb);
        pc=latura(xp,yp,xc,yc);
        s1=arie(pa,pb,c);
        s2=arie(pa,pc,b);
        s3=arie(pb,pc,a);
        s=arie(a,b,c);
        daca s=s1+s2+s3
            atunci scrie „Punctul este interior triunghiului”
            altfel scrie „Punctul este exterior triunghiului”
altfel scrie „Nu toate punctele sunt in primul cadran”
Stop
```

3. *Enunţ*. Fie *n* numere naturale, $n \leq 100$. Să se afle numărul de zerouri cu care se sfârşeşte produsul celor *n* numere, fără a efectua produsul.

Exemplu numeric. Fie $n = 4$ şi cele patru numere citite: 12, 7, 35, 25. Rezultă un produs cu două zerouri finale.

Rezolvare. Problema nu se poate rezolva prin simpla efectuare a produsului celor *n* numere, deoarece acest produs se poate să nu încapă în reprezentarea internă a unui întreg. Spre exemplu, dacă presupunem $n = 100$ şi fiecare dintre cele 100 de numere are două cifre, atunci produsul ar avea maximum 200 de cifre, o valoare imposibil de stocat fără erori de trunchiere, într-un tip de dată numerică elementară recunoscut de limbajul de programare (int, long, unsigned, float, double).

Rezolvarea constă în determinarea numărului de perechi de divizori 2 şi 5 pe care le conţine produsul celor *n* numere. Pentru exemplul luat, cele patru numere participă în total cu doi de doi (din 12) şi trei de cinci (din 35 şi 25). Se pot face astfel două perechi de factori care să genereze valoarea 10.

Rezultă următoarea listă de acţiuni:

- citirea numărului *n* şi verificarea acestuia – dacă se află în limitele date în enunţ: $2 \leq n \leq 100$ (problema are sens de la două numere în sus);
- citirea succesivă a *n* numere naturale, în variabila *a*, cu verificarea încadrării fiecăruia între limitele de număr natural, conform tipului de reprezentare *unsigned*: $0 \leq a \leq 65535$;
- aflarea numărului de apariţii ale divizorului 2 pentru fiecare număr *a* dintre cele *n* numere;
- alfarea numărului de apariţii ale divizorului 5 pentru fiecare număr *a* dintre cele *n* numere;
- adunarea numerelor de apariţii aflate la sumele totale de divizori 2, în s2 şi, respectiv, de divizori 5, în s5;
- compararea celor două sume, alegerea minimului şi afişarea rezultatului.

Din lista de mai sus se pot determina următoarele grupări de acţiuni generale (modulelor subordonate li s-au dat nume în paranteze):

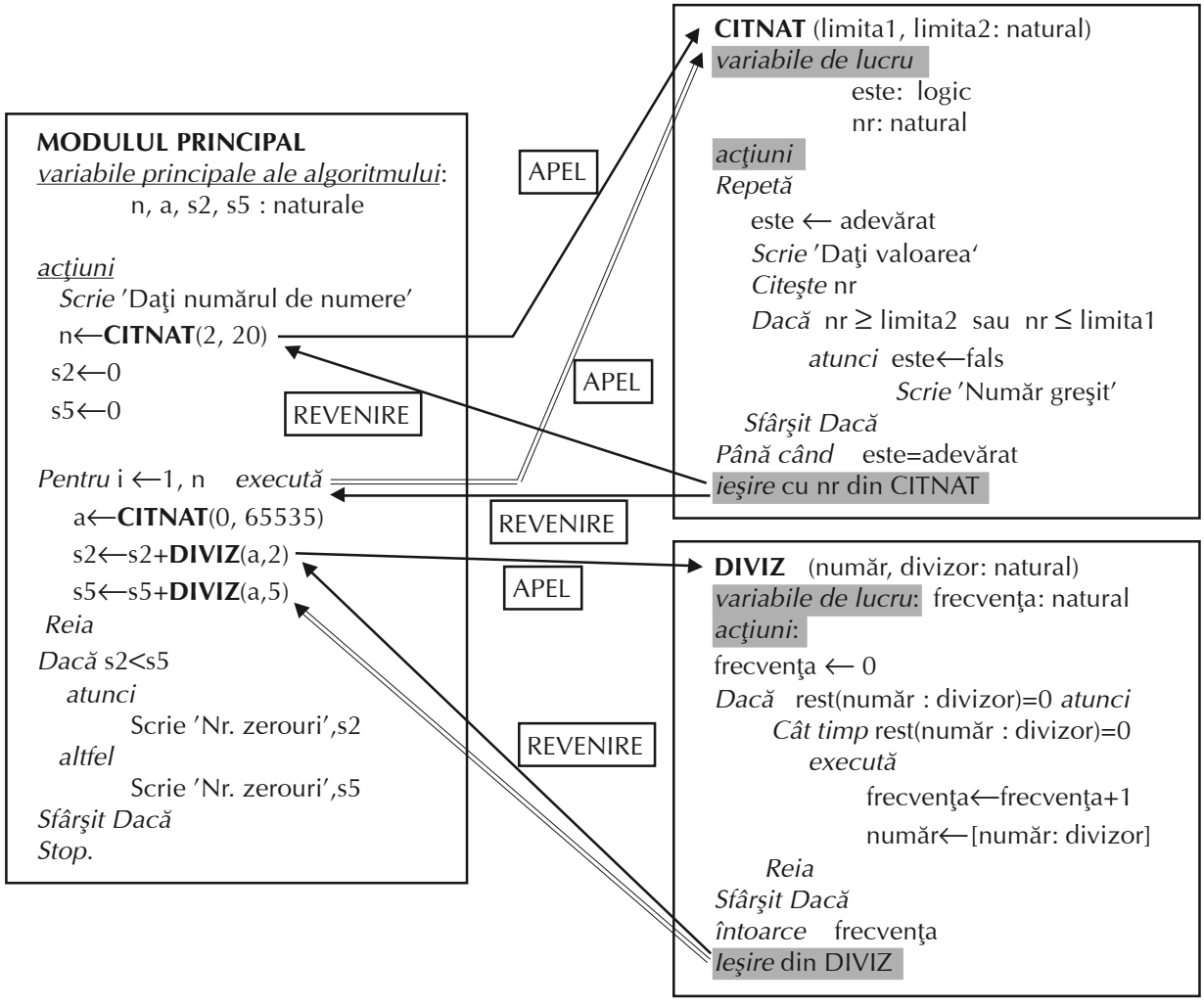
- citirea unui număr natural cu verificarea încadrării lui între anumite limite valorice (**CITNAT**);
- determinarea divizibilităţii unui număr natural dat cu un alt număr şi aflarea numărului de apariţii ale acelui divizor în numărul dat (**DIVIZ**);
- cumularea numărului de apariţii aflat, care se va realiza în modulul principal;
- compararea şi afişarea rezultatului comparării, care se va realiza în modulul principal.

Toată structura prelucrării va fi compusă din trei module: modulul **PRINCIPAL**, modulul **CITNAT** şi modulul **DIVIZ** , iar modulele **CITNAT** şi **DIVIZ** vor fi folosite de câte două ori.

Modulele trebuie să se asambleze perfect în noua construcţie realizată cu ele. Astfel, vor trebui asigurate legăturile, „lianţii". Pentru acest lucru vor fi folosite *variabile de legătură sau de comunicare*, numite **parametri**. Fiecare modul subordonat întoarce un răspuns modulului principal.

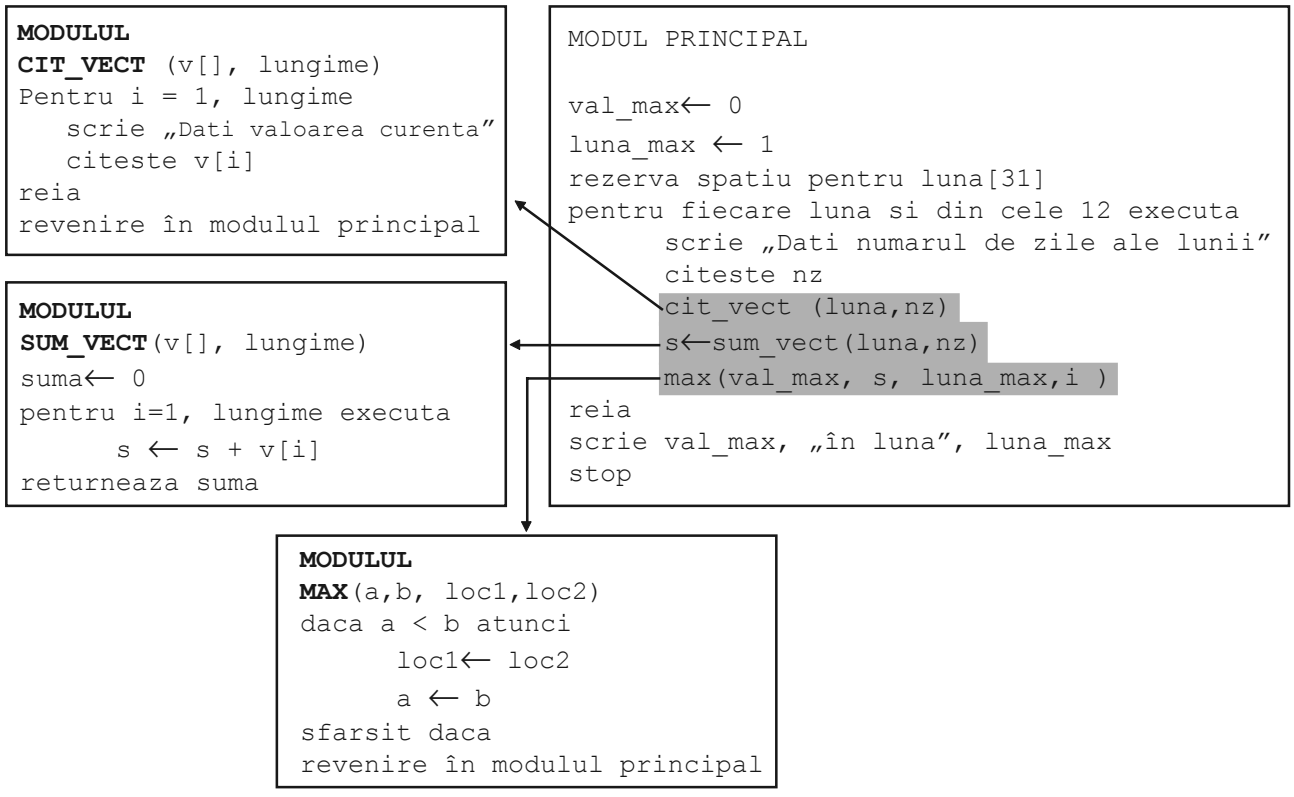
• În cazul modulului **CITNAT**, legăturile sunt realizate prin variabilele: *număr, limita1 şi limita2*. În momentul asamblării lui în structură, la prima utilizare (**activare**) se vor face conexiunile: *număr* ↔ *n, limita1* ↔ *2, limita2* ↔ *100*, iar la a doua utilizare : *număr* ↔ *a, limita1* ↔ *0 şi limita2* ↔ *65535*. La fiecare dintre utilizări, răspunsul dat de modul este valoarea citită în variabila *nr*, pe care o comunică, la sfârşit, modulului principal.

• Pentru modulul **DIVIZ**, lianţii sunt asiguraţi de variabilele: *număr* şi *divizori*. Astfel, la prima lui utilizare, legăturile care se fac sunt: *număr* ↔ *a*, *divizor* ↔ *2*, iar la a doua utilizare: *număr* ↔*a*, *divizor* ↔ *5*. La fiecare dintre cele două utilizări, răspunsul modulului este frecvenţa calculată pentru divizorul încercat.



4. Se consideră datele despre vânzările zilnice efectuate la un magazin, pe parcursul unei luni. Se doreşte stabilirea lunii din cadrul anului în care vânzările au atins valoarea maximă.
Rezolvarea problemei necesită următoarele grupări de *prelucrări* pe module:
– citirea datelor despre valoarea vânzărilor zilnice efectuate într-o lună şi memorarea lor într-un vector (**CIT_VECT**);

– însumarea valorilor vânzărilor zilnice dintr-o lună, pentru a obţine totalul pe acea lună (**SUM_VECT**);
– stabilirea valorii maxime faţă de lunile precedente (**MAX**);
– modulul principal.



În rezolvarea de mai sus, în modulul **SUM_VECT**, s-a utilizat notarea `v[]` pentru a sugera că modulului i se transmite un vector, iar nu o variabilă simplă. La apel se va transmite lui `v` numărul de elemente alocate în `luna`. În modulul **MAX**, se stabileşte dacă `b` este mai mare ca `a` şi dacă da, `a` este înlocuit cu valoarea din `b`, iar `loc1` cu valoarea din `loc2`. Astfel, în modulul principal se vor prelua: în `val_max` valoarea din `s`, dacă aceasta a fost mai mare, iar în variabila `luna_max` se va muta valoarea lui `i`, deoarece luna curentă, `i`, produce o sumă mai mare decât până atunci.

1. Stabiliţi modulele necesare pentru calculul câtului a două numere complexe z1=a+ib şi z2=c+id, pentru care se citesc din mediul de intrare valorile reale *a, b, c, d*. Construiţi modulul principal în exprimare pseudocod şi verificaţi dacă structura lui este valabilă şi în cazul în care s-ar împărţi, pe rând, *n* numere complexe: primul la al doilea, câtul la al treilea şi aşa mai departe.
2. Stabiliţi modulele necesare pentru a verifica dacă două drepte sunt paralele. Dreptele sunt date prin ecuaţiile lor carteziene generale: $ax + by + c = 0$ şi $dx + ey + f = 0$, adică, din mediul de intrare se citesc coeficienţii. Construiţi modulul principal în exprimare pseudocod şi verificaţi dacă el este valabil şi pentru a testa paralelismul a *n* drepte.
3. Stabiliţi modulele care intervin în situaţia în care se citesc, pe rând, *n* vectori de numere naturale, se elimină din fiecare vector valorile multiple (transformarea vectorului în mulţime) şi se determină, progresiv, mulţimea lor intersecţie. Construiţi modulul principal în exprimare pseudocod. Exemplu: Fie $a = (1, 3, 6, 3, 7)$, $b = (2, 3, 6)$ şi $c = (7, 2, 3, 8, 3, 9, 6, 6)$. După eliminarea valorilor multiple din a şi b, IN=$a \cap b \Rightarrow (1, 3, 6, 7) \cap (2, 3, 6) = (3, 6)$. Apoi IN $\cap (7, 2, 3, 8, 9, 6) \Rightarrow$ IN = (3, 6).
4. Stabiliţi modulele care intervin în situaţia în care se doreşte determinarea diametrului minim al cercului care ar delimita în interior toate cele *n* puncte date prin coordonatele lor carteziene

(Indicaţie: se determină perechea de puncte între care distanţa este maximă). Construiţi modulul principal în exprimare pseudocod.

5. Stabiliţi care dintre modulele din coloana B se folosesc în prelucrările din coloana A

A – prelucrări	**B** – module
1. Suma termenilor unei progresii aritmetice pentru care se cunosc n, r şi a0	1. `SUM_VECT(v[],lungime)`
2. Calculul numărului de permutări	2. `CIT_VECT(v[],lungime)`
3. Calculul numărului de combinări $C_n^k = \frac{n!}{k! \times (n-k)!}$	3. `CITNAT(lim1,lim2,nr)`
4. Valoarea absolută a unui număr real	4. `CALCUL(a,b)`
5. Rădăcina ecuaţiei de gradul I	5. `CALCUL(x)`
6. Construirea unei liste înlănţuite de numere naturale	6. `FACT(p)`
7. Calculul gradului fiecărui vârf al unui graf neorientat cu *n* vârfuri şi matricea de adiacenţă A	7. `CALCUL(a,b,c)`

6.2. Subprograme

6.2.1. Exemplificarea structurii unui program cu subprograme în C/C++

Realizarea în limbaj de programare a problemei din exemplul 2 al paragrafului 6.1 este dată pentru a pune în evidenţă:

– declararea şi definirea modulelor ca subprograme;

– activarea modulelor;

– definirea şi funcţionarea parametrilor („lianţilor" între modulul subordonat şi cel principal).

Modulele proiectate apar ca subprograme ***descrise*** înaintea modulului (programului) principal care în **C/C++** este funcţia `main()`.

```
#include <iostream.h>
// zerouri_coada_produs;

unsigned citnat(unsigned limita1,unsigned
                                limita2)
{unsigned nr;
  do {
     cout<<"Introduceti numar:\n ";
     cin>>nr;
    } while ( ! (nr >= limita1 &&  nr<=
limita2));
return (nr);
}
unsigned diviz(unsigned numar,unsigned
divizor)
{unsigned frecventa=0;
 if (numar % divizor==0)
       while (numar % divizor==0)
          {frecventa ++;
            numar / = divizor;
          }
return frecventa;
}
```

```
//modulul principal

void main()
{
unsigned n,a,i,s2,s5;
cout<<"Dati numarul de numere \n";
   n=citnat(2,20);
   s2=0; s5=0;
   cout<<"Introduceti, pe rand, numerele
\n";

   for (i=1; i<=n; i++)
      { a=citnat(0,65535);
        s2 + = diviz(a,2);
        s5 + = diviz(a,5);
      }

if (s2<s5)
    cout<<"Produsul are "<<s2<<"zerouri";
   else
     cout<<"Produsul are "<<s5<<" zerouri";
}
```

6.2.2. Definiția și caracteristicile subprogramelor

Subprogramul este un **bloc de comenzi simple** care realizează o anumită **prelucrare generală** şi care este **distinct** descris în raport cu comenzile care îl utilizează.

În C/C++ subprogramele se numesc funcţii.

Caracteristicile subprogramului

I. Subprogramele reprezintă acele părţi ale unui program care corespund **modulelor de prelucrări** definite în structura rezolvării unei probleme complexe.

II. În cazul în care nu s-ar lucra cu programul structurat pe module (funcţii), acţiunile pe care ar trebui să le conţină un subprogram vor trebui programate **de mai multe ori**, **în mai multe locuri**, pentru a prelucra acelaşi tip de date, dar pentru alte variabile.

III. Acţiunile reflectate în subprogram definesc **rezolvarea unei subprobleme** necesare problemei date.

IV. Subprogramul are un **grad de generalitate maxim** pentru subproblema pe care o rezolvă. El poate fi utilizat în mai multe aplicaţii, programe, mai ales dacă este înglobat într-o bibliotecă de funcţii. Spre exemplu, subrogramul *sqrt* oferit din biblioteca standard a C/C++ în fişierul **`math.h`**.

V. Subprogramele se pot identifica în cadrul structurii printr-un **nume** care este utlizat atât la definirea lor, cât şi la activarea (apelul) lor.

VI. **Structura** unui subprogram cuprinde două părţi: **antetul** (linia de recunoaştere) şi **corpul** subprogramului, figurat ca instrucţiune compusă.

VII. **Corpul** subprogramului conţine *constantele* şi *variabilele* pentru lucrul intern şi *acţiunile* pe care le realizează acesta, cuprinse toate între acoladele de definire a unei instrucţiuni compuse.

VIII. Pentru asigurarea asamblării corecte a subprogramului în structura programului trebuie definite legăturile, adică **parametrii de comunicare** între subprogram şi restul programului (există situaţii particulare în care nu se utilizează parametri şi pentru care specificaţiile de comunicare se supun unor convenţii cu caracter restrictiv).

IX. **Metoda TOP-DOWN** este metoda dezvoltării pas cu pas a programelor, descompunând rezolvarea unei probleme pe niveluri de complexitate până la nivelul de operaţii elementare. Ea impune alcătuirea de subprograme prin care prelucrarea este mai uşor de realizat corect şi de urmărit apoi pentru eventuala depanare a programului.

Urmărind aceste caracteristici în programul dat pentru exemplul 2 din paragraful 1, observăm:
– modulele **CITNAT** şi **DIVIZ** au fiecare un titlu, sau linie de recunoaştere, marcată în text prin culoare;
– în titlu se regăsesc numele şi parametrii, puşi între paranteze – de exemplu, la **CITNAT** apar ca parametri *limita1 şi limita2*. Asemănător şi pentru **DIVIZ**;
– blocul modulelor menţionate conţine, pentru fiecare, o listă de variabile de lucru intern şi acţiunile de realizat.

Identificaţi caracteristicile date mai sus în exemplele 1, 3 şi 4 din paragraful 6.1.

6.2.3. Ierarhizarea subprogramelor. Elementele unui subprogram

a) Ierarhizarea modulelor

În ansamblul de module generate prin descompunerea problemei trebuie stabilită clar relaţia ierarhică directă între oricare *două* module.

Astfel, pot fi:

- module subordonate direct altor module; acestea se numesc module **apelate** (puse în execuţie de către modulele cărora li se subordonează);

– module care au în subordine directă alte module; acestea se numesc module **apelante** (care pun în lucru modulele subordonate);

– module independente între ele; nu există o relaţie ierarhică directă între ele.

În exemplul pentru calculul sumei celor *n* fracţii ordinare (exemplul 1 din paragraful 6.1) a fost nevoie de proiectarea a trei module între care relaţiile ierarhice s-ar figura astfel:

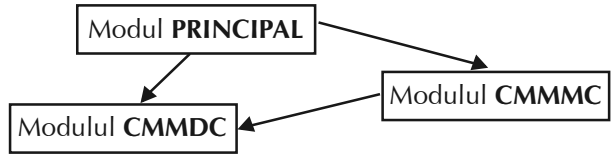


Figura 6.2

observă

Modulul **CMMMC** este un modul ***apelat*** în relaţia cu modulul **PRINCIPAL**, dar este un modul ***apelant*** în relaţia cu modulul **CMMDC**. Modulul **CMMDC** este subordonat atât modulului principal, cât şi modulului **CMMMC**. El este numai modul ***apelat***.

rezolvă

Pentru fiecare dintre exemplele 2, 3 şi 4 din paragraful 6.1 realizaţi câte o listă a statutului ierarhic al modulelor desemnate şi câte un desen asemănător celui din figura 6.2.

Concluzie. Structura ierarhică determinată pentru module va fi şi structura în care se vor găsi subprogramele (funcţiile) corespunzătoare. Întotdeauna modulul principal este un modul ***apelant***.

b) Funcţionarea relaţiei modul apelant – modul apelat

Între cele două tipuri de module activitatea se desfăşoară în trei faze:

– faza1 – modulul *apelant* activează, apelează pe cel subordonat şi îi predă controlul prelucrării;

– faza 2 – modulul *apelat* lucrează şi modulul *apelant* îl aşteaptă să termine activitatea;

– faza 3 – modulul *apelat* predă controlul înapoi modulului *apelant* şi încetează activitatea, iar modulul *apelant* reia lucrul.

Graficul fazelor de lucru pentru cele două tipuri de module este desenat în figura 6.3.

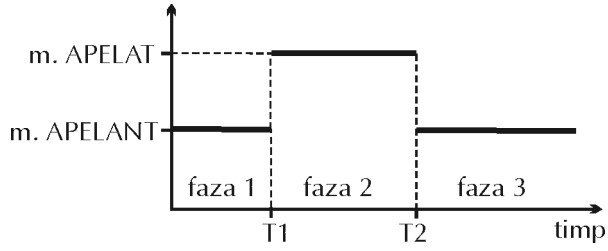


Figura 6.3

c) Elementele constructive ale unui subprogram (funcţie)

– **antetul** sau titlul subprogramului, necesar recunoaşterii de către compilator;

– **corpul** subprogramului, care este reprezentat printr-o instrucţiune compusă.

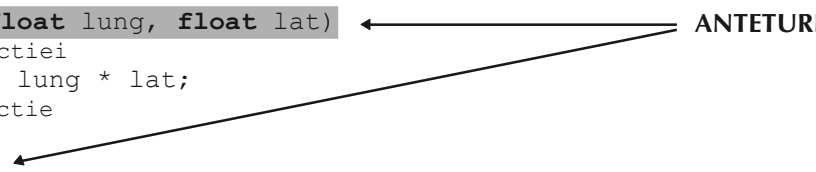
exemplu

În programul de mai jos se calculează perimetrul şi aria unui dreptunghi, utilizându-se un modul pentru calculul perimetrului şi un modul pentru calculul ariei. Modulul principal are în sarcină comunicarea cu utilizatorul pentru datele care intră şi ies din program şi activează modulele subordonate la momentul potrivit.

```
#include<iostream.h>
float perim(float lung, float lat)
{ //incepe corpul functiei
    return 2*(lung+lat);
```

ANTET

```
//sfarsitul corpului functiei
}
float arie(float lung, float lat)
{//corpul functiei
      return lung * lat;
//sf.corp functie
}
void main()
{//corpul functiei
  float a, b;
  cout<<"Dati lungimea si latimea dreptunghiului ";
  cin>>a>>b ;
  cout<< "Perimetrul este : "<<perim(a,b)<<endl ;
  cout<<"Aria este " <<arie(a,b) ;
//sfarsitul corpului functiei
}
```



6.2.4. Clasificarea subprogramelor (funcţiilor)

a) Modalitatea de returnare a rezultatului

După *modul în care sunt returnate rezultatele* către modulele apelante, subprogramele se clasifică în două grupe. În consecinţă, diferă şi *tipul de apel* (activare):

- Subprograme de tip ***funcţie operand***, care au ca sarcină principală calcularea şi returnarea *unui rezultat* într-o variabilă ce reprezintă chiar *numele subprogramului*.

Un astfel de subprogram acţionează ca o aplicaţie matematică definită pe o mulţime *domeniu* şi cu valori într-o mulţime *codomeniu*.

Subprogramul se activează prin apariţia lui ca *termen* într-o expresie formulată de către modulul apelant (de exemplu, utilizarea funcţiei *sqrt(x)* din biblioteca *math*).

- Subprograme de tip ***funcţie procedurală***, care calculează şi *returnează mai multe valori* prin intermediul parametrilor *sau nu returnează nimic*.

Un astfel de subprogram se activează prin apariţia numelui lui ca instrucţiune de sine stătătoare (de exemplu, apelul *clrscr()* din biblioteca *conio*).

În exemplul 2 din paragraful 6.1, **CITNAT** şi **DIVIZ** au caracteristica de funcţii operand. Ele calculează şi returnează o valoare în variabila numelui funcţiei:

– **CITNAT** face calcule logice asupra limitelor valorice ale unui număr natural citit şi returnează modulului apelant numărul citit în momentul în care acesta este corect, în variabila cu acelaşi nume ca şi modulul – **CITNAT**;

– **DIVIZ** face calcule numerice şi logice (testarea restului şi numărarea apariţiilor divizorului), returnează modulului apelant frecvenţa găsită, prin intermediul variabilei **DIVIZ**.

Stabiliţi tipul fiecărui modul care intervine în cadrul exemplelor 1, 3 şi 4 din paragraful 6.1.

b) Originea subprogramului

Subprogramele se clasifică în două grupe, în funcţie de *originea (autorul) lor*:

• Subprograme ***standard*** (predefinite şi puse la dispoziţie în bibliotecile limbajului de programare). Aceste subprograme nu trebuie declarate sau definite de către programator, deoarece acest lucru este făcut în cadrul bibliotecii de care aparţin. Este suficient ca pentru ele să fie scris corect apelul, în una din variantele:

– ca operand, într-o expresie;

– simplu, prin menţionarea numelui urmat de parantezele cu sau fără parametri, după cum a fost definit în biblioteca din care este luat.

Pentru ca ele să poată fi utilizate, în program trebuie menţionate bibliotecile din care fac parte, dacă bibliotecile respective nu se încarcă automat în memorie odată cu încărcarea programului.

• Subprograme ***nestandard***, sau create de utilizatorul limbajului de programare. Aceste subprograme trebuie definite în program.

Modulele **CITNAT** şi **DIVIZ** definite în exemplul 2 din paragraful 6.1 fac parte din grupa subprogramelor nestandard.

Stabiliţi originea fiecărui modul care intervine în cadrul exemplelor 1, 3 şi 4 din paragraful 6.1.

REGULĂ : niciun subprogram nu poate fi utilizat înainte ca el să fie declarat în unul din modurile:
– implicit – prin biblioteca din care face parte;
– explicit – direct în program, înaintea modulului apelant.

6.2.5. Proiectarea şi utilizarea unui subprogram nestandard

6.2.5.1. Definirea subprogramului

Definirea unui subprogram înseamnă scrierea antetului şi a corpului subprogramului.

a) Definirea antetului

Definirea antetului unei ***funcţii operand***:	Definirea antetului unei ***funcţii procedurale***:
tip_rezultat nume_functie(lista parametrilor) codomeniu domeniu	**void** nume_functie(lista de parametri)

• *Tip rezultat* poate fi: un tip standard, o adresă a unui tip definit anterior (printre care pot fi tablouri, alte funcţii), o structură de tip articol.

• *Lista de parametri* are sintaxa:

$$tip_1\ \ param_1\ ,\ tip_2\ param_2\ ,...,\ tip_n\ \ param_n$$

unde:

- tip_i, dintre oricare din listă, este un tip cunoscut dintr-o definire anterioară. Dacă subprogramul nu are parametri, atunci apar numai parantezele sau cuvântul **void**, adică tip nedefinit[1], între paranteze;
- $param_i$, oricare din listă, este un nume de variabilă.

ATENŢIE!
Dacă subprogramul nu are parametri, se trec totuşi cele două paranteze, atât la definirea cât şi la apelul lui.
Antetul nu este urmat de „;“ (punct şi virgulă) ci de „{“ (acoladă) pentru corpul subprogramului.

După cum s-a văzut până acum, pentru un program este definită o funcţie principală, *main*, asupra căreia sistemul de operare dă controlul prima dată la execuţia programului.

În **C++** funcţia *main* trebuie să aibă scris tipul **void**, altfel compilatorul cere existenţa unei instrucţiuni **return** în corpul funcţiei.

b) Definirea corpului

Corpul subprogramului este o instrucţiune compusă, adică un set de comenzi încadrate între acolade.
El apare ca o structură de bloc şi cuprinde:
– definiţii de constante şi declaraţii de variabile, proprii subprogramului, dacă este nevoie;
– definirea de tipuri de date ale utilizatorului *valabile numai* în cadrul subprogramului (**typedef**);
– grupul de instrucţiuni care reflectă sarcina executabilă a subprogramului.

Numele funcţiei poate apărea şi în cadrul unor expresii din subprogram, fapt care indică reluarea funcţiei pentru un nou calcul, adică autoapelul ei (tehnică numită *recursivitate* care va fi studiată în Capitolul 7).

[1] *void* = vid (engl.).

Dacă funcţia nu întoarce nici o valoare calculată, adică *tip_rezultat* este **void**, atunci ieşirea din funcţie se face la întâlnirea acoladei de închidere a blocului.

Dacă ea este o *funcţie operand,* atunci ea trebuie să conţină în cadrul corpului instrucţiunea **return** prin care este returnat rezultatul, prin numele funcţiei, în modulul apelant.

Instrucţiunea **return** are sintaxa:

```
return  expresie;  sau  return(expresie);
```

Se calculează valoarea expresiei şi aceasta este returnată modulului care a apelat funcţia operand.

Parantezele din a doua formă a instrucţiunii au doar efect de a mări lizibilitatea programului. Dacă lipseşte instrucţiunea **return**, se produce un mesaj de eroare din partea compilatorului. Tipul valorii returnate trebuie să corespundă specificaţiei *tip_rezultat* din antet. Dacă nu se asigură această corespondenţă, comunicarea nu se poate face în exterior şi compilatorul produce un mesaj de eroare. Instrucţiunea **return** se poate amplasa oriunde în cadrul blocului, acolo unde logica funcţionării decide reîntoarcerea în modulul care activează subprogramul. Acest lucru vine oarecum în contradicţie cu principiile programării structurate.

Figura 6.4 conţine exemple de definiri de anteturi şi corpuri pentru unele subprograme.

Anteturi	***Corpurile corespunzătoare***

`//1)functie de verificare de numar prim`
`int prim(int nr)`
	`{for(int d=2;d<=nr/2;d++)`
	`  if(nr%2==0)return 0;`
	`  return 1;`
	`}`

`//2) functie de afisare mesaj` `void scrie()`	`{cout<<" scriu din o functie procedurala";}`
`//3) functie de calcul medie aritmetica` `float medie(unsigned a,unsigned b)`	`{return (float)a/b;}`
`//4) functie de testare cifra` `int e_cifra(char ch)`	`{if(ch>='0' && ch <='9')return 1; else return 0;}`

`//5) functie de afisare divizori proprii`
`void diviz(unsigned a)`
	`{for(int d=2;d<=a/2;d++)`
	`  if(a%d==0)cout<<d<<" ";`
	`}`

Figura 6.4

c) Definirea parametrilor – tipurile de parametri

Asamblarea unui subprogram nestandard în cadrul textului sursă al programului presupune ca, înaintea definirii lui, să se judece foarte bine cum se vor realiza conexiunile între el şi funcţia apelantă. Aceste conexiuni funcţionează ca nişte *canale prin care circulă datele* între funcţia apelată şi cea apelantă în momentul activării subprogramului.

Datele care circulă prin conexiuni au următoarele semnificaţii:

– date de intrare – pe care le primeşte subprogramul de la modulul apelant, ca bază de lucru;

– date de intrare-ieşire, pe care subprogramul le primeşte, le modifică şi le returnează modificate;

– rezultate ale funcţiei, pe care le creează subprogramul de tip funcţie operand.

Cu excepţia rezultatului creat de o funcţie operand, datele care circulă pe celelalte conexiuni se vor referi prin denumirea de **parametri**.

Ca şi stăzile, conexiunile pot avea sens unic sau dublu sens în parcurgerea lor de către date.

➢ *Conexiunile cu sens unic* permit:

– transmiterea **parametrilor de intrare;**

Parametrii de intrare sunt **valori copie** (dublură) ale celor din modulul apelant. Copiile sunt furnizate modulului apelat, dar valorile originale, din modulul apelant, rămân neschimbate. Din acest motiv, ei sunt denumiţi şi **parametri-valoare.**

– transmiterea „**parametrului de ieşire".**

„Parametrul de ieşire" este rezultatul funcţiei operand, care este furnizat modulului apelant prin variabila care **denumește funcția**.

➢ *Conexiunile cu dublu sens* permit:
– transmiterea **parametrilor de intrare-ieşire.**

Parametrii de intrare-ieşire sunt **adresele** valorilor din modulul apelant, valori pe care modulul apelat le va prelucra şi modifica. Din acest motiv ei sunt denumiţi şi **parametri-variabilă**.

Dacă ne referim la exemplele tabelului din figura 6.4, vom determina folosirea următoarelor tipuri de parametri:

– pentru funcţia `prim`, variabila `nr` este parametru de intrare (fig. 6.5); ea „poartă" o valoare copie a unui număr din exterior asupra căreia aplică verificarea de număr prim; funcţia nu trebuie să actualizeze valoarea primită; de asemenea, numele funcţiei este variabila prin care rezultatul (codificarea numerică pentru adevărat/fals) „pleacă" în exterior (fig. 6.5).

– pentru funcţia `medie`, variabilele `a` şi `b` sunt parametri de intrare, iar denumirea `medie` este parametrul de ieşire (fig. 6.6).

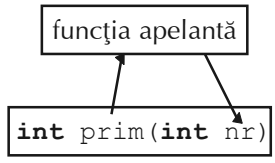


Figura 6.5

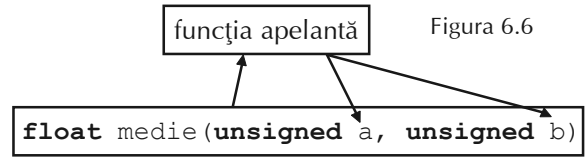


Figura 6.6

Stabiliţi tipul parametrilor utilizaţi în exemplele 1, 2, 3 şi 4 din paragraful 6.1.

d) Sintaxa pentru modul de transmitere a parametrilor

d.1) Parametrii de tip valoare

Printr-un *parametru de intrare,* ***parametru-valoare***, subprogramul primeşte de la modulul apelant o **copie a valorii originale** a variabilei care intră în prelucrare ca bază de calcul pentru un viitor rezultat. Variabila parametru trebuie să aparţină *aceluiaşi tip* de reprezentare ca şi valoarea primită din exterior.

Specificarea acestei comunicări în sintaxa antetului este:

`tip_rezultat nume_functie(tip_data nume_parametru`$_1$`, tip-data nume_parametru`$_2$`, …)`

Ca exemple, se pot considera funcţiile 1), 3), 4) şi 5) din tabelul din figura 6.4.

Probleme rezolvate exemplificând transmiterea parametrilor prin valoare.

1. *Enunţ.* Dorim ca afişarea în ordine descrescătoare a numerelor naturale, începând cu un număr citit, *init*, să se facă utilizând o funcţie *desc*, al cărei antet este: **`void`** `desc (`**`unsigned`** `numar)`.

Fişa subprogramului.

Funcţia			**Parametrul 1**		
Denumire	*Tip funcţie*	*Tip rezultat*	*Denumire*	*Tip parametru*	*Reprezentare*
`desc`	procedurală	**`void`**	`numar`	de intrare	**`unsigned`**

Explicaţii:

Tipul parametrului este de intrare, pentru că primeşte valoarea citită în modulul principal şi nu returnează nici o valoare. Acest lucru înseamnă că `numar` este o copie a valorii din variabila `init`.

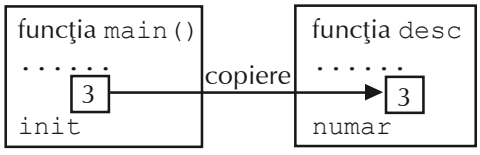
Exemplu numeric:

Presupunem că `init` primeşte valoarea 3 prin citire de la tastatură. În funcţia `desc`, copia lui, `numar`, primeşte valoarea 3 la început. Apoi ea scade cu câte o unitate, până ce `numar` devine 0, moment în care condiţia din **`for`** este falsă şi se iese din **`for`**. La ieşirea din **`for`** se întâlneşte acolada de încheiere a subprogramului, aşa că se predă controlul înapoi, funcţiei **`main`**(). Variabila `init`, a cărei valoare a fost copiată la intrarea în funcţie,

rămâne tot cu valoarea 3, pentru că nu asupra ei s-a acţionat prin sădere, ci asupra copiei. Copia ei, `numar`, dispare la ieşirea din funcţia ***desc***.

Etapele rezolvării:

init	număr	operaţia realizată
3	–	citire în **main**
3	3	apel **desc**
3	3	scrie3 în **desc**
3	2	scrie 2 în **desc**
3	1	scrie 1 în **desc**
3	0	oprire **for**
3	-	întoarcere în **main**




```
#include <iostream.h>
void  desc ( unsigned numar)
{ for ( ; numar ; —numar)
    cout<< numar << "\n";
}
void  main( )
{          unsigned  init;
  cout<<"Numarul de prelucrat";
  cin>>init;
  cout>>"init inainte de prelucrare "<<init<<endl;
  cout>>"Descresterile";
  desc(init);
cout>>"init dupa prelucrare "<<init;
}
```

2. *Enunţ.* Afişarea *cmmdc(a,b)*, unde *a* şi *b* sunt variabile naturale cu valori nenule se va face utilizând o funcţie care returnează cel mai mare divizor comun calculat prin algoritmul lui Euclid.

Antetul funcţiei este: **unsigned** `cmmdc(`**unsigned** `a,` **unsigned** `b)`

Fişa subprogramului

Funcţia			**Parametrul 1**			**Parametrul 2**		
Denumire	*Tip funcţie*	*Tip rezultat*	*Denumire*	*Tip parametru*	*Reprezentare*	*Denumire*	*Tip parametru*	*Reprezentare*
cmmdc	operand	**unsigned**	a	intrare	**unsigned**	b	intrare	**unsigned**

Explicaţii:

Tipul parametrilor este de intrare, pentru că şi `a` şi `b` primesc valorile citite în modulul principal şi nu trebuie să se întoarcă modificate. Acest lucru înseamnă că `a` şi `b` din funcţia `cmmdc` sunt cópii ale valorilor `x` şi `y` din funcţia **main**.

Exemplu numeric:

Fie situaţia în care, în modulul principal, variabila `x` primeşte valoarea 15 şi variabila `y` primeşte valoarea 85, prin citire de la tastatură. În funcţia `cmmdc`, copiile lor, numite *a* şi *b*, primesc valorile 15 şi, respectiv, 85, la început. Apoi ele se modifică prin operaţiile aplicate în algoritmul lui Euclid, pînă se obţine `b`=0, moment în care instrucţiunea **while** nu se mai execută. Urmează returnarea rezultatului obţinut în variabila `a`, adică numele funcţiei ia valoarea din `a` (*cmmdc* ← *a*). Se trece la acolada de încheiere a subprogramului, aşa că se predă controlul înapoi, funcţiei **main**(). Variabilele `x` şi `y`, ale căror valori au fost copiate în `a` şi `b` din funcţie, rămân tot cu valorile 15, respectiv, 85, pentru că asupra lor nu s-a acţionat prin împărţirile algoritmului, ci asupra cópiilor. Copiile dispar la ieşirea din funcţia `cmmdc`.

x	y	cmmdc (x,y)	a	b	r	operaţia realizată
15	85	–	–	–	–	citire în **main**
15	85	cmmdc (15,85)=?	15	85	–	apel **cmmdc**
15	85	cmmdc (15,85)=?	15	85	15	calcul rest 15 ≠ 0
15	85	cmmdc (15,85)=?	85	15	10	calcul rest 10 ≠ 0
15	85	cmmdc (15,85)=?	15	10	5	calcul rest 5 ≠ 0
15	85	cmmdc (15,85)=?	10	5	0	calcul rest 0 = 0
15	85	cmmdc (15,85)=5	5	0	–	cmmdc ← 5
15	85	5	–	–	–	întoarcere în **main** şi scrie 5

```
#include <iostream.h>
unsigned  cmmdc(unsigned a, unsigned b)
// a si b sunt parametrii de intrare ai
//functiei, din domeniul unsigned.
//Ei sunt  diferiti de x si y citite in
//modulul principal.
//Parametrul de iesire se poate spune că
//este variabila a,
```

```
//a carei valoare este intoarsa de
//return si astfel ea
//devine valoarea functiei în cadrul
//codomeniului unsigned

{ unsigned r;
//variabila r este proprie modulului, nu
//se poate folosi în main
   while (b)
      { r = a % b;
        a = b;
        b = r;
      }
return a;
}
void  main( )
{ unsigned x,y;
   cin>>x>>y;
   cout<< "cmmdc ="<<cmmdc(x,y);
}
```

Exerciţii rezolvate

Pentru fiecare dintre exerciţiile de mai jos se va rula programul pas cu pas, urmărindu-se în fereastra **watch** evoluţia variabilelor. Se va alcătui pe caiet tabelul etapelor rezolvării.

1. Să se construiască un program pentru calculul perimetrului şi ariei unui dreptunghi, cunoscându-se laturile acestuia. Calculele cerute se vor face în câte un subprogram corespunzător.

Rezolvare. Analiza problemei conduce la proiectarea a două module de tip funcţii operand, `perim` şi `arie`.

Aceste module vor primi ca parametri-valoare lungimea şi lăţimea dreptunghiului, citite în funcţia `main`, în variabilele `lung` şi `lat`.

```
#include <iostream.h>
// definirea functiei perim
float perim(float lungime, float latime)
{
      return (lungime + latime) * 2;
}
//definirea functiei arie
float arie(float lungime, float latime)
{
      return lungime*latime;
}
```

```
void  main( )
{
  float  lung, lat;
  cout<< "Dati lungimea si apoi latimea;
  cin>>lung>>lat;
  cout<< "Perimetrul este:"<< perim(lung,
                          lat)<<"\ n";
  cout<< "Aria este:"<< arie(lung,
                          lat)<<"\ n";
}
```

2. Se consideră *n* numere naturale. Să se afişeze acele numere care sunt identice faţă de ordinea de citire.

De exemplu *n*=4 şi numerele sunt: 12, 34243, 45, 1221. Se vor afişa valorile: 34243, 1221.

Rezolvare. Se va organiza un modul de tip funcţie operand, **invers**, care primeşte un număr natural printr-un parametru valoare şi returnează valoarea inversată ca ordine a cifrelor.

```
#include <iostream.h>
// definirea functiei invers
unsigned  invers(unsigned  numar)
{ unsigned inv=0;
  while (numar)
      {inv=inv*10+numar%10;
      numar/=10;
      }
  return inv;
}
```

```
void main()
{
  unsigned n, a;
  cout<<"Dati numarul de valori ";
  cin>>n;
  for(int i=1;i<=n;i++)
      {cin>>a;
      if(a==invers(a)) cout<<a<<' ';
      }
}
```

3. Se consideră `n` numere naturale citite pe rând, cu valori între 2 şi 65000. Se cere aflarea ultimei cifre a sumei puterilor `n` ale celor `n` numere.

Fie `n`=4 şi numerele: 23984, 65532, 71,6. Se cere ultima cifră a sumei $23984^4 + 65532^4 + 71^4 + 6^4$. Cifra căutată este 9.

Rezolvare. Din enunţ se vede că numerele pot fi de valori foarte mari. Rezultă că ideea efectuării ridicării la puterea `n` a fiecărui număr, adunarea rezultatului în suma de până atunci şi, în final, aflarea

ultimei cifre, nu se poate realiza, deoarece se vor ivi depăşiri ale submulţimii de numere naturale pe care o presupune tipul **unsigned**. Se va organiza un modul de tip funcţie operand, **putere**, care va primi numărul curent şi valoarea lui **n** prin doi *parametri de tip valoare*. Modulul va furniza cifra puterii a *n*-a a numărului curent. Această cifră se va adăuga la cifra finală a sumei de până atunci.

Pentru exemplul luat, va fi următorul proces:

s=(s+putere(23984,4))%10 → s=0+6; s=(s+putere(71,4))%10 → s=(2+1)%10 → 3;

s=(s+putere(65532,4))%10 →s=(6+6)%10 →2; s=(s+putere(6,4))%10 → s=(3+6) → 9

```
#include <iostream.h>
// definirea functiei putere
unsigned putere(unsigned numar, unsigned n)
{ unsigned c=1,i;
  numar=numar%10;
  for (i=1;i<=n;i++)
      c=c*numar%10;
  return c;
}
```

```
void main()
{  unsigned n, a,i,s=0;
  cout<<"Dati numarul de valori ";
  cin>>n;
  for(i=1;i<=n;i++)
      {cin>>a;
      s=(s+putere(a,n))%10;
      }
  cout<<"Ultima cifra="<<s;
}
```

4. Se citesc n numere naturale. Se cere afişarea numărului cu cele mai multe cifre distincte. Dacă sunt mai multe astfel de numere, se va afişa unul dintre ele.

Pentru n=4 şi numerele 2345, 1772, 111, 172, se va afişa 2345, deoarece are 4 cifre distincte.

Rezolvare. Se va organiza un modul de tip funcţie operand, **nr_cif_dist**, care va avea sarcina să calculeze numărul de cifre distincte ale unui număr pe care îl primeşte ca *parametru-valoare*. Numărul de cifre distincte calculat va fi returnat modulului principal. Calculul intern modulului se bazează pe organizarea unui vector de 10 elemente, corespunzător celor zece cifre. În vector se va înregistra valoarea 1 pe poziţia fiecărei cifre care apare în număr. Urmează apoi să se facă suma valorilor 1 din vector şi aceasta să fie returnată modulului principal. Modulul principal va compara fiecare număr de cifre, c, primit de la **nr_cif_dist** cu maximul de până atunci şi va reţine noul maxim, în variabila max şi numărul aferent acestuia, în variabila nr_max.

```
#include <iostream.h>

unsigned nr_cif_dist(unsigned numar)
{
      int v[10]={0},nc=0,i;
   //variabila nc numara aparitiile
   //cifrelor
      while(numar)
        {v[numar%10]=1 ;
        numar /= 10 ;
       }
      for(i=0;i<10;i++)
         nc += v[i] ;
      return nc;
   }
```

```
void main()
{  unsigned n, a,i,max=0,nr_max;
      cout<<"Dati numarul de valori ";
      cin>>n;
      for(i=1;i<=n;i++)
            {cin>>a;
             c= nr_cif_dist(a);
            if(max<c)
                  {max=c;
                   nr_max=a;
                  }
             }
       cout<<"Numarul cu cele mai multe
            cifre distincte= "<<nr_max;
    }
```

5. Se dau două puncte în plan, **p0** şi **q0**, care vor genera o dreaptă numită *reper*. Se dau, apoi, **n** perechi de puncte în plan. Fiecare pereche de puncte determină o dreaptă. Să se determine câte dintre cele **n** drepte sunt paralele cu dreapta *reper*. Pentru fiecare punct din plan se citesc două valori reale, reprezentând coordonatele lui în plan.

Fie p0(3,0) şi q0(0,3) punctele care determină dreapta *reper*, faţă de care se verifică paralelismul celorlalte. Fie n=2, numărul de drepte care vor fi verificate şi anume d1(p1(4,1), q1(2,3)) şi d2(p2(5,1), q2(2,4)). Cele două drepte sunt paralele cu dreapta *reper*.

Rezolvare. Se calculează panta dreptei martor şi se verifică dacă pentru fiecare dreaptă citită panta aceleia este egală, în limitele unui ε=0,001, cu panta dreptei martor, contorizând în variabila **nr** situaţia de egalitate.

Panta dreptei martor este **m0**= $(y_{p0}-y_{q0})/(x_{p0}-x_{q0})$, iar panta unei drepte oarecare se va calcula, asemănător, în variabila **m**. Rezolvarea nu tratează cazul de drepte verticale, ceea ce va rămâne ca *temă de laborator*.

Pentru rezolvarea de mai sus s-a ales stabilirea unui nou tip de dată, al utilizatorului, numit **punct**. Noul tip de dată a fost definit ca înregistrare, folosind declaraţia **struct**.

```
#include<iostream.h>
#include<math.h>
#include<conio.h>

typedef struct punct{float x,y;};
const float eps=0.001;

float panta(punct a,punct b)
{return (b.y-a.y)/(b.x-a.x);}

int vertical(punct a, punct b)
{ return fabs(a.x-b.x)<eps;}

void main()
{punct p0,q0,p,q;
 float m0,m;
//m0 este panta primei drepte
//m este panta dreptei curente
 int n,i,nr=0;
 do
{cout<<"Dati punctele primei drepte,
                  nu verticala ";
 cin>>p0.x>>p0.y>>q0.x>>q0.y;
}while(vertical(p0,q0));
 m0=panta(p0,q0);
 cout<<"dati numarul de drepte de
      verificat ";
 cin>>n;
 for(i=1;i<=n;i++)
   { do
      {cout<<"Dati punctele dreptei
       curente, nu verticala ";
       cin>>p.x>>p.y>>q.x>>q.y;
      }while(vertical(p,q));
     m=panta(p,q);
     if(fabs(m0-m)<eps) nr++;
    }
 cout<<"Sunt "<<nr<<" paralele la
prima
       dreapta"; getch();
   }
```

Din rezolvarea de mai sus se observă modalitatea de a transmite ca parametru o dată de tip **struct**.

6. Se consideră un şir de litere şi cifre citit în variabila **sir**, având maximum 20 de caractere. Se cere să se stabilească numărul de cifre şi numărul de litere pe care le conţine.

Dacă se citeşte şirul *12mai2006*, atunci se vor contoriza: 6 cifre şi 3 litere.

Rezolvare. Este util să organizăm o funcţie, **ciflit**, care să testeze dacă un caracter primit este cifră sau literă. Răspunsul funcţiei va fi tot un caracter, şi anume, litera **C**, pentru situaţia de cifră, şi litera **L**, cînd a găsit literă. În orice altă situaţie, funcţia returnează spaţiu.

```
#include<iostream.h>
#include<ctype.h>
#include<conio.h>
char ciflit(char c)
{
  if(c>='0'&&c<='9') return 'C';
  if(toupper(c)>='A' && toupper(c)<='Z')
            return 'L';
  return ' ';
}
void main()
{
  char sir[21];
  int nC=0,nL=0,i;
  clrscr();
  cout<<"dati sirul ";
  cin>>sir;
  for(i=0;sir[i]!=0;i++)
   if(ciflit(sir[i])=='C')nC++;
     else if(ciflit(sir[i])=='L')nL++;
  cout<<"S-au gasit "<<nC<<" cifre\n";
  cout<<"S-au gasit "<<nL<<" litere";
  getch();
  }
```

În subprogramul **ciflit**, **if**(toupper(c)>='A' && toupper(c)<='Z') nu are else pentru cazul în care nu este îndeplinită condiţia, deoarece pentru fiecare dintre cele două situaţii (adevărat sau fals) firul prelucrării se „rupe" prin trimitere necondiţionată în afara modulului, înapoi la modulul apelant. Această trimitere este făcută prin instrucţiunea **return**, astfel că va executa **return ' '** doar în cazul în care condiţia este falsă, în celălalt caz executând **return 'L'**.

test

I.

1. Care dintre următoarele declarări reprezintă antetul corect al unei funcţii reale `f` cu doi parametri întregi, `a` şi `b`?
a) `float f(int a,b);` **b)** `float f(int a,int b);` **c)** `float f(int a;int b);`
d) `void f(int a,int b);` **e)** `double f(int a,int b);` **f)** `float f(long a,long b);`
g) `double f(long a; b);` **h)** `long double f(long a, int b);` **i)** `f(long a, long b).`

2. Care este antetul corect al subprogramului `dist` ce returnează valoarea expresiei `E=| a – b |`, modulul diferenţei, pentru valorile reale `a` şi `b` transmise ca parametri?
a) `void dist(float a,float b);` **b)** `float dist (float a,float b);`
c) `void dist(float a,floatb,float e);` **d)** `int dist(float a,float b);`
e) `float dist(double a, double b);` **f)** `double(float a,float b,double e)`

3. Se consideră definită funcţia `max` care returnează valoarea maximă dintre cele două valori transmise ca parametri. Pentru un număr natural cu cel mult două cifre, memorat în variabila întreagă `n`, stabiliţi care este expresia care returnează cea mai mare cifră a numărului.
a) `max (n%10,n%100);` **b)** `max (n%10, n/100);` **c)** `max(n/10, n%10);` **d)** `max(n/10,n%100).`

4. Se consideră definită funcţia `min` care returnează valoarea minimă dintre cele două valori reale transmise ca parametri. Stabiliţi care dintre următoarele expresii nu este egală cu cea mai mică dintre valorile reale `a,b,c`.
a) `a+b+c-min(a,b)-min(a,c);` **b)** `min(min(a,b),min(a,c));` **c)** `min(min(a,b),c);`
d) `min(a,min(b,c)).`

5. Se consideră definită funcţia `min` care returnează valoarea minimă dintre cele două valori reale transmise ca parametri. Stabiliţi care dintre următoarele expresii este egală cu cea mai mare dintre valorile `a` şi `b`.
a) `min(a, b);` **b)** `min(a,b)-a-b;` **c)** `a- min(a,b)+b-min(b,a);` **d)** `a+b-min(a,b).`

6. Pentru valori strict pozitive ale parametrului `a`, funcţia `f`, figurată în caseta de mai jos, returnează valoarea 1, dacă şi numai dacă valoarea lui `a` este un număr natural care:
a) are ultima cifră mai mică sau egală cu 5;
b) are cel puţin o cifră mai mică sau egală cu 5;
c) are prima cifră mai mică sau egală cu 5;
d) are cel mult o cifră mai mică sau egală cu 5;

```
int f(int a)
   { while(a%10>5)
       a/=10;
     return a >0 }
```

7. Variabila întreagă `perf` trebuie să reţină câte numere naturale pătrate perfecte mai mari decât 0 şi mai mici sau egale cu `n` există. Care este expresia cu care trebuie completată atribuirea `perf= ?`
a) `sqr(n));` **b)** `sqr(n)+1;` **c)** `sqrt(n)+1;` **d)** `sqrt(n).`

8. Se consideră două funcţii, `s` şi `p`, care calculează suma, respectiv produsul numerelor întregi `x` şi `y` care sunt transmise ca parametri. Antetul funcţiilor este:
`int s(int x, int y);` `int p(int x, int y).`
Care dintre apelurile de mai jos calculează valoarea expresiei `2(ab+ac+bc)`, unde `a, b` şi `c` sunt variabile întregi citite în modulul apelant?
a) `p(2, ab+ac+bc);` **b)** `p(2,s(p(a,b), p(a,c),p(b,c)));`
c) `p(2,s(s(p(a,b),p(a,c)),p(b,c)));`
d) `s(s(s(p(a,b),p(a,c)),p(b,c)),s(s(p(a,b),p(a,c)),p(b,c)))`

II.

1. Scrieţi un subprogram care primeşte un număr natural *x* şi returnează cel mai mare număr care se poate forma din cifrele numărului `x`.

2. Scrieţi un subprogram care returnează valoarea din şirul lui Fibonacci care se află pe locul `n`, număr natural primit ca parametru.

3. Scrieţi un subprogram care afişează toate numerele pătrate perfecte aflate între *1* şi valoarea naturală `n`, primită ca parametru.

4. Scrieţi un subprogram care returnează a*devărat* sau *fals*, după cum un număr natural `n` are exact `k` divizori proprii sau nu. Subprogramul se va folosi de convenţia `C/C++` pentru valorile logice cerute.

5. Scrieţi un subprogram care determină dacă un caracter este cifră, literă sau alt semn. *Indicaţie*: se va codifica răspunsul prin literele `'C'`, `'L'` şi `'S'`.

6. Scrieţi un subprogram care calculează `n!`.

7. Scrieţi un program care foloseşte funcţia de calcul pentru `n!` în scopul calculării numărului C_n^k.

d.2) Parametrii de tip variabilă

Pentru un *parametru de intrare-ieşire*, **parametru-variabilă**, este nevoie de precizarea sintactică a faptului că prin el se transmite modulului apelat doar o **referire** la **adresa** şi caracteristicile **variabilei efective** din modulul apelant.

Regula sintactică aplicată în definirea antetului constă în a preceda parametrul de operatorul de adresare **&**.

```
tip_rezultat nume_functie(tip_data & nume_parametru1, tip-data & nume_parametru2, …)
```

Subprogramul va opera cu valoarea acelei variabile direct în locul în care ea a fost instalată în memorie de modulul apelant. Astfel, el o poate schimba într-o valoare cu rol de rezultat.

```
void sp (int & x)
{……
     x++;
}
void main()
{…… int p=0;……
sp(p);
cout<<p ;}
```

Figura 6.7

La apel, valoarea efectivă se comunică *numai* prin intermediul unei ***variabile de acelaşi tip*** cu cel cu care parametrul a fost definit. Se spune că parametrul este un **sinonim** al variabilei efective din apel.

Astfel, de exemplu, subprogramul **sp** (figura 6.7) are nevoie de o valoare întreagă, **x**, pe care o creşte cu o unitate. În modulul apelant, această variabilă este denumită **p** şi are ca valoare iniţială 0. Deoarece subprogramul acţionează prin modificarea valorii primite prin parametru cu păstrarea modificării la ieşire, el va primi din partea lui **p** din **main** o referinţă la adresa de memorie a acestuia. Apelul lui **sp** nu poate folosi valoarea directă, **0**, ca pentru un apel cu transmitere prin valoare de forma **sp(0)**, deoarece **0** nu este o variabilă alocată în memorie, ci este un conţinut al lui **p**.

Probleme rezolvate pentru exemplificarea transmiterii parametrilor de tip variabilă.

1. *Enunţ.* Se doreşte realizarea unui interschimb a valorilor a două variabile reale, *a* şi *b*, citite de la tastatură, iar interschimbul să fie realizat de către un subprogram numit **schimb**.

Antetul funcţiei este:

```
void  schimb ( float &x, float &y)
```

Fişa subprogramului

Funcţia			**Parametrul 1**			**Parametrul 2**		
Denumire	*Tip funcţie*	*Tip rezultat*	*Denumire*	*Tip parametru*	*Reprezentare*	*Denumire*	*Tip parametru*	*Reprezentare*
schimb	procedural	**void**	x	intrare/ieşire	**Referinţă adresă pentru float**	y	intrare/ieşire	**Referinţă adresă pentru float**

Explicaţii:

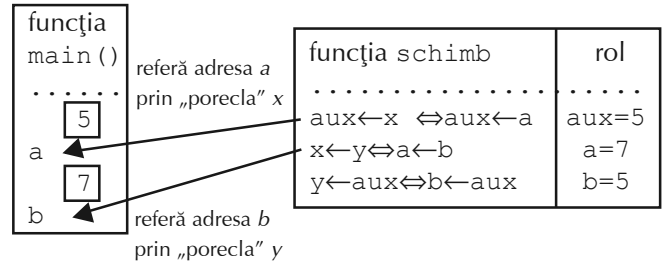
Interschimbul valorilor între cele două variabile, `a` şi `b`, se face prin intermediul funcţiei `schimb` care nu întoarce un rezultat propriu (este de tip **void**), dar primeşte valori *pe care le întoarce modificate*.

Tipul parametrilor este de intrare-ieşire pentru că `x` şi `y` sunt referinţe la adresele valorilor citite în modulul principal în variabilele `a` şi `b`. Aceste valori trebuie să se întoarcă modificate, prin interschimb. Acest lucru funcţionează ca şi cum `x` şi `y` din funcţia **schimb** sunt „al doilea nume, porecle" ale variabilelor `a` şi `b` din funcţia **main**.

Exemplu numeric:

Fie situaţia în care **a** primeşte valoarea 5 şi **b** primeşte valoarea 7, prin citire de la tastatură. În funcţia **schimb** se comunică referinţele la locurile de memorie în care au fost instalate variabilele **a** şi **b**. În subprogram, aceste locuri sunt cunoscute prin denumirile **x** şi **y**, care joacă, fiecare, rolul unui al doilea nume al acelor zone, sau rolul unor porecle: **a** este poreclit **x** şi **b** este poreclit **y**. Variabilele **x** şi **y** **nu sunt** cópii ale lui **a** şi **b**. Astfel că interschimbul se realizează ca şi cum s-ar lucra direct în funcţia **main**.

a	**b**	**x**	**y**	*operaţia*
5	7	-	-	citire (main)
5	7	5	7	intrare în schimb
5	7	5	7	aux ←x (5)
7	**7**	**7**	**7**	x←y
7	5	7	5	y ← aux
7	5	-	-	ieşire în main




```
#include <iostream.h>
void  schimb ( float & x, float & y)
{      float aux;
       aux=x;
       x=y;
       y=aux;
}
void  main( )
{   float  a,b,c,d;
    cout<<"Numerele de interschimbat ";
    cin>>a>>b;
    cout>>"numerele inainte de prelucrare "<<a<<' '<<b<<endl;
    schimb(a,b);
    cout>>"numerele dupa prelucrare "<< a<<' '<<b<<endl;
    cout<<"Dati al doilea set de numere de interschimbat ";
    cin>>c>>d;
    schimb(c,d);
    cout<<"\n Al doilea set, dupa prelucrare "<<c<<' ,<<d;
}
```

În caseta explicativă, perechile din coloanele **a** cu **x** şi **b** cu **y** au marcate anteturile cu aceeaşi umbrire, pentru a pune în evidenţă faptul că zona de memorie a lui **a** este folosită şi sub numele **x**, iar cea pentru **b** şi sub numele ***y***.

Avantajul abordării de mai sus a interschimbului este acela că subprogramul preia la apel referinţele de adrese de care are nevoie modulul apelant în momentul respectiv. Este cazul părţii a doua a lui **main** din program, în care se cere interschimbul între **c** şi **d**. Acum lui **x** şi lui **y** li se vor transmite referinţe la locurile lui **c** şi **d**.

2. *Enunţ:* Se doreşte să se afişeze, în ordine crescătoare, valorile a trei variabile reale, *a*, *b*, *c*, citite de la tastatură. Pentru realizarea ordonării se va folosi ca modúl elementar de calcul logic, o funcţie de stabilire a ordinii crescătoare între două valori, **ordine2.**

Antetul funcţiei este: **void** ordine2(**float** x, **float** y).

Fişa subprogramului

Funcţia			**Parametrul 1**			**Parametrul 2**		
Denumire	*Tip funcţie*	*Tip rezultat*	*Denumire*	*Tip parametru*	*Reprezentare*	*Denumire*	*Tip parametru*	*Reprezentare*
ordine2	procedural	**void**	x	intrare/ieşire	**Referinţă adresă pentru float**	y	intrare/ieşire	**Referinţă adresă pentru float**

Explicaţii:

Pentru a ordona crescător valorile a trei variabile, se va aplica, pe rând ordonarea a câte două. Stabilirea ordinii între două variabile **x** şi **y** presupune testarea relaţiei **x > y** şi interschimbul valorilor între cele două variabile, dacă inegalitatea este adevărată. Interschimbul se va face prin intermediul funcţiei **schimb** care a fost prezentată în *problema 1* de mai sus.

Funcţia **ordine2** nu întoarce un rezultat propriu (este de tip **void**), dar primeşte valori pe care le întoarce modificate.

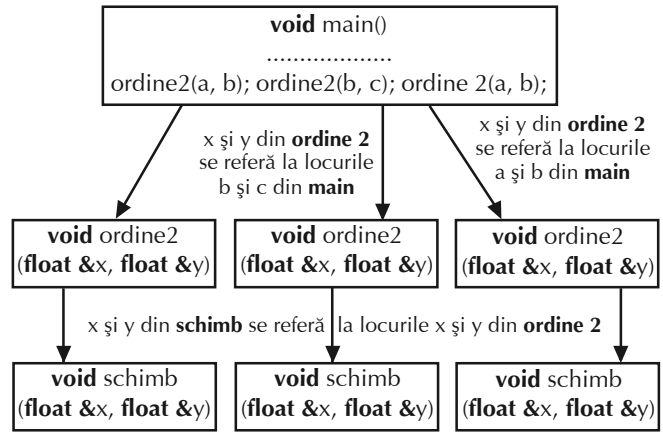
Tipul parametrilor este de intrare-ieşire, pentru că **x** şi **y** sunt referinţe la adresele valorilor citite în modulul principal în variabilele **a, b** *şi* **c** . Aceste valori pot să se întoarcă modificate, prin interschimb.

Exemplu numeric:

Presupunem că după citiri, variabilele au următoarele valori: a=5, b=3, c=2. Aplicând **ordine2**(a,b) se va obţine a=3 şi b=5. Apoi **ordine2**(b,c) va produce b=2 şi c=5. Ultima operaţie este **ordine2**(a,b), de unde rezultă a=2 şi b=3.

Din explicaţiile de mai sus rezultă că vom obţine între module o structură ierarhică pe două niveluri.

```
//utilizare subprograme ierarhizate
//pe doua niveluri
#include<iostream.h>
void  schimb ( float & x, float & y)
{     float aux;
      aux=x; x=y; y=aux;
}
void ordine2(float & x, float & y)
{
      if(x>y)schimb(x,y);
}
void main()
{float a,b,c;
 cout<<"Dati cele trei numere reale ";
      cin>>a>.b>.c ;
      ordine2(a,b);
      ordine2(b,c);
      ordine2(a,b);
      cout<<"\n Ordinea crescatoare a celor trei valori: "<<a<<' '<<b<<' '<<c;
}
```



Exerciţii rezolvate

Pentru fiecare dintre exerciţiile de mai jos se va rula programul pas cu pas, urmărindu-se în fereastra **watch** evoluţia variabilelor. Se va alcătui pe caiet tabelul etapelor rezolvării.

1. Să se construiască un program pentru calculul perimetrului şi ariei unui dreptunghi cunoscându-se laturile dreptunghiului, folosind un singur subprogram pentru aceste calcule.

Rezolvare. Problema a mai fost prezentată în paragraful *d.1*, cu scopul de a vedea descompunerea rezolvării în module elementare. Acum este transformată prezentarea, numai cu scop didactic, pentru a pune în evidenţă transmiterea parametrilor de tip variabilă. Astfel, se va organiza un singur modul de tip funcţie procedurală care va avea doi parametri de tip valoare, **lungime** şi **latime**, şi doi parametri de tip variabilă, **perim** şi **arie**.

```
#include <iostream.h>
void dreptunghi(float lungime, float
      latime, float &perim,float &arie)

{ perim=2*(lungime+latime);
  arie= lungime*latime;
}
void main()
{ float lung, lat, p,a;
   cout<<"Dati lungimea si latimea ";
  cin>>lung>>lat;
  dreptunghi(lung,lat,p,a);
  cout<<"Perimetrul="<<p<<endl;
  cout<<"Aria="<<a;
}
```

2. Se doreşte determinarea acelui număr natural din cele n citite care are cel mai mic divizor propriu cu frecvenţa (ordinul de multiplicitate) cea mai mare.

Exemplu. Dacă n=4 şi numerele citite sunt 15, 49, 27 şi 18, atunci numărul afişat va fi $27=3^3$, pentru că are cel mai mic divizor propriu 3, la puterea cea mai mare, 3, faţă de celelalte numere $15=3^1*5^1$, $49=7^2$, $18=2*3^2$.

Rezolvare. Se va proiecta un modul de tip funcţie operand, **citnat**, care va citi un număr natural **a**. Apoi, se va proiecta o funcţie operand, **frecv_d**, care va afişa divizorii proprii ai unui

număr primit ca parametru de tip valoare şi va întoarce în parametrul de tip variabilă, frecvenţa celui mai mic divizor, adică al primului găsit. Dacă numărul este prim, funcţia va intoarce 0, iar ca frecvenţă, tot 0. Dacă numărul nu este prim, funcţia va întoarce 1.

```
#include<iostream.h>
#include<conio.h>
unsigned citnat(unsigned i)
{unsigned nr; long val;
 cout<<"Dati numere naturale mai mici
                  decat 65535 ";
 cout<<endl;
 do
  {
    cout<<"Introduceti a "<<i<<"-a
valoare ";
    cin>>val;
    }while(val<0||val>65535);
 nr=val;
 return nr;
}
unsigned frecv_d(unsigned nr, unsigned &f)
{unsigned d;
 for(d=2,f=0;!f && d<=nr/2;d++)
    while(nr%d==0)
      {f++;
       nr/=d;
      }
 if(f) return 1;
  //else
 return 0;
}
```

```
void main()
 {
   unsigned n,a,max=0,dmax,d,f,nr_f;
   clrscr();
   cout<<"Dati numarul de valori ";
   cin>>n;
   for(int i=1;i<=n;i++)
     {a=citnat(i);
      if(frecv_d(a,f))
       if(max<f)
         {max=f;
          nr_f=a;
         }
      }
  if(!max) cout<<"S-au citit numai numere
                       prime\n";
   else

   {cout<<"Nr. "<<nr_f;
    cout<<" are cel mai mic divizor
propriu ";
    cout<<" cu frecventa maxima="<<max;
   }

getch();
 }
```

3. Se pune problema rezolvării ecuaţiei de gradul al doilea $ax^2 + bx + c = 0$. Coeficienţii ecuaţiei se citesc de la tastatură.

Rezolvare. Se vor determina următoarele funcţii care participă la rezolvarea ecuaţiei, pe lângă modulul principal:

– Funcţia operand **ec_gr_I**, care rezolvă situaţia în care s-a introdus valoarea 0 pentru coeficientul **a** şi returnează 0 dacă *nu există soluţie*, –1, dacă există o *infinitate de soluţii*, şi 1, dacă în parametrul **x** se *intoarce valoarea* rădăcinii. Antetul funcţiei este: **int ec_gr_I(float b, float c, float & x)**

– Funcţia operand **delta**, care calculează şi returnează discriminantul ecuaţiei, declarată cu antetul următor:

float delta(float a, float b, float c)

– Funcţia procedurală **radacina**, care calculează valoarea unei rădăcini. Funcţia va întoarce în parametrul **x** valoarea primei rădăcini sau a celei de-a doua, după cum parametrul **semn** este 1 sau –1. Antetul funcţiei este **void radacina(float d, float a, float b,float &x, int semn)**

```
#include <iostream.h>
#include <math.h>
void radacina(float d,float a, float b,
              float & x, int semn)
{
 x=(-b+ semn*sqrt(d))/(2*a);
}
float delta(float a,float b, float c)
{
 return b*b-4*a*c;
}
```

```
int ec_gr_I(float b, float c, float & x)
{
 if(b){x=-c/b;return 1;}
   if(c) return 0;
      return -1;
}
void main()
{ float x, a,b,c,d;
  int s;
  cout<<"Dati coeficientii";
  cin>>a>>b>>c;
```

```
if(a)

{d=delta(a,b,c);
 if (d<0)cout<<"Radacini complexe";
   else if(d>0)
     {radacina(d,a,b,x,1);
     cout<<"Prima radacina="<<x<<endl;
      radacina(d,a,b,x,-1);
      cout<<"A doua radacina=";
      cout<<x<<endl;
      }
      else {radacina(d,a,b,x,1);
            cout<<"Radacina dubla "<<x;
                }
}
 else
 { s=ec_gr_I(b,c,x);
   switch(s)
  {case 0:cout<<"Radacina imposibila";
          break;
   case -1:cout<<"O infinitate de
                 soluaii"; break;
   case 1:cout<<"Radacina="<<x;
   }
  }
}
```

Testul 2

I.

1. Să se determine ce afişează programul de mai jos:

```
include<iostream.h>
void a(int n)
int j;for(j=1;j<=n;j++)cout<<"*";
cout<<endl;
}

void main()
{a(15);
cout<<"exemplu simplu"<<endl;
 a(15);
}
```

2. Să se determine ce afişează programul dat în forma de mai jos şi apoi în forma conţinând modificarea **int** f(**int** &i):

```
#include<iostream.h>
int x;
int f(int i)
{int lucru;
 lucru=i+x;i++;x+=3;
 return lucru;}

void main()
{x=0;
cout<<x<<'\n';
cout<<f(x)<<" "<<x;
}
```

3. Să se determine ce afişează programul de mai jos:

```
#include<iostream.h>
void a(int n)
{n++;cout<<n<<'\n';}

void main()
{int n=3;a(1);
cout<<n<<'\n';}
```

4. Să se figureze, cu ajutorul tabelului de valori, modificarea valorilor variabilelor **h** şi **i** pe parcursul executării următorului program, precizându-se ce se afişează:

```
#include<iostream.h>
int i;
void y(int &h)
{int j=3*i; h=j+2;i+=3;
 cout<<h<<endl;
}

void main()
{i=4;
y(i);
cout<<i<<endl;
}
```

5. Să se determine care dintre variantele date este afişată de programul de mai jos:

```
#include<iostream.h>
void f(int a, int &b)
{a++;b++;
}

void main()
{int x=1,y=2;
 f(x,y); cout<<x<<y;
 f(x,x);cout<<x<<y;}
```

a) 1223; **b)** 1333; **c)**2322; **d)** 2333.

test

6. Să se determine care dintre variantele date este afişată de programul de mai jos:

```
#include<iostream.h>
int y;
int f(int &x)
{return ++x;}
int g(int x,int y)
{return x+y;}
void main()
{y=2;cout<<f(y);
cout<<g(y,y);
cout<<y;
}
```

a) 23; **b)** 233; **c)** 363; **d)** 232.

7. Se consideră următoarea funcţie procedurală:

```
void P(int n, int &f)
{
 int d=2,a;
 a=(int)sqrt(n),
 f=1;
 while(d<=a && f)
    {f=f&& n%d;
     d++;}
}
```

Prelucrarea realizată de funcţie constă în:

a) Determinarea factorilor primi ai lui n;

b) Determinarea numerelor prime până la rădăcina pătrată din n;

c) Verificarea numărului n dacă este prim;

d) Verificarea împărţirii exacte a lui n la d.

II.

Rescrieţi programele cerute la paragraful *d.1,* secţiunea II din Test 2, astfel încât rezultatele cerute subprogramelor să fie acum transmise prin parametri de intrare-ieşire.

6.2.5.2. Transmiterea parametrilor de tip tablou

După cum se ştie, ***numele tabloului*** reprezintă ***adresa lui simbolică*** în memoria internă. Rezultă următoarea particularitate pentru un parametru de tip tablou:

Un parametru de tip tablou este comunicat automat prin **referinţă,** numele lui în lista de parametri fiind un simbol al adresei de memorie la care tabloul este alocat.

exemplu

1. *Enunţ:* Se citesc de la tastatură notele primite de un elev la o materie la care se poate să se dea şi teză. Se doreşte calcularea şi afişarea mediei semestriale a acelui elev. Se consideră că nu pot fi mai mult de şapte note în rubrica de note la oral şi că datele de intrare sunt corecte.

Rezolvare. Se vor proiecta trei funcţii necesare pentru: *citirea* unui vector de numere naturale, de maximum 7 elemente, *calculul mediei la oral*, *calculul mediei semestriale*, în caz că există, şi nota la teză.

– Funcţia procedurală **cit_vect** cu antetul

```
void cit_vect (unsigned v[], unsigned & lung, int limita,char nume[])
```

va face citirea unui vector oarecare, v, care conţine lung numere naturale nu mai multe de limita, iar numele dat de utilizator vectorului se află în şirul nume.

– Funcţia operand **oral** cu antetul

```
float oral(unsigned v[], unsigned lung)
```

va returna media notelor la oral, fără rotunjire;

– Funcţia operand **medie**, cu antetul

```
unsigned medie(float mo, unsigned teza)
```

va returna media semestrială, rotunjită, cu sau fără teză.

```
#include <iostream.h>
#include <conio.h>
void cit_vect(unsigned v[],unsigned
        &lung, int limita, char nume[])
{
do
 {cout<<"Dati numarul de elemente ";
  cin>>lung;
```

```
 }while(lung<1 || lung>limita);
for(int i=0;i<lung;i++)
 {cout<<nume<<"["<<i+1<<"]=";
  cin>>v[i];
 }
}
float oral(unsigned v[],unsigned lung)
{ float mo=0;
  for(int i=0;i<lung;i++)
      mo+=v[i];
  mo/=lung;
  return mo;
}

unsigned media(float mo, unsigned teza)
{ float m;
  if(teza)m=(3*mo+teza)/4;
      else m=mo;
  return (int)(m+0.5);
}

void main()
{clrscr();
char den[12];
unsigned note[7], n,t;
float m_oral;
cout<<"Dati numele materiei ";
cin>>den;
cit_vect(note,n,7,den);
cout<<"Dati nota tezei sau 0 daca nu
                          exista ";
cin>>t;
m_oral=oral(note,n);
cout<<"Media semestriala la "<<den;
cout<<" este "<<media(m_oral,t);
getch();
}
```

La apelul funcţiei **cit_vect**, parametrului **v** i se transmit atât adresa vectorului **note** cât şi mărimea spaţiului maxim alocat (şapte elemente) deoarece **v** devine un **sinonim** al lui **note**. Din acest motiv, nu mai este nevoie să se completeze parantezele drepte ale lui **v** din lista de parametri. Acelaşi lucru se întâmplă şi cu parametrul **nume** care devine un sinonim al lui **den** – denumirea materiei pentru care se calculează media.

exemplu

2. *Enunţ:* Se citesc de la tastatură valorile naturale **m** şi **n** care reprezintă numărul de linii şi, respectiv, de coloane ale unei matrice **A**. Se citesc apoi, linie cu linie, valorile elementelor matricei, numere întregi. Să se afişeze numărul liniei cu cele mai multe elemente negative.

Rezolvare. Se va utiliza proprietatea de structură a tabloului bidimensional şi anume faptul că el este un vector de vectori. În cazul dat, **A** este o structură de **m** vectori a câte **n** elemente întregi fiecare. Din fiecare linie a matricei se formează o valoare de tip frecvenţă care reţine numărul de valori negative din acea linie. Pentru a determina linia cu cele mai multe elemente negative, aici, în scop didactic, se va organiza un vector al acestor frecvenţe. Prelucrarea se poate desfăşura şi fără acest vector de frecvenţe, prin reţinerea maximului în mod progresiv şi, evident, a liniei în care acesta apare.

Se vor proiecta următoarele funcţii:

- Funcţia procedurală **cit_linie**, asemănătoare funcţiei **cit_vect** de mai sus, singura deosebire fiind în textul care se afişează utilizatorului privind elementul curent citit;
- Funcţia operand **nr_neg**, care returnează numărul de valori negative găsite într-un vector. Antetul funcţiei este **unsigned nr_neg(int v[], unsigned lung)** şi, la apel, **v** va deveni sinonim, pe rând, cu fiecare linie a matricei;
- Funcţia operand **max_v**, care returnează valoarea maximă găsită printre elementele unui vector, aici fiind vorba de vectorul frecvenţelor de numere negative din fiecare linie a matricii. Antetul funcţiei este: **unsigned max_(unsigned v[], unsigned lung)**.

```
#include <iostream.h>
#include <conio.h>
void cit_linie(int v[],unsigned lung,
               char nume, unsigned lin)
{
for(int i=0;i<lung;i++)
{cout<<nume<<"["<<lin+1<<","<<i+1<<"]=";
 cin>>v[i];
 }
}
unsigned nr_neg(int v[],unsigned lung)
{
      unsigned nr=0;
      for(int i=0;i<lung;i++)
            if(v[i]<0) nr++;
      return nr;
}
unsigned max(unsigned v[], unsigned
                  lung, unsigned &loc)
{ unsigned m=0;
  for(int i=0;i<lung;i++)
  if(v[i]>m)
            {m=v[i];loc=i;}
  return m;
```

```
}
void main()
{
const dim=3;
clrscr();
int A[dim][dim];
unsigned frecv[dim],m,n,i,j;
do
  {cout<<"Dati numarul de linii ";
      cin>>m;
  }while(m<2 || m>dim);
do
  {cout<<"Dati numarul de coloane ";
      cin>>n;
  }
while(n<2 || n>dim);

for(i=0;i<m;i++)
  {
      cit_linie(A[i],n,'A',i);
      frecv[i]=nr_neg(A[i],n);
  }
cout<<"Numarul maxim de negative=
        "<<max(frecv,m,j);
cout<<" si corespunde liniei "<<j+1;
getch();
}
```

3. *Enunţ:* Se citesc de la tastatură valorile naturale **m** şi **n** care reprezintă numărul de linii şi, respectiv, de coloane ale unei matrice **A**. Se citesc apoi, linie cu linie, valorile elementelor matricei, numere întregi. Să se afişeze linia şi coloana în care se găseşte elementul minim din matrice.
Rezolvare. Se va utiliza o funcţie care returnează elementul minim dintr-un tablou bidimensional primit ca parametru. Linia şi coloana minimului sunt returnate prin parametrii de intrare-ieşire `L` şi `C`. Funcţia este de tip operand şi are antetul:

```
int minim(int matr[][10],unsigned lin,unsigned col,unsigned &L,unsigned &C)
```

Se observă că, de această dată, funcţiei i se comunică întreaga matrice. Parametrul **matr** va trebui să specifice, drept **constantă de lucru**, numărul maxim de elemente ale unei linii a matricei (**coloanele**) aşa cum s-au alocat la declararea matricei, în modulul apelant. Parametrii **lin** şi **col** sunt necesari pentru ca în funcţie să se parcurgă numai spaţiul ocupat de valori din spaţiul total alocat matricei. Deoarece în funcţia **main** se pot declara variabilele şi se pot citi valorile fără dificultate, se vor prezenta detaliat numai funcţia şi apelul ei.

```
int minim(int matr[][10],unsigned lin,
 unsigned col, unsigned &L, unsigned &C)
{ int m=matr[0][0],i,j;
        for(i=0;i<m;i++)
          for(j=0;j<n;j++)
            if(matr[i][j]<m)
              {m=matr[i][j];L=i; C=j;}
        return m;}
```

Apelul funcţiei se va face în operaţia de afişare, de exemplu, în forma următoare:

```
cout<<"Elementul minim este "<<A<<"["<<L+1<<","<<C+1<<"]="<<minim(A,m,n,L,C);
```

În exemplele de mai sus, deoarece în adresarea internă indicii pornesc de la 0, pentru afişarea rezultatelor, aceştia au fost scrişi prin adăugarea unei unităţi, astfel încât utilizatorul să îi poată citi conform obişnuinţei lui de numărare, de la 1.

6.2.5.3. Aspecte suplimentare în lucrul cu subprograme

a) Domeniul de vizibilitate al identificatorilor

Într-un program sunt declaraţi identificatori, denumiri, pentru a fi recunoscute: variabile, constante, funcţii, tipuri de date.

De exemplu:

`typedef unsigned us;` declară tipul de date `us` prin care se identifică tipul implicit **unsigned**;
`us a,b;` declară identificatorii `a` şi `b` pentru două variabile de tipul `us`;
`const dim=10;` declară constanta 10 care se va identifica în program prin numele `dim`;
`int adun(int a, int b) {return a+b;}` declară o funcţie de adunare care se identifică prin numele `adun`.

În cadrul organizării modulare a programului este foarte importantă ***aria din program*** în care un identificator *acţionează*, este *recunoscut* şi *folosit*, adică aria lui de **vizibilitate**.

Din acest punct de vedere, identificatorii sunt clasificaţi în:

– Identificatori ***locali*** – care sunt vizibili numai în blocul (instrucţiunea compusă sau modulul) în care sunt declaraţi. *Parametrii* unui subprogram sunt variabile *locale*.

– Identificatori ***globali*** – care sunt vizibili în toate blocurile din cadrul programului.

În programul din figura 6.8 este folosită o funcţie **med** care realizează media aritmetică dintre elementele unui vector de numere întregi. De asemenea, pentru fiecare element al vectorului, după citire, se determină şi afişează divizorii lui.

```
#include<iostream.h>
#include<conio.h>
 int i,k,m,n;
int med(int v[])
{int i;
 float m=0;
 for(i=0;i<n;i++)
  m+=v[i];
 m/=n ;
 return int(m+0.5);
 }
 void main()
{clrscr();
 int T[10];
 for( k=1;k<=3;k++ )
  {cout<<"Nr elem. ";cin>>n;
   for( i=0;i<n;i++  )
    {cout<<"T["<<i+1<<"]=";cin>>T[i];
     int k;
     for(k=2;k<=T[i]/2;k++)
      if(T[i]%k==0)
       cout<<T[i]<<" divizor=k="<<k<<"; " ;
     }
     m =med(T,n);
    cout<<endl;
    cout<<"medie="<<m<<" pasul k="<<k<<endl;
   } getch();
 }
```

v[] – parametru = variabilă locală
i , m= variabile ***locale***, declarate şi folosite numai în **modulul med**. Sunt proprii modulului **med**, chiar dacă există altele cu acelaşi nume în exteriorul modulului.

variabile *globale*

k = variabilă ***locală***, declarată şi folosită numai în **blocul for**(i=0; i<n; i++), în exteriorul blocului funcţionând variabila globală cu acelaşi nume.

Figura 6.8

În general, nu se proiectează variabile globale pentru a scăpa de grija parametrilor. Forţa unui subprogram rezidă tocmai în posibilitatea de a parametriza acţiunea pe care el o realizează, astfel câştigându-şi generalitatea.

Pentru exemplificarea utilităţii folosirii subprogramelor, ne referim la problema 3 din categoria ***Teme date*** în Capitolul 3, paragraful 3.3.c, suma a două polinoame `P(X)` şi `Q(X)`. Pentru organizarea prelucrărilor, s-au declarat doi vectori, corespunzători polinoamelor. Apoi, fiecare în parte a fost citit, în mod explicit, în cadrul funcţiei **main**. Acum, cunoscând subprogramele, programul se poate transforma astfel:

– se introduce în text funcţia cunoscută de citire a unui vector, **cit_vect**, căreia i se va comunica, pentru fiecare polinom, vectorul în care se citesc valorile, limita maximă de elemente, variabila în care se va citi numărul concret de elemente şi numele polinomului.

– se vor înlocui secvenţele de citire explicită cu apelul acestei funcţii adaptat corespunzător polinomului citit.

Reguli de acces

1) Identificatorii declaraţi şi definiţi în exteriorul tuturor modulelor programului au statut de elemente **globale** şi sunt cunoscuţi în tot fişierul text al programului.

2) Un subprogram are acces la identificatorii declaraţi în cadrul lui şi la cei ai subprogramului care îl înglobează.
3) Identificatorii care desemnează parametrii unui subprogram au statut de variabile locale.
4) În situaţia în care într-un bloc se declară un identificator local cu acelaşi nume ca şi cel global, în acel bloc va funcţiona identificatorul local, declarat acolo, iar în afara blocului va funcţiona identificatorul global.
5) Elementele standard sau predefinite (subprograme standard, tipuri de date predefinite, constante predefinite) sunt considerate de nivel 0, adică elemente globale. Ele sunt întotdeauna accesibile şi pot fi redefinite la orice nivel inferior.

b) Comunicarea între module

Apelul unui subprogram provoacă o serie de operaţii secundare, pe care programatorul nu le vede direct, dar de care trebuie să ţină seama pentru a nu se afla în situaţii pe care nu le poate rezolva.

Pentru aceasta este nevoie, la început, ca programatorul să aibă o idee asupra modului în care este repartizată memoria de lucru pentru programul său. Reluăm, în figura 6.9, o diagramă care a fost folosită, pentru o lămurire sumară asupra acestui lucru:

– Variabilele globale sunt stocate în segmentul de date globale pe toată execuţia programului.

– La execuţia unui subprogram, se alocă în memoria internă, în segmentul de stivă a sistemului (**STACK**), un spaţiu de lucru pentru variabilele locale şi adresa de revenire din subprogram. Când s-a terminat execuţia codului subprogramului, acest spaţiu este eliberat şi returnat sistemului pentru a fi reutilizat la apelul şi execuţia altui subprogram. Această implementare permite o economie de spaţiu de care programatorul e bine să ţină seama.

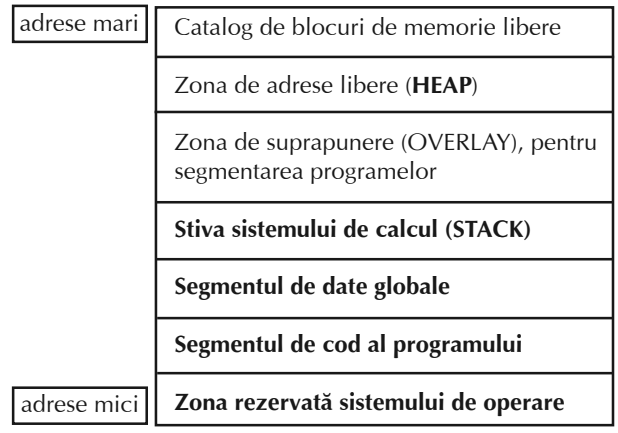


Figura 6.9

Să considerăm un program care utilizează trei subprograme de tip procedură, `sk`, `sm` şi `sn`. Fiecare subprogram prelucrează câte un tablou unidimensional, `k`, `m` şi, respectiv, `n`. Se prezintă două implementări ale programului, a) şi b).

a) Fiecare subprogram va lucra cu tabloul asociat lui, declarat, însă, ca variabilă globală. Zona alocată pentru program este de 15000 x 2 octeţi (suma mărimilor celor trei tablouri x lung_tip_întreg) şi se repartizează fix, pe durata programului, în *segmentul de date globale* (să ne închipuim că tablourile ar fi mai mari. Atunci suma de mai sus ar putea fi mai mare decât 65535 octeţi, capacitatea unui segment de memorie şi astfel programul nu se poate încărca pentru a fi executat!).

```
#include <...>
int k[5000], m[4500],n[5500];
void sk( )
{...}
void sm( )
{...}
void sn( )
{...}
void main ( )
{ sk; sm; sn; ...//apelurile}
```

b) Fiecare subprogram va avea tabloul respectiv definit ca variabilă locală. Programul va consuma un spaţiu de lucru, pentru fiecare funcţie, *pe rând*, de 5500 x 2 octeţi, adică lungimea maximă dintre cele trei tablouri, spaţiu care va fi alocat în mod dinamic, în timpul execuţiei, în *segmentul de stivă* a sistemului, fără a „sufoca" segmentul de date globale.

```
#include <...>
void sk( )
{ int k[5000]; ...}
void sm ( )
{ int m[4500]; ...}
void sn( )
{ int n[5500];...}
void  main ( )
{ sk; sm; sn;...}
```

Etapele execuţiei unui apel de subprogram:

1. Se întrerupe execuţia modulului apelant.
2. Se pregăteşte adresabilitatea parametrilor şi se aşază parametrii în STACK.
   Dacă parametrii sunt transmişi prin referinţă sau subprogramul lucrează cu variabilele globale, pe stivă sunt aşezate adresele acestora. Dacă parametrii sunt transmişi prin valoare, pe stivă se copiază valorile transmise în zone special alocate acestor parametri.
3. Se plasează în STACK, imediat după parametri, adresa de retur, adică adresa instrucţiunii ce trebuie realizată după întoarcerea din subprogram. Această adresă de retur ocupă 2 octeţi (adresa offset[1]), dacă subprogramul este în acelaşi segment de cod cu modulul apelant[2] . Dacă ele sunt în segmente de cod diferite, atunci adresa de retur ocupă 4 octeţi (2 pentru adresa de segment)[3] .
4. Se alocă spaţiu pentru variabilele locale.
5. Se lansează în execuţie codul maşină al subprogramului.

Etapele ieşirii din subprogram şi revenirii în modulul apelant:

1. Eliberarea spaţiului variabilelor locale alocat în STACK.
2. Reîncărcarea adresei instrucţiunii la care se face revenirea în modulul apelant.
3. Eliminarea parametrilor din STACK.
4. Revenirea în modulul apelant.
5. Dacă subprogramul este de tip funcţie operand, rezultatul calculat este comunicat modulului apelant într-o zonă temporară a cărei adresă este furnizată – în cazul rezultatului de formă dată structurată – sau în regiştrii UCP, dacă rezultatul este dată elementară.

Indicaţii pentru laborator. Pentru a observa cum trece prelucrarea unui program prin stările de: program (modul) principal, apoi comută în subprogram şi se instalează informaţiile în stivă, apoi cum descarcă stiva la ieşirea din subprogram ş.a.m.d, se va exersa rularea pas cu pas (**Trace into – F7**) a unor programe dintre cele date în capitol, având organizate pe ecran următoarele ferestre: fereastra de **editare**, fereastra **watch**, fereastra **output** şi fereastra **stivei** (activată, pe moment, din meniul principal **Debug→Call Stack**).

6. 3. Probleme rezolvate pentru fixarea noţiunilor prezentate

1) Se consideră un tablou unidimensional având maximum 100 de elemente reale. Să se realizeze o ***inversare a amplasării*** elementelor tabloului, fără a utiliza tablouri auxiliare.

Rezolvare. Se declară un tip de date numit `tabl` pentru structura vector ce va fi folosită. Se va organiza o funcţie *procedurală*, care va avea *un parametru de intrare-ieşire de tip* `tabl`. Acest parametru va primi tabloul iniţial şi va întoarce acelaşi conţinut, dar aşezat în ordine inversă. Unul dintre obiectivele problemei propuse este şi acela de a utiliza *modulele generale* pentru *citirea* şi *afişarea unui tablou* unidimensional.

```
//program inversare tablou
#include <iostream.h>
typedef float tabl[20];
void invtab(tabl t,unsigned dim)
{unsigned short i;float aux;
for (i=0;i<=dim/2;i++)
  { aux=t[i];
    t[i]=t[dim-i-1];
    t[dim-i-1]=aux;
    }}
void citire(tabl t,unsigned &dim,
            unsigned lim,char nume)
{unsigned i;
do {cout<<"Lungime tablou:";cin>>dim;
  } while (dim<2 || dim>lim);
for (i=0;i<dim;i++)
    {cout<<nume<<"["<<i+1<<"]=";
     cin>>t[i];
    }
}
void afisare(tabl t,unsigned dim)
{unsigned i;
  for (i=0;i<dim;i++) cout<<t[i]<<' ';
}
//progr. princ.
void main()
{ tabl a;unsigned lung;
citire(a,lung);
 afisare(a,lung);
 invtab(a,lung);
 afisare(a,lung);
}
```

[1] offset – adresa relativă în cadrul segmentului (vezi Informatică – manual pt. cl. a IX-a)
[2] Modelul de memorie NEAR = aproape (engl.)
[3] Modelul de memorie FAR = departe (engl.)

1. Scrieţi lista variabilelor locale existente în programul de mai sus, grupate după aria de vizibilitate.
2. Determinaţi dacă este util sau nu statutul variabilelor **a** şi **lung** şi dacă nu, scrieţi cum trebuie corectat programul.
3. Determinaţi dacă este utilă declarea tipului **tabl** prin comparare cu forma programului în cazul în care nu ar exista această declarare.

2) Se doreşte afişarea ***numerelor perfecte*** care se regăsesc între primele `n` numere naturale, `n` fiind citit (un număr perfect este acel număr pentru care suma divizorilor proprii naturali la care se adaugă valoarea 1 este egală cu numărul însuşi; de exemplu 6=1+2+3 este primul număr perfect).

Rezolvare. Numerele perfecte au fost definite de Pitagora. Problema are scopul de a figura într-o *funcţie operand* aflarea şi însumarea divizorilor proprii ai unui număr. Deoarece pentru fiecare număr natural, până la atingerea pragului `n`, se va face verificarea numărului cu suma divizorilor proprii `+1`, se deduce necesitatea unui subprogram care să calculeze această sumă pentru un număr dat. Fiind vorba de un singur rezultat, acest subprogram va primi o configurare de tip funcţie operand cu un singur parametru de intrare, anume numărul căruia trebuie să-i calculeze suma cerută.

```
#include <iostream.h>
unsigned n,nr;
unsigned suma_diviz(unsigned n)
{unsigned s,d;
      d=2;s=1;
      while (d<=n/2)
         {
             if (n % d==0)   s+=d;
             d++;
          }
return s;
}
void main()
{    do {
      cout<<"Numarul prag ";cin>>n;
      } while (n<6 || n>65535);
      for (nr=6;nr<=n;nr++)
       if (suma_diviz(nr)==nr)
        cout<<nr<<'\n';
}
```

1. Scrieţi lista variabilelor locale existente în programul de mai sus, grupate după aria de vizibilitate.
2. Scrieţi lista variabilelor globale existente în program.
3. Determinaţi dacă este util statutul variabilelor **n** şi **nr** sau nu şi dacă nu, scrieţi cum trebuie corectat programul.
4. Explicaţi de ce **n** parcurge valorile de la 6 la 65535.

3) Dându-se data zilei curente, să se calculeze ***data zilei următoare***. Valorile elementelor datei, **zi**, **lunã**, **an**, se introduc cu câte două cifre, respectiv, patru cifre pentru an.

Rezolvare. Se utilizează o *funcţie operand*, **nz,** pentru determinarea numărului permis de zile pentru luna citită. Dacă valoarea **luna** este greşită, rutina va întoarce valoarea zero, iar dacă valoarea **zi** este greşită, rutina întoarce –1. Calculul de succesiune pentru ziua următoare este realizat în programul principal.

```
#include <conio.h>
#include <iostream.h>
typedef unsigned us;
us nz(us zi,us luna,us an)
{ us z;
switch (luna)
{case 1 :
 case 3:
 case 5:
 case 7:
 case 8:
 case 10:
 case 12: z=31;break;
 case 2 :{z=28;if (an%4==0) z++;break;}
 case 4 :
 case 6 :
 case 9 :
 case 11 : z=30;break;
 default: z=0;
}
if (z && ! (zi>0 && zi<=z)) z=-1;
return z;
}

void main()
{us za,la, aa;
```

```
clrscr();
cout<<"Introduceti data pe rand :";
cout<<" zi,luna,an;anul cu patru cifre";
cin>>za>>la;
do { cout<<"anul: "; cin>>aa;
   }while (aa / 100== 0);
switch (nz(za,la,aa))
{case 0: cout<<"luna eronata";break;
 case -1: cout<<" zi eronata";break;
 default:
 {
 if (za+1>nz(za,la,aa))
   if (la==12) {aa++;la=1;za=1;}
     else { la++;za=1;}
  else za++;
 cout<<"Data zilei de maine:";
 cout<<za<<' '<<la<<' '<<aa; break;
 }sf. default
}//sf. switch
}
```

1. Scrieţi lista variabilelor locale existente în programul de mai sus, grupate după aria de vizibilitate.
2. Determinaţi dacă este util tipul de date **us**.
3. Scrieţi o variantă a programului în care funcţia **nz** să fie apelată o singură dată în funcţia **main**.
4. Scrieţi o variantă a programului care să nu mai folosească instrucţiunea **switch** pentru a determina, în funcţia **nz**, numărul **z.**

4) O problemă frecvent întâlnită este determinarea ***numărului de zile*** scurse între două date calendaristice.
Rezolvare. Rezolvarea se poate schiţa ca în figura 6.10.

Rezultă că între anii celor două date, notaţi în program prin **an1** şi **an2**, numărul zilelor scurse este calculat prin *(diferenţă între ani –1) x 365 + numărul de ani bisecţi*. La această valoare trebuie adăugată partea de zile scurse din fiecare an. Această parte, însă, se poate calcula la fel pentru cei doi ani, având în vedere că pentru primul ea se obţine din diferenţa între numărul de zile ale acelui an şi numărul de zile scurse până la **data 1**, iar pentru cel de-al doilea an, ea este chiar numărul de zile scurse din acel an până la **data 2**.

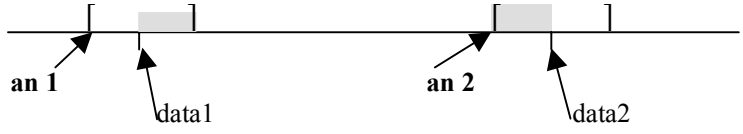


Figura 6.10

Astfel, rutina **piece**, de tip *funcţie operand*, va calcula pentru fiecare an numărul de zile scurse din acel an şi apoi, în programul principal valoarea calculată se va scădea sau nu, în funcţie de calculul pentru primul sau pentru al doilea an. De asemenea, **piece** realizează un calcul eficient pentru suma zilelor scurse din prima parte a anului, fără a mai necesita o structură de date sau de instrucţiuni prin care să se afle numărul de zile calendaristice ale fiecărei luni. În mod auxiliar, este definită o rutină de tip funcţie, **feb**, pentru situaţia lunii februarie.

```
#include<iostream.h>
#include<conio.h>
int feb(int anul)
{ if (anul%4==0)  return 1;
   return 2;  }
int piece(int anul,int luna,int ziua)
{int z;
  z=(luna-1)*30+luna/2;
  if (luna>2)  z=z-feb(anul);
  z=z+ziua;
 return z;
}
void main()
{ int  zz1,ll1,aa1,zz2,ll2,aa2,na,nza,
                    z1,z2,nrz;
clrscr();
cout<<"dati cele doua date ";
cin>>zz1>>ll1>>aa1>>zz2>>ll2>>aa2;
na=aa2-aa1-1;

if(aa1==aa2)  na=na+1;
nza=na*365+na/4;
z1=piece(aa1,ll1,zz1);
z2=piece(aa2,ll2,zz2);
if (aa1==aa2) nrz=z2-z1;
   else nrz=nza+366-z1+z2-1;
cout<<"nr de zile este "<<nrz;
getch();
}
```

rezolvă

1. Scrieţi lista variabilelor locale existente în programul anterior, grupate după aria de vizibilitate.
2. Determinaţi modul de calcul al valorii **z** din funcţia **piece**.
3. Modificaţi programul, astfel încât să verifice dacă anul este introdus cu patru cifre.

5) Se consideră maximum 20 de caractere care alcătuiesc o mulţime (nici un caracter nu se repetă). Să se afişeze toate ***submulţimile acestei mulţimi***.

Rezolvare. De la matematică se ştie că numărul de submulţimi ale unei mulţimi date, de la mulţimea vidă şi fără mulţimea totală, este de 2^n, unde **n** este cardinalul mulţimii de referinţă (totală). La prima vedere, este dificil de construit o prelucrare de selecţie a tuturor submulţimilor de câte un element, apoi de câte două ş.a.m.d.

Fără a avea pretenţia ca submulţimile să apară în ordinea numărului de elemente, dar verificând să nu apară soluţii identice, se poate imagina un proces de selecţie bazat pe **vectorul caracteristic asociat unei mulţimi**. Astfel, dacă mulţimea totală este A şi are conţinutul A={'a','d','g','i','s'}, la început, când formăm submulţimea vidă, vectorul caracteristic este V=(0,0,0,0,0). Dacă dorim submulţimea A_1={'a','s'}, atunci vectorul caracteristic trebuie să marcheze alegerea celor două elemente din A printr-o valoare 1 în poziţia corespunzătoare elementelor în mulţimea totală, adică V=(1,0,0,0,1). Dacă privim conţinutul vectorului caracteristic ca un număr binar cu 5 ordine, la început 5 de 0, atunci, prin adunarea unei valori 1, se va obţine V=(0,0,0,0,1), adică s-a selectat elementul de pe poziţia 5 din A. Adunând încă un 1, se obţine V=(0,0,0,1,0), adică s-a selectat submulţimea {'i'}. Continuând, apare V=(0,0,0,1,1), adică submulţimea {'i','s'} şi aşa mai departe până la V=(1,1,1,1,1), adică mulţimea totală.

Pe această idee se bazează rezolvarea propusă în programul de mai jos, criteriul de oprire din adunări fiind momentul în care tot vectorul conţine valoarea 1 în toate elementele sale. Adunarea nu se desfăşoară ca o simplă sumă de numere, ci *este simulată adunarea binară pentru un număr ale cărui cifre sunt elemente de tablou unidimensional*, subprogramul **adun**. Înainte de a se începe prelucrarea propriu-zisă, se verifică datele de intrare – dacă acestea alcătuiesc o mulţime.

```
#include <iostream.h>
typedef unsigned tab[20];
typedef char multime[20];
int verif_multime(multime x,unsigned
                  card)
{unsigned i,j;
 int este;
  i=0;
  while (i<card-1 && este)
    {j=i+1;
      while (j<card && este)
        {if (x[i]==x[j]) este=0;
         j++;
        }
      i++;
    }
return este;
}
void adun(tab v,unsigned n,unsigned &t)
{ unsigned i;
  t=1; //transport in ordinul urmator
  for (i=n-1;i>-1;i—)
    {v[i]=v[i]+t;
     if (v[i]>1)
         { v[i]=v[i]-2;
          t=1;
         }
      else t=0;
    }
}
void scrie(tab v,multime x,unsigned n)
{unsigned i;
 cout<<"{ ";
 for (i=0;i<n;i++)
  if (v[i]) cout<<x[i]<<',';
cout<<'\b'<<" }\n";//sterge virgula
}

void main()
{ tab v;multime m;
  unsigned nr,i,t;
do {
  cout<<"Cardinalul multimii ";cin>>nr;
  } while(nr<2 || nr>20);
do
 { cout<<"Elementele multimii ";
   for (i=0;i<nr;i++)
    {
     cout<<"m["<<i+1<<"]=";
     cin>>m[i];v[i]=0;
    }
 } while(!verif_multime(m,nr));
t=0;
while (!t)
 {adun(v,nr,t);
  scrie(v,m,nr);
 }
}
```

1. Scrieţi lista variabilelor locale existente în programul de mai sus, grupate după aria de vizibilitate.
2. Determinaţi modul de calcul al valorii **v[i]** din funcţia **adun**.
3. Scrieţi un alt program care, inspirându-se din acesta şi din funcţia **adun**, să realizeze adunarea a două numere naturale cu câte maximum 50 de cifre fiecare.

6) Se citeşte o linie de text. Se cere, pentru fiecare tip de vocală întâlnită, afişarea unei bare orizontale cu atâtea asteriscuri câte apariţii are acea vocală în text (***Histograma frecvenţelor***).

Rezolvare. Textul dat va fi încărcat într-o variabilă de tip şir de caractere, **c**. Un tablou, **nr**, cu 5 elemente, va conţine frecvenţele celor 5 vocale. Pentru determinarea vocalelor din şir s-a proiectat o *funcţie procedurală*, **vocale**, care are pe **s** şi pe **n** ca parametri de intrare-ieşire, sinonime ale variabilelor **c** şi **nr**. Afişarea barelor de asteriscuri este realizată de subprogramul **asterisc**.

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
typedef char sir[200];
sir voc="aeiou";

void vocale(sir s,int n[5])
{int lung,i,j;
 lung=strlen(s);
 for (j=0;j<5;j++)
    {  n[j]=0;
       for(i=0;i<lung;i++)
        if (voc[j]==s[i]) n[j]++;
    }
}
void asterisc(int n,char v)
{  cout<<v<<" ";
   while (n>0) {
      cout<<"*";
      n—;
     }
   cout<<"\n";
}

void main()
{ sir c;int nr[5],i;
  cout<<"Dati sirul de litere mici \n";
  cin>>c;
  vocale(c,nr);
  for(i=0;i<5;i++)
  asterisc(nr[i],voc[i]);
}
```

1. Scrieţi lista variabilelor locale şi globale existente în programul de mai sus, grupate după aria de vizibilitate.
2. Determinaţi modul de calcul al valorii **n[j]** din funcţia **vocale**.
3. Determinaţi sarcinile variabilei parametru **n** din funcţia **asterisc**.

6.4. Lucrul cu bibliotecile

6.4.1. Subprograme standard (vezi Anexa 1)

1. Program pentru ***testarea gradului de atenţie*** (scop interdisciplinar)

Rezolvare. Pe ecran se va afişa, în mod grafic, un cuvânt care pulsează. La anumite momente cuvântul se schimbă cu altul de aceeaşi lungime şi asemănător ca arie a semnificaţiei şi litere conţinute. Se cere utilizatorului să introducă numărul de apariţii ale celuilalt.

Cuvintele alese pentru testare sunt *găină* şi *gâscă*. În mod grafic, s-au ales un font şi o mărime de afişare convenabile. S-a proiectat o întârziere (funcţia delay) pentru pulsaţii (250 ms). Într-un vector, **m**, se contorizează de câte ori apare cuvântul *gâscă* pentru a se putea apoi compara cu numărul introdus de utilizator în variabila **r**. Executarea testului se face predefinit, de 20 de ori.

Scopul exemplului este de a pune în evidenţă utilizarea unor funcţii standard, aici din biblioteca `graphics.h`.

```
#include<graphics.h>
#include<conio.h>
#include<stdlib.h>
#include<iostream.h>
#include<dos.h>

void main()
{ char text[2][6]={"gaina","gasca"};
int dg=DETECT,mg,i,m[20],ori=0,r;
randomize();
initgraph(&dg,&mg,"d:\\borlandc\\bgi");
```

```
settextstyle(7,0,8);
//font,aliniere,marimea
for(i=0;i<20;i++)
{     setcolor(WHITE);
      m[i]=random(2);
      outtextxy(200,200,text[m[i]]);
      delay(250);
      setcolor(BLACK);
      outtextxy(200,200,text[m[i]]);
      delay(500);
}
delay(200);
closegraph();
```

```
cout<<"De cate ori a aparut cuvantul
GASCA ?";
cin>>r;
for(i=0;i<20;i++)
  if(m[i]==1)ori++;
if(ori==r)cout<<"Ati observat bine ";
      else
         cout<<"Nu ati observat bine ";
cout<<"\n Apasati o tasta pentru
                  iesire";
getch();
}
```

2. Program pentru ***testarea capacităţii de memorare vizuală***
Rezolvare. Programul prezintă 7 pătrate de culori diferite, alese aleatoriu. După un timp de 7 secunde (`delay(7000)`) ele dispar şi i se cere utilizatorului să introducă numele culorilor reţinute de el, în ordinea în care au apărut. La sfârşit, utilizatorului i se dă un punctaj. La început se prezintă lista de culori cu denumirile şi pătratele colorate, pentru ca utilizatorul să poată introduce denumirile corecte ale culorilor.

```
#include<graphics.h>
#include<conio.h>
#include<stdlib.h>
#include<iostream.h>
#include<dos.h>
#include<string.h>
void patrat1(int x,int y, int lat,int i)
{     setfillstyle(SOLID_FILL,i);
      bar(x,y,x+lat,y+lat);
}//deseneaza patrate-lista de culori
int patrat(int x,int y, int lat)
{  int cul;
      do
       cul=random(15);while(cul==0);
      setfillstyle(SOLID_FILL,cul);
      bar(x,y,x+lat,y+lat);
      return cul;
}//deseneaza patrate colorate aleatoriu
void main()
{  char culori[16][10]={" ",
"albastru","verde","turcoaz", "rosu",
"visiniu","maro","gri","fumuriu","bleu",
"vernil","azuriu","corai","mov","galben",
"alb"};
 int v[7];
 char r[10];
 randomize();
 int dg=DETECT,mg,i,x,y,lat,cul,p=0;
 x=30;y=30;lat=20;
 initgraph(&dg,&mg,"d:\\borlandc\\bgi");
```

```
 outtextxy(10,10,"Lista culorilor");
for(i=1;i<16;i++)
  {patrat1(x,y,lat,i);
   outtextxy(50,y,culori[i]);
   y+=30;
   }
outtextxy(300,200,"Apasati o tasta pentru
         a incepe");
getch();
cleardevice();
y=30;
for(i=0;i<7;i++)
      {v[i]=patrat(x,y,lat);
       x+=30;
      }
delay(7000);
closegraph();
cout<<"Scrieti lista culorilor
                        prezentate ";
for(i=0;i<7;i++)
  {cin>>r;
   if(strcmp(culori[v[i]],strlwr(r))==0)
       p++;
  }
cout<<"Aveti punctajul "<<p<<" din 7
                              posibil";
cout<<"\n Apasati o tasta pentru
                                 iesire";
getch();
}
```

6.4.2. Crearea de biblioteci proprii

În limbajul **C/C++** se pot crea biblioteci proprii prin construirea unui text sursă în care se vor scrie funcţiile ce trebuie cuprinse în bibliotecă şi, eventual, definiţiile de structuri sau tipuri de date necesare în prelucrări. Se vor îngloba şi bibliotecile limbajului care sunt necesare în prelucrările din funcţiile definite. Fişierul sursă creat astfel se va salva cu extensia **h**.


exemplu

Dacă se doreşte construirea unei biblioteci care să cuprindă funcţii de prelucrare a matricelor de tipul ($M_{m\times n}$), fişierul cu textele funcţiilor de prelucrare se poate salva cu numele **matrice.h**. Structura principală a bibliotecii **matrice.h** ar putea fi următoarea, în ce s-a considerat că se lucrează cu matrice întregi, cu *m* linii (maximum 20) şi *n* coloane (maximum 20):

```
#include <math.h>
void citire(int &m, int &n, int mat[][20], char nume[])
{ // functie pentru citirea dimensiunilor si valorilor matricei mat
      do {cout<< "Dati numarul de linii, maximum 20 :  " ;
            cin>>m ;}while(m<2||m>20) ;
      do {cout<< "Dati numarul de coloane, maximum 20 :  " ;
            cin>>n ;}while(n<2||n>20) ;
      for(int i=0;i<m;i++)
            for(int j=0;j<n;j++)
                  {cout<<nume<<"["<<i+1<<","<<j+1<<"]=";
                    cin>>mat[i][j];
                  }
}
```


observă

Se observă că prin citirea datelor matricei se vor completa cu valorile concrete date de utilizator atât variabilele simple **m** şi **n,** cât şi data structurată **mat,** fapt ce necesită transmiterea lor ca *parametri de intrare-ieşire*. Acest lucru este pus în evidenţă prin referinţa la adresele fizice ale lui **m** şi **n,** iar pentru matrice prin însuşi numele **mat,** ce reprezintă adresa simbolică a structurii.

```
void afisare(int m, int n, int mat[][20])
{     ... secventa pentru afisare  }
void adun(int m, int n, int mat1[][20],int mat2[][20],int suma[][20] )
{........secventa pentru adunare  }
int simetrica(int m, int mat[][20])
{.......secventa pentru verificarea a_ij==a_ji, }
etc.//alte prelucrari
```

Utilizarea bibliotecii create

Programul în care este folosită biblioteca trebuie să cuprindă indicaţia pentru compilator:

```
#include "matrice.h"
```

iar biblioteca trebuie să existe în acelaşi director cu programul respectiv.

De exemplu, se poate crea următorul program:

```
#include "matrice.h"
#include<iostream.h>
void main()
{
      int m,n,p,q,a[5][7],b[5][7],s[5][7],c[10][10];
      cout<<"Dati matricea A"<<endl;
      citire(m,n,a,"A");
      afisare(m,n,a);
      cout<<"Dati matricea B"<<endl;
      citire(p,q,b,"B");
      afisare(p,q,b);
      if(m==p && n==q)
```

```
            {adun(m,n,a,b,s);
             cout<<"Suma matricelor:"<<endl;
             afisare(m,n,s);
            }
          else cout<<"Matrice de dim. # si nu se pot insuma " <<endl ;
      cout<<"Dati matricea C, patratica"<<endl ;
      citire(m,n,c,"C") ;
      if (m==n)
            if(simetrica(m,c))
                  cout<< "Matricea C este simetrica " <<endl;
               else cout<<"Matricea C nu este simetrica"<<endl;
       else cout<< "Matricea C nu e patratica "<<endl;
...alte prelucrari.....
}
```

1. Bibliotecă de lucru cu numere întregi

Biblioteca de funcţii matematice, **math**, pusă la dispoziţie de limbajul **C/C++,** nu cuprinde o serie de funcţii de prelucrare elementară a numerelor întregi care sunt destul de frecvent utilizate de începători: verificarea parităţii unui număr, stabilirea semnului său, determinarea numărului de cifre, a sumei cifrelor, a cifrei de control, a inversului lui în citire etc. (cifra de control este obţinută prin repetarea însumării cifrelor numărului, apoi a cifrelor acelei sume ş.a.m.d., până se ajunge la o singură cifră; de exemplu, pentru 132457 se obţine 4: suma cifrelor numărului este 22, apoi suma cifrelor lui 22 este 4 şi procesul se opreşte).

APLICAŢIE

Mai jos este prezentată o bibliotecă care a fost numită **întregi** şi care conţine câteva dintre prelucrările elementare asupra numerelor întregi.

```
#include<math.h>
int par(long n)
//stabileste paritatea
{     return n%2==0;
}
int sgn(long n)
{ //stabileste semnul numarului
      if(n<0) return -1;
      if(n==0) return 0;
      return 1;
}
unsigned nr_cif(long n)
{ //stabileste numarul de cifre
      unsigned nc=0;
      do{
        nc++;
        n/=10;
        }while(n);
      return nc;
}
unsigned prima(long n)
{ //stabileste prima cifra a numarului
      while(n>9)
            n/=10;
      return n;
}
unsigned sum_cif(long n)
{ //stabileste suma cifrelor
      unsigned s=0;
      long aux=labs(n);
//se lucreaza cu modulul unui numar long
      while(aux)
            {s+=aux%10;
             aux/=10;
            }
      return s;
}
long invers(long n)
{
//stabileste inversul in citire

long inv=0,aux=labs(n);
 if(nr_cif(aux)==10 && aux%10>1) return 0;
      else
        {while(aux)
            {inv=inv*10+aux%10;
             aux/=10;
            }
         return sgn(n)*inv;
        }
}

unsigned control(long n)
{
//stabileste cifra de control
  unsigned s=sum_cif(n);
```

```
      while(s>9)
           s=sum_cif(s);
      return s;
}

void diviz(long n,long v[],unsigned &k)
{
//determina divizorii proprii pozitivi
 long d;
 for(d=2;d<=abs(n)/2;d++)
      if(n%d==0)
            {v[k]=d;
             k++;
            }
}
```

a) Deoarece împărţirea întreagă *a : b* este greşit proiectată în limbajele **C/C++** pentru operatorii / şi %, nerespectând definirea matematică a restului: $0 \le r < |b|$, funcţiile pentru suma cifrelor şi inversatul numărului lucrează cu valoarea absolută a numărului *n* primit ca parametru.
b) Inversatul numărului poate depăşi reprezentarea **long**, astfel că s-a prevăzut un test „grosier" în acest sens, situaţie în care funcţia întoarce valoarea 0.
c) inversatul numărului nu va fi folosit pentru numere de o singură cifră.
d) inversatul numărului este returnat de către funcţie după ce i s-a aplicat semnul numărului n.
e) funcţia **diviz** are de completat **k** poziţii în vectorul **v** cu divizorii proprii găsiţi, astfel că **v** şi **k** sunt parametri de intrare-ieşire, ceea ce obligă să fie comunicaţi prin referinţă la adresele lor fizice.

În programul de mai jos este folosită această bibliotecă pentru o valoare citită în variabila **nr**.

```
#include<iostream.h>
#include<conio.h>
#include "intregi.h"
void main()
{long nr,rev,div[20];
 clrscr();
 cout<<"Dati numarul de prelucrat ";
 cin>>nr; cout<<endl;
 if(par(nr))cout<<"Numar par";
   else cout<<"Numar impar";
cout<<endl<<"Numarul are "<<nr_cif(nr)
                <<" cifre"<<endl;
cout<<"Suma cifrelor este: "
                  <<sum_cif(nr)<<endl;
if(nr>9)
 {rev=invers(nr);
  if(rev==0)
  cout<<"Inversul depaseste reprez.
                       long"<<endl;
  else cout<<"Numarul inversat este:"
                       <<rev<<endl;
  }
 else cout<<"aveti un numar de o cifra"
                         <<endl;
cout<<"Cifra de control asociata:
            "<<control(nr)<<endl;
unsigned k=0;
diviz(nr,div,k);
if(k)
 {cout<<"Lista divizorilor proprii
              pozitivi: "<<endl;
  for(int i=0;i<k;i++)
      cout<<div[i]<<" ";
}
else cout<<"Numarul este prim ";
//sau "Numarul nu are divizori proprii"
getch();
}
```

Rezultatele afişate pentru nr=12357 sunt prezentate în figura 6.11.

```
Output
Dati numarul de prelucrat 12357

Numar impar
Numarul are 5 cifre
Suma cifrelor este: 18
Numarul inversat este:75321
Cifra de control asociata: 9
Lista divizorilor proprii pozitivi:
3 9 1373 4119
```

Figura 6.11

rezolvă

Definiţi şi adăugaţi în fişierul **intregi.h** următoarele funcţii :

- **cat** – pentru calculul corect al câtului împărţirii întregi dintre variabilele *a* şi *b*;
- **rest** – pentru calculul corect al restului împărţirii întregi dintre variabilele *a* şi *b*;
- **prim** – pentru determinarea proprietăţii de număr prim a valorii din *n*, fără a folosi funcţia `diviz` ;
- **prim_n** – pentru determinarea primelor *n* numere prime şi înregistrarea lor într-un vector *p*, valoarea maximă pentru *n* fiind 30;
- **suma_n** – pentru determinarea sumei primelor *n* numere naturale;
- **palind** – pentru determinarea proprietăţii de număr palindrom a lui *n* (de exemplu, *n*=1221) ;
- **cmmdc** – pentru determinarea c.m.m.d.c (a,b), unde *a* şi *b* sunt de tipul **long**;
- **cmmmc** – pentru determinarea c.m.m.m.c (a,b), unde *a* şi *b* sunt de tipul **int**;
- **power** – pentru puterea a^b, realizată prin înmulţiri repetate, unde *a* şi *b* sunt de tipul **int**;
- **factori** – pentru descompunerea în factori primi a numărului *n* de tipul **long**; funcţia va avea antetul:

```
void factori (long n, long f[], unsigned m[], unsigned & k)
```

unde `f` este vectorul factorilor primi, `m` este vectorul ordinelor de multiplicitate corespunzătoare şi `k` este numărul de factori găsiţi.

Construiţi un program prin care să încercaţi funcţionarea bibliotecii pentru toate situaţiile.

exemplu

2. Biblioteca de lucru cu numere compelxe

Un număr din mulţimea numerelor complexe, notat frecvent prin **z = a + i b**, este o construcţie de tip structură de date pentru un program care trebuie să realizeze operaţii cu astfel de numere. Compilatoarele nu dispun de un tip implicit predefinit pentru un număr complex, aşa cum este cazul numerelor naturale, întregi şi reale.

Pentru compilatorul **C++** există definită biblioteca **complex,** care lucrează cu numărul complex definit prin structura

```
struct complex {double x, double y;};
```

unde `x` şi `y` sunt partea reală, respectiv, partea imaginară ale numărului complex.

Această structură este definită în cadrul bibliotecii **math** şi este folosită de către clasa **complex** pentru a genera un număr complex şi pentru a opera cu el.

De exemplu, fie două numere complexe, **z1** şi **z2**. În programul de mai jos, aceste numere vor fi introduse în operaţiile de adunare, înmulţire şi împărţire utilizând operatorii +, * şi /. Aceşti operatori **capătă** proprietăţile de adunare, înmulţire şi, respectiv, împărţire de numere complexe prin funcţii speciale din cadrul bibliotecii **complex**. Pentru programator, acest fenomen este transparent, el scriind semnele respective şi în cazul expresiilor de calcul numeric obişnuite. În realitate, se lucrează în programare pe obiecte cu clasa de obiecte **complex**. Biblioteca operează cu obiectele **z1** şi **z2**, definite ca variabile de tipul nou introdus, tipul **complex**, şi aplică operatorii de calcul prin intermediul unor funcţii de supraîncărcare a operatorilor obişnuiţi cunoscuţi pentru numerele reale.

În textul programului, după ce sunt citite valorile părţilor reală şi imaginară ale numărului complex, este nevoie de activarea unei funcţii care să construiască numărul complex în structura definită pentru el. Astfel, linia:

```
z1=complex(a,b);
```

realizează această construire a variabilei (obiectului) `z1` din valorile introduse în variabilele reale **x** şi **y**. La fel se întâmplă şi pentru **z2**.

```
#include<complex.h>
#include<iostream.h>
#include<conio.h>
void main()
{
      double a,b;
      complex z1,z2;
```

```
clrscr();
      cout<<"Dati numarul complex 1:  ";cin>>a>>b;
      z1=complex(a,b);
      cout<<"\nDati numarul complex 2:  ";cin>>a>>b;
      z2=complex(a,b);
      cout<<"\nSuma celor doua numere: "<<z1+z2<<endl;
      cout<<"Produsul celor doua numere: "<<z1*z2<<endl;
      cout<<"Catul celor doua numere: "<<z1/z2<<endl;
      cout<<"Modulul primului numar: "<<abs(z1)<<endl;
      cout<<"Conjugatul celui de-al doilea numar: "<<conj(z2)<<endl;
getch();
}
```

Rezultatele pe care le afişează programul pentru numerele `z1=2 + i 3` şi `z2= 1 - i 2` vor fi ca în figura 6.12.

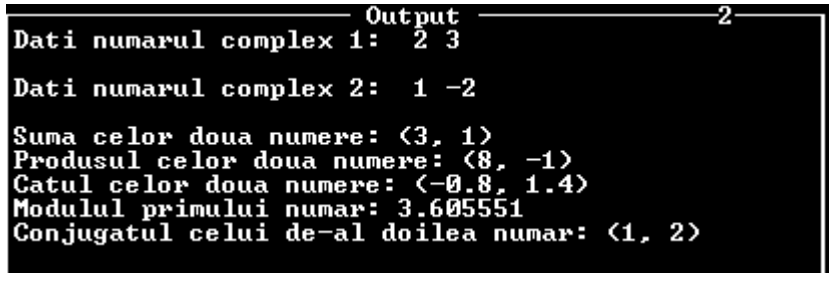


Figura 6.12

Pentru calculul modulului numărului complex `z1` s-a folosit funcţia **`abs(z1)`** care este definită special în **`complex.h`** conform calculului matematic: $|z1| = \sqrt{a^2 + b^2}$.

Pentru calculul conjugatului numărului `z2` s-a folosit funcţia **`conj(z2)`** din aceeaşi bibliotecă, care pentru numărul $z = a + ib$ realizează conjugatul său: $\overline{z} = a - ib$.

Biblioteca **`complex`** conţine şi o funcţie pentru afişarea unui număr complex prin prezentarea între paranteze rotunde a valorilor celor două părţi ale numărului.

APLICAŢIE

În situaţia în care nu dispunem de biblioteca **`complex`** (cazul limbajului **`C`** standard) sau dorim să creăm propria noastră bibliotecă, fără a cunoaşte tehnica programării pe obiecte, atunci vom defini următorul text în cadrul fişierului cu numele, de exemplu, **`biblcplx.h`:**

```
#include <math.h>
typedef struct complx{float re,im;};
float modul(complx z)
{
      return sqrt(z.re*z.re+z.im*z.im);
}
complx conjugat(complx z)
{  complx cj=z;
      cj.im=-cj.im;
      return cj;
}
complx suma(complx z1,complx z2)
{  complx s;
      s.re=z1.re+z2.re;
      s.im=z1.im+z2.im;
      return s;
}
complx produs(complx z1,complx z2)
{
      complx p;
      p.re=z1.re*z2.re-z1.im*z2.im;
      p.im=z1.re*z2.im+z2.re*z1.im;
      return p;
}
complx cat(complx z1,complx z2)
{
      complx c,cj;
      float m=modul(z2);
      m*=m;        //patratul modulului
      cj=conjugat(z2);
      c=produs(z1,cj);
      c.re/=m;
      c.im/=m;
      return c;
}
//sfarsit biblcplx
```

Pentru definirea formatului de număr complex s-a creat structura **complx**, în care sunt declarate două câmpuri de tip **float**: **re** şi **im**, pentru cele două părţi ale numărului complex.

Deoarece utilizarea bibliotecii a fost făcută de un program din acelaşi director în care ea a fost salvată, textul de mai jos foloseşte ghilimelele în cadrul directivei de compilare **include**.

```
#include "biblcplx.h"
#include <conio.h>
#include<iostream.h>
void main()
{  complx z1,z2,z3;char semn;
      clrscr();
      cout<<"Dati primul numar complex ";
      cin>>z1.re>>z1.im;
      cout<<"\n Modulul primului numar: "<<modul(z1);
      cout<<"\n Dati al doilea numar complex ";
      cin>>z2.re>>z2.im;
      cout<<"\n Modulul celui de-al doilea numar: "<<modul(z2);
      z3=suma(z1,z2);
      if(z3.im<0) semn='-';else semn='+';
      cout<<"\n Suma celor doua numere: "<<z3.re<<semn<<"i"<<fabs(z3.im)<<endl;
      z3=produs(z1,z2);
      if(z3.im<0) semn='-';else semn='+';
      cout<<"Produsul celor doua numere: "<<z3.re<<semn<<"i"<<fabs(z3.im)<<endl;
      z3=cat(z1,z2);
      if(z3.im<0) semn='-';else semn='+';
      cout<<"Catul celor doua numere: "<<z3.re<<semn<<"i"<<fabs(z3.im)<<endl;
      getch();
}
```

Biblioteca creată mai sus nu conţine toate funcţiile necesare prelucrării numerelor complexe. Se vor adăuga următoarele funcţii:

– **real** – furnizarea părţii reale a unui număr complex;
– **imag** – furnizarea părţii imaginare a unui număr complex;
– **unghi** – furnizarea unghiului unui număr în planul complex (care corespunde funcţiei arg din **complex.h**).
(Indicaţie – vezi figura 6.13, unde atan2(a,b) realizează *arctg a/b*)

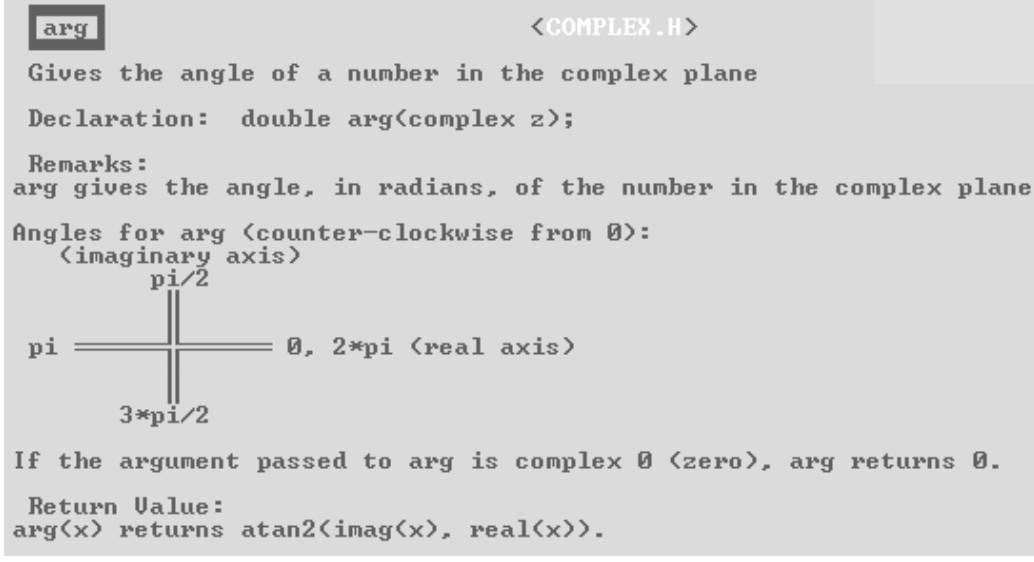


Figura 6.13

– **ln** – furnizarea logaritmului natural al unui număr complex: ln z = ln(| z |) + i * unghi;
– **lg** – furnizarea valorii lg z = ln z / ln 10;
– **exp** – furnizarea valorii e^z : $e^{(x+iy)} = e^x \bullet (\cos y + i \sin y)$;
– **sqrt** – furnizarea valorii $\sqrt{z}$: $\sqrt{z} = \sqrt{|z|} * \left(\cos\left(\text{unghi}(z)/2\right) + i\sin\left(\text{unghi}(z)/2\right)\right)$;
– **sin** – sinusul lui z : sin z = (e^{i*z} - e^{-i*z})/(2 i);
– **cos** – cosinusul lui z : cos z = (e^{i*z} + e^{-i*z})/ 2 ;
– **tan** – tangenta lui z : tg z = sin z / cos z;
– **power** – valoarea puterii $z1^{z2} = e^{z2 \ln z1}$;
– **trig** – pentru transformarea în formă trigonometrică a numărului complex.

Probleme propuse

1) Se vor distribui elevilor, individual, programele prezentate şi comentate în capitolele 1–4 pentru a le transforma în manieră modulară (cu subprograme) şi a le îngloba în portofolii.
2) Se vor transforma programele prezentate în Capitolul 5 în forma modulară.
3) Să se construiască un program pentru un graf neorientat, care conţine un subprogram pentru citirea muchiilor existente şi completarea matricei de adiacenţă şi un subprogram care determină dacă un graf neorientat este complet sau nu.
4) Să se alcătuiască un program care realizează interclasarea a doi vectori ordonaţi crescător, utilizând subprograme pentru: citirea unui vector, adăugarea unui element nou în vectorul în care se construieşte rezultatul, afişarea unui vector.
5) Să se alcătuiască un subprogram care citeşte un şir de caractere şi returnează acel şir inversat.
6) Să se alcătuiască un subprogram care verifică dacă un vector de numere naturale alcătuieşte o mulţime şi să se înglobeze într-un program care citeşte n vectori şi realizează mulţimea lor de intersecţie dacă aceştia respectă condiţia de mulţime.
7) Să se alcătuiască o bibliotecă, **vector.h**, care va conţine funcţii pentru următoarele prelucrări efectuate asupra unui vector de numere întregi: citirea, afişarea, suma elementelor, determinarea elementului maxim/minim, ordonarea crescătoare/descrescătoare, statistica elementelor nule, pozitive şi negative.
8) Să se realizeze un program pentru calculul unei aproximante a numărului transcendent **e**, după formula de dezvoltare în serie:

   $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \ldots + \frac{1}{n!}$ cu o eroare **eps** citită, utilizând un subprogram pentru calculul **n!**
9) Să se scrie o funcţie care primeşte adresele (indicii) de început a două liste simplu-înlănţuite, **p** şi **q**, şi realizează concatenarea celei de-a doua liste la prima. Funcţia va întoarce adresa primului element al listei finale sau –1 în situaţia în care listele primite sunt vide.

Capitolul 7

Tehnica recursivității

În acest capitol veţi învăţa despre:

- Tehnica recursivităţii aplicată în rezolvarea problemelor care conţin procese recurente
- Organizarea prelucrărilor recurente ca subprograme recursive
- Proiectarea subprogramelor recursive
- Mecanismul de lucru recursiv

7.1. Noțiuni introductive. Recurență – Recursie

Să presupunem că observăm o plantă, în timp, din momentul în care este pusă sămânţa în pământ. La scurt timp după însămânţare răsare o mlădiţă şi se înalţă rapid, în intervalul de timp t_1 (fig. 7.1), după care apare un mugure. Din acesta se dezvoltă o frunză. După câteva zile, se vede cum se ramifică mlădiţa şi apar alte frunze, într-un ritm exploziv, până la nivelul n_1. După un timp t_2, se atinge nivelul n_2 şi procesul de apariţie de rămurele şi frunze încetineşte progresiv, pe timpii t_3 şi t_4, diferenţele de nivel fiind din ce în ce mai mici. Pe timpii t_5 şi t_6 procesul de creştere încetineşte până la oprire. Este momentul în care informaţia genetică moştenită din sămânţă comandă plantei încetarea creşterii, deoarece a ajuns la maturitate: **nivelul final**.

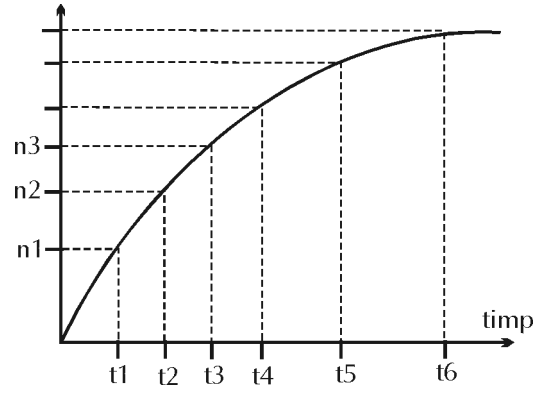


Figura 7.1

Din aceste observaţii se vede cum efortul plantei pentru a urca un nivel se bazează pe efortul depus anterior, pe cât a acumulat din efortul de la nivelurile inferioare acestuia, iar dozarea efortului (viteza de creştere) este măsurată prin compararea cu informaţia genetică.

Exemplul de mai sus **pare** o prezentare simplă a unui proces repetitiv de tipul **`while`**.

Proiectarea unei acţiuni repetitive presupune ca un fenomen ***să fie condus*** prin următoarele etape:

e1) Stabilirea procesului care trebuie repetat.

e2) Configurarea elementului motor al repetării (care este variabila ce determină trecerea în altă stare).

e3) Fixarea condiţiei de reluare a repetiţiei.

e4) Comandarea reluării.

Aceste lucruri sunt deja cunoscute şi exploatate în programarea *explicită* a fiecărui pas al prelucrărilor cerute de rezolvarea unei probleme, adică modul **iterativ** de programare. Deşi etapa **e4** (*trimiterea la reluarea procesului)* nu apare printr-o comandă proprie, trebuie totuşi pusă în evidenţă, fiind un pas important de prelucrare în abordarea iterativă.

De exemplu, procesul simplu de ***numărare,*** din 1 în 1, până la 100, este condus *iterativ*, punându-se în evidenţă funcţia matematică de succesiune definită pe mulţimea numerelor naturale:

i←1 Cât timp i ≤ 100 execută i←i+1 **Reia**	*motorul este i* *condiţia de continuare* *procesul* comanda explicită de *reluare*

Exprimat în pseudocod, proiectul *iterativ* al unei acţiuni repetitive apare mai clar decât într-un limbaj de programare în care, prin convenţiile de sintaxă, se „absoarbe" comanda explicită *Reia*.

Să presupunem acum că numărarea este realizată de un **aparat** – de exemplu, un cronometru fixat să sune după minutul 100. Numărul va creşte cu o unitate la fiecare minut (care va însemna un proces subordonat de numărare a secundelor până la 60). Fiecare nouă creştere se va face după ce aparatul compară cât **a acumulat** până atunci cu valoarea 100 la care a fost fixat. Dacă prin comparare a rezultat că mai trebuie să crească, aparatul se **autocomandă** pentru o nouă adunare cu 1. Nu trebuie ca cineva să pornească din nou cronometrul după trecerea fiecărui minut.

Ceea ce diferenţiază o acţiune repetitivă iterativă de acţiunea repetitivă realizată de aparat sau de plantă este faptul că în funcţionarea aparatului (sau a plantei) *procesul îşi comandă lui însuşi reluarea,* dacă nu este îndeplinită condiţia de oprire, fără să fie nevoie de o comandă exterioară de reluare.

Un astfel de proces este denumit *recursiv*.

Prin **recursivitate** se înţelege proprietatea intrinsecă a unui proces de a i se putea descrie funcţionarea pe baza unor referiri la el însuşi.

Procesele recursive sunt utilizate în aplicaţiile în care soluţia unei probleme se poate exprima prin termenii unor aplicări succesive ale aceleiaşi rezolvări la subproblemele delimitate în cadrul problemei date.

În termeni matematici, recursivitatea se regăseşte ca o operaţie de compunere a procesului (în particular, funcţie) cu el însuşi, compunere întâlnită în **formulele recurente**.

Definirea mecanismului inducţiei complete ascunde un proces recurent: dacă o proprietate este adevărată pentru $k = 0$ şi, considerând-o adevărată până la $k = n - 1$ se poate demonstra că este adevărată şi pentru cazul $k = n$, atunci se poate spune că acea proprietate este adevărată pentru orice $k \geq 0$.

Mecanismul *compunerii funcţiilor*, învăţat la matematică, este de foarte mare ajutor în urmărirea execuţiei unui proces recursiv şi, apoi, în programarea lui corectă.

În matematică, termenul de recurenţă este utilizat cu ocazia extinderii unei proprietăţi sau a unui calcul asupra aceloraşi obiecte/procese din aproape în aproape, pe baza calculelor anterioare.

În programare, ca termen, ***recursivitatea denumeşte recurenţa***.

1) Ca un contraexemplu, într-un dicţionar nu se va găsi nici un cuvânt care să fie definit pe baza lui însuşi, cum ar fi, de exemplu, „*casa* este un obiect care are destinaţia de *casă* pentru o persoană”(!).
2) Dacă am defini un arbore, se poate observa că definiţia este recursivă. Astfel, un arbore este format dintr-un nod rădăcină şi, eventual, unul sau mai multe noduri descendente. Orice nod descendent este un arbore.

3) Procesul de evaluare a unei expresii fără paranteze se bazează pe regulile de ordine a operaţiilor. Evaluarea unei expresii conţinând paranteze presupune evaluarea expresiilor dintre paranteze, în ordinea imbricării acestora, iar evaluarea întregii expresii date devine un proces prelucrat recursiv (de fiecare dată fiind prelucrată paranteza determinată a fi prioritară în acel moment, până la rezultatul final al întregii expresii date).

4) Calculul pentru factorialul unui număr natural, n, reprezintă o prelucrare recursivă, deoarece:

– dacă $n = 0$ atunci $n! = 1$;

– dacă $n > 0$, atunci $n! = n*(n-1)!$, adică factorialul unui număr se obţine prin înmulţirea acelui număr cu factorialul predecesorului său.

De exemplu, $5!= 5* 4!= 5*4*3! = 5*4*3*2!=5*4*3*2*1!=5*4*3*2*1*0!=5*4*3*2*1*1=120$.

Definirea recurentă a procesului de calcul al factorialului : $n! = n \bullet (n-1) \bullet (n-2) \bullet \ldots \bullet 2 \bullet 1$ se cunoaşte ca fiind: $n! = \begin{cases} 1 & \text{dacă } n = 0 \\ n \times (n-1)! & \text{dacă } n > 0 \end{cases}$. Funcţia asociată va fi definită: $fact(n) = \begin{cases} 1 & \text{dacă } n = 0 \\ n \times fact\ (n-1) & \text{dacă } n > 0 \end{cases}$.

Pentru exemplul de mai sus, calculul lui 5! înseamnă parcurgerea inversă a compunerii funcţiei **`fact`**.

fact(5)= 5• fact(4) = 5•4•fact(3)= 5•4•3•fact(2)=5•4•3•2•fact(1)=5•4•3•2•1•fact(0)=5•4•3•2•1•1=120.

5) Calculul celui mai mare divizor comun a două numere naturale nenule date, a şi b, presupune o prelucrare recursivă.

Funcţia prin care s-ar defini procesul calculului celui mai mare divizor comun a două numere naturale, a şi b, conform algoritmului lui Euclid, este:

$$cmmdc\ (a,b) = \begin{cases} a & \text{dacă } b = 0 \\ cmmdc\ (b, rest\ (a,b)) & \text{dacă } b \neq 0 \end{cases}.$$

Exprimarea *cmmdc(a, b) = cmmdc(b, rest(a, b))*, arată că se calculează cmmdc între împărţitorul şi restul operaţiei de împărţire precedente. Când restul unei împărţiri din succesiunea de împărţiri devine 0, atunci se va reţine împărţitorul acelei împărţiri, care, între timp, s-a mutat în variabila *a*.

De exemplu, pentru *cmmdc(15,85)*=5, descompunerile sunt:

cmmdc(15,85) = cmmdc(85,15) = cmmdc(15,10)cmmdc(10,5) = cmmdc(5,0) = 5.

6) Înmulţirea a două numere naturale, *a* şi *b*, folosind adunarea repetată:

$$produs\ (a,b) = \begin{cases} 0 & \text{dacă } b = 0 \\ a + produs\ (a, b-1) & \text{dacă } b \neq 0 \end{cases}$$

De exemplu, pentru $a = 5$ şi $b = 4$, descompunerile recursiei *produs (5,4)*=5*4=20 sunt:
produs(5,4) = 5+produs(5,3) = 5+5+produs(5,2) = 5+5+5+produs(5,1) = 5+5+5+5+produs(5,0) = 5+5+5+5+0 = 20.

7) Ridicarea bazei *a* la puterea *b*, *a* şi $b \in \mathbb{N}$, folosind înmulţirea repetată:

$$putere\ (a,b) = \begin{cases} 1 & \text{dacă } b = 0 \\ a \times putere\ (a, b-1) & \text{dacă } b \neq 0 \end{cases}$$

De exemplu, dacă $a = 2$ şi $b = 4$, atunci descompunerile pentru *putere(2,4)* = 2^4 = 16 sunt:
putere(2,4) = 2*putere(2,3) = 2*2*putere(2,2) = 2*2*2*putere(2,1) = 2*2*2*2*putere(2,0) = 16.

8) Suma cifrelor unui număr natural, *n*

$$suma\_cifre(n) = \begin{cases} 0 & \text{dacă } n = 0 \\ rest\ (n:10) + suma\_cifre\ ([n:10]) & \text{dacă } n \neq 0 \end{cases}$$

De exemplu, dacă $n = 426$, atunci descompunerile pentru *suma_cifre(426)* sunt:
suma_cifre(426) = 6 + suma_cifre(42) = 6 + 2 + suma_cifre(4) = 6 + 2 + 4 + suma_cifre(0) = 12.

9) Calculul termenului de ordin *n* din şirul lui Fibonacci: 1, 1, 2, 3, 5, 8, 13, ……..

$$fib(n) = \begin{cases} 1 & \text{dacă } 0 \leq n < 2 \\ fib(n-1) + fib\ (n-2) & \text{dacă } n \geq 2 \end{cases}$$

De exemplu, pentru *fib(4) = 5*, descompunerile sunt:
fib(4) = fib(3) + fib(2) = [fib(2) + fib(1)] + [fib(1) + fib(0)] =
= [[fib(1) + fib(0)] + 1] + [1 + 1] = [1 + 1 + 1] + [1 + 1] = 5.

Se observă că, pentru calculul lui fib(5) s-au făcut, de mai multe ori, apeluri pentru aceeaşi valoare a funcţiei, cum ar fi, de exemplu, apelul lui fib(2). Pentru o valoare mai mare a lui *n*, întregul proces se dezvoltă arborescent şi devine, astfel, un mare consumator de timp. În figura 7.2 sunt puse în evidenţă cele 9 apeluri ale procesului, necesare calculului lui fib(4).

Acesta este un exemplu în care abordarea recursivă a calculului nu este avantajoasă. Sunt puse în evidenţă, pe fond gri, apeluri care s-au mai făcut o dată, în alt moment al calculului.

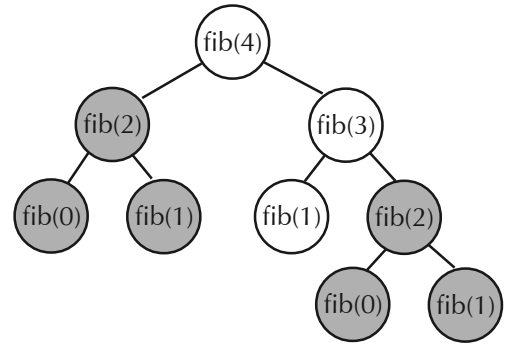


Figura 7.2

Recursivitatea este o tehnică de programare utilizabilă în acele limbaje de programare care acceptă implementarea recursivă a subprogramelor, adică apelul subprogramului în cadrul lui însuşi (autoapelul). Un algoritm este recursiv, dacă el este definit pornindu-se de la el însuşi.

În informatică, algoritmii recursivi sunt proiectaţi ca algoritmi care conţin apeluri la ei înşişi. Din acest motiv, recursivitatea va fi exprimată cu ajutorul **subprogramelor**, deoarece acestea pot conţine apeluri la ele însele, adică autoapeluri.

Autoapelul generează o nouă activare a aceluiaşi subprogram, adică execuţia instrucţiunilor lui începând cu prima, până la o nouă întâlnire a autoapelului. Se vede, deci, cât de importantă este existenţa unei condiţii de oprire a acestor reluări, altfel procesul degenerând în ciclare.

Odată îndeplinită condiţia de oprire a autoapelurilor, prelucrarea continuă cu operaţiile rămase de executat din cadrul subprogramului, pentru fiecare activare în parte, în ordinea inversă a activărilor.

Astfel, procesul de numărare până la 100, descris mai sus în formă iterativă, va apărea în manieră recursivă astfel:

Subprogram **numărare**(i: natural)	
Dacă i ≤ 100 *atunci*	condiţia de continuare
Scrie i	
numărare(i+1)	autoapelul; motorul i „mişcă" la i+1
Sfârşit Dacă	
Sfârşit **numărare**(*ieşire*)	întoarcere din subprogram la locul apelului
Modulul principal	
numărare(1)	Pornirea numărării
Stop.	

În forma de mai sus, funcţia de succesiune definită pe mulţimea numerelor naturale apare exprimată în autoapel prin numele subprogramului, *numărare(i+1)*, arătând că succesorul unui număr natural, dat, este tot un număr natural şi se obţine din numărul dat, *i*, prin adăugarea unei unităţi. Pornirea procesului de numărare se face cu valoarea 1.

Se poate observa că există o mare asemănare în scrierea algoritmului pentru un proces tratat iterativ şi scrierea algoritmului pentru un proces tratat recursiv, în amândouă formele punându-se în evidenţă cele trei elemente ale repetiţiei: *procesul*, *motorul* şi *condiţia de continuare*.

De altfel, pentru o acţiune repetitivă, se poate spune că orice algoritm iterativ se poate scrie în manieră recursivă şi invers, orice algoritm scris recursiv se poate transrforma în forma iterativă a acelei repetiţii.

Dacă figurăm modul de realizare a procesului recursiv pentru numărarea numai până la 4 pe baza algoritmului de numărare recursivă de mai sus, va apărea explicitarea prelucrării date în figura 7.3.

```
      .....
numarare(1) //apel din modulul principal.
.......
```

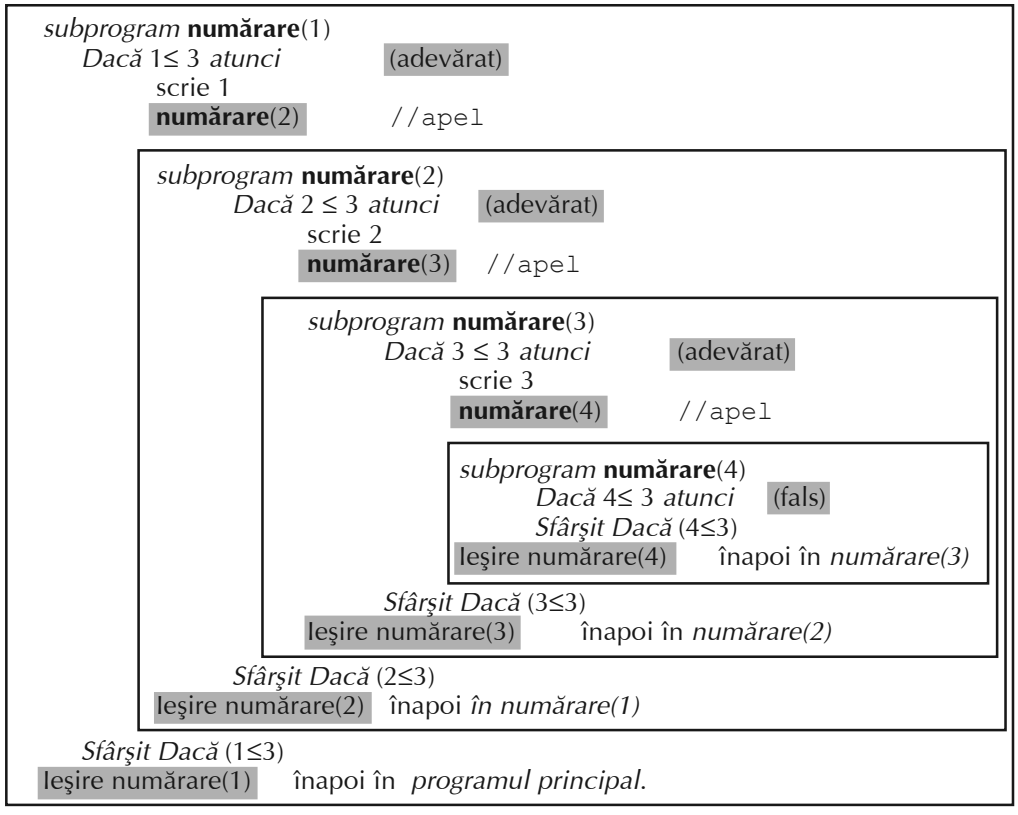


Figura 7.3

```
Stop. programul principal (apelant)
```

7.2. Execuţia programelor recursive

La execuţia oricărui apel al unui subprogram, sistemul de calcul se serveşte de zona rezervată în memoria de lucru numită **STACK** (zona stivei sistemului). În această zonă, sistemul aşază contextul apelului şi al accesului la subprogram: parametrii, adresa de retur, variabilele locale, iar la întoarcerea din subprogram, după execuţia lui, zona respectivă se „descarcă" de conţinut şi informaţiile stocate acolo nu mai sunt accesibile.

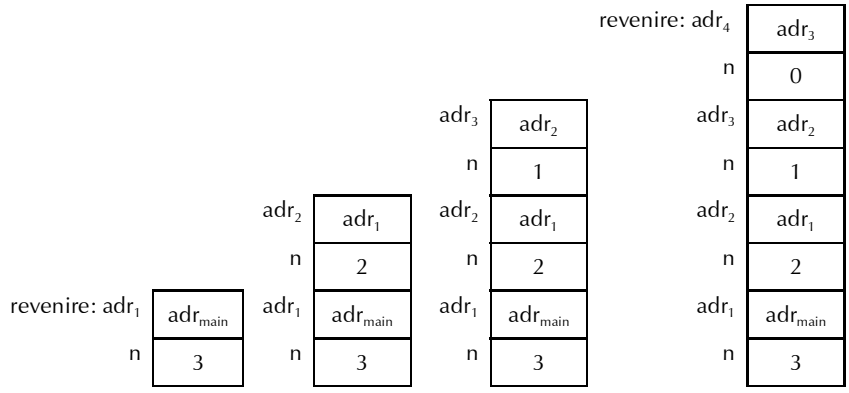


Figura 7.4

Aşadar, se deduce imediat că, în cazul execuţiei unui subprogram scris recursiv, informaţiile fiecărui apel se vor aşeza pe stiva sistemului, în ordinea autoapelurilor, pe principiul stivei (**LIFO**).

La revenirea din fiecare apel, stiva se descarcă progresiv până se goleşte, adică revine la starea iniţială.

Dacă nu este corectă condiţia de oprire sau dacă lipseşte, atunci lanţul autoapelurilor, teoretic, nu se sfârşeşte, iar procesul devine divergent. Practic, însă, programul se opreşte cu o eroare de tipul **STACK OVERFLOW** (depăşirea zonei alocate stivei sistemului).

Să analizăm ce se întâmplă la nivelul sistemului de calcul pentru rezolvarea recursivă a produsului 3!.

Funcţia este definită prin:

$$fact(n) = \begin{cases} 1 & \text{dacă} \quad n = 0 \\ n \times fact\ (n-1) & \text{dacă} \quad n > 0 \end{cases}$$

Apelul din funcţia **main** se va face prin **fact(3).**

În stiva sistemului se construieşte o zonă a contextului apelului ca în figura 7.4.

Astfel, la apelul din **main**, în stivă se creează două zone: pentru valoarea lui **n**=3 şi pentru adresa de revenire în **main**, zona $\mathbf{adr_1}$.

Deoarece **n** > 0 se execută un nou apel pentru valoarea **n-1**=2. În stiva sistemului se creează o nouă zonă pentru **n**=2 şi zona $\mathbf{adr_2}$ pentru memorarea adresei de revenire $\mathbf{adr_1}$.

În continuare **n** > 0, ceea ce va avea ca efect urcarea mai departe în stivă cu zona pentru **n**=1 şi zona $\mathbf{adr_3}$ pentru memorarea revenirii la $\mathbf{adr_2}$. Ultimul apel, pentru $n-1=0$ creează zona de la adresa $\mathbf{adr_4}$. Din acest moment, încep revenirile, prin descărcarea stivei, progresiv: de la $\mathbf{adr_4}$ se întoarce la $\mathbf{adr_3}$, apoi la $\mathbf{adr_2}$, $\mathbf{adr_1}$, $\mathbf{adr_{main}}$.

Mecanismul recursivităţii

Fie **P** un anume proces de prelucrare, care se realizează **dacă şi numai dacă** este îndeplinită o condiţie **C**, fixată. Procesul trebuie să conţină operaţii care să conducă la **convergenţa** lui, adică la atingerea stării ⌉**C**. Procesul este definit mai jos ca subprogram cu acelaşi nume, **P**.

Ce se va vedea în program	Ce se va realiza de către sistem
Subprogram P operaţii auxiliare1; **Dacă C atunci apelare P** **altfel** operaţii auxiliare 2; **SfârşitDacă** Operaţii auxiliare 3 ; **Întoarcere** la prelucrarea de după apelul anterior;	➢ Se încarcă contextul subprogramului **P** în stiva sistemului; ➢ Se încarcă noul context al subprogramului **P** în stiva sistemului ➢ Se descarcă din stiva sistemului ultimul context al subprogramului **P**, activându-se contextul anterior

Exemple de probleme rezolvate

1. Se consideră un şir de caractere primit de la tastatură. Se cere afişarea lui în ordine inversă.

Rezolvare. *Procesul* constă în citirea câte unui caracter în variabila *a*. Acest proces se repetă cât timp nu s-a întâlnit sfârşitul de linie de intrare, care reprezintă *condiţia de continuare* a citirii.

• În manieră iterativă, ar trebui ca fiecare caracter citit să fie reţinut explicit, deci să se organizeze un tablou de caractere, din care, apoi, după terminarea citirii, acestea să fie luate la afişare în ordine inversă, urmărind tabloul cu un indice de la sfârşit către început. De asemenea, ar trebui prevăzută o alocare statică pentru acest tablou.

• Rezolvată recursiv, problema implică organizarea unui subprogram, de tip funcţie procedurală, **`sirinv`**, care citeşte un caracter nou şi se va autoapela până la citirea caracterelor de sfârşit de linie. La fiecare apel, pe stiva sistemului se creează contextul apelului, informaţii printre care şi caracterul curent citit (adică valoarea curentă a variabilei locale *a*), astfel că, la ultima citire, în vârful stivei se află ultimul caracter citit. Revenirile din apeluri, începând cu ultimul apel, vor descărca succesiv stiva, aducând pe rând la scriere caracterul memorat pe nivelul respectiv în stivă, astfel că la afişare ele vor apărea în ordinea inversă faţă de citire. Cu alte cuvinte, sarcina de reţinere a caracterelor citite a preluat-o stiva sistemului, pe care programul a exploatat-o dublu: o dată ca instrument de gestionare a apelurilor şi a doua oară ca tablou pentru caracterele citite; astfel programul „scapă" de alocarea statică a unui tablou şi de sarcinile de manevrare a indicilor în tablou (atât la completarea acestuia cu caractere, cât şi la urmărirea lui în ordine inversă pentru afişare).

```
#include <iostream.h>
//inversarea unui sir //de caractere
void sirinv()
{char a;  cin>>a;
if (a!='\n') sirinv();
 cout<<a;
}
 void main()//modul principal
{
  sirinv();//apel principal
}
// pentru sirul citit "car" va afisa sirul "rac"
```

Fie şirul „car". Zona **`STACK:`**

\n	intoarceri →	↓
r	apel `sirinv` ↑	scrie **r** ↓
a	apel `sirinv` ↑	scrie **a** ↓
c	apel `sirinv` ↑	scrie **c** ↓
	apel `sirinv`↑	`main`
caracter citit	**acţiuni la apeluri şi la reveniri din apeluri**	

2. Dându-se un număr natural *n*, să se afişeze valoarea care se află în şirul lui Fibonacci pe acel loc *n*. Acest exemplu este dat pentru a discuta unele aspecte legate de avantajele şi dezavantajele aplicării recursivităţii.

Rezolvare. Deoarece legea creşterilor organice, adică şirul lui Fibonacci, este un proces matematic recurent, definit ca funcţie de *n* astfel:

$$fib(n)=\begin{cases}1 & \text{dacă} \quad 0 \le n < 2\\ fib(n-1)+fib\ (n-2) & \text{dacă} \quad n \ge 2\end{cases}$$

atunci este evidentă o abordare recursivă şi în programul care rezolvă problema propusă. În program s-a limitat valoarea lui **`n`** la 24, pentru că şirul are o creştere rapidă şi ajunge la valori care nu se mai pot reprezenta exact în mulţimea **`unsigned`**.

```
#include <iostream.h>
//program fibonacci
unsigned n;
unsigned fib(unsigned n)
{if (n>1)
  return (fib(n-1)+fib(n-2));
  else return 1;
}
void main()
{do
  {cout<<"Locul: ";
   cin>>n;
  } while (n>=24);
//pentru a fi reprezentat in
//tipul unsigned
cout<<"Numarul de pe locul "<<n;
cout<<" este "<<fib(n);
}
```

Etapele ocupării zonei **STACK** pentru apelurile calculului **fib(4)**

1	fib(0)+fib(1)=fib(0)+1	0	fib(0)=1 ↓				
2	fib(2) ↑	2	1+fib(0) ↓→	2	fib(2)=2 ↓		
3	fib(2)+fib(3) ↑	3	fib(2)+fib(3)	3	fib(3) ↓→	3	2+fib(3) ↑
4	fib(4) ↑	4	fib(4)	4	fib(4)	4	fib(4)
n	**apel**	**n**	**apel**	**n**	**apel**	**n**	**apel**

1	fib(0)+fib(1)=fib(0)+1	0	fib(0)=1 ↓				
2	fib(2) +fib(1) ↑	2	1+1 ↓	2		1	fib(1)=1 →
3	2+fib(3) ↑	3	2+fib(3) →	3	2+2+fib(1)↑	3	2+2+fib(1)
4	fib(4)	4	fib(4)	4	fib(4)	4	fib(4)
n	**apel**	**n**	**apel**	**n**	**apel**	**n**	**apel**

3	2+2+1 ↓		
4	fib(4)	4	fib(4)=5 ↓
n	**apel**	**n**	**apel**

sfârşit apeluri

Figura 7.5

Din studiul configuraţiei stivei pe parcursul autoapelurilor executate (fig. 7.5) se observă o risipă de memorie pe stivă şi o risipă de timp, deoarece anumite apeluri au mai fost generate o dată la alte etape ale descompunerii (zonele pe fond gri din tabele). Se poate spune că acest exemplu ar apărea ca un contraexemplu al primei probleme, inversarea şirului de caractere. Totuşi, ca exerciţiu didactic pentru acomodarea cu tehnica recursivităţii, se poate imagina o variantă recursivă economică.

Varianta recursivă – forma economică. Rezolvarea se bazează pe procesul de calcul al termenului de pe locul curent, *fn*←*a+aa*, unde *aa* este termenul anteanterior şi *a* este termenul anterior.

Astfel, pentru calculul următor, variabilele îşi schimbă conţinutul, *aa*←*a*, *a*←*fn*. Motorul repetiţiei îl asigură variabila *i*, care numără locul în şir al termenului curent calculat. La fiecare apel, după actualizarea variabilelor, pe stivă se vor aşeza, deci, valorile expresiilor *aa*, *a* şi *i*.

Pornirea procesului se face pentru *aa*←*0*, *a*←*1* (primele valori din şirul lui Fibonacci) şi *i*←*2* (deoarece pentru primul loc valoarea a fost deja considerată în *a*).

Programul foloseşte variabila globală *fn* pentru a memora noul termen calculat pe care îl va utiliza procesul de la etapa următoare.

O altă variantă ar fi fost ca variabila *fn* să fie parametru transmis prin referinţă, în loc de variabilă globală.

Pentru a simplifica scrierea cuvântului **unsigned** de către fiecare variabilă sau parametru declaraţi, s-a ales definirea unui tip de dată nestandard, al utilizatorului, numit `us`, dar care, de fapt, redenumeşte tipul **unsigned**.

```
#include <iostream.h>
//Fibonacci- calcul rapid
typedef unsigned us;
us fn,n;
void fib(us aa,us a,us i)
{
 if (i<=n)
  {fn=aa+a;//procesul
   fib(a,fn,i+1);}
}
void  main()
{ do
   {cout<<"Dati pragul n: ";
    cin>>n; }while (!(n>1));
fib(0,1,2);
cout<<"Pe locul "<<n;
cout<<" este "<<fn;
cout<<"\n Apasati o tasta";
cin.get();
}
```

7.3. Recursivitate versus iterație

În general, programele recursive pot fi scrise şi nerecursiv, adică sub formă iterativă. Programele recursive nu realizează, de obicei, economie de memorie şi nici nu sunt mai rapide în execuţie decât variantele lor iterative. În acest sens, s-a discutat exemplul 9 din paragraful 7.1, unde se observă cum procesul de calcul al unui termen al şirului lui Fibonacci, deşi mai clar în exprimarea recursivă, este un mare consumator de timp.

Recursivitatea permite, totuşi, o descriere mai compactă şi mai clară a funcţiilor, cum se vede în exemplele din paragraful 7.1.

Fiind dată o anume problemă, alegerea între a o rezolva iterativ sau recursiv depinde de cât timp necesită execuţia fiecărei variante şi de pretenţia de claritate a textului programului, varianta recursivă producând un text mult mai clar, mai uşor de urmărit.

În tabelul de mai jos prezentăm tipurile de probleme rezolvate în ambele variante.

PROCES	RECURSIV	ITERATIV

FACTORIAL	`//program factorial;`
	`#include<iostream.h>`
	`unsigned fact(unsigned n)`
	`{`
	`  if(n==0) return 1;`
	`  else return n*fact(n-1);`
	`}`
	`void main()`
	`{ unsigned n;`
	`cin>>n;`
	`cout<<"Fact="<<fact(n));`
	`}`
		`//program factorial;`
		`#include<iostream.h>`
		`unsigned fact(unsigned n)`
		`{unsigned i,calcul;`
		`  calcul=1;`
		`  for(i=2;i<=n;i++)`
		`  calcul=calcul * i;`
		`  return calcul;`
		`}`
		`void main()`
		`{ unsigned n;`
		`cin>>n;`
		`cout<<"Fact="<<fact(n));`
		`}`

CMMDC	`unsigned cmmdc(unsigned a,unsigned b)`
	`{`
	`if(b!=0) return cmmdc(b,a % b);`
	`  else return a;`
	`}`
		`unsigned cmmdc(unsigned a,unsigned b)`
		`{ unsigned rest=a;`
		`while (rest != 0)`
		`{rest=a % b;`
		`a=b;`
		`b=rest;`
		`}`
		`return a; }`

a*b prin adunare repetată	`unsigned produs(unsigned a,unsigned b)`
	`{`
	`if(b==0) return 0;`
	`  else return a+produs(a,b-1);`
	`}`
		`unsigned produs(unsigned a,unsigned b)`
		`{unsigned p;`
		`p=0;`
		`while(b!=0)`
		`{ p=p + a;`
		`b-- ;`
		`}`
		`return p; }`

a^b prin înmulţire repetată	`unsigned putere(unsigned a,unsigned b)`
	`{`
	`if(b==0) return 1;`
	`  else return a*putere(a,b-1);`
	`}`
		`unsigned putere(unsigned a,unsigned b)`
		`{unsigned p;`
		`p=1;`
		`while(b!=0)`
		`{ p=p * a;`
		`b--; }`
		`return p;`
		`}`

suma cifrelor unui număr	`unsigned suma_cifre(unsigned n)`
	`{`
	`if(n==0) return 0;`
	`  else return n % 10 +`
	`  suma_cifre (n / 10);`
	`}`
		`unsigned suma_cifre(unsigned n)`
		`{unsigned sum;`
		`sum=0;`
		`while(n!=0)`
		`{sum=sum + n % 10;`
		`n=n / 10;`
		`}`
		`return sum; }`

PROCES	RECURSIV	ITERATIV

FIBONACCI	//program fibonacci;
	#include<iostream.h>
	unsigned fib(unsigned n)
	{
	  if(n>1) return fib(n-1)+fib(n-2) ;
	  else return 1;
	}
	void main()
	{ unsigned n;
	  do {cout<<"Locul: ";
	  cin>>n;
	  }while(n>=24);
	  //pentru a fi reprezentat in word
	cout<<"Numarul de pe locul "<<n<<"
	  este "<<fib(n));
	}
		#include<iostream.h>
		unsigned fib(unsigned n)
		{unsigned i,prec1,prec2,curent;
		prec1=1; curent=1; i=3;
		while(i<=n)
		  { prec2=prec1;
		  prec1=curent;
		  curent=prec1+prec2;
		  i++;
		  }
		return curent;
		}
		void main()
		{ unsigned n;
		cin>>n;
		cout<<"Pe locul "<<n<<" se afla "
		  <<fib(n));}

Tipuri de recursii

Există două forme de recursivitate:

– recursivitatea directă, care se prezintă prin autoapelul subprogramului în interiorul lui;

– recursivitatea indirectă, în care un subprogram apelează un alt subprogram, iar acesta îl apelează pe primul. Procesul se încheie în momentul în care este îndeplinită condiţia de ieşire din subprogramul care a iniţiat procesul de apeluri reciproce.

7.4. Probleme rezolvate prin recursivitate directă

Programele vor fi verificate la laborator prin rulare pas cu pas (F7) şi urmărire în ferestrele **watch** şi **stack**.

1) Fără a realiza explicit adunarea *a+b*, să se calculeze şi afişeze ***suma a două numere*** naturale, *a* şi *b*.

Rezolvare. Vom folosi operatori predefiniţi prin care să poată fi proiectat recursiv acest proces. Ideea de bază este că adunarea lui *a* cu *b* înseamnă adunarea unui număr de *b* valori 1 la valoarea *a*. De exemplu, *a* = 5 şi *b* = 6. Atunci *a* + *b* = 5 + 1 + 1 + 1 + 1 + 1 + 1. Procesul trebuie să aplice funcţia de succesiune pe mulţimea numerelor naturale începând cu valoarea *a*. Motorul îl va asigura mişcarea variabilei *b* de la valoarea citită descrescător până la 0 (condiţia de oprire), moment în care nu mai este nici o unitate de adunat.

```
#include <iostream.h>
unsigned s(unsigned a,unsigned b)
{
 if (!b) return a;
   else return s(++a,--b);
}
```

```
void main()
{
 unsigned a,b;
 cin>>a>>b;
 cout<<"suma "<<s(a,b);
}
```

2) Dându-se un număr natural, *n*, să se afişeze numărul obţinut prin ***inversarea lui n*** în raport cu scrierea.

Rezolvare. Deoarece abordarea recursivă presupune organizarea unui subprogram, proiectarea procesului de calcul începe cu alegerea între a construi un subprogram de tip funcţie operand sau a construi un subprogram de tip funcţie procedurală. După cerinţa din enunţ, rezultă imediat că ar trebui construit un subprogram de tip funcţie operand. Astfel, utilizând scrierea polinomială a unui număr şi gruparea convenabilă a monoamelor, gruparea Horner, avem:

$$n = (\ldots(a_s\, 10 + a_{s-1})10+a_{s-2})10+\ldots)10+a_1)10+a_o,$$

unde a_i, (i = 0, 1, …, s) reprezintă cifra de ordin i a numărului.

Această formă pune în evidenţă recurenţa. Astfel, procesul repetitiv va avea ca operaţii:

$$\begin{cases} \text{inv} \leftarrow \text{inv} \bullet 10 + \text{rest}(n:10) \\ n \leftarrow [n:10], \text{ starea iniţială fiind } \text{inv} \leftarrow 0, \end{cases}$$

care, transformate în recursie, cu **inv** ca subprogram, se vor scrie imediat astfel:

$$\begin{cases} \textit{Dacă } n \neq 0 \textit{ atunci } inv \leftarrow rest(n:10) + inv([n:10]) \bullet 10 \\ \textit{altfel } inv \leftarrow 0. \\ \textit{Sfârşit Dacă} \end{cases}$$

Problema este însă „urcarea" în stiva sistemului pentru fiecare autoapel, deoarece ultima cifră înregistrată în stivă va fi ultimul rest obţinut, adică cifra cea mai semnificativă. Astfel, când se descarcă stiva, cifra se înmulţeşte cu 10 şi se adună următoarea găsită în coborâre, încât la final se obţine numărul iniţial nu cel inversat. În *varianta 1* a rezolvării se prezintă o soluţie care evită acest neajuns prin organizarea unei funcţii procedurale care are un parametru de intrare-ieşire, *inv*, în care se acumulează, ordin cu ordin, numărul inversat.

Dacă totuşi ţinem să realizăm procesul în modul recurenţei date de parantezarea polinomului, utilizând o funcţie operand, atunci trebuie calculat ordinul de mărime al numărului (variabila *zq*) şi utilizat pentru a obţine cifrele şi a le înregistra pe stivă în ordinea descrescătoare a rangului. Astfel, ultima cifră înregistrată va fi cifra unităţilor, iar la „descărcarea" stivei, ea va căpăta rangul cel mai mare datorită înmulţirilor progresive cu 10. În rezolvarea de mai jos, nu s-a explicitat partea de operaţii pentru obţinera ordinului de mărime, acest lucru fiind neesenţial acum şi poate fi foarte uşor de realizat (*Varianta 2*).

Varianta 1

```
// nrinversat cu functie procedurala
#include <iostream.h>
void invers(int n,int &inv)
{ if (n>0) {
      inv=inv*10 +n%10;
      invers(n/10,inv);
       }}
void main()
{int n,inv;
cin>>n;
inv=0;
invers(n,inv);
cout<<"Numarul inversat:"<<inv;
}
```

Varianta 2

```
// nrinversat cu functie operand
#include <iostream.h>
int inv(int n,int zq)
{if(n>0)
  return (n/zq +inv(n%zq,zq/10)*10);
else return 0;}
void  main()
{int n,zq;
zq=1000;//pentru 4 ordine;
//altfel trebuie determinat zq in
//functie de marimea numarului
cout<<"Dati  numarul cu 4 cifre";
cin>>n;
cout<<inv(n,zq);}
```

3) Fie o înşiruire de *n* numere naturale. Să se afişeze ***divizorii proprii pozitivi*** ai fiecăruia dintre cele *n* numere.

Rezolvare. După cum se ştie, divizorii proprii ai unui număr sunt numerele care îl divid, în afară de 1 şi numărul însuşi. De exemplu, numărul 12 are ca divizori proprii pozitivi pe 2, 3, 4 şi 6. Pentru organizarea calculului recursiv, trebuie ales întâi între a proiecta o funcţie operand, sau una procedurală. Cerinţa din enunţ conduce la o prelucrare individuală a fiecărui număr, în care să se obţină divizorii respectivi şi să se afişeze (*procesul*). Astfel rezultă un tip procedural. De asemenea, enunţul nu precizează nimic privind sursa numerelor şi nici dacă ele mai participă la alte prelucrări, aşa că nu vom organiza o structură de date pentru ele (cel mult, în cazul în care sunt multe numere de prelucrat, se poate organiza un fişier text din care aceste numere să fie preluate). Acum, odată definit procesul, se va determina *motorul* lui. Evident, acesta constă în obţinerea unei noi valori a numărului natural, *d*, care este încercată drept divizor (primul divizor încercat fiind valoarea 2). Condiţia de oprire apare în expresia logică $d \leq [n:2]$ şi se întâmplă când această expresie este falsă.

```
#include <iostream.h>
#include <conio.h>
unsigned i,a,n;
void diviz(unsigned n,unsigned d)
{if (d<=n/2)
  {if (!(n%d)) cout<< d<<' ';
   diviz(n,d+1);
  }
}
void main()
{clrscr();
do {
  cout<<"Nr de numere";cin>>n;
   }while (n<1);
for (i=1;i<=n;i++)
 {
  do {
   cout<<"Nr curent, >3 ";cin>>a;
  }while (a<=3);
 cout<<"Divizorii proprii sunt:";
 diviz(a,2);cout<<'\n';
 }
getche();
}
```

4) Se citeşte un număr natural, *n*. Să se determine dacă acesta este un număr ***prim***.

Rezolvare. Pentru ca un număr natural să fie prim, el nu trebuie să se dividă cu nici un alt număr natural mai mare ca 1 şi mai mic decât el. Această verificare (*procesul*) va produce un singur rezultat, *adevărat* sau *fals*, drept pentru care este evidentă organizarea unei funcţii operand, care va întoarce o valoare logică. Motorul procesului (desemnat prin *evoluţia variabilei d*) va fi precum cel din problema precedentă. Condiţia de oprire poate fi momentul când s-a atins valoarea *n* (numărul dat) sau mai repede, eliminând testări inutile, când *d > [sqrt(n)]*.

```
#include <iostream.h>
#include <math.h>
int prim (unsigned d,unsigned n)
{ if (d<=(int)sqrt(n))
    return (n%d && prim(d+1,n));
  else  return 1;
}
void  main()
{ unsigned n;
 do
    {cout<<"n=";cin>>n;
    } while (n<2);
if (prim(2,n))
  cout<<n<<" este numar prim";
else cout<<n<<" nu este prim";
}
```

5) Dându-se (prin coeficienţii lui) un ***polinom*** cu gradul maxim *10*, într-o singură nedeterminată să se calculeze valoarea polinomului într-un punct *x0* real, dat.

Rezolvare. Fiind creată obişnuinţa de a citi un polinom de la monomul de grad maxim către cel de grad zero, coeficienţii vor intra, în această ordine, într-un tablou de numere reale, *a*, inclusiv pentru monoamele care nu apar în polinom şi pentru care coeficienţii sunt 0 (se poate realiza prelucrarea şi fără organizarea coeficienţilor în tablou, şi anume, citindu-i pe rând). Deoarece rezultatul procesului este unul singur, valoarea polinomului în punctul *x0*, se organizează o funcţie operand care întoarce un rezultat real. Motorul procesului este asigurat de mişcarea valorii *n* care parcurge descrescător parantezele grupării Horner realizate pentru polinom.

```
#include <iostream.h>
float a[10],x0;
float eval(int n)
{ if (n<0) return 0;
   else
    return (a[n]+x0*eval(n-1));
}
void main()
{
 int i,n;
 cout<<"Gradul polinomului: ";
 cin>>n;
 cout<<"coeficientii,";
 cout<<"in ordine descrescatoare";
 cout<<"a gradului monomului:\n";
 for (i=0;i<=n;i++) {
      cout<<"a["<<n-i<<"]=";
      cin>>a[i];
      }
 cout<<"Valoarea de evaluare:";
 cin>>x0;
 cout<<"Valoarea polinomului:";
 cout<<eval(n);
}
```

6) Parcurgerea în adâncime a unui ***arbore binar***: RSD, SRD şi SDR

Rezolvare. Parcurgerea arborelui binar este cunoscută din capitolul 5. De exemplu, pentru arborele binar înregistrat prin **S** = (2, 4, 6, 0, 0, 0, 0) şi **D** = (3, 5, 0, 7, 0, 0, 0), vor rezulta: **RSD** = 1, 2, 4, 7, 5, 3, 6; **SRD** = 4, 7, 2, 5, 1, 6, 3 şi **SDR** = 7, 4, 5, 2, 6, 3, 1.

```
//program parcurg_arbore;
#include<iostream.h>
#include<conio.h>
int s[10],d[10];
/*s=vectorul descendentilor stanga
s[i]=0 daca nodul i nu are
                      descendent stanga
d= vectorul descendentilor dreapta
d[i]=0 daca nodul i nu are
                     descendent dreapta
rad-nodul radacina
n-numar de noduri*/
//parcurgere in preordine=RSD
void RSD(int k)
{     cout<<k<<' ';
      if(s[k]!=0)  RSD(s[k]);
      if(d[k]!=0)  RSD(d[k]);
}
//parcurgere in inordine = SRD
void SRD(int k)
{      if(s[k]!=0) SRD(s[k]);
      cout<<k<<' ';
      if(d[k]!=0) SRD(d[k]);
}
//parcurgere in postordine
void SDR(int k)
```

```
{     if(s[k]!=0) SDR(s[k]);
      if(d[k]!=0) SDR(d[k]);
      cout<<k<<' ';}
 void main()
{
 int n, k,i,rad;
 clrscr();
cout<<"Introduceti numarul de varfuri n: ";
cin>>n;
cout<<"Introduceti nodul radacina: ";
cin>>rad;
 for(i=1;i<=n;i++)
  {
  cout<<"\n Pentru nodul "<<i<<"
                          introduceti";
  cout<<" descendent stang: ";cin>>s[i];
  cout<<"\n descendent drept: ";
  cin>>d[i];
  }
  cout<<endl;
  cout<<" * Parcurgere in preordine
                          (RSD) **\n";
  RSD(rad);cout<<endl;
  cout<<" * Parcurgere in inordine (SRD)
                                **\n";
  SRD(rad);cout<<endl;
  cout<<" * Parcurgere in postordine
                          (SDR) **\n";
  SDR(rad);
  getch();
}
```

Să se completeze programul cu celelalte trei funcţii pentru parcurgerile RDS, DRS şi DSR.

7) Dându-se un tablou unidimensional, cu maximum 100 de elemente întregi, să se verifice dacă acesta conţine ***elemente pozitive***.

Rezolvare. Problema are scopul de a pune în discuţie situaţia în care condiţia de oprire a procesului ierarhizează situaţiile critice. Aici procesul recursiv constă în verificarea unui element din tablou, începând cu primul, dacă este pozitiv. În momentul găsirii unui element pozitiv, procesul se opreşte din autoapeluri, deoarece s-a putut calcula un rezultat pentru problema enunţată şi, astfel, se poate returna răspunsul. Se poate considera această situaţie ca fiind condiţia de oprire din autoapeluri. Dacă, însă, în tablou nu există niciun element pozitiv, atunci formularea condiţiei de oprire în maniera de mai sus nu mai este valabilă şi procesul conduce la „stack overflow". Rezultă că *principala grijă pentru a opri procesul* este ca să nu se depăşească tabloul, adică să nu se continue verificarea dacă s-au epuizat elementele tabloului. În consecinţă, găsirea unui element pozitiv devine o *condiţie secundară*, suplimentară, care serveşte ideii de eficienţă (nu se mai continuă comparările dacă s-a ajuns deja la un rezultat; *este ceea ce în maniera iterativă s-a numit ieşire forţată din repetiţie*). Prin urmare, **procesul recursiv are acum o condiţie de oprire compusă prin subordonare** din condiţia principală şi condiţia secundară.

```
//verificare existenta elem. >0
#include <iostream.h>
typedef int v[100];
unsigned n;
int poz(v a,unsigned i)
{
 if (i>n-1) return 0;
   else if (a[i]>=0) return 1;
     else return poz(a,i+1);
}
void tipar(v a,unsigned lung)
{
 unsigned k;
 for (k=0;k<lung;k++)
       cout<<a[k]<<' ';
}
```

```
void  main()
{unsigned i;
v a;
cout<<"Introduceti numarul de elemente
                                      ";
cin>>n;
cout<<"Introduceti numerele";
for (i=0;i<n;i++)
  {cout<<"a["<<i+1<<"]=";
   cin>>a[i];
  }
if (poz(a,0))
  cout<<"Exista nr. pozitiv";
else cout<<"Nu exista numar pozitiv";
tipar(a,n);
}
```

8) Pentru două tablouri unidimensionale, de lungimi egale, se defineşte ***produsul scalar*** ca fiind un număr rezultat din însumarea produselor elementelor de acelaşi indice din cele două tablouri. De exemplu, fie *a* şi *b* cele

două tablouri de câte *n* elemente fiecare. Atunci, produsul scalar este suma produselor de tip a_ib_i, adică: $a_1b_1 + a_2b_2 + \ldots + a_nb_n$. Dându-se cele două tablouri, să se afişeze produsul lor scalar[1].

Rezolare. Problema are scopul de a exemplifica un proces recursiv, **pr_sc**, în care participă două structuri de date. În esenţă, acţiunile se supun schemei generale de însumare.

```
// produsul scalar a doua tablouri
//de aceeasi lungime
#include <iostream.h>
#include <conio.h>
typedef int vector[20];
vector a,b;
int m,n;
void cit_vec(vector x, int lung,char
                                num[2])
{ int i;
  for (i=0;i<lung;i++)
   {cout<<num<<"["<<i+1<<"]=";
    cin>>x[i];
   }}
int pr_sc(int i)
{if (i<n)
  return (a[i]*b[i]+pr_sc(i+1));
 else return 0;
}
void  main()
{
clrscr();
 do {
  cout<<"lung. lui a=";
  cin>>n;
  cout<<"lung. lui b=";
  cin>>m;
 }while (m!=n);
cit_vec(a,n,"a");
cit_vec(b,m,"b");
cout<<"Produsul scalar este: ";
cout<<pr_sc(0);
getch();
}
```

9) Dându-se un tablou unidimensional de maximum 20 de elemente întregi, să se determine şi să se afişeze elementul de ***valoare maximă***.

Rezolvare. Din enunţ se deduce imediat că procesul, producând un singur rezultat – valoarea maximă din tablou –, conduce la organizarea unei funcţii operand, **m**. Problema are ca prim scop să pună în evidenţă modul mai greoi în care se proiectează recursiv procesul. Un al doilea scop care este urmărit, îl reprezintă modul în care, prin tehnica recursivităţii, problema găsirii maximului se descompune progresiv în subprobleme de dimensiune mai mică, adică $\mathbf{m(a_1,a_2,\ldots,a_n)=m(a_1,m(a_2\ldots a_n))}$. Dimensiunea curentă de explorat este măsurată între **i** şi **n** (**n-1** pentru indicii în limbajul de programare). Valoarea curentă a maximului este reţinută în variabila **f** transmisă funcţiei **m** ca parametru de intrare-ieşire şi iniţializată cu primul element din vector, înainte de apelul din **main**.

```
//maximul dintr-un vector
#include <iostream.h>
typedef int tabl[20];
tabl a; int n;
int m(int &f,int i)
{
 if (i>n-1) return f;
  else
   {
   if (f<a[i]) f=a[i];
   return m(f,i+1);
   }
}
void main()
{int f,i;
do {
  cout<<"Numar de elemente";
  cin>>n;
  }while (!(n>0 && n<=20));
for (i=0;i<n;i++)
 { cout<<"a["<<i<<"]=";
   cin>>a[i];
 }
f=a[0];//initializare maxim
cout<<"Max este:"<<m(f,1);
}
```

10) ***Ordonarea crescătoare*** a elementelor unui tablou unidimensional de maximum 20 de elemente întregi.

Rezolvare. Există foarte mulţi algoritmi de sortare[2]. Aici este exemplificat clasicul algoritm al ordonării prin interschimb, până ce tabloul este adus la forma ordonată. Procesul este preluat de către un subprogram de tip funcţie procedurală, `ordon`, care, în momentul realizării unui interschimb se autoapelează, reluând cercetarea

[1]Dacă am considera cele două tablouri ca fiind coordonatele a doi versori în spaţiul cu *n* dimensiuni şi produsul lor scalar ar fi zero, atunci cei doi versori sunt perpendiculari.

[2] În *Tratatul de programare,* Knuth precizează că aceştia sunt în număr de peste 1000.

tabloului de la început: **i** ia valoarea 0. Pentru o pereche de valori găsită în relaţie de ordine, **x[i]<=x[i+1],** se autoapelează pentru perechea următoare (**i** creşte cu o unitate). Se elimină astfel acţiunile de urmărire a variabilei logice care marca starea de ordine din tablou la un anume moment, acţiuni realizate în forma iterativă a algoritmului, deoarece se merge pe ideea că, odată făcut un interschimb, înseamnă că el poate genera dezordine în raport cu elementele vecine perechii tratate şi este firesc să se reia vectorul de la început.

```
//ordonarea crescatoare
//metoda interschimbului
#include <iostream.h>
typedef int vect[20];
vect a;
void ordon(vect x,unsigned lung,
                          unsigned i)
{
int aux;
if (i<lung-1)
   if (x[i]>x[i+1])
     { aux=x[i];
       x[i]=x[i+1];
       x[i+1]=aux;
       ordon(x,lung,0);
     }
    else ordon(x,lung,i+1);
}
void  main()
{unsigned lung,i;
do
 {cout<<"Numar de elemente=";
  cin>>lung;
 }while(!(lung>0 && lung<=20));
for (i=0;i<lung;i++)
    {cout<<"a["<<i+1<<"]=";
      cin>>a[i];
    }
ordon(a,lung,0);
for (i=0;i<lung;i++)
   cout<<a[i]<<' ';
}
```

11) Se citeşte o ***bază de numeraţie***, *b*, $2 \le b < 10$. De asemenea, se citeşte un număr natural, **n**, de maximum nouă cifre. Să se afişeze valoarea numărului **n** în baza de numeraţie **b**.

Rezolvare. Transformarea unui număr din baza 10 într-o bază de numeraţie revine la a împărţi succesiv numărul şi câturile intermediare la acea bază de numeraţie. Acest proces reflectă grupările celor **n** obiecte în grupe de câte **b**, iar grupele rezultate, la rândul lor, în grupe de câte **b** ş.a.m.d.

Pentru rezolvarea ***iterativă*** a acestei probleme este nevoie de un vector de numere naturale în care să se reţină resturile împărţirii progresive la **b** a numărului **n** şi apoi a câturilor rezultate. Apoi, pentru afişarea rezultatului, vectorul este parcurs de la sfârşit, adică de la ultimul rest obţinut la cel mai înalt nivel de grupare, către început. ***Recursiv***, procesul se simplifică foarte mult. Rolul vectorului îl va juca stiva sistemului. În stivă se vor aşeza câturile intermediare (valorile curente ale lui **n**), iar afişarea resturilor în ordinea inversă a obţinerii lor nu înseamnă decât ca, la descărcarea stivei, valoarea curentă a lui **n** să fie împărţită la **b** şi să se scrie restul obţinut.

```
#include<iostream.h>
#include<conio.h>
typedef unsigned us;
void conversie(us n,us b)
{
      if(n) {conversie(n/b,b);
                  cout<<n%b;}
}
void main()
{us n,b;
 clrscr();
 cout<<"Dati valoarea numarului ";
 cin>>n;
 do
      {cout<<"Dati valoarea bazei <10 ";
      cin>>b;}while(b<2||b>9);
 cout<<endl;
 conversie(n,b);
 getch();
}
```

12) Se consideră primele maximum 20 de numere naturale, începând cu 1, reprezentând etichetele unor obiecte. Interesează să se alcătuiască toate aşezările posibile ale acestor numere într-o înşiruire.

Rezolvare. Pentru un *n* fixat, maximum 20, cerinţa problemei se mai exprimă şi ca determinarea tuturor ***permutărilor*** celor **n** numere naturale. Astfel, dacă **n=3**, atunci există 6 aşezări ale numerelor 1, 2 şi 3 şi anume: 123, 132, 213, 231, 312, 321 (adică, pentru prima poziţie candidează trei numere – deci trei variante – apoi, pentru a doua, două numere, cele rămase – deci două variante, iar pentru ultima poziţie cadidează un număr din unul rămas – deci o variantă; sunt, deci, în total 3x2x1 variante, adică 3!).

Această rezolvare implică, însă, un număr mare de operaţii. Iniţial, tabloul **a**, variabilă globală, este iniţializat cu numerele de la 1 la *n*. Apoi, intervine subprogramul **perm**, care interschimbă elementul de pe poziţia *i* cu elementul de pe poziţia **k** (la început **k=n**=3 şi **i**=1, interschimbă pe 1 cu 3, deci se formează 3, 2, 1) şi prin autoapeluri realizează toate permutările pentru celelalte elemente (în exemplu – variantele 2, 3, 1 şi 3, 2, 1). Urmărind ieşirile progresive din autoapeluri se reface aşezarea iniţială (1,2,3, în cazul exemplului nostru) pentru a aşeza pe poziţia **n** o valoare de pe următoarea poziţie **i** din şir (în exemplu – numărul 2 –, generând variantele 3, 1, 2 şi 1, 3, 2). Pentru o nouă valoare a lui **i** (formată de instrucţiunea **for**), procesul se repetă (pentru exemplul luat mai rămân 2, 1, 3 şi 1, 2, 3).

Scopul principal al exemplului constă în a crea obişnuinţa de urmărire a două prelucrări repetitive imbricate (subordonate una alteia), una realizată iterativ şi cealaltă, subordonata, realizată recursiv.

```
#include <iostream.h>
typedef int tabl[20];
int n;tabl x;
void perm(int k)
{
 int i,j,a;
 if (k>0)
   for(i=0;i<k;i++)
      {a=x[i];x[i]=x[k];x[k]=a;
       perm(k-1);
       a=x[i];x[i]=x[k];x[k]=a;
      }
  else
   for(j=0;j<n;j++)
    cout<<x[j]<<' ';
   cout<<'\n';
}
void main()
{int i;
 cout<<"Lungime tablou:";
 cin>>n;
 for(i=0;i<n;i++)x[i]=i;
 perm(n-1);
}
```

13) Se citesc perechi de numere naturale de forma *(x,y)*, reprezentând faptul că persoana *x* simpatizează persoana *y*. Să se stabilească dacă există cel puţin un ***leader*** în grupul de persoane (persoana simpatizată de către toate celelalte persoane). Citirea datelor se opreşte la întâlnirea sfârşitului fişierului de intrare. Persoanele sunt numerotate începând cu 1, în mod secvenţial. Dacă este localizat un leader, numărul acestei persoane va fi afişat după mesajul corespunzător. Se cere o singură soluţie.

Rezolvare. Relaţiile de simpatie se pot modela ca un graf orientat în care un arc **(x,y)** reprezintă relaţia de simpatie a lui **x** către **y**.

Datele de intrare se vor organiza într-o structură bidimensională, matricea de adiacenţă **a**. Cum persoanele din grup au număr de ordine, dimensiunea utilă în matricea **a** este determinată de numărul cel mai mare citit pentru o persoană. Organizate în acest mod, datele unei coloane **j** din tablou vor arăta ce persoane simpatizează pe **j** (prin valorile de tip „adevărat") şi ce persoane nu-l simpatizează (prin valori de tip „fals"). Rezolvarea constă în determinarea existenţei unei coloane a tabloului care are toate elementele adevărate (variabila **ld** rămâne **1**), în afara elementului de pe diagonala principală (nu se consideră cazul în care o persoană se autosimpatizează). Dacă se găseşte o astfel de coloană, atunci se afişează numărul ei împreună cu mesajul, iar dacă nu, atunci se afişează „Nu există leader".

Programul utilizează o funcţie operand, **leader**, pentru determinarea coloanei cu proprietatea menţionată. Numărul coloanei găsite se reţine într-o variabilă globală, **ex**.

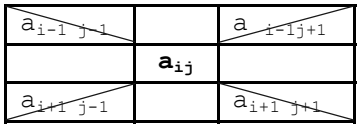
Pe lângă punerea în evidenţă a recursivităţii, scopul exemplului este şi de a prezenta o modalitate eficientă de organizare a datelor de intrare pentru probleme în care intervin relaţii de grup, dependenţe.

```
#include <iostream.h>
int a[10][10]={0};
unsigned n=0,ex;
int leader(unsigned  k)
{
 unsigned  ld, j;
 if (k>n-1) return 0;
   else
   {ld=1;
    for(j=0;j<n;j++)
      if(k!=j) ld=ld && a[j][k];
    if (ld) { ex=k+1; return 1;}
       else return(leader(k+1));
    }
}
void main()
{unsigned i,j;
 cout<<"Dati perechile (la sfarsit
                          CTRL^Z)\n";
 while (cin.good())
```

```
{cin>>i>>j;
 a[i-1][j-1]=1;
 if (i>=j && i>n) n=i;
   else if (j>i && j>n) n=j;
}
```

```
 cin.clear();
 if (leader(0))
   cout<<"exista un leader "<<ex<<'\n';
      else cout<<"Nu exista leader\n";
}
```

14) Pe o tablă ca de şah de dimensiune *n* (maximum 10) se află un nebun şi nişte ***obstacole*** codificate cu numărul 1 (în rest 0). Coordonatele obstacolelor de pe tablă se citesc de la tastatură sub formă de perechi de numere *(i,j),* $1 \le i \le n$, $1 \le j \le n$, pe principiul coordonatelor unui tablou bidimensional, până la introducerea sfârşitului de fişier de intrare (**CTRL si Z**). Coordonatele nebunului se citesc ulterior, în acelaşi mod. Să se afişeze, utilizând un proces recursiv de căutare, dacă există obstacole pe traseele atacate de nebun şi dacă da, câte astfel de obstacole există pe fiecare traseu.



Rezolvare. Tabla de şah este codificată sub forma unui tablou bidimensional pătratic, astfel încât localizările se fac pe principiul coordonatelor într-o astfel de structură de date şi nu al codificării practicate în şah. Se păstrează regula de deplasare a unui nebun pe tabla de şah, adică pe traseul celor patru segmente diagonale ce pornesc din locul lui de amplasament. Pentru a proiecta, indiferent de traseu, un proces general de căutare în scopul *autodefinirii lui*, s-a organizat tabloul unidimensional **depl** cu creşterile algebrice ale indicilor de linie şi de coloană pentru fiecare sens de deplasare (traseu). De exemplu, perechea ($\mathbf{depl}_1$,$\mathbf{depl}_5$) reflectă sensul **N-V**; astfel, dacă nebunul se află pe locul de coordonate **(i,j)**, se va obţine **(i-1,j-1)** din **(i+depl$_1$,j+depl$_5$)**.

Subprogramul de tip funcţie **obst** va urmări recursiv un traseu precizat prin parametrii **lin** şi **col** (motoarele procesului), numărând toate obstacolele întâlnite pe acel traseu (se adună fiecare **a[lin][col]=1** găsit pe acel traseu). Selecţia traseului atacat o face parametrul **i**, care ia, pe rând, cele **4** valori ale traseelor. Condiţia de oprire este asigurată de testul ieşirii din tabla de şah pentru un anume traseu.

Exemplul este dat şi în scopul obişnuirii cu o modalitate de a codifica convenabil deplasările într-o structură de date de tip tablou bidimensional, astfel încât să se poată da o formă generală prelucrărilor de tip parcurgere. În program, indicii încep de la 0.

```
#include <iostream.h>
const int depl[8]={-1,-1,1,1,-1,1,-1,1};
int a[10][10]={0},n;
int obst(int lin,int col,int i)
{
if(lin>n-1||lin<0||col>n-1||col<0)
                return 0;
   else
return(a[lin][col]+obst(lin+depl[i],col+
depl[i+4],i));
}
void main()
{int i,j,k;
  cout<<"Dimensiune tabla:";cin>>n;
  cout<<"Dati coordonatele
          obstacolelor,CTRL^Z la sf.";
  while (cin.good())
  {
     do  cin>>i>>j;
     while ((i>n ||i<1)&&(j<1||j>n));
  a[i-1][j-1]=1;}
  cin.clear();
  cout<<"dati coordonatele nebunului: ";
  do  cin>>i>>j;
  while ((i>n ||i<1)&&(j<1||j>n));
  i--;j--;
  a[i][j]=0;//se considera pozitie
neocupata de obstacol
  cout<<"pe directia N-V sunt:
                "<<obst(i,j,0)<<'\n';
  cout<<"pe directia N-E sunt:
                "<<obst(i,j,1)<<'\n';
  cout<<"pe directia S-V sunt:
                "<<obst(i,j,2)<<'\n';
  cout<<"pe directia S-E sunt:
              "<<obst(i,j,3)<<'\n';  }
```

15) Se consideră o ***fotografie alb-negru*** codificată într-un tablou bidimensional pătratic cu elemente 0 şi 1. Codificarea constă în faptul că un punct negru din fotografie se pune în corespondenţă cu valoarea 1 în tablou, iar un punct alb, cu valoarea 0. Este evident că un grup compact de puncte negre vor da imaginea unui obiect din fotografie. Acestui lucru îi corespunde în tablou faptul că, dacă o valoare 1 are vecini de valoare 1 (pe linie, coloană şi diagonală), atunci el aparţine acelui obiect. Se cere ca, utilizând această codificare a fotografiei, să se numere câte obiecte distincte conţine aceasta.

$a_{i-1\ j-1}$	$a_{\ i-1\ j+1}$	$a_{\ i-1j+1}$
$a_{\ i+0\ j-1}$	$\mathbf{a_{ij}}$	$a_{\ i+0\ j+1}$
$a_{i+1\ j-1}$	$a_{\ i+1\ j+0}$	$a_{i+1\ j+1}$

Rezolvare. Problema este foarte întâlnită în codificarea imaginilor, iar algoritmul pentru determinarea unui obiect este clasic – ***algoritmul de umplere*** (**FILL**). Obişnuiţi cu problema precedentă, a urmăririi obstacolelor din calea nebunului pe o tablă de şah, acum este uşor să ne imaginăm această urmărire pe toate cele **8** sensuri de plecare dintr-o valoare **1** din tablou. În momentul în care, pe un anume sens de parcurs, nu se ajunge într-un alt **1**, atunci se consideră atinsă o margine de obiect şi trebuie reluată parcurgerea pe alt sens, tot din aproape în aproape. Diferenţa este că, pentru a nu lua în considerare locurile prin care s-a trecut deja, se marchează drumul la fiecare înaintare cu o valoare, **nrob**, care nu este în tablou (în exemplu **2**) (procesul de umplere). Dacă pe toate sensurile se ating marginiale obiectului, înseamnă că s-a inspectat tot obiectul şi se caută un altul, adică o valoare **1** următoare existentă în tablou. Evident că, trecând la alt obiect, se va incrementa un contor de numărare a obiectelor distincte găsite, **nrob**.

Se va regăsi şi aici ideea organizării valorilor relative ale deplasării, dar acum vor fi **8** perechi de astfel de valori.

Astfel, apare imediat necesară organizarea a două tablouri, unul pentru creşterile relative ale liniei şi celălalt pentru creşterile relative ale coloanei, **dl** şi **dc**. În programul de mai jos, fiecare obiect nou se „umple" cu **nrob+1** pentru a pune în evidenţă, la afişarea tabloului, obiectele găsite.

```
#include <iostream.h>
#include <stdlib.h>
//program imagini
typedef int depl[8];
const depl dl={-1,-1,-1,0,1,1,1,0};
      depl dc={-1,0,1,1,1,0,-1,-1};
int poza[20][20],n,i,j,nrob;
void fill(int lin,int col)
{int i;
if (lin<n && lin>=0 && col>=0 && col<n)
  if (poza[lin][col]==1)
    {poza[lin][col]=nrob+1;
     for(i=0;i<8;i++)
       fill(lin+dl[i],col+dc[i]);
    }
}
void  main()
{randomize();
//datele fotografiei pot fi citite din
//exterior;aici sunt generate aleator
cout<<"Dimensiune fotografie:";
cin>>n;
for (i=0;i<n;i++)
    for(j=0;j<n;j++)
         poza[i][j]=random(2);
cout<<"Afisare poza \n";
for(i=0;i<n;i++)
 {for(j=0;j<n;j++)
       cout<<poza[i][j]<<' ';
  cout<<'\n';
}
nrob=0;
for (i=0;i<n;i++)
 for(j=0;j<n;j++)
  if (poza[i][j]==1)
    {nrob++;
     fill(i,j);
    }
cout<<"Afisarea dupa marcare\n";
for(i=0;i<n;i++)
 {for(j=0;j<n;j++)
       cout<<poza[i][j]<<' ';
  cout<<'\n';
 }
cout<<"Numar obiecte: "<<nrob;
}
```

7.5. Recursie ierarhizată

exemplu

Se consideră un sistem de n ecuaţii algebrice liniare, având n necunoscute notate $x_1, x_2, \ldots, x_n$. Sistemul are aspect triunghiular:

$$\begin{cases} a_{11}x_1+a_{12}x_2+\ldots+a_{1n}x_n = b_1 \\ \quad a_{22}x_2+\ldots+a_{2n}x_n = b_2 \\ \quad \ldots\ldots\ldots\ldots\ldots\ldots \\ \quad a_{nn}x_n = b_n \end{cases},$$

unde a_{ij} sunt coeficienţii necunoscutelor, iar b_i sunt termenii liberi, i şi j având valori de la 1 la n.
Se cere afişarea rădăcinilor sistemului.

Rezolvare. Din forma particulară a sistemului se poate vedea că, pornind de la ultima ecuaţie şi aplicând metoda substituţiei, soluţiile sistemului se obţin foarte uşor de la x_n la x_1:

$$x_i = \frac{b_i - \sum_{k=i+1}^{n} a_{ik} \times x_k}{a_{ii}}.$$

Prin urmare, este nevoie de două funcţii care sunt organizate ierarhic:

– funcţia operand pentru însumare, **suma**;

– funcţia procedurală care calculează o rădăcină, x_i, **calculx**. Funcţia **calculx** apelează funcţia **suma**. Însumarea se execută recursiv.

```
#include <iostream.h>
float a[10][10],b[10],x[10];
int n;
float suma(int j,int i)
{ if (j<=i) return 0;
   else return (a[i][j]*x[j]+suma(j-
1,i));
}
void calculx(int c)
{
  if (c>-1) {
      x[c]=(b[c]-suma(n,c))/a[c][c];
      calculx(c-1);
      }
}
void main()
{int i,j;
 cout<<"Dati dimensiunea sistemului:";
 cin>>n;
 cout<<"coeficientii: ";
 for(i=0;i<n;i++)
  for(j=i;j<n;j++)
  {
   cout<<"a["<<i+1<<','<<j+1<<"]=";
   cin>>a[i][j];}
 cout<<"Termenul liber: ";
 for(i=0;i<n;i++) {
     cout<<"b["<<i+1<<"]=";
     cin>>b[i];}
 cout<<"Vectorul radacinilor: ";
 calculx(n-1);
 for(i=0;i<n;i++)
   cout<<"x["<<i+1<<"]="<<x[i]<<' ';
}
```

7.6. Exerciții și probleme propuse

1. Fie funcţia *f* definită astfel:

```
int f(int x)
{ if(x<=0) return 3;
     return f(x-1)*2;}
```

Stabiliţi ce valoare întoarce pentru apelul f(4). Daţi o exprimare iterativă funcţiei.

2. Alegeţi expresia cu care se vor înlocui punctele de suspensie din definiţia funcţiei **cif,** astfel încât, după apel, **cif(n)** să returneze numărul de cifre ale unui număr natural nenul:

```
unsigned cif(long n)
{if(n==0) return 0;
return...;
}
```

a) `cif(n/10)+1;` **b)** `n/10+cif(n%10);` **c)** `n%10+cif(n/10);` **d)** `1+cif(n%10).`

3. Să se scrie subprogramul recursiv pentru calculul valorii funcţiei $f : \mathbb{R} \to \mathbb{R}$, definită mai jos:

$$f(x) = \begin{cases} x - 1, dacă & x \geq 12 \\ f(f(x+2)) & pentru \quad x < 12 \end{cases}$$

şi să se calculeze:

a) valoarea funcţiei pentru $x = 9$;

b) numărul de apeluri ale funcţiei pentru $x = 9$.

4. Următorul subprogram calculează valoarea funcţiei $f : \mathbb{Z} \to \mathbb{Z}$. Să se prezinte etapele calculului pentru $x = 5$ şi numărul de apeluri necesare.

```
int f(int x)
{if(x>=15) return x-2;
          return f(f(x+4))
          };
```

5. Se consideră funcţia *s*: ℕ* x ℕ*→ℕ*, definită prin:

$$s(x,y)=\begin{cases} y, \text{ dacă } x=1 \\ x, \text{ dacă } y=1 \\ m\cdot s(x-2,y)+n\cdot s(x,y-2), \text{ în rest} \end{cases}, \text{ unde } m \text{ şi } n\in\mathbb{Z}.$$

a) Scrieţi un subprogram recursiv care să determine valoarea acestei funcţii pentru *x,y,m* şi *n* comunicaţi din exteriorul subprogramului;
b) Care este valoarea funcţiei şi câte apeluri se fac pentru *x* = 3, *y* = 5, *m* = 1 şi *n* = 1?
i) 12 cu 5 apeluri; ii) 11 cu 4 apeluri; iii) 8 cu 3 apeluri; iv) 11 cu 5 apeluri; v) 9 cu 5 apeluri.

6. Fie următorul subprogram:

```
int x (int n)
{  if (n<3) return 1;
      else
    return (x(n-1)+x(n-3)+1);
           }
```

Să se calculeze numărul de apeluri pentru `x(x(5))`.

7. Fie funcţia *m* definită astfel:

```
int m(int i)
{ if(i==0) return 0;
     return m(i-1)+i;
}
```

Stabiliţi ce valoare întoarce pentru apelurile `m(8)` şi `m(50)`.
Daţi o exprimare iterativă funcţiei.

8. Fie funcţia *m* definită astfel:

```
int m(int i)
{ if(i==0) return 0;
   if()i%2==0) return m(i-1)+i;
      return m(i-1)-1;
}
```

Stabiliţi ce valoare întoarce pentru apelurile `m(8)` şi `m(520)`.
Daţi o exprimare iterativă funcţiei.

9. Se consideră algoritmul scris în pseudocod:

```
citeste n, a (numere naturale)
este ← 0
executa
      daca n=a atunci este←1
      n←[n/10]
cât timp (n=0  ∨  este=1)
scrie este
```

a) Să se determine ce afişează algoritmul pentru `n=1462` şi `a=14`;
b) Să se transforme într-o funcţie recursivă apelat din `main( )`, după citirea datelor şi care returnează valoarea din variabila `este`.

10. Se consideră funcţia recursivă de mai jos. Ce se va afişa la apelul `scrie(21)` dintre variantele date?

```
void scrie(int x)
{
if(x>1) scrie(x/2);
   cout<<x%2;
}
```

a) 10101 **b)** 00000 **c)** 11111 **d)** 01010
Daţi o exprimare iterativă funcţiei.

11. Se consideră funcţia lui Ackermann, funcţie utilizată în testarea vitezei de lucru a unui calculator. Ea are definiţia:

$$ac(m,n)=\begin{cases} n+1, \text{ dacă } m=0 \\ ac(m-1,1), \text{ dacă } n=0 \\ ac(m-1,ac(m,n-1)) \text{ în rest} \end{cases}$$

Scrieţi pe caiet etapele de calcul pentru ac(3,0) şi să se determine numărul de apeluri efectuate.

12. Să se determine ce se afişează în urma rulării programului de mai jos:

```
#include<iostream.h>
#include<conio.h>
void scrie(int n)
{ cout<<"n="<<n<<endl;
 if(n)
 {cout<<"Autoapel"<<endl;
  scrie(n-1);
  cout<<"Revenire"<<endl;
 }
}
void main()
{ int n;
  clrscr();
  cout<<"nr= ";
  cin>>n;
  scrie(n);
  getch();
}
```

13. Se consideră funcţia *u* definită astfel:

```
void u(int a)
{
 if(a/8>0)
  u(a/8);
  cout<<a%8;
}
```

Stabiliţi:
a) Ce se afişează pentru u(39)?
b) Numărul de autoapeluri realizate.
a) 74 şi 2; b) 74 şi 1; c) 47 şi 2; d) 47 şi 1; e) 47 şi 3.

14. Se consideră funcţia *u* definită astfel:

```
void u(int k)
{int d,p;
 if(k>1)
  {d=s+1;p=0;
    while(k%d==0)
   {k=k/d;
       p++;
        }
  cout<<d<<" "<<p<<" ";
  s=d;
  u(k);
  }
else cout<<k<<endl;
}
```

unde *s* este variabilă globală. Dacă *s* = 1 şi *k* = 18, care dintre variantele de mai jos este rezultatul afişărilor?
a) 1 2 9 3 2 1; **b)** 2 1 3 2 1; **c)** 18 2 1 3 2 1; **d)** 2 1 9 3 1; **e)** 2 1 3 1 1.

15. Formulaţi un enunţ ce se poate aplica prelucrării realizate de funcţia *u* din exerciţiul precedent.

16. Se consideră programul de mai jos. Să se analizeze funcţionarea lui şi să se precizeze ce afişează pentru *n* = 3.

```
#include <iostream.h>
unsigned short n;
void p(unsigned short n)
{if(!n) cout<<'B';
  else
  {
   p(n-1);
   cout<<'C';
   p(n-1);
   cout<<'G';
  }
}
void main()
{
 cin>>n;
 p(n);
 cout<<'\n';
}
```

17. Se consideră funcţia *c* definită astfel:

```
void c(unsigned b, unsigned a)
{if(a<=b/2) {if(!(b%a)) cout<<a<<" "; c(b,a+1);}}
```

Ce se va afişa pentru apelul c(15,2) şi care este numărul de apeluri ?
a) 3 5 15 şi 17 apeluri; b) 2 4 6 7 şi 7 apeluri; c) 3 5 şi 8 apeluri; d) 1 3 5 şi 8 apeluri; e) 2 4 6 7 şi 6 ap.
Care este condiţia de oprire din autoapeluri?
a) $a > b$; **b)** $a > [b/2]$; **c)** $a = b$; **d)** $a <= [b/2]$; **e)** $a! = b$.

18. Se defineşte şirul recurent a_n astfel: $a_{n+1} = \begin{cases} m, \text{ dacă } n = 0 \\ \dfrac{a_n}{n}, \text{ dacă } n = par \\ 4a_n - 1 \text{ în rest} \end{cases}$

Scrieţi un subprogram care calculează a_n, pentru $n > 0$ dat, şi $m \in \mathbb{R}$ citit.

Probleme propuse

19) Să se realizeze varianta economică de calcul al unui termen din şirul lui Fibonacci prin proiectarea unui subprogram recursiv de tip funcţie operand.
20) Să se modifice programul pentru evaluarea unui *polinom de grad n* în punctul *x0,* astfel încât utilizatorul să poată introduce coeficienţii în ordinea crescătoare a gradelor monoamelor.
21) Să se construiască o formă echivalentă a programului de verificare de număr *prim,* astfel încât să apară o condiţie subordonată de oprire din recursie (se va adăuga situaţia de ieşire din recursie dacă s-a descoperit că numărul nu este prim până ce *d* atinge valoarea lui finală).
22) Dându-se un două numere naturale, *n* şi *d*, să se organizeze un proces recursiv de determinare dacă *d* este divizor al lui *n* şi de câte ori (ordinul de multiplicitate).
23) Construiţi un program care să calculeze numărul de aranjamente de n elemente luate câte p, pentru n şi p numere naturale citite, utilizînd funcţia recursivă *fact* prezentată în capitol.
24) Construiţi un program care să calculeze numărul de combinări de n elemente luate câte p, pentru n şi p numere naturale citite, utilizînd funcţia recursivă *fact* prezentată în capitol.
25) Dându-se *n* numere naturale, să se proiecteze un proces recursiv, *p*, pentru calculul celui mai mare divizor comun al lor. *Indicaţie*: se va folosi funcţia *cmmdc(a,b)*. Apelul lui *p(a, n)* din main se face după ce s-au citit *n* şi primul număr în *a*. În *p* se citeşte un număr următor în *b* şi se aplică *p(cmmdc(a,b),n-1),* dacă *n>0*.
26) Se citeşte un cuvânt de maximum 16 litere. Să se definească o funcţie care verifică dacă acest cuvânt este palindrom. De exemplu, cuvântul *cojoc* este palindrom.
27) Dându-se un tablou unidimensional de maximum 100 de elemente întregi, să se organizeze un proces recursiv de verificare, a elementelor sale, dacă sunt aşezate în ordine crescătoare.
28) Din fişierul *numere.in* se citesc numărul şi elementele unui tablou unidimensional de maximum 100 de întregi. Să se organizeze un proces recursiv de căutare a unei valori, *v*, citite de la tastatură.
29) Dându-se un tablou unidimensional de maximum 100 de elemente reale, să se organizeze un proces recursiv de mutare a elementelor sale nule în poziţiile finale.
30) Din fişierul *numere.in* se citesc numărul şi elementele unui tablou unidimensional de maximum 100 de întregi. Să se organizeze un proces recursiv de verificare a aşezării elementelor în ordinea: pozitiv, negativ, pozitiv, negativ etc.
31) Să se scrie o funcţie recursivă care primeşte două şiruri de caractere şi întoarce valoarea logică corespunzătoare situaţiei, dacă şirurile sunt egale sau nu.
32) Să se scrie o funcţie recursivă care să primească adresa de început a unei liste simplu-înlănţuite şi să afişeze elementele listei.
33) Transformaţi funcţia cerută la problema 11 pentru a returna numărul de elemente din listă.
34) Să se definească două funcţii recursive pentru citire, respectiv, afişarea elementelor unei matrice de dimensiuni m × n, citite.
35) Se consideră o matrice pătratică. Să se definească o funcţie recursivă care realizează suma elementelor de pe diagonala principală.
36) Se consideră o matrice pătratică. Să se definească o funcţie recursivă care realizează matricea transpusă a celei date.
37) Se citeşte un şir de caractere, maximum 100. Să se definească două funcţii recursive pentru determinarea numărului de cifre din text şi, respectiv, pentru verificarea existenţei în text a tutuor vocalelor.
38) *Se consideră un număr natural nenul. Să se determine, utilizând un proces recursiv, acel număr m, cu m<n, care are numărul maxim de divizori proprii.
39) *Dându-se două numere naturale, a şi b, să se determine printr-un proces recursiv, dacă ele sunt termeni consecutivi ai şirului lui Fibonacci.
40) *Se consideră două şiruri de caractere. Să se verifice, printr-un proces recursiv, dacă unul din şiruri este anagrama celuilalt.
41) *Se consideră un cuvânt. Să se construiască un proces recursiv care transformă cuvântul în „limba păsărească". De exemplu, *acasa* devine *apacapasapa*. Transformarea constă în dublarea fiecărei vocale şi intercalarea literei *p*.
42) *Se citeşte un număr în scriere romană. Să se transforme recursiv într-un număr în scriere cu cifre arabe.
43) *Se citesc o frază şi un cuvânt separat. Să se determine dacă se regăseşte cuvântul în frază (în ordinea literelor lui dar nu obligatoriu consecutiv).

PARTEA a III-a

Elaborarea algoritmilor de rezolvare a problemelor

Capitolul 8 Metode de rezolvare a unor probleme

În acest capitol veţi învăţa despre:

- Algoritmi de rezolvare specific unor tipuri de probleme – „*Divide et impera*" şi „*Backtracking*"
- Divizarea unei probleme în subprobleme cu grade de complexitate scăzute progresiv
- Rezolvarea problemelor elementare la care s-a ajuns şi combinarea soluţiilor
- Construirea tuturor soluţiilor valide prin încercări şi reveniri la valorile anterioare

Există anumite probleme care se pot rezolva prin metode standardizate, la fel ca în cazul unor reţete de prelucrare. Acestea generează algoritmi specifici pentru a fi programate pe calculator. De exemplu, la rezolvarea sistemelor de ecuaţii, există metode specifice de rezolvare – metoda eliminării, metoda substituţiei etc.

Punerea în algoritm a unei astfel de metode necesită aplicarea unor tehnici de programare, cum ar fi: structurarea datelor, organizarea de module (subprograme) – fiecare cu sarcină precisă, corespunzătoare unei faze a rezolvării –, aplicarea recursivităţii, tehnici prin care se poate face o proiectare mai clară a programului sau o execuţie a lui mai rapidă. Lista acestor metode este destul de întinsă, din ea fiind cuprinse în programa şcolară pentru clasa a XI-a doar două, şi anume, metoda „***Divide et impera***" şi metoda„***Backtracking***".

8.1. Metoda „DIVIDE ET IMPERA"

8.1.1. Exemple

1. Într-un plic a fost ascuns un număr natural, nu mai mare decât 100. Doi copii se întrec să obţină un premiu pus în joc, care se acordă celui care ghiceşte numărul ascuns. Ca să afle numărul, copiii pot pune întrebări despre număr la care se poate răspunde numai cu *Da* sau *Nu*. Cum vor proceda copiii?

Rezolvare.

• La prima vedere, s-ar părea că se pot întrece prin întrebări de tipul „este numărul…?", fiecare mizând pe noroc, numerele fiind luate la întâmplare.

• Dacă stau şi se mai gândesc puţin, ar putea să meargă pe acelaşi model de întrebare, dar parcurgând din 1 în 1 valorile până la ghicire. În acest mod unul dintre ei va câştiga, tot la noroc, dar fără a avea ocazia de a mai pierde timpul cu întrebări pe care le-au mai pus anterior, însă le-au uitat sau nu le-au dat atenţie. Nu vor scăpa totuşi de plictiseală în situaţia în care numărul ascuns este chiar 100.

• La o gândire mai profundă, măcar unul dintre ei va observa că se poate servi de faptul că numerele pe care le are de încercat *sunt ordonate crescător*, fiind o submulţime a numerelor naturale. În acel moment, va descoperi că poate pune întrebările în genul „este numărul ascuns egal cu 50?" Dacă se răspunde NU, poate întreba „este numărul ascuns >50?", iar dacă se răspunde „DA", poate elimina definitiv orice întrebare din zona numerelor între 1 şi 50. Poate întreba, deci, în continuare: „este numărul ascuns egal cu 75?" ş.a.m.d. În câteva astfel de „salturi" logice, înjumătăţind mereu spaţiul de explorat, el va determina numărul ascuns.

exemplu

Dacă numărul ascuns este 37, vor fi necesare 5 întrebări:
– „Este numărul egal cu 50?" – răspuns „NU";
– „Este numărul > 50?" – răspuns „NU";

– „Este numărul = 25?" – răspuns „NU";
– „Este numărul > 25 ?" – răspuns „DA";
– „Este numărul egal cu [(25+50):2]=37 ?" – răspuns „DA".

Pentru a apela la intuiţia generală, ***maniera de rezolvare*** prin salturile logice de mai sus este referită şi sub numele de „căutarea leului în Sahara". Pe scurt, aceasta se enunţă astfel: „pentru a găsi un leu în Sahara, se împarte Sahara în două (fig. 8.1). Dacă o jumătate este prea mare pentru a fi explorată, se împarte în două la rândul ei şi se repetă raţionamentul în noile condiţii de căutare. La fiecare înjumătăţire a suprafeţei de explorat se pune întrebarea: *când se opreşte căutarea într-o anumită jumătate*? Când, prin înjumătăţirile respective, s-a ajuns să se inspecteze suprafeţe de 1 m^2, care, teoretic, pot fi ocupate de un leu".

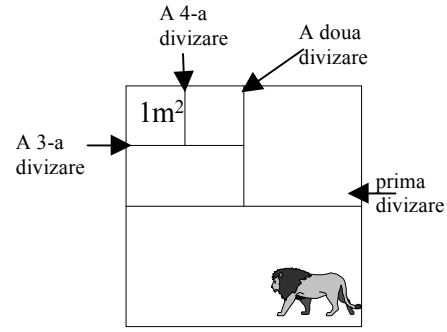


Figura 8.1

2. *Căutarea binară*. Fie o succesiune de numere naturale ***crescătoare***, notate a_1, a_2, …, a_n, unde $n < 100$. Se pune problema dacă printre aceste *n* numere există şi numărul natural notat *b*.

Rezolvare 1.

Din experienţa acumulată din exemplul prezentat mai sus, se poate încerca ceva mai mult. Se consideră fie să nu mai fie acceptată pierderea timpului prin căutarea numărului *b* în mod succesiv printre cele **n** numere până când acesta este găsit fie să nu (deci maximum *n* comparaţii). Se va impune utilizatorului să furnizeze **numerele ordonate crescător,** iar programul să caute numărul în şirul ordonat.

exemplu

Fie următorul vector: A=(1,3,7,19,20,24). Desfăşurarea procesului este controlată prin trei variabile: variabila *m* prin care se determină indicele elementului de mijloc din vector, şi variabilele *s* şi *d*, prin care se delimitează indicii stânga şi dreapta ai subvectorului în care se va face căutarea curentă. Vom analiza cele două cazuri: a) căutarea fără succes; b) căutarea cu succes. În figura 8.2, a) şi b), s-au notat cele trei etape de lucru prin indici ataşaţi variabilelor de urmărire a procesului.

Pentru cazul a) se va căuta numărul 18. Desfăşurarea procesului se bazează pe valorile pe care le iau cele trei variabile: *m*, *s* şi *d*. Se observă cum *s* ajunge să-l depăşească pe *d*. În cazul b) se caută numărul 19, care va fi găsit în vector.

a) Se caută numărul 18; Căutarea fără succes, 18 ∉ vectorului, procesul se opreşte pentru că s3 > d3.

			m_1	s_1		d_1
indicii	0	1	**2**	3	4	5
valorile	*1*	*3*	***7***	*19*	*20*	*24*

				s_3,d_3	m_2	
indicii	0	1	2	3	**4**	5
valorile	*1*	*3*	*7*	*19*	***20***	*24*

			d_3	m_3=s_3		
indicii	0	1	2	**3**	4	5
valorile	*1*	*3*	*7*	***19***	*20*	*24*

m	*s*	*d*	*operaţia*
[(0+5)/2]=2	0	5	18== A[2] nu
2	0	5	18 < A[2]=7 nu
2	m+1=3	5	18 > A[2]=7
[(3+5)/2]=4	3	5	18==A[4]=20 nu
4	3	3=m-1	18<A[4]=20 nu
[(3+3)/2]=3	3	3	18==A[3]=19 nu
3	3	3	18<A[3]=19 nu
3	3	2=m-1	**spaţiu inexistent**

b) Se caută numărul 19; Căutarea cu succes, A[m3]=19.

			m_1	s_1		d_1
indicii	0	1	**2**	3	4	5
valorile	*1*	*3*	***7***	*19*	*20*	*24*

				s_3,d_3	m_2	
indicii	0	1	2	3	**4**	5
valorile	*1*	*3*	*7*	*19*	***20***	*24*

			d_3	m_3		
indicii	0	1	2	**3**	4	5
valorile	*1*	*3*	*7*	***19***	*20*	*24*

m	*s*	*d*	*operaţia*
[(0+5)/2]=2	0	5	19== A[2]=7 nu
2	0	5	19 < A[2]=7 nu
2	m+1=3	5	19 > A[2]=7
[(3+5)/2]=4	3	5	19==A[4]=20 nu
4	3	3=m-1	19<A[4]=20 nu
[(3+3)/2]=3	3	3	19==A[3]=19
			găsit pe locul 3

Figura 8.2

Programul de mai jos face verificare intrării în ordine crescătoare a valorilor vectorului **a**.

```
#include <iostream.h>
//program cautare binara;
unsigned a[100],b;
int s,d,m,n,i, este;
void  main()
{ do
 {cout<<"Dati numarul de elemente ";
  cin>>n;
 }while (!(n>0 && n<100));
cout<<"Dati primul element:";
cin>>a[0];
cout<<"Restul elementelor crescatoare";
for (i=1;i<n;i++)
 do
  {cout<<"a["<<i+1<<"]=";
   cin>>a[i];
  }while(!(a[i]>=a[i-1]));
cout<<"Dati numarul de cautat :";
cin>>b;
s=0;d=n-1;este=0;

while (s<=d && !este)
{
  m=(s+d)/2;
  if (a[m]==b) este=1;
    else
      if (b>a[m]) s=m+1;
       else d=m-1;
}
if (este)
  cout<<"Gasit pe locul "<<m+1;
 else cout<<"Nu exista";
}
```

Analizând procesul de căutare realizat în ambele situaţii: cazul căutării unui număr într-un şir ordonat de numere şi cazul căutării leului în Sahara, se observă următoarele:

Asemănări:

– ***aria*** de inspectat este ***succesiv restrânsă*** prin înjumătăţiri ale ei;

– scopul restrângerii este de a căuta mai uşor şi mai sigur (un tip de efort presupune o căutare într-un spaţiu de 1000 m^2 sau de 50 de numere, şi un alt fel de efort este cerut de o căutare într-un m^2 sau între 2 numere);

– ***prelucrarea*** în sine, de căutare, este ***aceeaşi***, ***indiferent de aria inspectată,*** şi anume, sunt realizate comparări cu scopul identificării "obiectului" căutat;

– procesul se ***încheie*** în momentul ***localizării*** "obiectului" sau în momentul în care a fost ***epuizată*** aria de căutare (nu se mai pot face divizări şi obiectul nu a fost găsit).

Deosebiri:

– în cazul căutării între numere ordonate, odată ce se descoperă că într-o jumătate sigur nu se află numărul, prin testarea valorii căutate cu numărul amplasat „în tăietură", *în acea jumătate nu se mai întoarce procesul căutării*, fiind, evident, o pierdere de timp;

– în cazul căutării leului însă, nu se poate trage concluzia, în momentul divizării ariilor, pe care dintre jumătăţi să o eliminăm, ci trebuie inspectată fiecare arie în parte prin repetarea divizării, până ce se ajunge la aria unitate, când se poate trage o concluzie. ***Apoi se combină rezultatele*** inspectării ariilor în mod invers ordinii de divizare.

Se observă că metoda de localizare a unui element în cadrul unei înşiruiri ordonate de valori este un caz particular al metodei căutării leului în Sahara.

Să asociem un *arbore binar* procesului de căutare explicat până acum, pentru situaţia mai simplă a căutării unei valori într-o înşiruire ordonată **a1, a2, …, an** (figura 8.3).

Fie **n=7**. În nodurile arborelui se vor trece indicii (locurile) de *mijloc* obţinuţi prin divizarea intervalului de indici, adică indicii notaţi cu **m** în program şi rezultaţi din împărţirea întreagă **(s+d):2**.

În parantezele care însoţesc fiecare nod s-au trecut valorile variabilelor **s** şi **d**, adică indicii care marchează capetele spaţiului inspectat.

La început **s=1** şi **d=7**. Pe măsură ce procedul avansează în divizare, **s** şi **d** îşi schimbă valorile, după cum se vede menţionat lângă fiecare nod, în paranteze. Descendenţii stânga vor conţine nodurile divizărilor de tip stânga pentru fiecare set **(s,d)**, iar descendenţii dreapta vor conţine nodurile divizărilor de tip dreapta.

În pătratele notate **f** sunt figurate situaţiile de fals, adică insucces în căutare. Se observă că, în cazul unei căutări cu succes, se coboară în arbore maximum 3 niveluri, pe nivelul 3 încheindu-se explorarea. În cazul unei căutări fără succes, se coboară obligatoriu trei niveluri, pe al patrulea nu se mai ajunge, coborârea fiind stopată de relaţia de oprire, **s>d**.

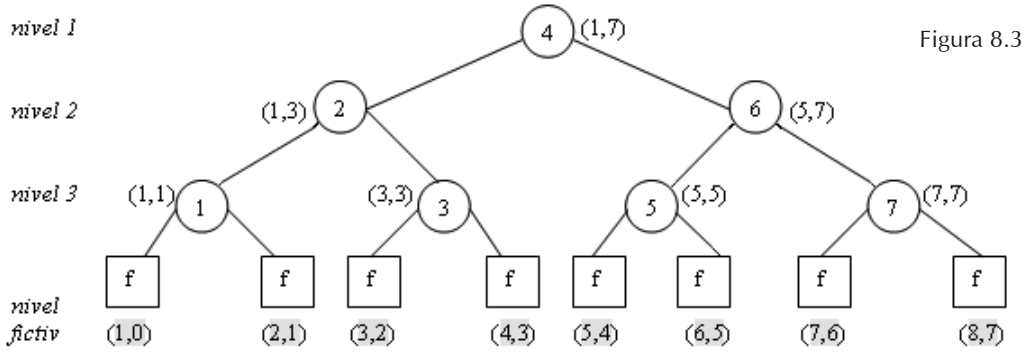


Figura 8.3

Cum descompunerea pe arborele binar a procesului generează un număr de variante, care este o putere a lui 2, se poate calcula numărul maxim de operaţii de divizare pentru identificarea numărului.

Cea mai mare putere a lui 2, care este cuprinsă în numărul de elemente, **n**, indică numărul de niveluri ale arborelui. Astfel, dacă $2^{k-1} \le n \le 2^k$, atunci numărul maxim de operaţii de identificare va fi **k**. Într-adevăr, pentru exemplul luat, $2^2 \le 7 < 2^3$, numărul maxim de operaţii de comparare este 3, adică nivelul maxim de descompunere binară.

Această modalitate de figurare a procesului căutării printr-un arbore binar descompus pe **k** niveluri, unde **k**, în relaţie cu dimensiunea problemei, a dat şi numele matematic al metodei de căutare – **căutarea binară**.

În mod ***recursiv***, procesul apare mult mai clar, deoarece se pune în evidenţă imediat mecanismul reducerii dimensiunii ariei de căutare şi, deci, împărţirea problemei în subprobleme de acelaşi tip, rezolvate în acelaşi mod, principiu *determinant* pentru tehnica recursivităţii.

```
#include <iostream.h>
//program cautare binara - recursiv;
unsigned a[100],b;
int m,n,i;
int caut_bin(int s,int d)
{int m;
 if (s>d)  return -1;
  else
   {
    m=(s+d)/2;
    if (a[m]==b)
       return m;
    else
       if (b>a[m])
        return caut_bin(m+1,d);
       else return caut_bin(s,m-1);
   }
}
void main()
{
do
 {cout<<"Dati numarul de elemente ";
  cin>>n;
 }while(!(n>0 && n<100));
cout<<"Dati primul element:";
cin>>a[0];
cout<<"Restul elementelor";
for (i=1;i<n;i++)
  do
   {cout<<"a["<<i+1<<"]=";
    cin>>a[i];
   }while (!(a[i]>=a[i-1]));
cout<<"Dati numarul de cautat :";
cin>>b;m=caut_bin(0,n-1);
if (m>-1)
  cout<<"Gasit pe locul "<<m;
 else cout<<"Nu exista";
}
```

Rezolvare 2. Aplicăm tot principiul divizării problemei în subprobleme, ca şi în cazul *Rezolvare 1*, dar acum modalitatea de divizare nu va mai fi pe baza mijlocului intervalului de indici, ci prin descompunerea practicată în majoritatea exemplelor date la tehnica recursivităţii.

Astfel, pentru vectorul **a** de **n** elemente, la început **s=0** şi d**=n-1**. Fie **b** numărul căutat. Prima divizare va pune primul element de-o parte şi restul de **n-1** elemente, de cealaltă parte. Se va testa dacă **a[0]==b**. Dacă nu sunt egale, se va proceda asemănător în spaţiul delimitat de **s=1** şi **d=n-1**. Se observă rapid că, în cazul cel mai nefavorabil, se fac **n** comparări. Procedeul este consumator de timp, revenind la a testa secvenţial elementele.

Ambele rezolvări au procedat la reducerea progresivă a ariei de prelucrare, dar una dintre modalităţi este mai eficientă decât cealaltă.

3. *Sortare rapidă.* Dacă problema căutării unui element într-un şir dat, de *n* elemente, are o rezolvare rapidă prin algoritmul de căutare binară, cu condiţia ca înşiruirea de elemente să fie ordonată, atunci se pune întrebarea dacă nu ar exista un algoritm care să ordoneze o înşiruire de *n* elemente (poate în vederea unei căutări binare) mai rapid decât prin metoda bulelor sau metoda clasică a interschimbului. Atât metoda clasică a interschimbului, cât şi metoda bulelor sunt metode care se bazează pe divizarea ariei de lucru în subarii elementare (de câte 2 elemente), în care ordonarea se reduce la un simplu interschimb a două elemente.

Reamintim că, prin *metoda clasică a interschimbului*, în cel mai nefavorabil caz şirul de n valori trebuie reluat de *n* ori pentru a stabili dacă a ajuns să fie ordonat. La fiecare reluare se trece prin tot şirul pentru a vedea dacă există vreo pereche de valori succesive în neordine (crescătoare sau descrescătoare, după cum s-a ales iniţial), deci *n* – 1 comparaţii. În total sunt **`n(n-1)`** comparaţii.

Prin *metoda „bulelor"*, în cel mai nefavorabil caz, şirul de valori este reluat de fiecare dată, însă, mai puţin cu un element faţă de trecerea anterioară. Acest lucru se întâmplă datorită interschimbului, care „mână" progresiv pe ultimul loc cea mai mare valoare din şir (într-o ordonare crescătoare). La trecerea următoare prin şir, această valoare nu mai este necesar a fi comparată, considerându-se ieşită din discuţie (asemenea celei mai mari bule de aer care iese prima dintr-un lichid). Astfel, şirul poate fi „scurtat" de ultimul element. Se realizează astfel (*n* – 1) + + (*n* – 2) + … + 2 + 1 comparaţii, adică **`n(n-1)/2`** comparaţii.

Principiul sortării rapide

– Se desparte înşiruirea celor **`n`** valori în două subşiruri, printr-un element de mijloc, **`a[m]`**.

– Apoi se ordonează valorile, astfel încât partea stângă a diviziunii să conţină numai valori mai mici (nu strict), iar partea dreaptă a diviziunii să conţină numai valori mai mari (nu strict) decât elementul **`a[m]`**.

– Procesul continuă apoi în fiecare diviziune şi aşa mai departe, până ce se ajunge la lungimea minimă de unu sau două elemente.

indici	s=0	1	2	3	4	5	6	7	d=8
iniţial	(4	2	7	5	6	1	3	9	8)
m=4	(4	2	7	5	6	1	3	9	8)
indici	i=0		i=2				j=6		j=8
schimb 7 cu 3	(4	2	**7**	5	6	1	**3**	9	8)
indici					i=4	j=5			
schimbat 6 cu 1	(4	2	3	5	**1**	**6**	7	9	8)
indici	s_{11}=0				d_{11}=4		s_{12}=6		d_{12}=8
divizare I	(4	2	3	5	1)	**6**	(7	9	8)
m_1=2, m_2=7	(**4**	2	3	5	**1**)	6	(7	9	8)
schimbat 4 cu 1	(1	2	3	5	4)	6	(7	9	8)
indici	s_{21}=0		d_{21}=2	s_{22}=3	d_{22}=4		s_{12}=6		d_{12}=8
divizare II	((1	2)	3	(5	4))	6	(7	9	8)
m3=1, m4=3	((1	2)	3	(5	4))	6	(7	9	8)
schimb 4 cu 5	((1	2)	3	(**5**	**4**))	6	(7	9	8)
primul segment e ordonat	(1	2	3	4	5)	6	(7	9	8)
indici							s_{12}=6		d_{12}=8
schimb 9 cu 8	(1	2	3	4	5)	6	(7	**9**	**8**)
tot vectorul e ordonat	1	2	3	4	5	6	7	8	9

Fie, *de exemplu*, înşiruirea de 9 valori întregi: *a* = (4, 2, 7, 5, 6, 1, 3, 9, 8).

• Pentru început, indicii care delimitează înşiruirea sunt: **`s`**=0 şi **`d`**=8. Elementul din mijloc are indicele **`m`**= 4 =[(0+8):2]. Deci **`a[4]`**=6.

• Acum, faţă de valoarea **`a[m]`** din şir, vom analiza elementele din stânga şi pe cele din dreapta. Scopul este ca elementul **`a[m]`** să fie plasat pe locul lui final în şirul ordonat.

Pentru ajutor, se iau doi indici de urmărire **i**, respectiv, **j**, care pornesc unul spre celălalt. La început, **i**←**s** şi j←**d**. Se compară 4<6 ? da; 2 < 6 ? da; 7 < 6 ? NU! se reţine locul lui 7, *adică* **i**=*2*.

Acum se caută în partea dreaptă elementul cu care se va interschimba numărul 7. Deci: 8 > 6 ? da; 9 > 6? da; 3>6? NU ! se reţine locul lui 3, *adică* **j**=*6*. Se face interschimbul şi succesiunea va arăta astfel: *a* = (4, 2, 3, 5, 6, 1, 7, 9, 8).

Se observă că după un interschimb nu este sigur că în stânga lui 6 sunt valori mai mici sau egale cu el şi în dreapta lui, valori mai mari sau egale. Deci, va trebui ca **i** şi **j** să se apropie *până se petrec,* pentru a şti că în spaţiul delimitat de actualii **(s,d)** elementul de mijloc cu care s-a început, aici 6, se află pe locul lui final în şir, adică, faţă de valoarea lui, cele două jumătăţi respectă regula.

Continuăm deci, **i**←3 şi **j**←5 şi cum **i**≤**j**, 5 < 6? da; 6 < 6? NU! şi **i**=4. Acum, pentru partea rămasă în dreapta: 1 > 6 ? NU! şi **j**=5. Deci se vor interschimba **a[4]** cu **a[5]**, adică valorile 6 cu 1.

Noul şir este: *a* = (4, 2, 3, 5, 1, 6, 7, 9, 8) şi i←i+1 adică **i**=5 şi j←j-1, adică **j**=4, deci **i**>**j**. Se vede că 6 a ajuns într-o poziţie pentru care elementele din stânga lui sunt mai mici, iar elementele din dreapta lui sunt mai mari. *Acum se poate spune că numărul 6 provoacă divizarea în fapt a înşiruirii în două subşiruri*: *a'* = (4, 2, 3, 5, 1) şi *a"* = (7, 9, 8), care vor fi delimitate de capetele **s**=0 şi **d**=4 şi respectiv, **s**=6 şi **d**=8.

• Pentru fiecare diviziune în parte, se aplică apoi acelaşi raţionament ca până acum.

În program s-a proiectat funcţia ***recursivă*** **sortare** prin care se pune în evidenţă procesul divizării progresive a problemei date în subprobleme de ordonare crescătoare a unor subşiruri din ce în ce mai scurte, până la maximum 2 elemente. Astfel, recursia continuă cât timp **s**<**j** sau **d**>**i**. Altfel, asociind cu imaginea arborelui binar al procesului divizării, s-ar ajunge pe nivelul fictiv, adică subşir de un element. Odată cu calculele se numără şi comparaţiile făcute în variabila **nrcomp**.

```
#include <iostream.h>
//program sortare rapida_mijloc;
const dim=100;
int a[dim];
unsigned nrcomp;
int n,i;
void sortare(int s,int d)
{int m,i,j,aux;
i=s;j=d;m=(s+d)/2;
do {     //prelucrare elem. de mijloc
  while (a[i]<a[m])
    {i++; nrcomp++; }
  while (a[j]>a[m])
     {j-;nrcomp++;}
  nrcomp+=2;
//pana aici s-au stabilit locurile
//elementelor pentru interschimb
  aux=a[i];a[i]=a[j];
  a[j]=aux;
   i++; j-;
 }while( i<=j);
// o diviziune este epuizata
   if (s<j) sortare(s,j);
   if (d>i) sortare(i,d);
//cand s=j sau d=i s-ar ajunge pe
//nivelul fictiv de divizare, subsir
//de un element!!
}
void main()
{do
{cout<<"Dati numarul de elemente ";
cin>>n;
}while(!(n>0 && n<dim));
cout<<"Dati elementele";
for (i=0;i<n;i++)
  {cout<<"a["<<i+1<<"]=";
   cin>>a[i];
}
sortare(0,n-1);
cout<<"Nr comparatii "<<nrcomp;
cout<<"\nSirul ordonat";
for (i=0;i<n;i++)
   cout<<a[i]<<' ';
}
```

Prin aplicarea procedeului de sortare descris mai sus, numărul de comparaţii scade simţitor faţă de metodele de sortare amintite mai sus. Se demonstrează că acesta ajunge în jurul valorii $n \log_2 n$, pentru cazul cel mai nefavorabil. Procedeul este cunoscut sub numele de **sortarea rapidă** (quick sort) şi se regăseşte sub diverse variante, în funcţie de modul de alegere a elementului cu care se începe lanţul comparărilor într-un şir (elementul iniţial, elementul de mijloc, elementul minim din acel subşir etc.).

4. **Aproximarea unei rădăcini a unei ecuaţii algebrice**. Există multe metode de calcul aproximativ pentru rădăcinile unei ecuaţii algebrice. O metodă de aproximare a uneia dintre rădăcinile unei ecuaţii algebrice, pentru care se ştie faptul că ea există în mod unic în intervalul **[a,b]** este şi metoda înjumătăţirii intervalului.

Astfel, ştiind că funcţia, **f**, asociată ecuaţiei este continuă pe intervalul **[a,b]** dat şi că în cadrul lui ea are o rădăcină, atunci înseamnă că pe acel interval funcţia are o schimbare de semn, adică **f(a)*f(b)<0**. Procesul de aproximare se bazează pe această proprietate astfel:

– se împarte intervalul **[a,b]** în jumătate, adică în intervalele $\left[a, \frac{a+b}{2}\right]$ şi $\left[\frac{a+b}{2}, b\right]$, prin **x=(a+b)/2**.

– se calculează **f(x)**; dacă **f(x)=0** procesul se opreşte şi **x** e rădăcina;

– dacă nu, atunci se repetă înjumătăţirea pentru intervalul **[a,x]**, dacă **f(a)*f(x)<0**, sau pentru **[x,b]**, în caz contrar.

– procesul înjumătăţirilor se opreşte în momentul în care intervalul la care s-a ajuns este **<ε**, unde **ε** este precizia acceptată pentru aproximantă.

În programul de mai jos se prezintă aproximarea unei rădăcini a ecuaţiei de gradul al II-lea. Precizia se citeşte în variabila **eps**, coeficienţii în variabilele **c1**, **c2**, **c3** şi capetele intervalului în variabilele **a** şi **b**.

```
#include<iostream.h>
#include<math.h>
#include<conio.h>
float f(float x, float a, float b,float
                                  c)
{ return a*x*x+b*x+c;}
void main()
{
float c1,c2,c3,x,fa,fb,a,b,eps;
clrscr();
cout<<"Dati precizia de calcul ";
cin>>eps;
cout<<"Dati coeficientii ";
cin>>c1>>c2>>c3;
cout<<"Dati capetele intervalului ";
cin>>a>>b;
while(fabs(a-b)>eps)
      {x=(b+a)/2;
      fb=f(x,c1,c2,c3);
      if(fb==0)break;
      fa=f(a,c1,c2,c3);
      if(fa*fb<0)  b=x;
            else a=x;
      }
cout.precision(3);
cout<<"Aproximanta radacinii="<<x;
getch();
}
```

8.1.2. Definirea metodei „divide et impera"

Din exemplele date se poate desprinde esenţa metodei de rezolvare aplicate. Dictonul latin[1] "divide et impera" – în traducere „dezbină şi stăpâneşte" – se potriveşte foarte bine acestei metode care se bazează pe ideea de a separa problema de rezolvat în fragmente ce pot fi „stăpânite" (rezolvate) cu uşurinţă.

Divide et impera este o metodă de rezolvare a unei **probleme complexe** prin descompunerea ei în **subprobleme** de aceeaşi natură, dar de dimensiuni mai mici, **ierarhizate ca arbore binar**, până la nivelul care permite rezolvarea imediată. **Soluţia** problemei iniţiale se obţine din **combinarea soluţiilor** subproblemelor din descompunere, urmărind înapoi arborele până în rădăcină.

Cele mai frecvente probleme studiate în liceu conduc la prelucrări asupra unor date organizate într-o structură de tablou unidimensional, $a_1, a_2, \ldots, a_n$.

Să presupunem o prelucrare, **PREL,** asupra secvenţei de valori $a_1, a_2, \ldots, a_n$. Presupunem, de asemenea, că pentru orice **s** şi **d** (indicii stânga şi dreapta) numere naturale, cu **1≤ s < d ≤ n**, există un indice **m** ∈ {**s, …, d–1**}, astfel încât prelucrarea secvenţei $a_s, a_{s+1}, \ldots, a_d$ se poate face prelucrând secvenţele $a_s, \ldots, a_m$ şi $a_{m+1}, \ldots, a_d$ cu obţinerea rezultatelor **rez1** şi **rez2**. Apoi, combinând printr-un proces, **COMBIN**, rezultatele, **rez1** şi **rez2**, se obţine *rezultatul* prelucrării întregii secvenţe $a_s, a_{s+1}, \ldots, a_d$. Secvenţele $a_s, \ldots, a_m$ şi $a_{m+1}, \ldots, a_d$ vor rezulta prin divizarea şirului, realizată printr-un proces **DIVIZ** până la o secvenţă de lungime **lung**, pentru care **PREL** se poate face imediat (nivelul elementar care, în arbore, corespunde nodurilor terminale).

Cu aceste notaţii, schema generală ***recursivă*** a metodei *„divide et impera"* se prezintă, în pseudocod, astfel:

[1]Ridicat la principiu de guvernământ, atribuit lui Niccolo Machiavelli, dar de fapt mult mai vechi.

Mecanismul „divide et impera”

Fie **PREL** un anume proces de prelucrare, care se realizează **dacă şi numai dacă** este îndeplinită o condiţie **C privind spaţiul valorilor delimitat de s şi d**. Procesul trebuie să conţină operaţii care să conducă la **convergenţa** lui, adică la atingerea stării ⌉**C**.

funcţia **DEI** (s, d, rezultat)
  Dacă d-s ≤ lung *atunci cheamă* **PREL**(s,d, rezultat)
    altfel
      cheamă **DIVIZ**(s,d,m)
      cheamă **DEI** (s, m-1, rez1)
      cheamă **DEI** (m+1, d, rez2)
      cheamă **COMBIN** (rez1, rez2, rezultat)
  Sfârşit Dacă
Ieşire din funcţie.

Din exemplele date până acum se observă că divizarea succesivă a problemei de rezolvat în subprobleme identice, ca metodă de rezolvare, dar de dimensiuni mai mici, până la o dimensiune pentru care calculele se pot face imediat, nu presupune întotdeauna subşiruri de aceeaşi lungime, rezultate prin separare exact în jumătăţi.

8.1.3. Probleme rezolvate

1) Maximul dintr-o înşiruire. Dându-se un tablou unidimensional *a*, de maximum 100 de elemente reale, să se obţină elementul de valoare maximă.

Rezolvare. Problema de faţă este o problemă foarte cunoscută şi, până acum, a fost rezolvată în mai multe moduri. Aici, ea va constitui motivul primei exemplificări a rezolvării după schema generală a metodei „divide et impera”. Astfel, **PREL** ar avea ca sarcină să determine elementul maxim dintre două valori. Funcţia **DIVIZ** va furniza indicele de mijloc din înşiruirea curentă, adică **m=[(s+d):2].** La început **s=0** şi **d=n-1**. Funcţia **DEI** va aplica prelucrarea în fiecare din cele două părţi care rezultă din **DIVIZ**, după care **COMBIN** va alege în **rezultat** pe cel mai mare dintre **rez1** şi **rez2**.

```
#include <iostream.h>
//program maxim_prin_DEI;
float a[100];
float PREL(float a,float b)
{
    if(a>b)return a;
    return b;
}
void DIVIZ(int s,int d,int &m)
{
 m=(s+d)/2;
}
void COMBIN(float r1,float r2,float &r)
{
      r=PREL(r1,r2);
}
void DEI(int s,int d,float &rezultat)
{float rez1,rez2;int m;
 if (d-s<=1) rezultat=PREL(a[s],a[d]);
      else
        { DIVIZ(s,d,m);
        DEI(s,m,rez1);
        DEI(m+1,d,rez2);
        COMBIN(rez1,rez2,rezultat);
       }
}
void  main()
{ int n,i;float rezult;
do
  {cout<<”Lungimea insiruirii:”;
   cin>>n;
   }while (!(n>1 && n<=100));
for (i=0;i<n;i++)
    {cout<<”a[”<<i+1<<”]=”;
     cin>>a[i];
     }
DEI(0,n-1,rezult);
cout<<”Maximul este:”<<rezult;}
```

Este evident faptul că s-a urmărit numai transpunerea unei prelucrări cunoscute în schema metodei, deoarece, în mod simplificat, toate prelucrările din program se pot face şi altfel, tot pe baza unei aplicări „divide et impera", dar într-un singur subprogram apelat recursiv şi în altă manieră de divizare determinată de formularea ($max(a_1, a_2, \ldots, a_n)=max(a_1,max(a_2,\ldots,a_n))$,până se ajunge la lung=2.

2) Dându-se **n** numere naturale, să se calculeze **cel mai mare divizor comun** al lor.

Rezolvare. Proiectarea unui subprogram pentru calculul cmmdc(a,b) fiind cunoscută, programul de mai jos este uşor de urmărit, el fiind dat ca *exerciţiu de fixare a noţiunilor capitolului de faţă*.

```
#include<conio.h>
#include <iostream.h>unsigned a[100];

unsigned PREL(unsigned a,unsigned b)
{ unsigned r=a;
  while(r){r=a%b;a=b;b=r;}
  return a;}

void DIVIZ(int s,int d,int &m)
{ m=(s+d)/2; }

void COMBIN(unsigned r1,unsigned r2,
                        unsigned &r)
{ r=PREL(r1,r2);}
void DEI(int s,int d,unsigned &rezultat)
{unsigned rez1,rez2;int m;
 if (d-s==1) rezultat=PREL(a[s],a[d]);
  else if(d==s) rezultat=a[s];
       else
        { DIVIZ(s,d,m);
          DEI(s,m,rez1);
          DEI(m+1,d,rez2);
          COMBIN(rez1,rez2,rezultat);
         }}

void  main()
{ unsigned rezult,i,n;
clrscr();
do  {cout<<"Lungimea insiruirii:";
      cin>>n;}while (!(n>1 && n<=100));
for (i=0;i<n;i++)
  {cout<<"a["<<i+1<<"]="; cin>>a[i];
}
DEI(0,n-1,rezult);
cout<<"CMMDC este:"<<rezult;getch();
}
```

3) Numărul de descompuneri. Orice număr natural se poate scrie ca sumă de numere naturale diferite de zero. Astfel, numărul 4 se poate scrie în patru moduri: 3+1, 2+2, 2+1+1 şi 1+1+1+1. Să se calculeze numărul de descompuneri diferite în care se poate scrie numărul natural **n**, **n**<50.

Rezolvare. Există mai multe metode de a rezolva problema. Aici se va aborda metoda „divide et impera", fiind şi mai eficientă ca timp de execuţie. Se descompune numărul **n** în două numere, **d1** şi **d2**, care la rândul lor se descompun în acelaşi mod, până când se ajunge la o diferenţă (**rest**) între ele care devine negativă (**d1<d2**). Iniţial, d1=1 şi d2=n-1 formează prima descompunere.

Pentru numărul 6, suita descompunerilor apare ca în arborele din figura 8.4. Sunt 10 variante de descompunere, ele obţinându-se prin parcurgerea aborelui până la un nivel ales. Pentru nivelul 2, 6=(5)+1=(4+1)+1, sau 6=(5)+1=((3+2)+1).

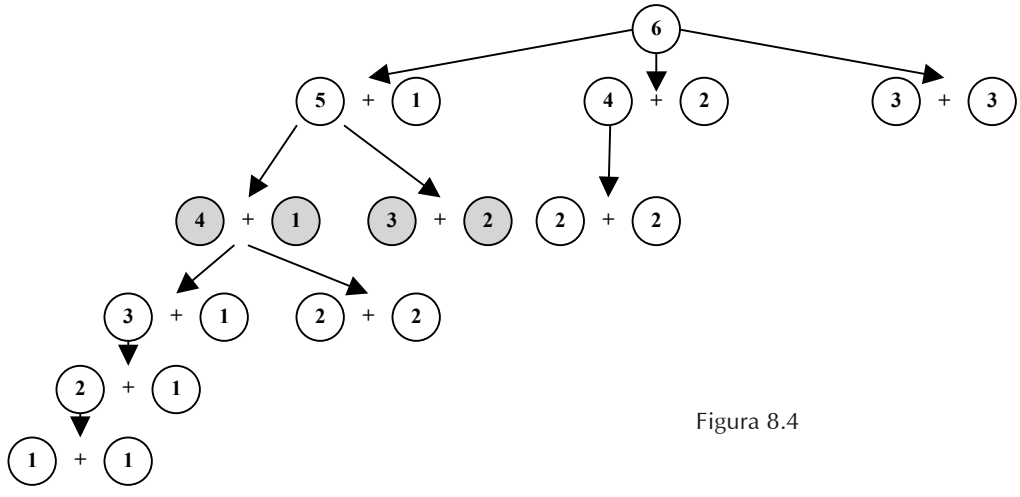


Figura 8.4

```
#include <iostream.h>
//program nr. de descompuneri;
int sum; int n,prim,n1;
void divide(int n,int prim)
{int d1,d2,rest;
d2=prim;
d1=n-d2;
while (d1>=d2)
  {sum++;rest=d1-d2;
   if (rest>prim)
     divide(d1,d2);
   d1-;d2++;
   }
}
void main()
{cout<<"Nr. de descompus, <50 ";
cin>>n;
n1=n;
prim=1;n1=n-prim;sum=0;
while (n1>=prim)
  {sum++;
   divide(n1,prim);
   n1-;prim++;
   }
cout<<n<<" are "<<sum<<" descompuneri";
}
```

4) Problema turnurilor din Hanoi. Sunt trei tije notate cu **a**, **b**, **c** şi **n** discuri perforate central, de diametre diferite, oricare două câte două. Iniţial, toate cele **n** discuri sunt aşezate pe tija **a**, în ordinea descrescătoare a diametrelor, de la bază spre vârf. Cu ajutorul tijei **c**, toate discurile trebuie mutate pe tija **b**, în aceeaşi ordine în care au fost iniţial, respectând următoarele reguli:

– la fiecare pas se mută un singur disc;

– nu se poate muta un disc mai mare peste un disc mai mic în diametru.

Se cere şirul mutărilor.

Rezolvare. Reducând dimensiunea problemei cu o unitate, mutarea celor **n** discuri de pe **a** pe **b** ar însemna mutarea celor **n-1** discuri de deasupra celui de la bază pe tija **c** şi mutarea discului de la baza tijei **a** pe tija **b**. Apoi, cele **n-1** discuri de pe tija **c** trebuie mutate pe tija **b**, cu ajutorul tijei **a**.

O mutare, însă, este permisă pentru a transfera numai un disc din vârful unei tije în vârful altei tije (respectarea stivei). Astfel că mutarea celor **n-1** discuri trebuie analizată ca mutare a **n-2** discuri separat şi al **n-1** –lea separat ş.a.m.d., până ce se ajunge la o mutare elementară, de un disc. Astfel, dacă şirul mutărilor este văzut ca un lanţ de acţiuni **H**, atunci **H(n,a,b,c)** ar reprezenta şirul de mutări a **n** discuri de pe tija **a** pe tija **b**, folosind tija auxiliară **c**.

exemplu

• Fie **n=2**, (fig. 8.5). Atunci **H(2, a,b,c)** va genera suita de acţiuni: **H(1,a,c,b) , a→b, H(1,c,b,a)**, din care rezultă lanţul de mutări: **(a→c) , (a→b) , (c→b)** .

• Fie **n=3**. Atunci **H(3,a,b,c)** va genera lanţul de mutări: **H(2,a,c,b) , a→b,H(2,c,b,a)**, care mai departe se descompune în: **H(1,a,b,c) , a→c,H(1,b,c,a) , a→b, H(1,c,a,b) , c→b, H(1,a,b,c)**

adică şirul: **(a→b) , (a→c) , (b→c) , (a→b) , (c→a) , (c→b) , (a→b)**

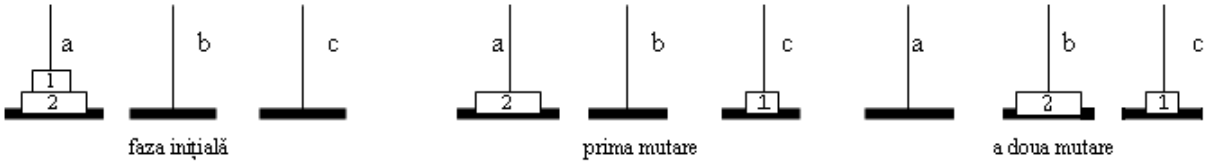


Figura 8.5

rezolvă

Să se realizeze practic mutările pentru n=3 şi apoi pentru n=4. Pentru aceasta, se vor numerota discurile şi se vor face desenele fiecărei faze a mutărilor.

În general, şirul mutărilor se va genera recursiv, micşorând progresiv dimensiunea problemei până la o mutare elementară, astfel:

$$\mathbf{H(n,a,b,c)} = \begin{cases} \mathbf{a \to b}, \text{ pentru } n = 1 \\ \mathbf{H(n-1,a,c,b), a \to b, H(n-1,c,b,a)}, \text{ pentru } n > 1 \end{cases}$$

În program, pentru uşurinţa înţelegerii schimbării sarcinilor între cele trei tije de la fază la fază, tijele au fost notate cu numerele 1, 2 şi, respectiv, 3, astfel că întotdeauna tija care este de serviciu (auxiliară) se poate calcula prin diferenţă, **6-(tp+ts)**, unde **tp** este tija de pe care pleacă discurile şi **ts** este tija pe care se mută discurile la un anumit moment. De exemplu, dacă mutăm de pe tija 2 pe 3, atunci tija auxiliară va fi 6-2-3=1. Pentru afişarea în notaţiile **a**, **b** şi **c** ale tijelor, s-a utilizat o constantă-şir de trei caractere, notată **sir**.

```
#include <iostream.h>
//program hanoi_recursiv;
const char sir[4]="abc";
int n;

void muta (int n, int tp, int ts)
//tp -tija de plecare
//ts - tija de sosire
{
cout<<"Disc "<<n<<" se muta din ";
cout<<sir[tp-1]<<" pe "<<sir[ts-1];
cout<<'\n';
}
void h(int n, int tp,int ts)
{ if (n==1) muta(1,tp,ts);
      else {
        h(n-1,tp,6-(tp+ts));
 //6-(tp+ts)- tija auxiliara
        muta(n,tp,ts);
        h(n-1,6-(tp+ts),ts);
        }
}
void main()
{cout<<"n=";
 cin>>n;
  h(n,1,2);
}
```

Prin inducţie matematică se poate demonstra că rezolvarea problemei prin metoda propusă, care generează lanţul minim de mutări, se realizează în 2^n-1 mutări. Deci, pentru n=3, vor fi 7 mutări necesare pentru a aduce discurile de pe tija *a* pe tija *b*.

1. Se citesc **n**, cuvinte din fişierul **cuvinte.txt,** n<100. Să se determine:
a) cel mai scurt cuvînt; b) cel mai mare cuvînt în sens lexicografic,
utilizând metoda *„divide et impera"*.

2. Aplicaţi schema metodei *„divide et impera"* pentru a determina numărul valorilor negative din vectorul **v**, cu **n** numere întregi, **n**<100.

3. Aplicaţi schema metodei *divide et impera* pentru a determina numărul valorilor pare din vectorul **v**, cu **n** numere întregi, **n**<100.

4. Fie **T** un tablou unidimensional cu **n** numere întregi distincte, **n**<100. Numerele din tablou sunt ordonate crescător. Să se aplice schema metodei *„divide et impera"* pentru a afişa toţi indicii **p**, 1≤**p**≤**n**, cu proprietatea că **T[p]=p**.

5. Aplicaţi schema metodei *„divide et impera"* pentru a verifica dacă toate elementele unui vector de **n** numere naturale, **n**<100, sunt numere prime.

6. Aplicaţi schema metodei *„divide et impera"* pentru a calcula suma elementelor pozitive dintr-un vector **v** cu **n**, numere întregi, **n**<100.

7. Aplicaţi schema metodei *„divide et impera"* pentru a calcula suma 1*2+2*3+…+n*(n+1).

8. Să se elaboreze programele pentru sortarea rapidă, pe rând, în cazurile în care alegerea elementului de referinţă în procesul ordonării unei secvenţe curente este: elementul minim, primul element, elementul maxim din secvenţă. Programele vor fi elaborate astfel încât să pună în evidenţă schema generală a metodei „d*ivide et impera"*.

9. Să se transpună în schema generală a metodei *„divide et impera"* algoritmul de căutare binară. Ce formă capătă rutina **COMBIN** ?

10. Fie **T** un tablou unidimensional cu **n** numere reale, **n**<100. Parcurgându-se o singură dată tabloul, să se determine valoarea maximă şi valoarea minimă.

11. Să se transpună în schema metodei *„divide et impera"* algoritmul de **FILL**.

[1]Pentru k=1 → 2^1 – 1=1 adică *H(0), o mutare elementară, H(0)*; pentru k=2 →2^2 – 1=3, adică *H(1), mutare elementară, H(1)*=(2^1-1)+1+(2^1-1), ş.a.m.d. pentru k=n→2^n – 1adevărat. Pentru k=n+1 ? *H(n), mutare elementară, H(n)* adică 2^n – 1+1+2^n – 1=2 2^n – 1=2^{n+1} – 1.

8.2. Metoda BACKTRACKING

8.2.1. Exemple

1. *Defilare.* Pentru a asigura un fundal muzical la o prezentare, un creator de modă vrea să pregătească o casetă cu trei melodii care să îi pună în valoare modelele. După ce a ales melodiile potrivite evenimentului, a apărut problema ordinii în care să aşeze aceste melodii pe casetă. Există vreo „reţetă" sigură, astfel încât să se poată alcătui **toate succesiunile celor trei melodii** pe casetă, **fără a repeta vreuna**, pentru a studia şi a alege combinaţia care îi place?

Rezolvare.

Pentru a simplifica lucrurile, melodiile vor fi numerotate cu 1, 2 şi, respectiv, 3. Figurând banda din casetă ca o zonă liniară împărţită în trei spaţii (LOC 1, LOC 2 şi LOC 3), pentru cele trei melodii, ar apărea succesiunea încercărilor din figura 8.6. În figură încercările cu soluţie sunt marcate în litere îngroşate, iar cele fără soluţie cu *.

LOC 1	LOC 2	LOC 3	**Comentariu**
			caseta vidă
1			*Ocupare* LOC 1 – aşezarea melodiei 1
1	*1		*Continuă* cu LOC 2: nu poate veni iar melodia 1 !
1	2		*Ocupare* LOC 2 – aşezarea melodiei 2
1	2	*1	*Continuă* cu LOC 3: nu poate veni iar melodia 1 !
1	2	*2	*Încearcă iar* pe LOC 3: nu poate veni iar melodia 2 !
1	**2**	**3**	**soluţia 1**
1	2	*4 ?	*Încearcă iar* pe LOC 3: există melodia 4 ?
1	3		**Întoarcere** pe LOC 2 şi aşezarea melodiei 3 !
1	3	*1	*Continuă* cu LOC 3: nu poate veni iar melodia 1 !
1	**3**	**2**	**soluţia 2**
1	3	*3	*Continuă* cu LOC 3: nu poate veni iar melodia 3 !
1	3	*4 ?	*Încearcă iar* pe LOC 3: există melodia 4 ?
1	*4 ?		**Întoarcere**. *Continuă încercările* pe LOC 2: există melodia 4 ?
2			**Întoarcere** pe LOC 1 şi aşezarea melodiei 2
2	1		*Continuă* cu LOC 2: aşezarea melodiei 1
2	1	*1	*Continuă* cu LOC 3: nu poate veni iar melodia 1 !
2	1	*2	*Încearcă* iar pe LOC 3: nu poate veni iar melodia 2 !
2	**1**	**3**	**soluţia 3**
2	1	*4 ?	*Încearcă iar* pe LOC 3: există melodia 4 ?
2	*2		**Întoarcere** pe LOC 2: nu poate veni iar melodia 2 !
2	3		*Încearcă* pe LOC 2 melodia 3 şi se poate aşeza !
2	**3**	**1**	**soluţia 4**
2	3	*2	*Încearcă iar* pe LOC 3: nu poate veni iar melodia 2 !
2	3	*3	*Încearcă iar* pe LOC 3: nu poate veni iar melodia 3 !
2	3	*4 ?	*Încearcă iar* pe LOC 3: există melodia 4 ?
2	*4 ?		**Întoarcere**. *Continuă încercările* pe LOC 2: există melodia 4 ?
3			**Întoarcere** pe LOC 1: poate veni următoarea melodie
3	1		*Continuă* pe LOC 2: aşază melodia 1
3	1	*1	*Continuă* cu LOC 3: nu poate veni iar melodia 1!
3	**1**	**2**	**soluţia 5**
3	1	*3	*Încearcă iar* pe LOC 3: nu poate veni iar melodia 3 !
3	1	*4 ?	*Încearcă iar* pe LOC 3: există melodia 4 ?
3	2		**Întoarcere** pe LOC 2: poate veni următoarea melodie
3	**2**	**1**	**soluţia 6**
3	2	*2	*Încearcă iar* pe LOC 3: nu poate veni iar melodia 2 !
3	2	*3	*Încearcă iar* pe LOC 3: nu poate veni iar melodia 3 !
3	2	*4 ?	*Încearcă iar* pe LOC 3: există melodia 4 ?
3	*3		**Întoarcere** pe LOC 2: nu poate veni iar melodia 3 !
3	*4 ?		*Încearcă iar* pe LOC 2: există melodia 4 ?
*4 ?			**Întoarcere** pe LOC 1: există melodia 4 ?
?	**?**	**?**	**nu mai este nici o posibilitate de încercat**

Figura 8.6

Din acest tabel se poate observa că se fac simple permutări a trei valori, 1,2 şi 3, pe trei locuri. De asemenea, pare că nu ar avea nici un rost, de exemplu, ca, după aşezarea lui 1 pe primul loc, să se verifice dacă 1 se poate aşeza şi pe locul doi. E doar evidentă folosirea lui! Aceeaşi părere se iveşte şi când, epuizându-se valorile posibile, se trece la valoarea 4, care este clar că nu există în discuţie.

Aceasta este însă o reţetă pentru roboţi, un algoritm pentru calculator, care exprimă tocmai ***mecanismul gândirii realizat în procesul generării acestor permutări, defalcat la nivel de operaţii bază***. Când trebuie aleasă o valoare pentru LOC 2, după ce pe LOC 1 s-a aşezat valoarea 1, *„evidentul" că nu-l aleg tot pe 1* trece prin operaţia de bază *„compar valoarea care candidează la alegere, 1, cu ceea ce am ales până atunci pentru a vedea dacă a mai fost aleasă"*. Pentru om, această operaţie se face instantaneu (şi pare evidentă), dar pentru o maşină (robot), care nu realizează decât operaţii elementare în unitatea de timp, ea trebuie să fie prevăzută în bagajul de cunoştinţe (aşa cum şi omul, copil fiind, a învăţat să realizeze instantaneu operaţii complexe trecând, din ce în ce mai rapid, prin suita operaţiilor elementare).

Figurând în pseudocod operaţiile din tabelul de mai sus, dar generalizând la ***n*** melodii, programul robotului apare ca în figura 8.7.

```
banda : tablou de n elemente naturale
k : indice pt. bandă, n : natural //numărul de melodii
citeşte n
k←1; banda [k] ←0 //nu s-a ales încă nimic pe poziţia k
cât timp  k > 0  execută
        f←fals        //ipoteza: nu este bun ce am ales pe locul k;
                                  la început valoarea 0
        cât timp  banda [k] < n  ∧  ¬f  execută
              // mai există valori nealese pentru locul k şi f încă e fals
            banda[k]←banda[k] + 1
              // continuă să aleagă o valoare următoare
            Dacă  valid (k)          atunci  f←adevărat
              //verificarea  dacă valoarea nu a mai fost aleasă
            Sfârşit Dacă
        reia
        Dacă f atunci
              Dacă  k = n  atunci scrie_soluţia_curentă
                  altfel
                    k←k+1      //înaintare pe poziţia următoare
                    banda[k]←0
              Sfârşit Dacă
            altfel
              k←k-1   //întoarcere
        Sfârşit Dacă
reia
Stop.

valid (k)
i : natural; v: logic
v ← adevărat
i←1
   cât timp   i<k ∧ v     execută
      dacă banda[ i ] = banda [k]
                  atunci v←fals
      Sfârşit Dacă
      i←i+1
   reia
valid ← v
ieşire
```

Figura 8.7

2) *Melodii în ordine.* După ascultarea separată a melodiilor, creatorul de modă doreşte ca, în toate variantele de aşezare a melodiilor pe casetă, melodia **3** să fie cântată înainte de a apărea melodia **1**.

Rezolvare. O soluţie ar fi ca, după ce se completează o permutare, să se verifice dacă este îndeplinită condiţia, şi dacă nu, să se renunţe la afişarea acelei permutări. Astfel, din tabelul de la exemplul 1, s-ar afişa doar soluţiile 4, 5 şi 6. Primele trei ar fi fost generate inutil!

O soluţie care economiseşte timp pentru condiţiile nou apărute, este ca, în „reţeta" prezentată mai sus, *restricţia de ordine să se includă în subprogramul* ***valid*** şi astfel, în momentul efectuării unei alegeri, *valid* va întoarce *fals* dacă, spre exemplu, melodia *x* nu a apărut pe un loc dinaintea melodiei ***y* care urmează a fi** aşezată. Apariţia melodiei *x* pe bandă este urmărită de variabila de stare *g*. Dacă *x* nu a apărut pe bandă şi melodia curentă de aşezat este *y*, se opreşte generarea în continuare a soluţiei şi se trece la o nouă încercare pe poziţia curentă. Noua formă a subprogramului *valid* este:

```
valid(k)
i : natural; g ,v : logic
v←adevărat          //ipoteza că alegerea de pe poziţia k va fi bună
g←fals              //ipoteza că melodia x a fost aşezată deja
i←1
Cât timp  i < k  ∧  v  execută
          Dacă  banda[ i ]=banda[ k ]
                atunci  v ← fals
                altfel
                       Dacă  banda[ i ]=x  atunci  g←adevărat
          Sfârşit Dacă
          Sfârşit Dacă
          i←i+1
reia
Dacă  banda[ k ] = y   atunci    v←v ∧ g
Sfârşit Dacă
valid ← v
ieşire
```

Se poate pune întrebarea: „dacă **y** se aşază pe prima poziţie, cum rezolvă *valid* situaţia dacă apoi vine **x** ?"

Se observă că pentru **k**=1 procesul nu va mai intra în repetiţie, pentru că **i**=1 *nu este mai mic strict* decât **k**=1 şi astfel **g** rămâne cu valoarea *fals*, dată iniţial.

La sfârşit, când se trage concluzia, va opera expresia **v**← **v** ∧ **fals** , deci **y** nu va fi amplasat pe prima poziţie şi, prin tranzitivitate, pe nici o altă poziţie până ce **x** nu apare şi este amplasat.

3. *Drapele.* Fie următoarele 9 culori: *alb, turcoaz, roşu, galben, negru, albastru, verde, mov şi maro*. Se cere compunerea tuturor drapelelor *tricolore* posibile, cu restricţia ca în mijloc să fie *alb* sau *negru*, prima să fie o culoare deschisă, iar ultima, o culoare închisă.

Rezolvare. Fiind vorba de drapele *tricolore*, se vor alege toate combinaţiile de trei culori distincte din cele 9 date, cu obligaţia ca în mijloc să fie doar alb sau negru; prima extremă să aibă culori deschise, a doua extremă, închise.

Datele de intrare sunt codificate cu indicii asignaţi culorilor într-un tablou de 9 şiruri de caractere. După cum sunt înşirate culorile în enunţ, primele patru culori sunt deschise, iar ultimele 5, închise. Poziţiile 1 şi 5 din tablou sunt ocupate de culorile de mijloc.

Datele de ieşire vor fi configuraţii de câte trei culori. Rezultă că un drapel se poate codifica sub forma unui tablou unidimensional cu trei elemente, *care vor fi indicii culorilor alese din tabloul de culori*.

Problema se poate programa foarte simplu, cu trei repetiţii compuse prin subordonare.

Dorim să aplicăm totuşi reţeta de tip robot configurată mai sus.

Apare o situaţie nouă faţă de exemplul precedent: în toate cele trei poziţii ale drapelului, **LOC1**, **LOC2** şi **LOC3**, vom avea acum de ales valori din mulţimi distincte de valori. Valorile pentru **LOC1** ∈ {**2,3,4**}; valorile pentru **LOC2** ∈ {**1,5**}; valorile pentru **LOC3** ∈ {**6,7,8,9**}. În acest sens, funcţia **valid** are o structură de selecţie – **switch** – prin care determină dacă alegerea de pe poziţia curentă, **k**, corespunde mulţimii din care trebuie să se facă sau nu.

Pentru controlul afişării, soluţiile sunt numerotate prin variabila **nr**, astfel încât funcţia **scrie** afişează şi numărul drapelului creat. O soluţie este afişată astfel ca utilizatorul să citească culorile din drapel şi nu indicii lor din tabloul de culori, aşa cum s-a lucrat intern în program. În tabelele de mai jos sunt figurate 33 de etape ale încercărilor de construire de drapele tricolore în condiţiile date. Etapele 9 – 12 şi 22 – 25 sunt încercări cu succes, constituind primele opt soluţii. Alegerile care nu conduc către soluţie sunt marcate cu *. În tabele s-au scris numele culorilor încercate şi nu indicii lor.

1	*alb	–	–
2	turcoaz	–	–
3	turcoaz	alb	–
4	turcoaz	alb	*alb
5	turcoaz	alb	*turcoaz

6	turcoaz	alb	*roşu
7	turcoaz	alb	*galben
8	turcoaz	alb	*negru
9	turcoaz	alb	albastru
10	turcoaz	alb	verde

11	turcoaz	alb	mov
12	turcoaz	alb	maro
13	turcoaz	*turcoaz	–
14	turcoaz	*roşu	–
15	turcoaz	*galben	–

16	turcoaz	negru	–
17	turcoaz	negru	*alb
18	turcoaz	negru	*turcoaz
19	turcoaz	negru	*roşu
20	turcoaz	negru	*galben
21	turcoaz	negru	*negru

22	turcoaz	negru	albastru
23	turcoaz	negru	verde
24	turcoaz	negru	mov
25	turcoaz	negru	maro
26	turcoaz	*albastru	–
27	turcoaz	*verde	–

28	turcoaz	*mov	–
29	turcoaz	*maro	–
30	rosu	–	–
31	rosu	alb	–
32	rosu	alb	*turcoaz
33	rosu	alb	*roşu

```
#include <iostream.h>
//drapele
int s[3],i,nr;
char cul[9][9];
void scrie() //afisarea solutiei nr
{int i;
 cout<<"drapel "<<nr;
 for (i=0;i<3;i++)
  cout<<cul[s[i]]<<' ';
 cout<<'\n';
}
int valid(int k)
{int i,f;
 f=1;i=0;
 while (i<k && f)
  {if (s[i]==s[k]) f=0;
   i++;
  }
if (f) switch (k)
 {case 0:if (!(s[k]<=3 && s[k]>=1))
            f=0;break;
   case 1:if(!(s[k]==0||s[k]==4))
            f=0;break;
   case 2:if(!(s[k]>=5 && s[k]<=8))
            f=0;break;
 }
 return f;
}
```

```
void drapel()
//funcþia de incercari i alegeri
{int k,f;
 k=0;s[k]=-1;
 while (k>-1)// -1 = indice inexistent
 {
   f=0;
   while (s[k]<8 && !f)
    {s[k]=s[k]+1;
     f=valid(k);
    }
  if(f)
   if (k==2)
      {nr++; scrie();
      }
     else
       {k++;s[k]=-1;
       }
  else k–;
 }
}
void main()
{cout<<"culorile: ";
 for(i=0;i<9;i++)
  {cout<<"nr["<<i+1<<"]";
  cin>>cul[i];
  }
 drapel();
}
```

rezolvă

Să se realizeze tabelul cu încercările necesare obţinerii următoarelor patru soluţii.

Concluzii

Din cele trei exemple prezentate mai sus se pot trage următoarele concluzii:

c1 – soluţiile problemelor s-au constituit fiecare dintr-un ansamblu de componente identice din punctul de vedere al formei şi tipului de reprezentare.

c2 – pentru o anume problemă, fiecare soluţie are acelaşi număr de componente, adică aceeaşi lungime.

c3 – în cadrul soluţiei, o componentă are dreptul la valori dintr-o anumită mulţime.

c4 – schema de generare a unei noi componente cere ca toate componentele precedente să fi fost generate până atunci şi constă în:

– alegerea pentru componenta curentă a unei următoare valori, dintre valorile nealese încă din mulţimea de valori destinate acelei componente.

– odată aleasă o valoare pentru componenta curentă, se face verificarea corectitudinii ei conform condiţiilor din enunţ şi noncontradicţiei cu valorile alese până atunci în componentele precedente;

– se fixează valoarea aleasă, dacă aceasta este admisă de testele de verificare, sau se reia alegerea cu o următoare valoare din mulţimea de valori ale componentei.

c5 – dacă, prin încercări succesive, se epuizează mulţimea de valori pentru o componentă, atunci când nici una nu este admisă de către testul de corectitudine, se face o întoarcere la componenta anterioară, ignorându-se tot ce s-a lucrat pentru componenta curentă. Întoarcerea la componenta anterioară are ca scop încercarea altor valori, încă nealese, în speranţa că, la înaintarea către finalul soluţiei, ea va conveni alegerii pentru componenta următoare. Prin tranzitivitate, această întoarcere poate să meargă înapoi cu mai mulţi paşi. La limită, întoarcerea poate conduce prelucrarea spre stânga la o componentă inexistentă (*k* devine –1, în condiţiile de indici), fapt ce poate însemna că s-au epuizat toate posibilităţile.

c6 – dacă o valoare aleasă este validată, atunci se produce înaintarea către construirea componentei următoare. Aceasta poate să existe sau nu (s-a atins lungimea soluţiei), moment în care se încheagă o soluţie ce poate fi scrisă.

c7 – modalitatea de rezolvare oferită de această „reţetă" generează *toate soluţiile valide* (care respectă relaţiile între componente) dintre soluţiile posibile ale problemei în cauză (în prima problemă, de exemplu, sunt 3^3 soluţii posibile, de la (1,1,1), (1,1,2),… până la (3,3,3), dar nu toate sunt valide, ci numai în 6 dintre ele nu se repetă vreo valoare = condiţiile între componente).

rezolvă

Pentru exerciţiile de mai jos *aplicaţi algoritmul de generare descris până acum* şi construiţi încercările cu ajutorul tabelului.

1. Se consideră numerele naturale **m** şi **n**, **0<m<n<12**. Se doreşte generarea tuturor şirurilor formate din **m** litere alese dintre primele **n** ale alfabetului englez. Pentru **m**=2 şi **n**=4 se generează: AA, AB,AC,AD,BA,BB,BC,BD,CA,CB,CC,CD,DA,DB,DC,DD.
   Construiţi tabelul etapelor de generare pentru cazul ales. Construiţi tabelul etapelor de generare pentru situaţia în care **m**=3 şi **n**=2. Determinaţi care este operaţia matematică pentru obţinerea mulţimii de valori care sunt soluţii ale problemei.
2. Se consideră numerele naturale **m** şi **n**, **0<m<n<12**. Se doreşte generarea tuturor şirurilor formate din **m** litere ***distincte*** alese dintre primele **n** ale alfabetului englez. Pentru **m**=2 şi **n**=4 se generează: AB,AC,AD,BA,BC,BD,CA,CB,CD,DA,DB,DC.
   Construiţi tabelul etapelor de generare pentru cazul ales. Construiţi tabelul etapelor de generare pentru situaţia în care **m**=3 şi **n**=5. Determinaţi care este operaţia matematică pentru obţinerea mulţimii de valori care sunt soluţii ale problemei.
3. Se citesc numerele naturale **n** şi **k**, cu **0<k≤10** şi **0<n≤10000**. Se doreşte generarea, în ordine crescătoare, a tuturor numerelor naturale de **k** cifre, care sunt formate numai din cifrele numărului **n**. Pentru **n**=327 şi **k**=2 se generează: 22, 23, 27, 32, 33, 37, 72, 73, 77.
   Construiţi tabelul etapelor de generare pentru situaţia în care **n**=3145 şi **k**=3. Determinaţi care este operaţia matematică pentru obţinerea mulţimii de valori care sunt soluţii ale problemei.
4. Se consideră mulţimea cifrelor **{1,2,3,4}**.
   Se doreşte generarea tuturor şirurilor de **n** cifre alese din această mulţime, astfel încât oricare două cifre alăturate sunt fie ambele pare, fie ambele impare. Pentru **n**=3 se vor genera: 111, 113, 131, 133, 222, 224, 242, 244, 311, 313, 331, 333, 422, 424, 442, 444.
   Stabiliţi lista încercărilor, subliniind soluţiile, pentru cazul în care **n**=4.
5. Se consideră numerele naturale **n** şi **k**, **0<n,k<1000**. Se doreşte generarea tuturor şirurilor strict crescătoare de lungime **k**, formate din divizori ai numărului **n**. Pentru **n**=8 şi **k**=2 se vor genera şirurile: 1 2; 1 4; 1 8; 2 4; 2 8; 4 8. Stabiliţi lista încercărilor, subliniind soluţiile, pentru cazul în care **n**=36 şi **k**=4.
6. Generarea tuturor numerelor de **3 cifre**, fiecare cifră putând fi orice număr din mulţimea **{5,3, 6}**, realizează ca prime 5 soluţii: 666, 663, 665, 636, 633. Care dintre numerele din lista următoare este soluţia a şasea: 563, 635, 536, 366?
7. Generarea tuturor numerelor de **trei cifre distincte**, fiecare cifră putând fi orice număr din mulţimea **{5,3, 6,7,2}**, realizează ca prime 5 soluţii: 276, 273, 275, 263, 265. Care dintre numerele din lista următoare este soluţia a şasea: **7**63, 635, 235, 735 ?

8. Pentru a obţine toate numerele formate din **3 cifre** alese din şirul **2,9,4**, numerele sunt generate în ordinea: 222, 229, 224, 292, 299, 294, 242, …, 494, 442, 449, 444. Aplicând acelaşi procedeu pentru a obţine numere de 4 cifre, după numărul 4944 va urma: 4422, 4949, 9222, sau 4924?

9. Dacă se generează **toate permutările** de **5 obiecte** şi primele patru permutări sunt: 5 4 3 2 1; 5 4 3 1 2; 5 4 2 3 1; 5 4 2 1 3, care este a cincea permutare dintre permutările din lista următoare: 5 4 3 2 1; 5 3 4 2 1; 5 4 1 2 3; 5 4 1 3 2 ?

10. Pentru a determina toate modalităţile de a scrie pe **9 ca sumă** de numere naturale nenule distincte, folosind mecanismul prezentat, vor fi generate următoarele soluţii: 1+2+6; 1+3+5; 1+8; 2+3+4; 2+7; 3+6; 4+5. Notaţi numărul de componente ale fiecărei soluţii. Aplicându-se acelaşi mecanism să se determine soluţiile pentru scrierea lui 12. Care este a 6-a soluţie pentru scrierea lui 18 ?

11. Se consideră numărul **s**=61 şi **8 numere naturale**: 12,61,22,57,10,4,23,30. Toate mulţimile de numere dintre cele opt date, cu proprietatea că pentru fiecare mulţime suma elementelor este **s**, sunt: 4 12 22 23; 4 57; 61. Notaţi numărul de componente ale fiecărei soluţii. Stabiliţi lista încercărilor pentru a obţine mulţimile care se formează dacă s=57.

8.2.2. Definirea metodei backtracking

Procedeul de rezolvare aplicat în exemplele precedente devine o metodă generală pentru orice problemă de tipul celor de mai sus şi poate fi o reţetă de programare. Metoda are elemente definitorii clare:

Soluţia este o structură de date omogenă. În general, pentru problemele mai simple, este un tablou unidimensional. Fie notaţia soluţiei $S = (s_1, s_2, s_3, \ldots s_n)$.

Lungimea soluţiei reprezintă numărul de componente ale structurii de date S. Notaţia folosită: *n* sau *lung*. Soluţiile pot avea acelaşi număr de componente sau pot avea un număr variabil de elemente, în funcţie de atingerea unui prag dat în enunţ.

Mulţimile de valori permise fiecărei componente sunt mulţimi finite, notate V_i, $i \in \{1,\ldots,n\}$.

Mulţimea $V_1 \times V_2 \times \ldots \times V_n$ alcătuieşte *spaţiul soluţiilor posibile*.

Condiţiile interne sunt relaţiile dintre componentele soluţiei. Aceste condiţii impun alegerea dintre soluţiile posibile a celor reale, ***valide***, care vor fi afişate ca rezultat.

Condiţiile de continuare sunt condiţii care stabilesc dacă are sens să se treacă la calculul următoarei componente a soluţiei, s_{k+1}, sau, orice valoare am alege pentru s_{k+1}, s_{k+2}, …,s_n, dacă nu se va închega o soluţie rezultat şi deci, va trebui să alegem o altă valoare pentru s_k din V_k, iar dacă nu mai sunt valori de ales în V_k, atunci să ne întoarcem la o nouă alegere pentru s_{k-1} (**backtracking**= urmărire înapoi) dintre cele rămase.

Backtracking este o **metodă de rezolvare** a acelor probleme ale căror soluţii sunt *structuri de date omogene* care aparţin unui spaţiu multidimensional. Mecanismul ei are ca scop căutarea, în spaţiul soluţiilor posibile *a tuturor soluţiilor rezultat* ale problemei în cauză.

Etapele metodei

I. Determinarea **tipului soluţiei:** vector sau matrice. În continuare, pentru soluţie tip vector, se va nota $S = (s_1, s_2, \ldots, s_{lung})$.

II. Determinarea **numărului de componente** ale soluţiei: *fix* sau *variabil* (până la o valoare maximă ce trebuie calculată pentru alocare), pe care o notăm *lung*.

III. Determinarea **mulţimii de valori permise fiecărei componente** a soluţiei: V_k, k fiind indicele componentei curente, k. ∈ {1,2,…,lung}.

IV. Stabilirea **condiţiilor interne** (relaţiile între componente) şi a celor de **continuare** pentru alegerea valorii componentei următoare (noncontradicţia cu alegerile din componentele anterioare):

valid – validitatea alegerii curente.

V. **Mişcarea: înaintare sau întoarcere,** în funcţie de adevărul predicatului **Q:**

Q : ($\exists$) o valoare **x** $\in$ $\mathbf{V}_k$ astfel încât **valid**(x).

– **înaintare:** *se trece la componenta* **următoare** dacă predicatul **Q** este adevărat

– **întoarcere:** *se trece la componenta* **anterioară** dacă predicatul **Q** nu este adevărat, adică:

($\forall$) **x** $\in$ $\mathbf{V}_k$ atunci $\neg$ **valid**(x).

La înaintare poate apărea situaţia în care k = lung, şi atunci înseamnă că s-a finalizat o soluţie ce trebuie afişată (înregistrată);

La întoarcere poate apărea situaţia în care *k = 0,* şi atunci înseamnă că s-au terminat toate soluţiile posibile.

Între condiţiile interne şi cele de continuare există o strânsă legătură, fapt pentru care testele de verificare, de obicei grupate în funcţia ***VALID***, vor fi astfel alcătuite încât să combine cele două tipuri de condiţii. Figurată în pseudocod, schema generală de rezolvare backtracking apare astfel:

```
funcţia BKT (s,lung)
k ←1
iniţializare s_k
Cât timp k > 0 execută
    f ← fals
    Cât timp (∃) valoare în V_k ∧ ¬ f execută
        s_k ← valoare
        Dacă VALID(k) atunci f ← adevărat
        Sfârşit Dacă
    Reia
    Dacă f atunci Dacă k=lung atunci SCRIE_SOL
                                altfel
                                    k←k+1
                                    iniţializare s_k
                Sfârşit Dacă
            altfel k←k – 1
        Sfârşit Dacă
Reia
Ieşire
```

```
funcţia VALID(k)
i : natural
valid ← adevărat
Pentru i = 1, k – 1 execută
    Dacă ¬ condiţie(s_i ,s_k)
        atunci valid←fals
    Sfârşit Dacă
Reia
Ieşire
```

```
funcţia SCRIE_SOL
i : natural
Pentru i =1, lung execută
    Scrie s_i
Reia
Ieşire
```

Identificarea elementelor definitorii ale metodei în problemele date în exemplele 1–3 şi în exerciţiile 1–5 şi 10, 11

1. *Defilarea.* Aşezarea celor trei melodii pe casetă cu generalizare la **n** melodii în **n** locuri pe casetă.

Tip soluţie	Lungime	Mulţimile $\mathbf{v_i}$	Condiţii interne şi de continuare pentru valoarea candidată pe poziţia **k**	Nr. soluţii posibile/valide (tip generare)
vector	fixă – **n** locuri	aceeaşi pentru toate componentele: V={1,2,…,n}	– valoarea să nu fi fost aleasă pentru o altă componentă anterioară – să existe valori nealese încă	***posibile***: n^n $V_1 \times V_1 \times \ldots \times V_1$ (de n ori) ***valide***: n ! ***generare***: permutări

2. Aşezarea melodiilor pe casetă cu restricţia ca melodia **y** să fie aşezată doar dacă melodia **x** a fost aşezată într-o etapă precedentă.

Tip soluţie	Lungime	Mulţimile $\mathbf{v_i}$	Condiţii interne şi de continuare pentru valoarea candidată pe poziţia **k**	Nr. soluţii posibile/valide (tip generare)
vector	fixă – n locuri	aceeaşi pentru toate componentele: V={1,2,…,n}	– valoarea să nu fi fost aleasă pentru o altă componentă anterioară – să existe valori nealese încă – dacă este la rând melodia y atunci melodia x să se găsească într-una dintre poziţiile de la 1 la k-1	***posibile***: n^n $V_1 \times V_1 \times \ldots \times V_1$ (de n ori) ***valide***: n-cele care au x după y ***generare***: permutări cu condiţie

3. Formarea drapelelor **tricolore** din culorile date respectând restricţiile de aşezare din enunţ.

Tip soluţie	Lungime	Mulţimile V_i	Condiţii interne şi de continuare pentru valoarea candidată pe poziţia **k**	Nr. soluţii posibile/valide (tip generare)
vector	fixă – 3 locuri	i=1, 2,…, 9, indice culoare V1={2,3,4} V2={1, 5} V3={6,7,8,9}	– valoarea să nu fi fost aleasă pentru o altă componentă anterioară – să fie aleasă din mulţimea Vi corespunzătoare	***posibile***: 9^3 {1,...,9} x {1,...,9} x {1,...,9} ***valide***: V1 x V2 x V3, adică 3 x 2 x 4=24 ***generare***: produs cartezian

1. Generarea şirurilor formate din **3 litere** alese dintre primele **2** ale alfabetului englez.

Tip soluţie	Lungime	Mulţimile V_i	Condiţii interne şi de continuare pentru valoarea candidată pe poziţia **k**	Nr. soluţii posibile/valide (tip generare)
vector	fixă – 3 locuri	aceeaşi pentru toate componentele: V={1,2}, adică literele A şi B	să existe valori nealese încă	***posibile***: 2^3 =V x V x V ***valide***: 2^3 ***generare***: produs cartezian

2. Generarea şirurilor formate din **3 litere distincte** alese dintre primele **5** ale alfabetului englez.

Tip soluţie	Lungime	Mulţimile V_i	Condiţii interne şi de continuare pentru valoarea candidată pe poziţia **k**	Nr. soluţii posibile/valide (tip generare)
vector	fixă – 3 locuri	aceeaşi pentru toate componentele: V={1,2,3,4,5}, adică { A,B,C,D,E}	– valoarea să nu fi fost aleasă pentru o altă componentă anterioară – să existe valori nealese încă	***posibile***: 5^3 V x V x V x V x V ***valide***: 60 ***generare***: aranjamente A_5^3

3. Generarea, în ordine crescătoare, a tuturor numerelor naturale de **3** cifre folosind cifrele numărului **3145**.

Tip soluţie	Lungime	Mulţimile V_i	Condiţii interne şi de continuare pentru valoarea candidată pe poziţia **k**	Nr. soluţii posibile/valide (tip generare)
vector	fixă – 3 locuri	aceeaşi: mulţimea cifrelor lui 3145, ordonate crescător. V={1,2,3,4} mulţimea locurilor, adică cifrele{1,3,4,5}	să existe valori nealese încă	***posibile***: 4^3 V x V x V ***valide***: 4^3 ***generare***: produs cartezian

4. Generarea tuturor numerelor naturale de **4** cifre folosind cifrele **1,2,3,4**, astfel încât oricare două cifre alăturate sunt fie ambele pare, fie ambele impare.

Tip soluţie	Lungime	Mulţimile V_i	Condiţii interne şi de continuare pentru valoarea candidată pe poziţia **k**	Nr. soluţii posibile/valide (tip generare)
vector	fixă – 4 locuri	aceeaşi: V={1,2,3,4}	– oricare două cifre alăturate sunt fie ambele pare, fie ambele impare – să existe valori nealese încă	***posibile***: 4^4 V x V x V x V ***valide***: $2^4+2^4=32$ ***generare***: produs cartezian: $\{2,4\}^4 + \{1,3\}^4$ [a]

5. Generarea tuturor şirurilor ***strict crescătoare*** de lungime **4** folosind **divizorii lui 36**.

Tip soluţie	Lungime	Mulţimile V_i	Condiţii interne şi de continuare pentru valoarea candidată pe poziţia **k**	Nr. soluţii posibile/valide (tip generare)
vector	fixă – 4 locuri	aceeaşi: V={1,2,3,4,5,6,7,8,9} indicii în mulţimea divizorilor {1,2,3,4,6,9,12,18,36}	– valoarea să nu fi fost aleasă pentru o altă componentă anterioară – să existe valori nealese încă – alegerea să se facă în ordinea *strict crescătoare* a divizorilor	***posibile***: 9^4 V x V x … x V (de 9 ori) ***valide***: C_9^4 = 126 ***generare***: combinări C_9^4

[a] Prin $\{2,4\}^4$ s-a notat pe scurt, din motive de spaţiu, produsul cartezian {2,4} x {2,4} x {2,4} x {2,4}

rezolvă

7. Generarea tuturor modalităţilor de scriere a numărului **12** ca sumă formată din numere naturale ***distincte, nenule***.

Tip soluţie	Lungime	Mulţimile $\mathbf{v_i}$	Condiţii interne şi de continuare pentru valoarea candidată pe poziţia **k**	Nr. soluţii posibile/valide (tip generare)
vector	variabilă – de maximum 4 locuri = ([12 /3])	aceeaşi: numerele V={1,2,3,4,5,6,7,8, 10,11}, care se pot regăsi ca termeni în sumele generate	– valoarea să nu fi fost aleasă pentru o altă componentă anterioară – să existe valori nealese încă – alegerea făcută, adăugată la suma alegerilor anterioare, să dea o valoare ≤ 12	***posibile***: $C_{11}^2+ C_{11}^3+...C_{11}^{11}$ ***valide***: 14 ***generare***: submulţimi ale lui V de minimum 2 elemente, care îndeplinesc condiţia de însumare

8. Generarea submulţimilor de numere dintre cele **8** date astfel ca suma elementelor selectate să fie **57**.

Tip soluţie	Lungime	Mulţimile $\mathbf{v_i}$	Condiţii interne şi de continuare pentru valoarea candidată pe poziţia **k**	Nr. soluţii posibile/valide (tip generare)
vector	variabilă – de maximum 8 locuri	aceeaşi: mulţimea indicilor V={1,2,3,4,5,6,7,8}, care desemnează ca termeni în sumele generate, elemente din lista: 12,61,22,57,10,4,23,30	– valoarea să nu fi fost aleasă pentru o altă componentă anterioară – să existe valori nealese încă – alegerea făcută, adăugată la suma alegerilor anterioare, să dea o valoare ≤ 57	***posibile***:$C_8^2 +C_8^3 +..C_8^8$ ***valide***: 3 ***generare***: submulţimi ale listei, de minimum 1 element, care îndeplinesc condiţia de însumare.

observă

O1. ***Metoda backtracking*** furnizează ***toate soluţiile de tip rezultat***, alegând, dintre toate soluţiile posibile, aşa cum s-a spus mai sus, numai pe acelea care respectă condiţiile interne. Dacă enunţul problemei nu ar sugera condiţii interne, atunci toate soluţiile posibile sunt şi soluţii rezultat. Aceasta revine la a considera că **VALID** întoarce întotdeauna valoarea adevărat, adică **VALID** apare ca o prelucrare de tip tautologie şi nu va mai fi necesară scrierea funcţiei în textul programului.

Dacă în aşezarea celor **n** melodii, exemplul 1, facem această presupunere, problema va rezolva, pentru n=6 de exemplu, simularea kilometrajului de pe bordul unei maşini, pornit de la valoarea 111.111, ceea ce înseamnă un alt enunţ.

O2. ***Mecanismul*** metodei poate fi figurat cu ajutorul unui ***arbore*** astfel:

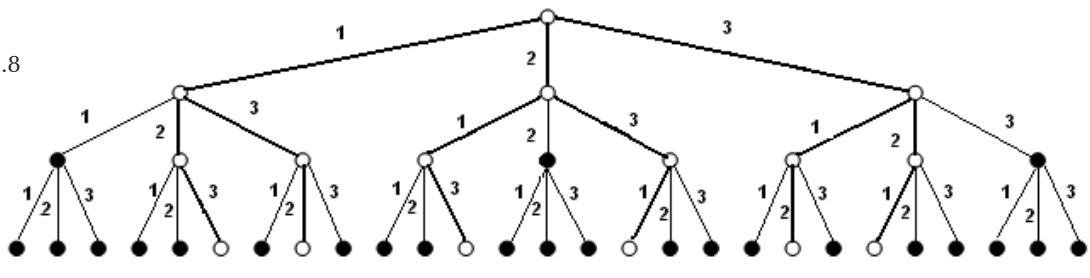
– Nivelul 1 conţine rădăcina procesului;

– Din orice nod de pe un nivel, **k,** pleacă atâtea muchii spre nivelul **k+1** câte valori conţine $\mathbf{V_k}$. Aceste muchii vor fi etichetate fiecare cu câte o valoare din $\mathbf{V_k}$.

– Pe ultimul nivel în arbore se găsesc nodurile terminale care încheie un lanţ-încercare (soluţie posibilă).

În figura 8.8 este prezentat arborele generat de aşezarea pe casetă a celor trei melodii din exemplul 1. Muchiile îngroşate pun în evidenţă drumurile către soluţiile de tip rezultat (soluţiile valide). Nodurile prin care trece un lanţ-soluţie sunt albe, iar nodurile negre figurează situaţia în care continuarea nu conduce la o soluţie validă.

Figura 8.8



Deplasările prin arbore ar arăta că la întâlnirea unui nod negru procesul nu mai poate înainta în subarborele acestuia pentru a închega o soluţie. Acesta este momentul în care căutarea face revenirea în nodul de nivel „tată" (de pe nivelul imediat inferior).

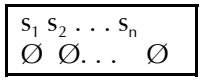


Figura 8.9

O3. ***Încheierea*** procesului de căutare a tuturor soluţiilor se realizează când se ajunge la *configuraţia finală*, adică în momentul în care ***s-au consumat mulţimile de valori*** pentru toate componentele şi se încearcă o întoarcere (deoarece toate valorile pentru s_1 au fost folosite) la o componentă din stânga lui s_1 (momentul k=0). Acest lucru se întâmplă deoarece mulţimile V_1, V_2, …,V_n sunt finite, iar prin modificările succesive de configuraţie nu este posibil să se ajungă de două ori la aceeaşi configuraţie. Situaţia poate fi reprezentată ca în figura 8.9.

O4. Prin modul de funcţionare, *metoda backtracking* completează componentele unei soluţii pe **principiul stivei**.

Tabloul-***soluţie*** poate fi considerat ca fiind un ***spaţiu alocat static unei stive*** în care *baza* se găseşte la nivelul de *stivă goală* (indicele -1), iar *vârful* este la *nivelul componentei curente*, ***k***.

Se alege şi se testează o valoare pentru componenta **k**, după ce au fost aşezate componentele de la **1** la **k-1**. Dacă nici o valoare din mulţimea de valori posibile pentru componenta k nu este validată, întoarcerea la componenta **k-1**, pentru alte încercări, echivalează cu descărcarea din stivă a elementului de ordin **k** şi deci coborârea vârfului la nivelul **k-1**.

Completarea stivei, astfel încât vârful să ajungă la nivelul **lung,** echivalează cu alcătuirea uneia dintre soluţiile de tip rezultat.

Trecerea la compunerea unei noi soluţii începe cu modificarea ultimei soluţii găsite prin încercarea altei valori pe poziţia **lung**. Dacă descărcarea stivei ajunge la baza acesteia, atunci nu mai sunt soluţii noi.

8.2.3. Probleme rezolvate

Pentru problemele de mai jos soluţia este liniară, componentele configurând un tablou unidimensional. Pe lângă exemple se propun teme în scopul fixării tipului de problemă exemplificat.

1) Produs cartezian. Dându-se ***n*** mulţimi diferite de elemente, notate ***A1, A2, …, An,*** se cere afişarea mulţimii produs cartezian al acestora, adică mulţimea ***A1*** × ***A2*** × … × ***An***.

Rezolvare. Fie **n**=2 şi A_1={a,b}, A_2={a,c,g}.

Atunci $A_1 \times A_2$ = {(a,a),(a,c),(a,g),(b,a), (b,c),(b,g)}, care are *card* $A_1 \times$ *card* A_2 elemente.

Pentru exemplul luat, numărul de elemente al produsului cartezian este $2 \times 3=6$.

După cum se ştie de la matematică, elementele produsului cartezian sunt perechi de valori alcătuite în ordinea: prima valoare este luată din prima mulţime, la rând, iar cea de-a doua din a doua mulţime. Rezultatul produsului cartezian constă în realizarea mulţimii tuturor perechilor posibile, conform ordinii impuse.

În caz general, un element al produsului cartezian pentru **n** mulţimi va avea forma unui tablou unidimensional de **n** elemente. Rezultă că *o soluţie construită prin metoda backtracking* este un astfel de *vector*. Pentru a simplifica proiectarea rezolvării prin metoda backtracking, se va lucra cu indicii elementelor în cadrul mulţimilor care intră în produsul cartezian. Astfel, componenta ***i*** din soluţie va lua valori de la ***1*** la $card_i$, unde $card_i$ este numărul de elemente ale mulţimii A_i. Definirea produsului cartezian conduce la concluzia că nu sunt condiţii interne pentru componente. Deci, metoda va genera toate soluţiile posibile ca şi soluţii rezultat. Este o situaţie în care ***VALID*** devine o tautologie.

În programul de mai jos este dată varianta în care, în produs, intră o singură mulţime, înmulţită cu ea însăşi de **m** ori. Mulţimea este citită în variabila vector **a** şi are **n** elemente de tip caracter, **n≤20**. Soluţiile sunt create în vectorul **s**. Evoluţia procesului pe stiva sistemului, pentru mulţimea ***A={1,2},*** este dată în figura 8.10.

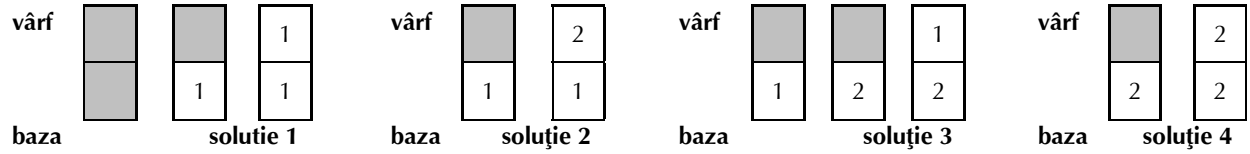


Figura 8.10

```
#include <iostream.h>
char a[20];
int  s[20], n,m;
unsigned nr;
void scr()
{int i;
 nr++;
 cout<<"Elementul: "<<nr<<' ';
 cout<<"( ";
 for(i=0;i<m;i++)
    cout<<a[s[i]]<<',';
 cout<<"\b)\n";
}
void cartezian()
{int k;
 k=0;s[k]=-1;
 while (k>-1)
  {
     if (s[k]<n-1)
       {s[k]++;//alege  alta valoare
          if (k==m-1) scr();
          else
                {k++;// urca pe stiva
                  s[k]=-1; }
        }
      else k-;
   }
}
void main()
{int i;
 cout<<"Dati numarul de elemente ";
 cin>>n;
 cout<<"Elementele: "<<endl;
 for (i=0;i<n;i++)
      {cout<<"elem["<<i+1<<"]=";
       cin>>a[i];
      }
cout<<"Dati numarul de factori ";cin>>m;
cout<<"Pt. afisarea solutiilor ";
cout<<"apasati o tasta\n";
cout<<"Multimea produs cartezian \n";
nr=0;
cartezian();
}
```

Din clasa de probleme produs cartezian fac parte exerciţiile 1, 3, 4, 6 şi 8 din paragraful 8.2.1

2) Se cere generarea tuturor şirurilor formate din cele 7+1 note ale gamei Do major (Do, re, mi, fa, sol, la, si, do) şi ilustrarea sonoră a fiecărei soluţii.

Rezolvare. Problema se reduce la construirea **permutărilor** a 8 elemente. Aceste elemente sunt alese din notele gamei Do major, la care se adaugă şi do de sus. Se va organiza un vector de şiruri, **a**, de caractere, care se va iniţializa cu numele notelor, şi un vector, **sunet**, de numere întregi, care reprezintă frecvenţa sunetelor acestor note redate de difuzorul calculatorului.

La etapa de scriere a unei soluţii, funcţia **scr()** va face atât afişarea listei de note din soluţia curentă, cât şi producerea sunetelor aferente acestei soluţii. Pentru operaţiile legate de redarea sunetelor s-a folosit biblioteca de funcţii de sistem a limbajului **C/C++**, **dos.h**.

Este prevăzută şi o întrerupere a execuţiei dacă aceasta devine plictisitoare, deoarece sunt ***8!*** soluţii de afişat şi cântat. Pentru întreruperea forţată s-a oferit utilizatorului posibilitatea de a apăsa tasta **S**. Oprirea forţată a procesului s-a făcut cu funcţia **exit(0)** din biblioteca **stdlib.h**. De asemenea, pentru a evita introducerea literei mici, pentru testul de apariţie a tastei **S** s-a folosit transformarea în literă mare cu funcţia **toupper(c)** din biblioteca **ctype.h**.

```
#include <iostream.h>
#include<stdlib.h>
#include<ctype.h>
#include<conio.h>
#include<dos.h>
//program gama
char a[8][4]={"Do","Re","Mi","Fa","Sol",
                        "La","Si","do"};
int sunet[8]={262,294,330,349,392,440,
                                  494,523};
int s[20],nr=0;
void scr()
{int i; char c;
nr++;
cout<<"\nLista notelor ";
cout<<"din solutia "<<nr<<"=\n";
for(i=0;i<8;i++)
 {cout<<a[s[i]]<<",";
  sound(sunet[s[i]]);delay(500);
  nosound();
  }
nosound();
cout<<"\b \nApasati o tasta, S pt.
                                  STOP";
c=getch();if(toupper(c)=='S')exit(0);
}
int valid(int k)
{int i;
 i=0;
while(i<k)
```

```
      {if(s[k]==s[i]) return 0;
       i++;
      }
return 1;
}
void gama()
{int k,f;
k=0;s[k]=-1;
while (k>-1)
  { f=0;
      while (s[k]<7 && !f)
       {s[k]++;
           f=valid(k);
       }
    if (f)
      if (k==7) scr();
        else
          {k++;
           s[k]=-1;
          }
    else      k–;
  }
}
void main()
{ int i,j,n,p;
clrscr();
cout<<"Lista solutiilor\n";
gama();
getch();
        }
```

3) *N* ture pe tabla de şah. Se consideră o „tablă de şah" având *n* linii şi *n* coloane. Se caută toate modalităţile de plasare a *n ture,* astfel încât să nu existe două ture care să se atace. Reamintim că regula după care se deplasează o tură la jocul de şah este în lungul liniei şi al coloanei pe care este aşezată.

Rezolvare. Este evident că pe fiecare linie şi coloană va fi plasată o singură tură. De exemplu, fie n=3 şi T simbolul pentru ture. Configuraţiile de amplasament vor fi în genul celor din figura 8.11.

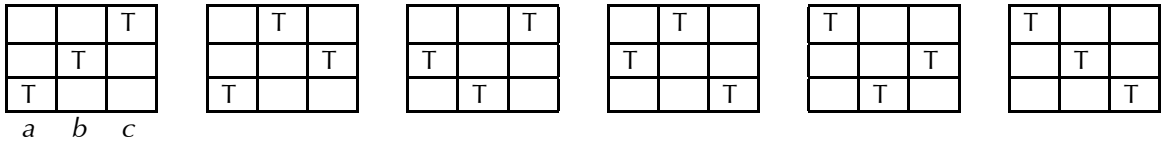
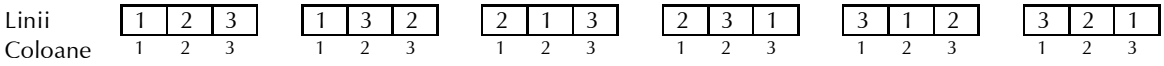


Figura 8.11

Se observă că are loc, de fapt, permutarea turelor pe locurile neutre din cele trei linii şi trei coloane. Dacă o tură este plasată pe câmpul a1, atunci a doua se poate plasa doar pe câmpul b2 sau b3. Odată plasată şi a doua tură, pentru a treia nu mai rămâne decât un loc valid.

S-ar părea că, pentru un *n* oarecare, este dificilă deplasarea pe tablă.

O codificare avantajoasă a datelor, însă, va simplifica foarte mult acest lucru. Astfel, atribuim *turelor numere de ordine de la 1 la n, corespunzătoare liniilor* pe care le vor ocupa. Considerăm *coloanele ca fiind numerotate şi ele de la 1 la n* şi desemnând indici într-un tablou unidimensional de lungime **n**. Elementele tabloului vor fi numai numere de la 1 la n şi vor corespunde numerelor turelor. În exemplul nostru, pentru **n=3**, tabloul va fi, pe rând, ocupat astfel:



Deci, un conţinut al tabloului, de exemplu de forma (2,3,1), va indica faptul că o tură se aşază în coloana 1 linia 2, altă tură, în coloana 2 linia 3 şi ultima, în coloana 3 linia 1. În acest mod se asigură şi condiţia ca oricare două ture să nu se atace: două sau mai multe ture nu pot fi aşezate pe aceeaşi coloană, deoarece un element de pe o poziţie din tablou nu poate lua decât o valoare la un moment dat, şi, de asemenea, nu se pot regăsi două sau mai multe ture pe aceeaşi linie, pentru că ar însemna ca două sau mai multe elemente din tablou să ia aceeaşi valoare, lucru interzis de permutări.

De aici la a configura rezolvarea backtracking nu mai este nici un pas, deoarece se vede o rezolvare de tip permutări ale primelor *n* numere naturale. Soluţia curentă va fi conţinutul vectorului la acel moment. La scrierea unei soluţii se va da utilizatorului posibilitatea de a citi pe înţelesul său, deci ceva în genul celor din interpretarea setului (2,3,1) făcută mai sus.

Rezolvaţi problema pentru n=4 ture, pe caiet, apoi realizaţi programul pentru cazul general şi comparaţi soluţiile date de program cu cele din caiet.

4) *N* regine. Se consideră o „tablă de şah" având *n* linii şi *n* coloane. Se caută toate modalităţile de plasare a *n regine* astfel încât să nu existe două regine care să se atace. Reamintim că regula de deplasare a unei regine la jocul de şah este pe linia, pe coloana şi pe diagonalele pe care este aşezată.

Rezolvare. Din exemplul precedent se preia modul de codificare a datelor. Astfel, în loc de ture, aici vor fi numerotate de la **1** la **n** reginele de plasat pe tablă. Soluţia va căpăta structura de tablou unidimensional, fiecare element primind valori din aceeaşi mulţime de valori, numere de la **1** la **n**, care reprezintă numere de regine. Apare o condiţie internă în plus faţă de problema precedentă, şi anume, „atacul pe diagonale". În figura 8.12 sunt puse în evidenţă două situaţii de conflict pe diagonale.

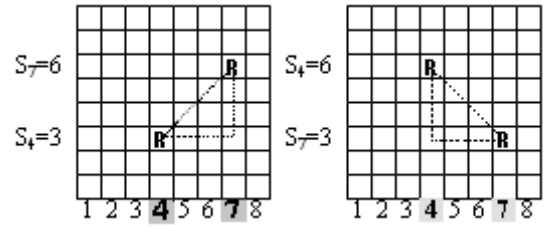


Figura 8.12

Plasarea pe o diagonală revine, geometric, la crearea unui triunghi dreptunghic isoscel, în care segmentul din diagonală este ipotenuza.

Rezultă coordonatele ipotenuzei pentru prima situaţie ca fiind perechile (4,s[4]) şi (7, s[7]), adică (4,3) şi (7,6). Pentru a doua situaţie: (7, s[7]) şi (4, s[4]), adică (7,2) şi (4,6). În general, pentru două regine plasate pe locurile **i** şi **k** din tabloul coloanelor, coordonatele ipotenuzei formate vor fi **(i,s[i])** şi **(k,s[k])**. Astfel, egalitatea catetelor revine la relaţia: **|i-k|=|s[i]-s[k]|**, oricare i,k = 1, 2, …,n.

În acest mod, **VALID** va putea determina şi condiţia de aşezare privind diagonala.

```
#include <iostream.h>
#include <math.h>
#include<conio.h>
//program n_regine
int s[20],i,j,k,n;
unsigned nr=0;
void scr()
{int i;
nr++;
cout<<"Solutia "<<nr<<'\n';
for (i=0;i<n;i++)
  {cout<<"R"<<s[i]+1<<" linia ";
   cout<<i+1<<" col "<<s[i]+1;
   cout<<'\n'; }
cout<<"\nApasati o tasta";
getch();
}
int valid(int k)
{int i,f;
f=1;
i=0;
while (i<k &&f)
  {if(s[k]==s[i]||abs(s[k]-s[i])==
                          abs(k-i))
       f=0;
   i++;
   }
return f;
}
void regine(int k)
{int f;
 s[k]=-1;
 while (k>-1)
  {f=0;
   while(s[k]<n-1 && !f)
     {s[k]++;
     f=valid(k);
     }
   if (f)if (k==n-1) scr();
      else{k++;s[k]=-1;}
   else k--;
  }
}
void  main()
{cout<<"Dati dimensiunea tablei ";
 cin>>n;
 regine(0);
}
```

Rezolvaţi problema pentru n=4, pe caiet, şi apoi comparaţi soluţiile cu cele afişate de program.

5) Numere neconsecutive. Se consideră un şir de ***n*** numere întregi. Să se afişeze toate variantele de aşezare a acestor numere, astfel încât în nicio aşezare să nu existe numere consecutive.

Rezolvare. Problema cere de fapt obţinerea tuturor permutărilor numerelor date, în care să nu apară numere consecutive – ***permutări cu condiţie***. Asemănător exemplului 2, al aşezării celor *n* melodii – astfel ca melodia **x** să fie înaintea melodiei **y** (tot un caz de permutări cu condiţie) –, soluţia va fi de tip tablou unidimensional

cu **n** componente. Pentru toate componentele, valorile se aleg din aceeaşi mulţime, {**1,2,…,n**}, *mulţimea indicilor numerelor* din şirul dat.

Funcţia **VALID** va prelua sarcinile de verificare a condiţiilor interne şi de continuare sub forma următoarelor teste:

– valoarea aleasă pentru poziţia curentă, **k**, nu trebuie să mai fi fost aleasă pe vreuna din poziţiile de la **1** la **k-1** (condiţie de continuare);

– diferenţa absolută dintre valoarea curentă şi precedenta să fie >1, adică

$|s_k - s_{k-1}| > 1$. (condiţie internă).

De exemplu, pentru n=4, se pot construi soluţiile: 2, 4, 1, 3 şi 3, 1, 4, 2

În programul de mai jos s-a utilizat notaţia de indici începând cu 0, astfel că indicele componentei curente, **k**, ia valori de la 0 la **n-1**. Din acest motiv, valoarea iniţială în s_k este **-1**.

```
// s-a tinut cont ca indicii incep cu 0
#include <iostream.h>
#include <math.h>
int a[20],s[20];
short i,n;unsigned nr;
void scr()
{short i;
nr++;
cout<<"Solutia: "<<nr<<'\n';
for(i=0;i<n;i++)
    cout<<a[s[i]]<<' ';
cout<<"\nApasati o tasta";
 cin.get();
}

int valid(short k)
{short i,f;
 f=1;i=0;
 while(i<k && f)
  {if(s[i]==s[k]) f=0;
   i++;}
 if (f && k>0)
  f=!(abs(a[s[k]]-a[s[k-1]])==1) && f;
return f;
}
void necons()
{short k,f;
 k=0;s[k]=-1; //initializari
 while (k>-1)
  {f=0;
   while(s[k]<n-1 && !f)
     {s[k]++;//alege valoarea urmatoare
      f=valid(k);
     }
   if (f) if(k==n-1) scr();
     else {k++;s[k]=-1;}//urca pe stiva
   else k--;
 }
}
void main()
{cout<<"Dati numarul total de numere ";
 cin>>n;
cout<<"numerele: \n";
for (i=0;i<n;i++)
  {cout<<"nr["<<i+1<<"]=";
   cin>>a[i];
  }
nr=0;
cout<<"Lista numerelor neconsecutive ";
necons();
}
```

Teme din clasa de probleme permutări. Determinaţi modul de structurare a datelor din fiecare enunţ, lungimea soluţiei, conţinutul mulţimilor de valori pentru fiecare componentă a soluţiei, condiţiile interne şi de continuare şi realizaţi programele.

a) Un grup de *n* persoane (n≤10) sunt aşezate pe un rând de scaune în ordinea sosirii. După un timp, între oricare două persoane vecine s-au ivit conflicte. Se cer toate modurile posibile de reaşezare a persoanelor, astfel încât, între oricare două persoane aflate la început în conflict, să stea una sau cel mult două persoane.

b) Generalizarea problemei de mai sus pentru diferenţa=*v*, unde *v* este citit.

c) Regele Arthur. La curtea regelui Arthur s-au adunat 2*n* cavaleri şi fiecare are printre cei prezenţi cel mult n-1 invidioşi. Arătaţi că Merlin, consilierul regelui, poate să-i aşeze pe cavaleri la o masă rotundă, astfel încât niciunul dintre ei să nu stea alături de vreun invidios al său. Găsiţi toate soluţiile posibile pentru un *n* dat.

d) Fie *n* muncitori care pot lucra pe oricare dintre *N* maşini avute la dispoziţie. Să se alcătuiască toate repartizările posibile ale acestor muncitori pe maşinile date.

6) Porturi. Într-o mare închisă sunt *n* porturi, maximum 20. Să se stabilească toate voiajele prin *p* porturi (p≤ n).

Rezolvare. Problema se bazează pe ***generarea de* aranjamente** a **n** obiecte luate câte **p**.

Soluţia, **s**, este sub forma unui tablou unidimensional şi reprezintă un voiaj (lista porturilor alese).

Lungimea soluţiei este **p** – numărul de porturi cerut pentru un voiaj.

Mulţimea valorilor pentru fiecare componentă este aceeaşi, mulţimea indicilor cu care s-au înregistrat porturile în ordinea citirii datelor de intrare.

Condiţiile interne şi de continuare sunt: un port ales să nu mai fi fost ales în construirea voiajului;

```
#include <iostream.h>
#include<conio.h>
//program voiaje
char a[20][10];//numele porturilor
int s[20],nr=0; //solutia = un voiaj

void scr(int p)
{int i;
nr++;
cout<<"\nLista porturilor ";
cout<<"din voiajul "<<nr<<'\n';
for(i=0;i<p;i++)
 cout<<a[s[i]]<<",";
cout<<"\b\b \nApasati o tasta";
getch();
}
int valid(int k)
{int i;
 i=0;
 while(i<k)
      {if(s[k]==s[i]) return 0;
       i++;
      }
return 1;
}
void voiaj(int n, int p)
{int k,f;
k=0;s[k]=-1;
while (k>-1)
  { f=0;
        while (s[k]<n-1 && !f)
         {s[k]++;
             f=valid(k);
         }
       if (f)
         if (k==p-1) scr(p);
             else
              {k++;
               s[k]=-1;
               }
       else   k–;
 }
}
void main()
{ int i,j,n,p;
clrscr();
cout<<"Numarul de porturi:\n";
cin>>n;
cout<<"Numele fiecarui port\n";
for (i=0;i<n;i++)
  {cout<<"port["<<i+1<<"]=";
  cin>>a[i];
  }
cout<<"Numar porturi in voiaj ";
cin>>p;
cout<<"Lista solutiilor\n";
voiaj(n,p);
        }
```

7) Turnul de cuburi. Se consideră *n* cuburi, iar pentru fiecare cub se cunosc: latura şi culoarea. Să se alcătuiască toate variantele de aşezare a acestor cuburi pentru a forma un turn stabil de *h* cuburi, astfel încât între oricare două cuburi succesive să difere culoarea.

Rezolvare. Problema se bazează pe generarea de aranjamente a **n** obiecte luate câte **h**. În plus, apar condiţiile de stabilitate şi culoare – **aranjamente cu condiţie**.

– Soluţia, **s**, este sub forma unui tablou unidimensional.

– Lungimea soluţiei este **h** – numărul de cuburi din turn.

– Mulţimea valorilor pentru fiecare componentă este aceeaşi, şi anume, mulţimea indicilor cu care s-au înregistrat cuburile în ordinea citirii datelor de intrare.

– Condiţiile de continuare sunt: un cub ales să nu mai fi fost aşezat în turn;

– Condiţiile interne: latura cubului curent ce trebuie aşezat să fie mai mică sau egală cu cea a cubului peste care se aşază (stabilitatea turnului) şi culorile să fie diferite.

Pentru definirea unui cub trebuie organizată o structură de date de tip înregistrare cu două câmpuri: latura, **lat**, şi culoarea, **cul**.

```
#include <iostream.h>
#include <string.h>
#include<conio.h>
struct cub
   {unsigned lat;char cul[10];};
struct cub a[20];int s[20],n,h,nr;
void scr()
{int i;
nr++;
cout<<"Lista cuburilor din turnul ";
```

```
cout<<nr<<"=\n";
for(i=0;i<h;i++)
{cout<<"Nr.cub "<<s[i]+1<<' ';
cout<<" latura "<<a[s[i]].lat;
cout<<" culoare "<<a[s[i]].cul<<'\n';}
cout<<"apasati o tasta"; getch();
}
int valid(int k)
{int i,f;f=1;i=0;
while (i<k && f)
 {if(s[i]==s[k]) f=0; i++;}
if (f && k>0)
 if(a[s[k]].lat>a[s[k-1]].lat ||
   strcmp(a[s[k]].cul,a[s[k-1]].cul)==0)
     f=0;
  else f=1;
return f;}
void turn()
{int k=0,f;s[k]=-1;
while (k>-1)
 {f=0;
  while(s[k]<n-1 && !f)
    {s[k]++;f=valid(k);}
  if (f) if (k==h-1) scr();
       else {k++;s[k]=-1;}
  else k--;
 }
}
void main()
{int i;
cout<<"Dati nr. total de cuburi ";
cin>>n;
cout<<"Cuburile ";
for(i=0;i<n;i++)
 {cout<<"latura ";
  cin>>a[i].lat;
  cout<<"culoarea ";
  cin>>a[i].cul;
  }
cout<<"Inaltimea turnului ";
cin>>h;
nr=0;
turn();
        }
```

8) Voiaje cu cost impus. Într-o mare închisă sunt *n* porturi, maximum 20. Staţionarea într-un port se taxează printr-un cost asociat. Să se stabilească toate voiajele prin *p* porturi (p≤ n) care nu depăşesc un *cost total* de staţionare dat.

Rezolvare. Rezolvarea problemei cere, ca şi problema anterioară, generarea aranjamentelor sub o condiţie dată. Sunt A_n^p soluţii dintre care vor fi alese cele care vor respecta şi limita de cost, **limcost**. Deosebirea constă însă în locul de abordare a acestei condiţii.

Există tentaţia ca, după închegarea unei soluţii, înainte de scriere, să se facă testul dacă se depăşeşte costul total de către costul, **cost**, acumulat pentru o soluţie (voiaj).

Dacă privim limita de cost ca fiind o ***condiţie de continuare***, o putem trece în **VALID**. În acest mod, dacă s-a descoperit că pentru poziţia curentă deja limita de cost este depăşită, nu mai are sens să se continue cu o nouă poziţie în compunerea soluţiei. Acţionând astfel, trebuie ca orice adăugare la **cost**, **c[s[k]]**, să fie anulată (scăzută) în momentul în care se renunţă la acea alegere. De asemenea, când s-a închegat o soluţie, pentru a trece la o altă soluţie, trebuie scăzut **c[s[k]]**, care a fost adăugat la ultima alegere.

```
#include <iostream.h>
#include<conio.h>
//program voiaje_ieftine
char a[20][10];
int s[20], nr=0;
float c[20],cost,limcost;
void scr(int p)
{int i;
nr++;
cout<<"Lista porturilor ";
cout<<"din voiajul "<<nr<<"=\n";
for(i=0;i<p;i++)
   cout<<a[s[i]]<<' ';
cout<<"\nCostul=";
cout.width(5);cout.precision(2);
cout<<cost;
cout<<"\nApasati o tasta\n";
getch(); }
```

```
int valid(int k)
{int i,f;
 f=1;i=0;
 while(i<k && f)
   {if(s[k]==s[i]) f=0;
    i++;
   }
 if (!f || cost+c[s[k]]>limcost)
      f=0;
     else cost+=c[s[k]];
return f;
}
void voiaj(int n,int p)
{int k,f;
k=0;s[k]=-1;cost=0;
while (k>-1)
  { f=0;
    while (s[k]<n-1 && !f)
```

```
      {s[k]++;
       f=valid(k);
      }
    if (f)
      if (k==p-1){scr();cost-=c[s[k]];}
       else
        {k++;s[k]=-1;}
    else
     {k–; cost-=c[s[k]]; }
   }
}

void main()
{int n,p,i; clrscr();
cout<<"Numarul de porturi:";
cin>>n;
cout<<"Numele fiecarui port ";
cout<<"si costul stationarii \n";
for (i=0;i<n;i++)
  {cout<<"port["<<i+1<<"]=";
  cin>>a[i];
  cout<<"cost["<<i+1<<"]=";
  cin>>c[i];
}
cout<<"Numar porturi in voiaj ";
cin>>p;
cout<<"Limita maxima a costului ";
cin>>limcost;
voiaj(n,p); getch();
}
```

Teme din clasa de probleme aranjamente. Determinaţi modul de structurare a datelor din fiecare enunţ, lungimea soluţiei, conţinutul mulţimilor de valori pentru fiecare componentă a soluţiei, condiţiile interne şi de continuare şi realizaţi programele.

a) Fie *n* muncitori care pot lucra pe oricare dintre *M* maşini avute la dispoziţie, M <n. Să se alcătuiască toate repartizările posibile ale acestor muncitori pe maşinile date.

b) Să se afişeze toate funcţiile injective $f : A \rightarrow B$, unde *A* şi *B* sunt două mulţimi cu *m*, respectiv *n* elemente din mulţimea literelor mici ale alfabetului englez.

c) Modificaţi programul permutărilor notelor muzicale, astfel încât să se cânte doar câte 5 note.

9) Comitete. Dintr-un număr de *n* elevi ai unei clase, maximum 30, se cere alcătuirea tuturor variantelor de comitete de *p* elevi, cu *p*<*n*, care ar putea să reprezinte clasa.

Rezolvare. Se observă că la bază stă problema generării tuturor **combinărilor** de **n** elemente luate câte **p**. Soluţia se regăseşte şi aici sub formă de tablou unidimensional.

Lungimea soluţiei este **p**, numărul elevilor dintr-un comitet format.

Toate componentele îşi iau valori din aceeaşi mulţime, şi anume, mulţimea indicilor primiţi de numele elevilor la citirea acestora într-un tablou de nume, **a**.

Condiţiile de continuare vor consta în alegerea unei valori care nu a mai fost aleasă.

Nu trebuie repetată generarea unei soluţii care conţine aceleaşi valori, dar altfel aşezate. Referindu-ne la definiţia combinărilor, aceasta revine la eliminarea permutărilor din cadrul aranjamentelor de **n** elemente luate câte **p**.

Fie n=4 şi p=3. Dintre aranjamentele generate vor fi excluse permutările tăiate cu o linie orizontală: (1,2,3), (1,2,4),~~(1,3,2)~~, (1,3,4), ~~(1,4,2)~~, (~~1,4,3~~), (2,3,4), (~~2,1,3~~), (~~2,3,1~~), (~~2,1,4~~), (~~2,4,1~~), (~~2,4,3~~), (~~3,1,2~~), (~~3,1,4~~), (~~3,2,1~~), (~~3,2,4~~), (~~3,4,1~~), (~~3,4,2~~), (~~4,1,2~~), (~~4,1,3~~), (~~4,2,1~~), (~~4,2,3~~), (~~4,3,1~~), (~~4,3,2~~).

Se observă că pentru a genera combinările este suficientă condiţia ca *valoarea curentă să fie mai mare strict decât valoarea de pe poziţia anterioară*. Astfel **VALID** va trebui să testeze acest lucru prin care se acoperă şi situaţia de a nu se repeta o valoare deja aleasă.

Mai mult, se poate ca valoarea de iniţializare a unei noi componente să fie făcută cu valoarea anterioarei componente, adică **s[k]=s[k-1]**, ceea ce asigură, la momentul **s[k]++**, alegerile în mod crescător; se poate renunţa astfel la funcţia **VALID**.

```
#include <iostream.h>
#include<conio.h>
char a[30][10];
int  s[30];
unsigned nr=0;
void scr(int m)
{int i;
nr++;
cout<<"Solutia: "<<nr<<' ';
for(i=0;i<=m;i++)
  cout<<a[s[i]]<<' ';
cout<<'\n';
}
void comitet(int n, int m)
{int k;
 k=0;s[k]=-1;
```

```
 while (k>-1)
  {if (s[k]<n-1)
     {s[k]++;
     if (f) if (k==m) scr();
      else
      {k++;  s[k]=s[k-1]; }
      }
     else k–;
  }
}
void main()
{int i,m,n; clrscr();
cout<<"Dati numarul de copii ";
cin>>n;
cout<<"numele copiilor:"<<endl;
for (i=0;i<n;i++)
   {cout<<"copil["<<i+1<<"]=";
    cin>>a[i];
}
cout<<"Dati numarul de membri ";
cin>>m;  m–;
cout<<"Pt. afisarea solutiilor ";
cout<<"apasati o tasta\n";
cout<<"Lista comitetelor \n";
comitet(n,m); getch();
}
```

10) Descompunerea unui număr natural. Dându-se un număr natural, *n*, să se genereze toate descompunerile lui distincte în sumă de numere naturale crescătoare (exerciţiul 10, paragraful 8.2.1).

Rezolvare. Problema descompunerii unui număr natural în sumă de numere naturale, fără a repeta respectivele descompuneri, termenii în ordine crescătoare, revine la a genera ***acele combinări*** care, la un moment dat, fiind însumate, dau numărul **n**. Nu toate variantele (soluţiile valide) au aceeaşi lungime. Deci, pe lângă scopul principal al realizării efective a descompunerilor, exemplul mai urmăreşte şi o situaţie nouă până acum: ***soluţia nu are aceeaşi lungime pentru toate cazurile***. Soluţia are tot configuraţia de tablou unidimensional ale cărui componente sunt termenii unei descompuneri. Calculul lungimii este făcut indirect, şi anume, chiar de către variabila **k**, în care se va găsi numărul de componente completate în tabloul soluţiei în momentul în care suma elementelor acestui tablou a ajuns egală cu numărul dat, **n**. În acest moment s-a închegat o soluţie, şi ea poate fi scrisă. Se regăseşte ideea de la problema 5, varianta optimizată, numai că aici este obligatoriu să se procedeze aşa: nu se poate genera o soluţie şi apoi să se testeze dacă suma elementelor ei formează numărul **n**, ci, progresiv, o componentă nou aleasă îşi aduce contribuţia la formarea numărului. Pentru a nu repeta descompunerile, o nouă componentă se iniţializează cu valoarea componentei anterioare.

```
#include<iostream.h>
#include<conio.h>
int i,j;
unsigned s[20],nr=0,p,n;
void scr(unsigned k)
{int i; nr++;
if (k>0)
 {cout<<"Lista descompunerilor ";
  cout<<"din solutia "<<nr<<'\n';
  for (i=0;i<=k;i++)
       cout<<s[i]<<' ';
  cout<<'\n'<<"Apasati o tasta ";
getch();}
}
int valid(int k)
{ if (p+s[k]>n) return 0;
return 1; }
```

```
void desc()
{int k,f;
k=0;s[k]=0;p=0;
while (k>-1)
  {s[k]++; f=valid(k);
   if (f)
    { p+=s[k];
      if (p==n) {scr(k);p-=s[k];}
        else
        {k++; s[k]=s[k-1]; }
     }
      else {k–;p-=s[k];}
  }}
void main()
{ cout<<"Numarul de descompus:";
  cin>>n;
        desc();getch();}
```

Teme din clasa de probleme combinări

a) Fie un grup de **p** persoane dintre care **f** femei. Dintre acestea, trebuie ales un comitet de **c** persoane, **c<n**, dintre care **nf** să fie femei, **nf<f**. Să se alcătuiască o listă cu toate comitetele realizabile.

(*Indicaţie*: se vor nota cu indici de la **1** la **f** numele femeilor şi de la **f+1** la **p**, numele bărbaţilor, astfel încât mulţimea numelor celor **p** persoane să se folosească, de fapt, ca două submulţimi; apoi, în construcţia soluţiei, se vor folosi primele **nf** locuri pentru a alege indici din mulţimea femeilor, iar locurile de la **nf+1** la **c** vor fi ocupate de indici din mulţimea bărbaţilor; pentru program, aceşti indici se vor calcula începând de la 0)

b) Să se descompună un număr natural, **N**, ca sumă de **p** numere naturale, în toate modurile distincte posibile, **p<N** (*Indicaţie*: se adaptează validarea din programul dat mai sus astfel încât, în momentul în care s-a realizat suma **N** să se fi ales şi **p** termeni).

c) Să se realizeze o listă cu toate ***submulţimile unei mulţimi*** de *n* elemente (*Indicaţie*: se vor genera mulţimile de lungimi progresive, începând cu 1 şi până la lungimea **n-1**; pentru aceasta, pe rând, numărul de elemente din mulţime, **m**, va creşte gestionat prin **for** în funcţia **main**).

1. Se doreşte generarea tuturor permutărilor mulţimii primelor *n* numere naturale, nenule. Dacă n=4, care dintre următoarele mulţimi nu constituie soluţii ?
a) {1,3,2}; **b)**{1,2,3,5}; **c)**{2,3,4,1}; **d)**{4,2,3,2}.

2. Fie mulţimile A={1,2,3}, B={2} şi C={1,2}. Care dintre următoarele grupuri nu fac parte din produsul cartezian A x B x C ?
a) (1,3,2); **b)** (1,1,1); **c)** (2,2,1); **d)** (3,2,2); **e)** (2,2,2) .

3. Se doreşte generarea tuturor permutărilor mulţimii primelor *n* numere naturale, nenule. Dacă n=4, care este înălţimea stivei corespunzătoare vectorului soluţie şi de câte ori este atinsă aceasta în procesul generării?
a) 3, 6; **b)** 16, 4 ; **c)** 4, 16; **d)** 4, 24.

4. Se doreşte generarea tuturor grupelor formate din p elemente alese dintre primelor *n* numere naturale, nenule. Dacă n=4 şi p=2, care este înălţimea stivei corespunzătoare vectorului soluţie şi de câte ori este atinsă aceasta în procesul generării?

5. Se citesc *n* şi *p*, numere naturale. Se cere generarea tuturor submulţimilor mulţimii primelor *n* numere naturale nenule, formate din *p* elemente. Care dintre formulările de mai jos este corectă pentru această problemă. Scrieţi toate soluţiile pentru formularea aleasă.
a) generarea combinărilor de *n* elemente luate câte *p*; **b)** generarea permutărilor de *p* elemente combinate câte *n*; **c)** generarea produsului cartezian de ordinul *p*; **d)** generarea aranjamentelor de n elemente luate câte *p*.

6. Se cere generarea tuturor submulţimilor distincte ale mulţimii primelor n numere naturale nenule, formate din *p* elemente. Precizaţi numărul soluţiilor pentru *n*=4 şi *p*=2:
a) 8; **b)** 6 ; **c)** 12; **d)** 24.

7. Se cere aplicarea metodei backtracking pentru generarea tuturor submulţimilor distincte ale mulţimii primelor *n* numere naturale nenule, formate din *p* elemente. Precizaţi care dintre următoarele mulţimi sunt soluţii pentru *n*=4 şi *p*=3, conforme mecanismului de generare:
a) {1,3,2}; **b)**{1,2,4}; **c)**{3,4,1}; **d)**{1,2,3}; **e)**{1,2,3}; **f)** {2,3,4} .

8. Se cere generarea tuturor submulţimilor distincte ale mulţimii primelor *n* numere naturale nenule, formate din *p* elemente. Pentru evitarea generării repetate a aceleiaşi soluţii, condiţia de continuare în poziţia *k* este alegerea ca valoare iniţială *a*:
a) valorii -1; **b)** *n-p*; **c)** valorii componentei *k*-1; **d)** valorii componentei *k*-1 micşorată cu o unitate.

9. Funcţia de validare a unei alegeri pentru generarea permutărilor de *n* obiecte faţă de generarea aranjamentelor de *n* obiecte luate câte *p*.
a) este inexistentă; **b)** este aceeaşi; **c)** la aranjamente se testează până la valoarea p; **d)** nu se pot compara.

10. Se cere aplicarea metodei backtracking pentru generarea tuturor submulţimilor distincte ale mulţimii primelor *n* numere naturale nenule, formate din *p* elemente distincte. Care dintre următoarele afirmaţii este corectă:
a) funcţia de validare nu este necesară; **b)** funcţia de validare este vidă, dar se scrie pentru a respecta schema metodei ; **c)** după generarea unei soluţii se aplică o funcţie de ordonare a elementelor ei; **d)** în soluţie, elementele sunt generate crescător.

11. Pentru determinarea tuturor descompunerilor numărului *n*=4 ca sumă de numere naturale vectorul soluţie este de tip stivă. Care dintre afirmaţiile următoare sunt adevărate?
a) vârful stivei ia valori între 2 şi 4; **b)** o valoare este aleasă în soluţie dacă, adăugată la suma anterioarelor determină, dă un număr mai mare decât *n*; **c)** o soluţie se obţine când suma este egală cu 4; **d)** în soluţie elementele sunt generate crescător.

12. Pentru generarea produsului cartezian, la alcătuirea soluţiei:
a) fiecare element depinde de elementele anterioare; **b)** fiecare element depinde de elementele următoare; **c)** fiecare element se alege din mulţimea corespunzătoare poziţiei lui în soluţie; **d)** orice element ales nu depinde de celelalte alese până atunci.

13. Asociaţi fiecărui caz din coloana A enunţul potrivit din coloana B pentru determinarea tuturor soluţiilor privind:

A	B
1.Modalităţile de aşezare a c copii în b bănci	1. Generarea produsului cartezian
2. Anagramele unui cuvânt	2. Generarea permutărilor
3. Rondurile de noapte pentru n paznici la n locuri de pază	3. Generarea combinărilor
4. Codurile ce se pot forma din n litere	4. Generarea aranjamentelor
5. Delegaţiile alese dintre n sportivi la un concurs internaţional	5. Generarea submulţimilor unei mulţimi
6. Cifrurile de 6 poziţii formate din 4 cifre	
7. Distribuirea a n cărţi la doi prieteni dintre care primului i se dau m cărţi	
8. Turnurile de n cuburi formate din n cuburi egale de culori diferite	

14. Pentru care dintre problemele enumerate mai jos se recomandă utilizarea metodei backtracking?
a) determinarea reuniunii a *n* mulţimi;
b) determinarea tuturor submulţimilor unei mulţimi;
c) generarea a *n* numere aleatoare într-un interval dat;
d) determinarea tuturor divizorilor unui număr;
e) determinarea tuturor variantelor ce se pot obţine la aruncarea a două zaruri;
f) determinarea tuturor elementelor şirului lui Fibonacci, care pot fi memorate în variabilă de tip long;
g) generarea tuturor numerelor din intervalul (1,1000) care au proprietatea de număr prim.

8.2.4. Forma recursivă a metodei backtracking

Mecanismul de lucru al metodei constă în repetarea unor încercări de plasare a valorilor pentru cele **lung** componente ale soluţiei, revenind la contextul poziţiilor anterioare în caz de insucces pe poziţia curentă. Trecând aceste repetiţii în sarcina recursivităţii, contextul fiecărui nivel al soluţiei poate fi înregistrat pe stiva sistemului.

Sintetizând esenţa procesului în exprimarea în pseudocod, schema generală a metodei apare acum ca în figura 8.13.

Subprogramul **BKT** se va autoapela pentru o nouă poziţie, **k**, iar eşecul alegerilor pentru acea poziţie va conduce la întoarcerea pe stiva sistemului la contextul apelului anterior, ceea ce echivalează cu trecerea pe nivelul **k-1** din soluţie.

```
funcţia BKT (k)

Dacă k=lung + 1
   atunci SCRIE_SOL
   altfel
      Cât timp (∃) valoare în V_k execută
         s_k ← valoare
         Dacă VALID(k) atunci BKT(k+1)
         Sfârşit Dacă
      Reia
Sfârşit Dacă
INITializează s_k
Ieşire
```

```
funcţia VALID(k)
i : natural
valid ← adevărat
Pentru i = 1, k – 1 execută
   Dacă ⌉ condiţie (s_i ,s_k)
        atunci valid←fals
```

```
funcţia SCRIE_SOL
i : natural
Pentru i =1, lung execută
   Scrie s_i
Reia
```

Figura 8.13

Probleme rezolvate

Teme: Se vor realiza şi încerca programele corespunzătoare problemelor date spre exemplificare**.**

1) Vecinii necertăreţi. Enunţul problemei apare mai sus, în clasa de probleme de tip permutări cu condiţie.

Rezolvare. Programul este aproape identic cu forma iterativă.

Va fi prezentat aici numai subprogramul **vecini** care suferă transformarea recursivă respectivă.

```
void vecini(int k)
{
 if(k==n)scr()
  else
    while(s[k]<n-1)
      {s[k]++;
       if(valid(k)) vecini(k+1);
      }
     s[k]=-1;
}
```

2) Voiaje pe o mare închisă. Problema a fost enunţată în seria de exemple de rezolvări iterative.

Rezolvare. Programul este aproape identic cu forma iterativă.

Va fi prezentat aici numai subprogramul *voiaj* care suferă transformarea recursivă respectivă.

```
void voiaj(int k)
{ if (k==p)  scr();
 else
   while (s[k]<n-1)
     {s[k]++;
      if (valid(k)) voiaj(k+1);
     }
s[k]=-1;}
```

3) N regine pe o tablă „de şah”.

Rezolvare. Programul este aproape identic cu forma iterativă. Va fi prezentat aici numai subprogramul **regine** care suferă transformarea recursivă respectivă.

```
void regine(int k)
{ if (k==n) scr();
   else
   while (s[k]<n-1)
      {s[k]++;
       if (valid(k)) regine(k+1);
      }
s[k]=-1;}
```

4) Colorarea hărţilor. Se consideră harta unei regiuni de pe glob ocupată de n ţări. Se doreşte determinarea tuturor soluţiilor de colorare a acestor ţări, utilizând m culori, $m<n$, astfel ca oricare două ţări vecine să fie colorate diferit.

Rezolvare. Problema revine la a realiza soluţii prin generarea de aranjamente de tipul A_n^m, cu condiţia să nu existe nicio soluţie care să conţină elemente vecine identice. Se va lucra şi aici cu indicii culorilor din vectorul denumirilor culorilor, **cul**.

Ce apare important, însă, este modul în care se codifică vecinătăţile ţărilor. Se observă că modelul cel mai potrivit este un *graf neorientat*, în care fiecare vârf este o ţară şi fiecare muchie **[i,j]** reprezintă vecinătatea ţării **i** cu ţara **j**. Astfel, utilizatorul va comunica numărul de ţări, **n**, şi apoi perechile de ţări vecine (muchiile). Introducerea datelor se opreşte în momentul în care utilizatorul tastează combinaţia de sfârşit intrare din tastatură, **CTRL** şi **Z**.

```
#include <iostream.h>
int a[10][10]={0},s[10];
char cul[5][10];
int i,j,m,n;
int valid(int k)
{ int i;
 i=0;
 while (i< k)
  {if (s[k]==s[i] && a[i][k]==1)
                    return 0;
   i++;
   }
return 1;
}
```

```
void scr()
{int i;
 cout<<'\n';
 for (i=0;i<n;i++)
   {cout<<"Tara "<<i+1;
   cout<<"culoare "<<cul[s[i]]<<'\n';}
 cout<<"Apasati o tasta";
 cin.get();
}
void colorare(int k)
{
 if(k==n) scr();
  else
   while (s[k]<m-1)
```

```
     {s[k]++;
     if (valid(k)) colorare(k+1);
     }
s[k]=-1;
}
void main()
{cout<<"numar de tari:";
 cin>>n;
 cout<<"perechi de tari vecine";
 cout<<" la sf. CTRL si Z\n";
 while  (cin.good())
 { do {cin>>i>>j;
  }while(!(i<=n&&j<=n&&i>=0&&j>=0));
 a[i-1][j-1]=a[j-1][i-1]=1;
 }
 cin.clear();
 cout<<"Dati numarul de culori ";
 cin>>m;
 cout<<"dati culorile: ";
 for (i=0;i<m;i++)
  cin>>cul[i];
cin.clear();
s[0]=-1;
         colorare(0);}
```

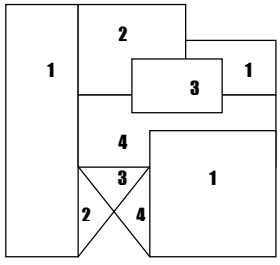


Figura 8.14

Matematic, se demonstrează că sunt suficiente 4 culori pentru a rezolva această problemă în orice condiţii de vecinătate.

De exemplu, fie harta stilizată a unei regiuni cu 9 ţări, aşa cum este prezentată în desenul din figura 8.14. Pe desen nu sunt figurate numele ţărilor. Sunt marcate, prin numere de la 1 la 4, culorile pe care le pot primi aceste 9 ţări, într-una dintre soluţiile valide ale problemei.

Atribuiţi numere de ordine ţărilor, apoi atribuiţi nume şi ţărilor şi culorilor.
Scrieţi lista valorilor pe care le va tasta utilizatorul pentru a rula programul pe acest exemplu.
Completaţi matricea de adiacenţă asociată grafului reprezentat de reţeaua ţărilor.
Stabiliţi, prin rularea programului, ce realizează comenzile `cin.good()` şi `cin.clear()`.

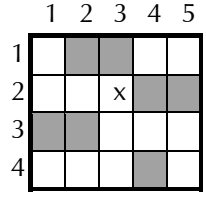


Figura 8.15

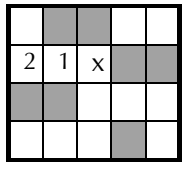


Figura 8.16

5) Labirint. Se consideră un labirint codificat într-un tablou bidimensional cu elemente din mulţimea {0,1}. Valoarea 1 semnifică drum liber, iar zerourile – ziduri. Se cer toate soluţiile de ieşire din labirint, pornind de la o poziţie *(i,j)* dată.

Rezolvare. Problema de faţă are o soluţie bidimensională, şi anume, matricea labirintului în care se vor pune în evidenţă *numai* elementele prin care se trece din poziţia de plecare către ieşire. Acest gen de soluţii dau rezolvării caracteristica de ***backtracking în plan***.

Deplasarea prin labirint este o deplasare de tip geometric în tabloul bidimensional. Astfel, pornind de la un element **loc(i,j)** specificat de utilizator, funcţia **încerc** realizează deplasările pe cele **4** direcţii – stânga, dreapta, sus, jos. În cadrul funcţiei, deplasările sunt calculate în variabilele **dx** – pe linie şi **dy** – pe coloană. Pentru a gestiona deplasările, se vor utiliza tablourile unidimensionale ale creşterilor relative pe direcţia de mers, aşa cum s-a întâlnit la problema obstacolelor în faţa nebunului pe tabla de şah sau la algoritmul de **FILL**.

De exemplu, în figura 8.15 plecarea se va face din elementul de coordonate (2,3). O soluţie în doi paşi poate fi cea din figura 8.16. Elementele „zid" s-au figurat prin culoarea gri. Soluţiile, **t**, vor fi generate recursiv şi aici, iar drumul către ieşire, dat de o soluţie, va apărea marcat în labirint prin numerele de ordine ale nivelurilor din soluţie (ale componentelor soluţiei).

În programul de mai jos labirintul este generat aleatoriu şi se face verificarea perechii **(i,j)** pentru a nu se referi un element marginal.

```
#include <iostream.h>
#include <stdlib.h>
#include<conio.h>
//program labirint
typedef int vector[4];
const vector a={-1,0,1,0};
      vector b={0,1,0,-1};
int loc[10][20],t[10][20],i,j,n,m;
int nr;
void scr()
{int i,j;
nr++;
```

```
cout<<"Solutia "<<nr<<'\n';
for (i=0;i<m;i++)
 {for (j=0;j<n;j++)
   cout<<' '<<t[i][j]<<' ';
 cout<<'\n';
 }
cout<<"Apasati tasta pt. ";
cout<<"o noua solutie\n";
getch();
}
void incerc(int i,int x,int y)
{int dx,dy,k;
for (k=0;k<4;k++)
 {dx=x+a[k];dy=y+b[k];
 if(dx>=0 && dx<m && dy>=0 && dy<n)
  if(loc[dx][dy]==1 && !t[dx][dy])
   {t[dx][dy]=i;
   if(dx==0||dx==m-1||dy==0||dy==n-1)
     scr();
    else incerc(i+1,dx,dy);
   t[dx][dy]=0;
   }
  }
}
```

```
void main()
{cout<<"dimens. loc: m si n";
 cin>>m>>n;randomize();
 for(i=0;i<m;i++)
   for(j=0;j<n;j++)
    {t[i][j]=0;
     loc[i][j]=random(2);
    }
 nr=0;
//afisarea matricei create
 for(i=0;i<m;i++)
   {for(j=0;j<n;j++)
     cout<<' '<<loc[i][j]<<' ';
     cout<<'\n';
   }
 do
  {cout<<"pozitia din labirint ";
  cin>>i>>j;i--;j--;t[i][j]=1;
 }while(!(i>0&&i<m-1 && j>0
   && j<n-1 && loc[i][j]==1));
 incerc(2,i,j);
 if(!nr)
   cout<<"Nu exista solutii";
}
```

Realizaţi, pe caiet, toate soluţiile de ieşire din labirint pentru exemplul dat mai sus.

6) Săritura calului. Dându-se o „tablă de şah" de dimensiune *n* şi un *cal*, să se afişeze toate variantele de trecere a calului prin toate câmpurile tabelei o singură dată. Reamintim, calul are o deplasare sub formă de *L*, atacând maximum *8* câmpuri în jurul său.

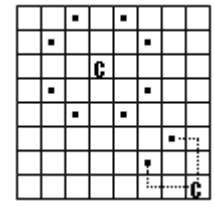


Figura 8.17

Rezolvare. Rezolvarea nu diferă ca principiu de precedenta. Aici însă, tablourile *deplasărilor relative*, **a** şi **b** de tip **rel**, vor cuprinde câte **8** valori, pentru cele **8** direcţii ale săriturii, calculate sub forma de **L**, aşa cum este desenat în figura 8.17, unde punctele reprezintă câmpurile atacate de calul **C**. Numărul acestor câmpuri este minimum 2 şi maximum 8.

Momentul realizării unei soluţii este remarcat prin testul între numărul componentei curente, **i** şi numărul câmpurilor de pe tablă, **np** care este valoarea $\mathbf{n^2}$.

O soluţie se obţine prin completarea, câmp cu câmp, a elementelor 0 ale matricei **tabla**. Un element nul din **tabla** primeşte numărul de ordine al pasului calului.

De exemplu, pentru dimensiunea 5 x 5 a tablei „de şah", primele două soluţii sunt date în figura 8.18.

```
dimens. tabla 5
Solutia 1
 1  20  17  12  3
 16  11  2  7  18
 21  24  19  4  13
 10  15  6  23  8
 25  22  9  14  5
Apasati o tasta pt. o noua solutie
Solutia 2
 1  18  23  12  3
 16  11  2  7  22
 19  24  17  4  13
 10  15  6  21  8
 25  20  9  14  5
Apasati o tasta pt. o noua solutie
```

Figura 8.18

Urmăriţi traseul calului pe rezolvarea din figura 8.18.
Încercaţi, pe hârtie o rezolvare pentru o tablă de 4 x 4. Notaţi ce observaţi şi comparaţi cu ceea ce afişează programul de mai jos.

```
#include <iostream.h>
#include<conio.h
//program calut
typedef int rel[8];
const rel a={-2,-1,1,2,2,1,-1,-2};
      rel b={1,2,2,1,-1,-2,-2,-1};
int tabla[20][20],n,i,j;
unsigned np,nr;
void scr()
{int i,j;
nr++;
cout<<"Solutia "<<nr<<'\n';
for (i=0;i<n;i++)
  {for(j=0;j<n;j++)
     if (tabla[i][j])
      {cout<<' ';
      cout<<tabla[i][j]<<' ';}
      else cout<<' ';
   cout<<'\n';
   }
cout<<"Apasati o tasta pt.";
cout<<" o noua solutie \n";
getch();
}
```

```
void incerc(int i,int x,int y)
{int dx,dy,k;
 for (k=0;k<8;k++)
  {dx=x+a[k];dy=y+b[k];
  if(dx<n &&dx>=0 && dy <n&&dy>=0)
   if (!tabla[dx][dy])
     {tabla[dx][dy]=i;
      if (i==np) scr();
      else incerc(i+1,dx,dy);
      tabla[dx][dy]=0;
     }
   }
}
void main()
{clrscr(); cout<<"dimens. tabla";
 cin>>n;
 np=n*n;
 for(i=0;i<n;i++)
  for(j=0;j<n;j++)
    tabla[i][j]=0;
 tabla[0][0]=1;nr=0;
 incerc(2,0,0);
 if (!nr)
   cout<<"Nu exista solutii";
}
```

1. Dacă se utilizează metoda backtracking pentru a genera anagramele cuvântului **caise**, ce combinaţii trebuie eliminate din lista de mai jos, astfel încât cele rămase să reprezinte o succesiune corectă ?
   a) ecias; **b)** ecisa; **c)** ecais; **d)** ecasi;**e)** ecsai; **f)** ecsia.
2. O urnă conţine patru bile negre şi opt bile albe. Se extrag cinci bile. Alegeţi din lista de mai jos răspunsul corect privind numărul de soluţii, astfel încât la extragere să apară două bile negre şi trei albe şi metoda de generare.
   a) 120, prin permutări; **b)** 792, prin combinări; **c)** 336, prin combinări; **d)** 95040, prin aranjamente.
3. La o pensiune meniul mesei de prânz este format din trei feluri de mâncare. Pentru servire, se pregătesc zilnic două sortimente pentru felul întâi, patru pentru felul al doilea şi patru pentru felul al treilea. Alegeţi din lista de mai jos răspunsul corect privind numărul de soluţii pentru obţinerea unor meniuri diferite şi metoda de generare.
   a) 120, prin combinări; **b)** 32, prin produs cartezian; **c)** 1024, prin submulţimi; **d)** 720, prin aranjamente.
4. Se generează toate numerele naturale care se pot forma cu cifrele 0, 1, 2, 3, fiecare cifră putând fi folosită cel mult o dată. Să se determine numărul precedent şi cel următor secvenţei de numere generate consecutiv: 1203, 1230, 1302, 1320.
   a) 1320 şi 2130; **b)** 1032 şi 2013; **c)** 1032 şi 2103; **d)** 1023 şi 2013.
5. La generarea submulţimilor mulţimii {1,2,3} prin metoda backtracking o parte din lista obţinută, în ordinea generării, este: {2}, {3}, {1,3}. Determinaţi dacă această listă este corectă şi care sunt submulţimile precedentă, următoare şi/sau lipsă din lista dată.
6. Se consideră mulţimea {1,2,3,4,5}. Se generează toate secvenţele aşezării acestor elemente, astfel încât în mijloc să fie cel mai mic sau cel mai mare element, iar celelalte să formeze, prin succesiunea valorilor, un aspect de "vale" sau, respectiv, "deal". Care este următoarea soluţie care se generează în lista de soluţii: {1,4,5,3,2}, {2,3,5,4,1}, {2,4,5,3,1}, {3,2,1,4,5}?
7. Se cere determinarea tuturor modalităţilor de planificare în zile diferite, într-o săptămână, a patru probe de concurs de gimnastică. Problema este echivalentă cu:
   a) generarea combinărilor de 7 obiecte luate câte 4; **b)** generarea aranjamentelor de 7 obiecte luate câte 4; **c)** generarea submulţimilor de 4 elemente; **d)** permutărilor de 4 elemente.

8. Pentru afişarea tuturor numerelor naturale formate din cel puţin 2 şi cel mult 8 cifre nenule distincte, o modalitate eficientă de rezolvare presupune:
a) aplicarea unei formule matematice; **b)** aplicarea metodei backtracking; **c)** parcurgerea numerelor din intervalul $[10,10^8-1]$; **d)** parcurgerea numerelor din intervalul[12,98765432].

9. Pentru afişarea numărului de perechi de numere naturale, (x,y), din intervalul [a,b], cu proprietatea că x<y, o modalitate eficientă de rezolvare presupune:
a) aplicarea unei formule matematice; **b)** aplicarea metodei backtracking; **c)** parcurgerea numerelor din intervalul [a,b] o singură dată; **d)** parcurgerea repetată a numerelor din intervalul[a,b].

10. Se consideră generarea, în ordine strict crescătoare, a tuturor numerelor formate din n cifre distincte, care în poziţiile pare au cifre impare. Poziţiile se numără de la stânga la dreapta, începând cu 1. Pentru n=5, al doilea număr generat este:
a) 21354; **b)** 21435; **c)** 12345; **d)** 13254.

Probleme propuse

1. Se citesc elementele unui vector de maximum 20 de numere întregi. Se doreşte afişarea tuturor modalităţilor de aranjare a valorilor din vector, astfel încât să nu existe două valori negative alăturate.

2. Să se genereze toate numerele de trei cifre distincte a căror sumă este pară, utilizând două variante de programare.

3. Să se afişeze toate funcţiile surjective $f : A \longrightarrow B$, unde A şi B sunt două mulţimi cu m, respectiv, n elemente din mulţimea literelor mici ale alfabetului latin.

4. Se consideră n ţări pe o hartă. Se cer toate variantele de colorare a hărţii, utilizând numai 4 culori, astfel încât două ţări vecine să fie colorate diferit.

5. În jurul unei mese rotunde trebuie aşezate n familii formate fiecare din soţ şi soţie. Să se construiască şi afişeze toate modalităţile de aşezare, astfel încât să nu existe doi soţi vecini şi fiecare bărbat să fie încadrat de două femei, iar fiecare femeie să fie încadrată de doi bărbaţi (*Indicaţie*: se vor aşeza pe poziţii impare bărbaţi şi pe cele pare femei, iar numerotarea persoanelor se va face de la 1 la n, pentru bărbaţi, şi de la n+1 la 2n, pentru femeii corespunzător înrudirii).

6. Să se afişeze toate numerele formate din p cifre nenule care au proprietatea că adunate cu inversul lor în scriere dau un pătrat perfect. Valoarea p este citită şi nu este mai mare de 9. Exemplu: p=2, 29+92=121.

7. Să se afişeze toate anagramele distincte formate pentru un cuvânt de maximum 5 litere şi minimum 3 litere, citit de la tastatură.

8. Să se genereze toate matricele pătratice binare de ordinul n, citit, care au proprietatea că atât fiecare linie, cât şi fiecare coloană conţin exact un element egal cu 1. Soluţiile generate se vor înregistra într-un fişier text.

9. O sesiune de examene ţine m zile consecutive. Trebuie programate n examene, n<[m/2], astfel încât să nu se susţină două examene în aceeaşi zi şi nici în zile consecutive.

10. * Se cere determinarea tuturor soluţiilor naturale ale ecuaţiei $a_1x_1+a_2x_2+\ldots+a_nx_n=b$. Valorile n, b şi coeficienţii se citesc din fişierul **`ecuatie.in`**, iar soluţiile se scriu în fişierul **`ecuatie.out`**. Valoarea n nu depăşeşte 25 (*Indicaţie*: se vor încerca divizorii naturali ai termenului liber).

11. * Se consideră o listă formată din maximum 20 de cuvinte. Să se formeze din aceasta cea mai lungă înşiruire de cuvinte în care fiecare cuvânt să înceapă cu litera cu care se termină predecesorul său, în afară de primul cuvânt care poate începe cu orice literă.

12. * Să se genereze cel puţin un şir de lungime dată, *n*, format din literele *x*, *y* şi *z*, astfel încât să nu existe două subşiruri alăturate identice. De exemplu, *xzxyxz* este un şir acceptat, dar *xzxyxy* sau *xzyxzy* sunt respinse.

ANEXE

1. Funcţii uzuale pentru calcule matematice

Antet	Semnificaţie

`int` **`abs`**`(int x)`
`long` **`labs`**`(long x)`
`double` **`fabs`**`(double x)`
	valoarea absolută a unui număr întreg
	valoarea absolută a unui număr întreg format lung
	valoarea absolută a unui număr real

`double` **`asin`**`(double x)`	arcsin x, întoarce valoarea $\in[-\pi/2, \pi/2]$
`double` **`acos`**`(double x)`	arccos x, întoarce valoarea $\in[0, \pi]$
`double` **`atan`**`(double x)`	arctg x, întoarce valoarea $\in(-\pi/2, \pi/2)$
`double` **`atan2`**`(double x, double y)`	arctg y/x, întoarce valoare $\in[-\pi,\pi]$
`double` **`ceil`**`(double x)`	rotunjire superioară: întoarce cel mai mic întreg mai mare sau egal cu x (ex. ceil(-1.2)= -1 ; ceil(1.2)=2)
`double` **`cos`**`(double x)`	cos x, cu x dat în radiani
`double` **`exp`**`(double x)`	e^x
`double` **`floor`**`(double x)`	rotunjire inferioară: întoarce cel mai mare întreg mai mic sau egal cu x. (ex. floor(-1.2)= -2; floor(1.2)=1)
`double` **`fmod`**`(double x, double y)`	calculează x modulo y
`double` **`log`**`(double x)`	ln x, x>0
`double` **`log10`**`(double x)`	log x, x>0
`double` **`modf`**`(double x, double *in)`	separă pentru x partea fracţionară, pe care întoarce ca rezultat de partea întreagă pe care o depune la adresa in (utilizarea conţinutului adresei se marchează prin operatorul * de referire adresă.)
`double` **`poly`**`(double x, int n, double coef [] )`	evaluează un polinom de grad n în punctul x. Coeficienţii sunt daţi în coef, crescător după rang
`double` **`pow`**`(double x, double y)`	x^y, pt. x<0 y trebuie să fie întreg.
`double` **`pow10`**`( int p)`	10^p
`double` **`sin`**`(double x)`	sin x, x dat în radiani
`double` **`sqrt`**`(double x)`	rădăcina pătrată din x, x pozitiv.
`double` **`tan`**`(double x)`	tg x

2. Câteva funcţii de ***verificare*** şi ***prelucrare de caractere***

Fişierul header este **ctype.h**. Argumentul c din tabelul de mai jos este un caracter de prelucrat. Amintim ca fiind caractere albe: spaţiu, tab, CR, LF, tab vertical şi form-feed.

Antet	Semnificaţie
`int` **`isalnum`**`( int c)`	întoarce valoarea 1 pentru c literă sau cifră
`int` **`isalpha`**`( int c)`	întoarce valoarea 1 pentru c literă
`int` **`isdigit`**`( int c)`	întoarce valoarea 1 pentru c cifră în baza 10
`int` **`iscntrl`**`( int c)`	întoarce valoarea 1 pentru c caracter de control
`int` **`isascii`**`( int c)`	întoarce valoarea 1 pentru c un caracter ASCII
`int` **`isprint`**`( int c)`	întoarce valoarea 1 pentru c un caracter imprimabil
`int` **`isgraph`**`( int c)`	întoarce valoarea 1 pentru c imprimabil, fără a fi spaţiu
`int` **`islower`**`( int c)`	întoarce valoarea 1 pentru c literă mică
`int` **`isupper`**`( int c)`	întoarce valoarea 1 pentru c litera mare
`int` **`ispunct`**`( int c)`	întoarce valoarea 1 pentru c semn de punctuaţie
`int` **`isspace`**`( int c)`	întoarce valoarea 1 pentru c un caracter alb
`int` **`isxdigit`**`( int c)`	întoarce valoarea 1 pentru c cifră a bazei 16
`int` **`toascii`**`(int c)`	conversie caracter ASCII cu codul >127 în caracter cu codul între 0 şi 127
`int` **`tolower`**`(int c)`	conversie caracter literă mare în literă mică
`int` **`toupper`**`(int c)`	conversie caracter literă mică în literă mare

3. Câteva funcţii generale

Fişierul header este **stdlib.h**. S-au ales câteva funcţii de interes din această grupă.

Antet	Semnificaţie
`double atof(const char *s)`	ASCII to float; conversia unui şir de caractere, a cărui valoare nu se poate modifica, alocat la adresa s, într-o valoare reală pe care funcţia o returnează în dublă precizie.[1]
`int atoi(const char *s)`	ASCII to int; conversia unui şir de caractere, a cărui valoare nu se poate modifica, alocat la adresa s, într-o valoarea întreagă.
`void exit(int stare)`	terminarea cu succes (0) sau cu eroare (1) a programului
`char * itoa(int valoare, char *sir,int baza)` `char * ltoa(long valoare, char *sir,int baza)`	converteşte o valoare într-un şir de caractere
`int random(int nr)`	generarea aleatorie a unui număr întreg între 0 şi nr – 1.
`void randomize(void)`	alegerea aleatorie a unei valori iniţiale (pe baza valorii ceasului intern) a seriei de numere aleatoare pe care le produce random
`div_t div(int num,int denom)`	produce câtul şi restul împărţirii întregi a lui *num* la *denom*. Rezultatul este comunicat sub formă de structură predefinită de tipul div_t existent în bibliotecă sub definiţia: *typedef struct {long num long denom;} div_t;*

4. Operaţii asupra ***şirurilor de caractere***. Funcţiile din tabelul de mai jos au prototipul în **string.h**

Antet	Semnificaţie
`char * strcat(char *d, const char * s)`	alipeşte şirul de la adresa s şirului de la adresa d şi returnează adresa d
`char * strncat(char *d, const char * s,size_t maxlen)`	alipeşte maxlen caractere din şirul de la adresa s şirului de la adresa d şi returnează adresa d
`char * strchr(const char *s, int c)`	verifică existenţa caracterului c în şirul s şi întoarce adresa primei apariţii
`int strcmp(const char *s1, const char *s2)`	compară şirul de la adresa s1 cu şirul de la adresa s2, întorcând un rezultat <0 pt. s1<s2, 0 pt. egalitate şi >0 pt. s1>s2
`char * strcpy (char *d, const char *s)`	copiază şirul sursă, nemodificabil, de la adresa s în şirul destinaţie alocat la adresa d, a cărui adresă e returnată

[1]Reamintim că şirurile de caractere sunt structuri de tip tablou şi, deci, sunt alocate la adrese fixe, a căror referire se face chiar prin numele tabloului. Aici, parametrul s este exprimarea simbolică a adresei la care este alocat spaţiu pentru şirul de caractere, deci transmitere prin referinţă. Dar *const* semnifică faptul că şirul nu se poate modifica, cum s-ar putea în mod obişnuit, deoarece parametrului i s-a comunicat adresa.

Antet	Semnificaţie
`size_t` **`strlen`**`(const char *s)`	este determinată lungimea şirului de la adresa s, adică numărul de caractere existente în şir până la întâlnirea terminatorului de şir '\0'. tipul size_t definit în string.h este un tip echivalent cu **unsigned**
`char*  strlwr(char *s)`	transformă literele mari din s în litere mici
`char* strrev(char *s)`	schimbă şirul s în ordine inversă
`char* strstr(const char *s1, const char *s2)`	întoarce adresa la care şirul s2 se regăseşte în şirul s1
`char* strupr(char *s)`	transformă literele mici din s în litere mari

5. Operaţii cu consola (***controlul afişării în mod text***)

Fişierul header este **conio.h**.

Antet	Semnificaţie
`void` **`clrscr`**`(void)`	umplerea ferestrei curente cu spaţii în culoarea de fond
`int` **`getch`**`(void)`	citirea unui caracter fără afişarea lui pe ecran (citire fără ecou)
`int` **`getche`**`(void)`	citirea unui caracter cu ecou pe ecran
`void` **`gotoxy`**`(int x, int y)`	poziţionarea cursorului la coloana (abscisa) x şi linia (ordonata) y
`int` **`putch`**`(int c)`	afişează un caracter în culoarea de scriere şi cea de fond existente în acel moment
`void` **`textbackground`**`(int fond)`	alegerea culorii de fond
`void` **`textcolor`**`(int scriere)`	alegerea culorii de scriere
`int` **`wherex`**`(void)` `int` **`wherey`**`(void)`	aflarea coordonatelor cursorului
`void` **`window`**`(int x1, int y1, int x2, int y2)`	definirea unei ferestre de coordonate NV date de (x1,y1) şi SE date de (x2,y2). O fereastră definită este vizibilă în urma unei operaţii clrscr(); în fereastra definită originea este la caracterul de coordonate(1,1).

RĂSPUNSURI

CAPITOLUL 1
Exerciţii de adresare: 1. b; **2.** c; **3.** b; **4.** c; **5.** d. **Test 1: 1.** b; **2.** b, e, f, g; **3.** c; **4.** b; **5.** e; **6.** b, e; **7.** e; **8.** c;
Test 2: 1. c; **2.** c; **3.** c; **4.** a; **5.** c.

CAPITOLUL 2
Paragraful 2.2.: 1. c,d; **2.** c,d; **3.** a,c; **4.** c,d; **5.** d. **Paragraful 2.3.: 1.** e; **2.** un, un, un ex, un exemplu. **Paragraful 3.4.: 1.** bengal; **2.** 5; **3.** c) 2; **4.** c; **5.** 11B; **6.** CLASA11B; **7.** 54321; **8.** 12 02; **9.** eroare, comparare greşită a şirurilor; **10.** afişează adres de memorie la care începe "ma". **Pag. 18: 1.** prima; **2.** a doua; **3.** fals; **4.** a, b; **5.** a,d; **6.** aia; **7.** aia.

CAPITOLUL 3
Raspunsuri exerciţii înregistrări: 1. b,d; **2.** a; **3.** b; **4.** c; **5.** c.

CAPITOLUL 4
Test1: 1. c; **2.** b; **3.** b; **4.** c; **5.** c; **6.** d; **7.** c; **8.** while (nat[s].urm) nat[s].nr=nat[nat[s].urm].nr.
Exerciţii şi probleme propuse. Exerciţii. 1. d; **2.** b; **3.** b.

CAPITOLUL 5
Exerciţii (pag.)
1. d); **2.**

A:	a	b	c	d	e	f	g	h
B:	d	f	i	-	a	j	b	c

3. d); **4.** a) ; b); d); e); **5.** b); **6.** 9; **7.** 10; **9.** c); **10.** c); **12.** c); **13.** a) Nu; b) 12; c) Nu; **14.** b); **15.** c); **16.** b).
Exerciţii (pag.)
1. a).

0	1	1	1
1	0	1	0
1	1	0	1
1	0	1	0

2. a).

0	1	1	0
1	0	1	0
1	1	0	0
0	0	0	0

3. Se foloseşte matricea de adiacenţă. **4.** Vârfurile au gradele: 0, 2, 2, 0, 0; **5.** a); **6.** Indicaţie: **c**) În matricea de adiacenţă (care este simetrică) se însumează numărul de elemente nenule situate deasupra (dedesubtul) diagonalei principale. **7.** Se foloseşte matricea de adiacenţă. **8.** Se foloseşte matricea de adiacenţă.
Exerciţii (pag. 18)
1. b); **2.** c); **3.** b).; c); **4.** Nu; **5.** Se determină componentele conexe ale grafului, reţinându-se mulţimile corespunzătoare de vârfuri ce le compun. Fie acestea în număr de p: C_1, C_2, ... ,C_p. Se completează cele n componente ale unui vector V astfel încât componenta k a acestuia să indice componenta conexă din care căreia îi aparţine vârful x_k. Pentru ca graful să devină conex, se adaugă p-1 muchii ce vol lega un vârf ce aparţine unei componente conexe (de exemplu vârful x_1 din C_1) cu câte un vârf din celelalte p-1 componente conexe. **6.** a), b), c), d); **7.** b); **8. a)** Da; **b)** Da; **c)** Nu; d) Da; **e)** Da; **f)** Nu; **g)** Nu; **i)** Nu; **j)** Nu; **9. a)** Da; **b)** Da; **c)** Da; **d)** Nu; **e)** Nu. **10.** Vezi programul.
11.

a	b	c
c	a	b

12. Se verifică îndeplinirea condiţiilor teoremelor prezentate; **13. a)** Da; **b)** NuU; **c)** Da; **d)** Da, **e)** Nu; **f)** Nu; **g)** Da
14. Vezi programul; **15.** Vezi programul.

Exerciţii (pag. 30)
1. RSD: 1, 2, 4, 3, 5, 7, 8, 6; SRD: 4, 2, 1, 7, 5, 8, 3, 6; SDR: 4, 2, 7, 8, 5, 6, 3, 1. Noduri terminale: 4, 7, 8, 6.
2. RSD: 2, 1, 5, 6, 9, 3, 4, 8, 10, 7; SRD: 6, 5, 9, 1, 3, 2, 8, 10, 4, 7; SDR: 6, 9, 5, 3, 1, 10, 8, 7, 4, 2. Noduri terminale: 6, 9, 3, 10, 7.
3. Deoarece G=(X, U) este un arbore, rezultă că subgraful H indus de Y este fără cicluri. Pentru a arăta că este şi conex se consideră două vârfuri x şi y în Y. Lanţurile care le leagă sunt lanţuri elementare în G_1 respectiv G_2 şi deci şi în G. Dar într-un arbore există un lanţ unuc ce leagă două vârfuri (altfel s-ar forma cicluri), de unde rezultă că cele două lanţuri coincid. Dar cum aceste lanţuri sunt alcătuite din vârfuri cer aparţin ambelor mulţimi X_1 şi X_2, rezultă că s-a obţinut un lanţ cu vârfuri din Y ce leagă cele două vârfuri, x şi y, deci H este conex.
4. Se reprezintă arborele precizând subarborii săi (stâng şi drept) şi se determină adâncimea maximă ca fiind maximul dintre cele două adâncimi, mărit cu o unitate.

5. Se verifică dacă este îndeplinită una dintre condiţiile teoremei de caracterizare a arborilor.
6. Valoarea maximă memorată în nodurile sale se determină parcurgând arborele.
7. ***Diametrul*** unui arbore se determină ca fiind valoarea maximă conţinută de matricea drumurilor (construită folosind algoritmul Roy-Floyd).
8. Se generează un arbore binar completând cei doi subarbori ai săi, stâng şi drept, cu „ascendenţii" pe linie maternă respectiv paternă a fiecărei persoane.
9. Se verifică dacă este îndeplinită una dintre condiţiile teoremei de caracterizare a arborilor.
10. Parcurgere SDR:

3	2	4	5	1	8	7	10	6	9	11
I	N	T	E	L	I	G	E	N	T	A

11. 2. **12.** Se reface arborele, considerând că a fost parcurs în preordine. **13.** procedează conform algoritmului prezentat
Exerciţii (pag.)
1. 1; **2.**1, 2, 3, 5, 1 şi 1, 2, 6, 7, 3, 5, 1; **3.**3; **4.**Se parcurge matricea de incidenţă a digrafului, şi pentru fiecare nod k se determină: gradul interior ca fiind numărul elementelor egale cu –1 şi gradul exterior ca fiind numărul elementelor egale cu 1. **5.** Se asociază problemei un digraf cu atâtea noduri câte persoane sunt în mulţime, iar un arc (x, y) este determinat de relaţia „x cunoaşte pe y". Pentru fiecare persoană neataşată unui grup se va constitui grupul din care face parte. **6. R** a); b); c) 2^m; **7.**Se înmulţeşte matricea de adiacenţă cu -1; **8.** Considerând că drumul D nu este elementar, fie z primul nod ce se repetă în şirul de noduri ce îl descriu pe D. Suprimând din D secvenţa de noduri dintre prima şi ultima apariţie a lui z. Se aplică acelaşi procedeu pentru drumul obţinut, până la determinarea unui drum elementar. **9.**Fiecare arc măreşte cu câte o unitate gradul celor două noduri cu care este adiacent. Tinând seama că arcele sunt în număr de m, se obţine relaţia cerută.

CAPITOLUL 6

Test 1: 1. b, e, f, h; **2.** b, e; **3.** d; **4.** a; **5.** d; **6.** b; **7.** d; **8.** a, b, c, d.
Test 2: 1. două rânduri cu 15 steluţe între care scrie *un exemplu simplu*; **2.** 0 0 3; **3.** 2 3; **4.** 17 17; **5.** d; **6. c; 7.** c.

CAPITOLUL 7

Exerciţii. 1. 48 : f(4)⟶f(3)×2⟶f(2)×2×2⟶f(1)×2×2×2⟶f(0)×2×2×2×2⟶3×16=48. **2.** a; **3.** 11, 7 apeluri; **4.** f=13, 11 apeluri: f(5)⟶f(f(9))⟶f(f(f(13)))⟶f(f(f(f(17))))⟶f(f(f(15)))⟶ f(f(13))⟶f(f(f(17)))⟶f(f(15))⟶f(13)⟶ f(f(17))⟶f(15)⟶13; **5.** b) iv; **6.** 24; **7.** 36, 1275; m(8)⟶m(7)+8⟶m(6)+7+8⟶m(5)+6+7+8⟶…⟶m(0)+1+2+3+ +4+5+6+7+8=36, adică suma primelor 8 numere naturale. La fel, pentru 50, suma primelor 50 de numere naturale; **8.** 4 şi 260. Numerele pare se adună şi cele impare se scad. Rezultă 8:2=4 de diferenţe 1 şi, respectiv, 520:2=260 diferenţe de 1; **9.** Verifică dacă n are prefix pe a; **10.** a; **11.** 5; **12.** de câte 4 ori autoapel şi apoi revenire; **13.** 47 şi 2 apeluri; **14.** b; **15.** Afişarea divizorilor numărului k împreună cu ordinul de multiplicitate al fiecăruia; **16.** BCBGCBCBGGCBCBGCBCBGGG; **17.** c, b.

25.
```
#include<iostream.h>
#include<conio.h>
typedef unsigned us;
us cmmdc(us a,us b)
{       if(b) return cmmdc(b, a%b);
  return a;}
us cmmdc_n(us a,us n)
{ us b;
  if(n)
   {cout<<"Numarul urmator ";
   cin>>b;
   return cmmdc_n(cmmdc(a,b),n-1);
  }
  return a;
}
void main()
{us n,a;
 clrscr();
 cout<<"Dati numarul de valori ";
 cin>>n;
 cout<<«Dati prima valoare «;
 cin>>a;
 cout<<endl;
 cout<<«CMMDC=»<<cmmdc_n(a,n-1);
getch();
}
```

CAPITOLUL 8

Test: **1.** a, b, d; **2.** a, b; **3.** d; **5.** a; **6.** b; **7.** b, d, e, f; **8.** c; **9.** c; **10.** a; **11.** a, c; **12.** c, d; **13.** a1b4, a2b2, a3b2, a4b1, a5b5, a6b1, a7b3, a8b2; **14.** e, g. **Exerciţii**: **1.** c, d; **2.** c; **3.** b; **4.** d; **7.** b; **8.** b; **9.** a; **10.** b.