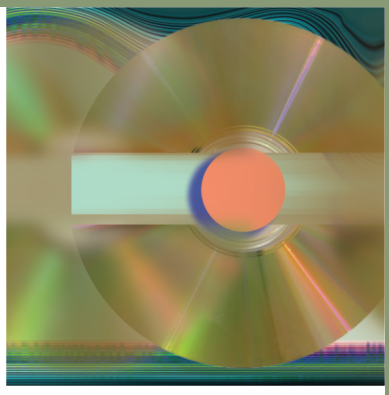
MINISTERUL EDUCAȚIEI ŞI CERCETĂRII



Informatică

Manual pentru clasa a XI-a

Mioara Gheorghe
(coordonator)
Monica Tătărâm
Corina Achinca
Constanța Năstase

CORINT EDUCAȚIONAL

MINISTERUL EDUCAȚIEI ŞI CERCETĂRII

Informatică

Manual pentru clasa a XI-a

Mioara Gheorghe
(coordonator)
Monica Tătărâm
Corina Achinca
Constanța Năstase

CORINT
EDUCAȚIONAL

Manualul a fost aprobat prin Ordinul ministrului Educației și Cercetării nr. 4446 din 19.06.2006 în urma evaluării calitative organizate de către Consiliul Național pentru Evaluarea și Difuzarea Manualelor și este realizat în conformitate cu programa analitică aprobată prin Ordin al ministrului Educației și Cercetării nr. 3252 din 13.02.2006.

MIOARA GHEORGHE – profesor gradul I la Colegiul Național „Mihai Viteazul" din București, metodist ISMB, membru în comisii la olimpiade și concursuri de informatică, autor de manuale și auxiliare școlare de informatică pentru gimnaziu și liceu, printre care: *Informatica pas cu pas*, apărută la Editura Corint, *Informatică,* manuale pentru clasa a IX-a și pentru clasa a X-a și *Tehnologia informației și a comunicării,* manuale pentru clasa a IX-a și a X-a.

Conf. Dr. **MONICA TĂTĂRÂM**, Catedra „Fundamentele informaticii", Facultatea de Matematică și Informatică, Universitatea București, cu preocupări în domeniile: Crearea aplicațiilor pentru soft educațional cu VBA și VB, Metodica predării informaticii, algoritmică și programare, Tehnici de proiectare, Analiza și proiectarea sistemelor. Autor a numeroase cărți, manuale și articole de specialitate în publicații de referință din țară și străinătate.

CORINA ACHINCA – profesor gradul II la Colegiul Național de Informatică „Tudor Vianu" din București. Autor de auxiliare școlare pentru liceu, membru în comisii la concursurile naționale de informatică.

CONSTANȚA NĂSTASE – profesor grad I la Colegiul Național „I.L. Caragiale" din București, coautor de manuale și auxiliare școlare pentru liceu și pentru examenul de bacalaureat, de ghiduri metodice pentru profesorii de informatică.

Referenți:
Prof. Dr. Ioan Tomescu, membru al Academiei Române, șeful Catedrei de informatică, Facultatea de Matematică și informatică, Universitatea București.
Prof. gradul I Doina Druță, metodist Inspectoratul Școlar al Municipiului București.

Contribuția autorilor:
Mioara Gheorghe – coordonator + capitolele 1, 4
Monica Tătărâm – capitolul 2, Anexe
Corina Achinca – capitolul 4
Constanța Năstase – capitolele 2.6, 3

Descrierea CIP a Bibliotecii Naționale a României
Informatică: manual pentru clasa a XI-a / Mioara Gheorghe (coord.), Monica Tătărâm, Corina Achinca, Constanța Năstase. – Ed. a 3-a. – București: Corint, 2008
Bibliogr.
ISBN 978-973-135-353-1

I. Gheorghe, Mioara (coord.)
II. Tătărâm, Monica
III. Achinca, Corina
IV. Năstase, Constanța

004(075.35)
004(075.35)

Redactare: **Sorin Petrescu**
Tehnoredactare și procesare computerizată: **Dana Diaconescu**
Coperta: **Valeria Moldovan**

Pentru comenzi și informații, contactați:
GRUPUL EDITORIAL CORINT
Departamentul de vânzări: Str. Mihai Eminescu, nr. 54A, sector 1, București, cod poștal 010517
Tel./Fax: 021.319.47.97; 021.319.48.20

Depozit: Calea Plevnei nr. 145, sector 6, București, cod poștal 060012, Tel.: 021.310.15.30
E-mail: vanzari@edituracorint.ro; Magazin virtual: www.edituracorint.ro

ISBN 978-973-135-353-1

Capitolul 1

STRUCTURI DE DATE

1. ORGANIZAREA DATELOR

STUDIU DE CAZ ***Compania Eficient***

Pentru reducerea cheltuielilor cu amenajarea spațiilor comerciale, reclamă și personal, compania Eficient selectează tineri distribuitori, absolvenți de liceu. Oferta companiei este foarte atrăgătoare; de aceea, numărul solicitanților depășește numărul de posturi oferite (n). Selecția candidaților se face în ordinea înregistrării scrisorii de intenție și a CV-ului. Dosarele candidaților sunt păstrate într-un fișet, unul peste altul, în ordinea angajării. Periodic, compania trimite un angajat la cursuri de formare; este trimis întotdeauna, ultimul angajat (fig. 1). Angajații sunt plătiți în funcție de numărul de produse distribuite (vândute) zilnic. Săptămânal, managerul companiei ține evidența pe zile a produselor distribuite de fiecare angajat. În orice moment, managerul poate determina angajatul cu cea mai bună activitate într-o zi sau ziua în care a fost distribuit cel mai mic număr de produse. La sfârșitul săptămânii, după ce aplică bonusuri sau penalizări, managerul centralizează aceste date într-un registru de evidență a vânzărilor realizate de către fiecare angajat.

Managerul companiei dorește să prelucreze cu calculatorul datele pentru selecția candidaților, evidența angajaților și evidența vânzărilor.

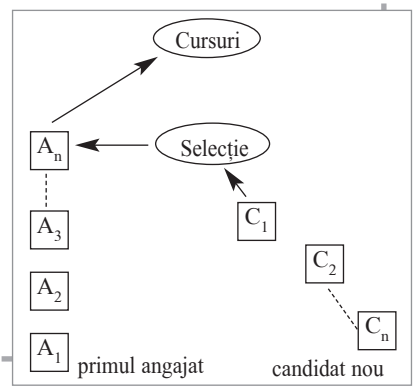


Figura 1

1.1. Analiza problemei

☑ datele despre candidați; din CV-ul fiecărui candidat vor fi reținute următoarele informații:
- numele,
- anul nașterii,
- media la examenul de bacalaureat;

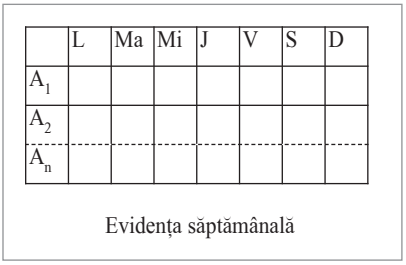


Figura 2

☑ selecția candidaților se face în ordinea înregistrării datelor personale;

☑ după examinarea unui candidat (selecție), datele acestuia nu mai sunt necesare; va intra în selecție candidatul următor;

☑ întotdeauna, datele unui nou candidat sunt așezate după datele ultimului candidat care așteaptă pentru selecție;

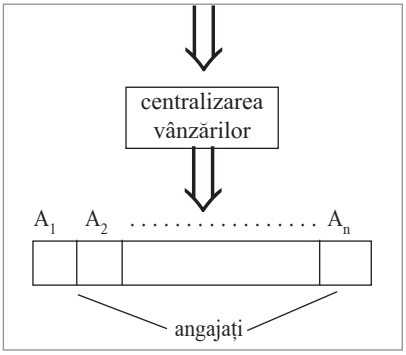


Figura 3

☑ datele angajaților sunt păstrate într-o ordine care să permită numirea rapidă a ultimului angajat în vederea trimiterii sale la cursuri;

☑ pentru evidența săptămânală a vânzărilor, se reține numărul de produse distribuite (vândute) zilnic de către fiecare angajat (fig. 2);

☑ pentru evidența centralizată a vânzărilor, totalul vânzărilor realizate într-o săptămână de fiecare angajat se adaugă la vânzările realizate de acel agajat până la momentul respectiv (fig. 3).

1.2. Soluția problemei

☑ problema propusă de managerul companiei Eficient necesită prelucrări matematice cu un grad mic de dificultate: centralizarea săptămânală, prin însumare, a valorilor reprezentând vânzările realizate de către fiecare angajat;

☑ datele specifice problemei trebuie organizate avantajos, astfel încât folosirea calculatorului să înlocuiască fișetele sau mapele în care sunt păstrate dosarele candidaților/angajaților și să respecte cerințele de așteptare sau prioritate specifice problemei.

1.3. Organizarea datelor

- Din analiza problemei urmărind *semnificația și complexitatea datelor*, rezultă două categorii de date:

☒ *date elementare*, spre exemplu: numărul de produse vândute de un angajat la un moment dat;

☒ *date grupate (structurate):*

– *date cu aceeași semnificație* (de același tip) – numite și *date omogene* –, spre exemplu: evidența săptămânală a vânzărilor pe zile și angajați, evidența vânzărilor pe angajați, datele despre toți angajații/candidații;

– *date cu semnificații diferite* – numite și *date neomegene* – spre exemplu, datele prin care este descrisă o persoană (candidat sau angajat): nume, an, medie.

De reținut!

O persoană este descrisă prin mai multe tipuri de date, ceea ce determină aspectul neomogen al grupului de date.

Candidații sau angajații formează grupuri de același tip – *persoană* – de unde rezultă aspectul omogen al acestui grup de date.

- Din analiza problemei urmărind *restricțiile de intrare-ieșire* (așteptare sau prioritate), după care sunt organizate datele dintr-un grup (structură), rezultă două categorii de date:

☒ date organizate după disciplina specifică unei *cozi* sau *fir de așteptare*; spre exemplu, candidații care așteaptă pentru selecție. Într-o astfel de structură, întotdeauna, un element nou este așezat (intră) după ultimul element. Dintr-o astfel de structură, iese, întotdeauna, primul element – în exemplul nostru, candidatul aflat „la rând".

☒ date organizate după disciplina specifică așezării obiectelor unul peste altul, în *stivă*; spre exemplu, dosarele angajaților. Într-o astfel de structură, întotdeauna, un element nou este așezat deasupra celorlalte, în *vârful* stivei. Dintr-o astfel de structură, iese, întotdeauna, elementul din *vârful* stivei – în exemplul nostru, va fi prelucrat dosarul ultimului angajat.

Concluzie

- organizarea datelor specifice unei probleme se poate face în locații de memorie independente – *date elementare* – sau în locații grupate – *structuri de date;*
- structurile de date pot fi:
  – *structuri omogene* (date cu aceeași semnificație/tip),
  – *structuri neomegene* (date cu semnificații/tipuri diferite);
- structurile omogene se numesc *tablouri;*
- într-un tablou, datele pot fi organizate astfel:

– după un singur criteriu – *tablouri unidimensionale* sau *vectori*; spre exemplu, tabloul pentru evidența vânzărilor realizate de fiecare angajat,

– după două criterii – *tablouri bidimensionale* sau *matrice*; spre exemplu, tabloul pentru evidența vânzărilor realizate zilnic (criteriul 1 – zilele săptămânii) de către fiecare angajat (criteriul 2 – angajații);

- într-un tablou unidimensional, datele pot fi organizate astfel încât să respecte o disciplină de intrare/ieșire de tip *coadă* sau *stivă*;
- structurile neomogene se numesc *articole* sau *înregistrări*; mai multe articole care descriu aceeași entitate (obiect, persoană etc.) reprezintă un grup de date cu aceeași semnificație și poate fi organizat într-o structură omogenă de tip tablou unidimensional (*vector de articole*).

TEME

1. Explicați deosebirea dintre analiza datelor din punct de vedere al complexității și analiza datelor din punct de vedere al regulilor de organizare (intrare/ieșire) într-o structură (grup de date).

2. Formulați un exemplu care să evidențieze diferența dintre datele omogene și datele neomegene.

3. Formulați un exemplu care să necesite organizarea după mai multe criterii a datelor cu aceeași semnificație.

4. Formulați un exemplu care să necesite organizarea datelor cu aceeași semnificație în structuri de tip *coadă*.

5. Formulați un exemplu care să necesite organizarea datelor cu aceeași semnificație în structuri de tip *stivă*.

6. Explicați disciplina structurii de date de tip coadă (FIFO – First Input First Output = *primul intrat, primul ieșit*).

7. Explicați disciplina structurii de date de tip stivă (LIFO – Last Input First Output = *ultimul intrat, primul ieșit).*

8. Formulați exemple de oganizare a datelor după alte reguli decât cele specifice structurilor de tip *coadă* sau *stivă*.

TEMĂ de GRUP

Densitatea straturilor geologice

*Un grup de geologi studiază densitatea straturilor geologice. În acest scop, s-a extras câte o probă pentru fiecare dintre cele **n** straturi geologice studiate. Fiecare probă este transmisă la un laborator specializat; pentru fiecare probă se fac mai multe teste, cel mult **m**. Întrucât nu există suficiente aparate, analiza de laborator durează. La sfârșitul activității, geologii păstrează probele în ordinea naturlă a straturilor geologice* (fig. 4)*. Rezultatele testelor se păstrează astfel încât să se poată determina, cu ușurință:*

– zăcământul la care s-au obținut aceleași valori la toate testele;

*– zăcământul cu cea mai mare densitate medie la cele **m** teste;*

*– zăcământul la care s-a obținut cea mai mare diferență între densitatea maximă și densitatea minimă din cele **m** teste.*

Cerințe:

1. Identificați formele de organizare a datelor specifice problemei propuse.

2. Fiecare membru al grupului descrie una dintre formele de organizare a datelor identificate (justificare, necesitate, proprietăți, prelucrări specifice și alte aspecte).

3. Grupul compune un scenariu pentru prelucrarea cu calculatorul a datelor specifice acestei probleme; scenariul poate fi prezentat narativ sau organizat pe secvențe (pași).

4. Grupul realizează o prezentare PowerPoint care să ajute la susținerea temei.

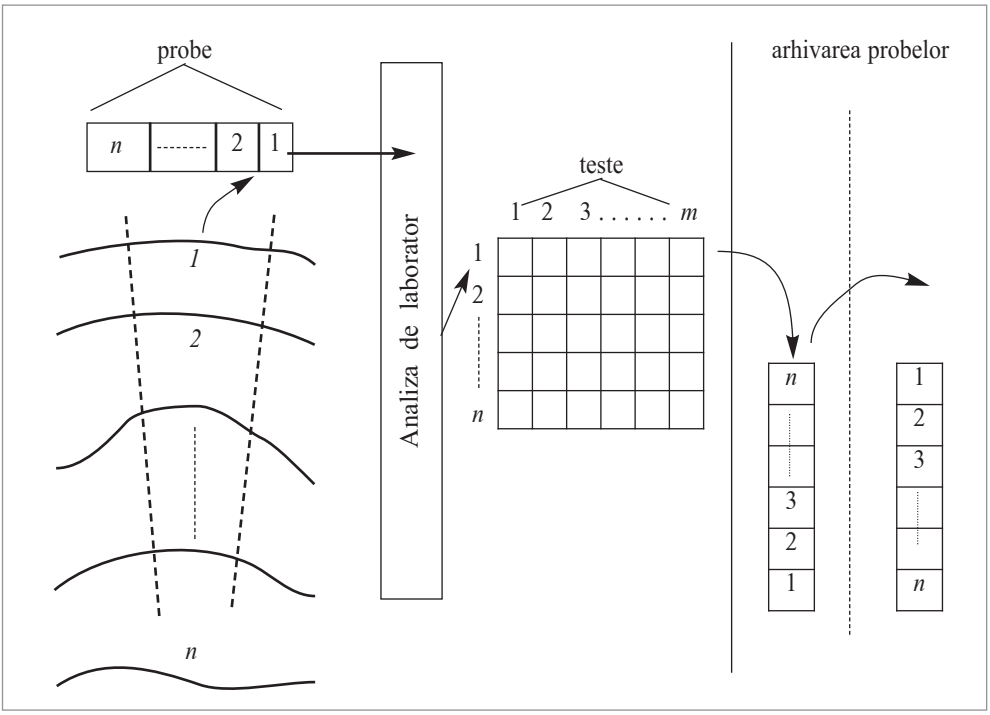


Figura 4

SUGESTIE DE PREZENTARE:

Prezentările pot fi expuse și analizate în laboratorul de Informatică; se va urmări corectitudinea rezolvării, creativitatea prezentării și eficiența lucrului în grup.

2. ORGANIZAREA DATELOR CU ACEEAȘI SEMNIFICAȚIE ÎN TABLOURI BIDIMENSIONALE

APLICAȚIA 1 FLOARE DE COLȚ

Membrii clubului „Floare de colț" participă la o tabără de vară, timp de o săptămână, într-o zonă montană greu accesibilă. Scopul lor este să înregistreze în fiecare zi temperatura aerului și să determine temperatura minimă, temperatura maximă și zilele în care s-au atins aceste temperaturi.

1. *Analiza problemei*

- *Date de intrare*
  – 7 valori reprezentând temperatura înregistrată în fiecare zi a săptămânii
- *Date de ieșire*
  – temperatura minimă (**tmin**);
  – temperatura maximă (**tmax**);
  – ziua/zilele în care s-au atins **tmin** și **tmax.**

2. *Organizarea datelor*

Datele de intrare au aceeași semnificație și vor fi înregistrate într-un vector (**T**) cu 7 elemente.

De exemplu:

T	18	15	12	14	18	15	12

tmin = 12 înregistrată miercuri și duminică
tmax = 18 înregistrată luni și vineri

3. *Raționamentul problemei*

Se înregistrează temperaturile zilnice în vectorul **T**. Se determină, printr-o singură parcurgere, valoarea minimă și cea maximă.

Pentru afișarea zilei/zilelor se mai fac două parcurgeri ale vectorului. Ziua va fi afișată ca indice (1, 2 etc.). Pentru afișarea zilei prin nume (luni, marți etc.) se recomandă introducerea structurii selective după **zi**.

4. *Reprezentarea algoritmului*

```
începput temperaturi1
alocă T[7]
pentru zi = 1, 7 execută citește T[zi] sfârșit pentru
tmin ← T[1]
tmax ← T[1]
pentru zi = 2, 7 execută
    bloc
        dacă T[zi] < tmin atunci tmin ← T[zi]
        sfârșit dacă
        dacă T[zi] > tmax atunci tmax ← T[zi]
        sfârșit dacă
    sfârșit bloc
```

```
sfârșit pentru
        scrie tmin
        |pentru zi = 1, 7 execută
        |       |dacă T[zi] = tmin atunci scrie zi
        |       |sfârșit dacă
        |sfârșit pentru
        scrie tmax
        |pentru zi = 1, 7 execută
        |       |dacă T[zi] = tmax atunci scrie zi
        |       |sf dacă
        |sfârșit pentru
        sfârșit temperaturi1
```

APLICAȚIA 2 FLOARE DE COLȚ

Măsurarea temperaturii aerului în zonele greu accesibile a fost foarte apreciată de ecologiști. Numărul voluntarilor dornici să participe la astfel de acțiuni a crescut. Instructorul clubului a organizat zece echipe care să măsoare temperatura zilnică în diverse zone de interes.

Prelucrarea valorilor înregistrate se va face astfel:

– se afișează temperatura minimă și temperatura maximă înregistrate în cele zece zone pe parcursul săptămânii;

– se afișează temperatura medie, pentru oricare dintre zone, la solicitarea celui interesat;

– se afișează zona în care s-a atins cea mai mică, respectiv cea mai mare temperatură, pentru orice zi a săptămânii solicitată de cel interesat.

1. *Analiza problemei*

Datele problemei au aceeași semnificație – temperatura zilnică –, dar se referă la zone diferite; de aceea, pentru organizarea lor se introduce și criteriul **zonă**. Rezultă un tablou (**T**) cu două dimensini: **zona** (linie) și **ziua** (coloană) (fig. 5).

T

Zona							
1						21	
2						18	
3	18	15	12	14	18	15	12
4						16	
5						17	
6						19	
7						20	
8						20	
9						17	
10						18	
	1	2	3	4	5	6	7

Ziua

Figura 5

Cum organizăm datele în tablouri

1. Dacă într-o problemă este necesară memorarea mai multor valori cu aceeaşi semnificaţie, aceste valori vor fi grupate într-un ansamblu de tip **tablou**.
2. În funcţie de cerinţele problemei şi de semnificaţia datelor, acestea sunt organizate în tablouri cu o singură dimensiune, numite **vectori**, sau în tablouri cu două dimensiuni, numite **matrice**.
3. Elementele unui tablou se identifică printr-o adresă formată din numele tabloului şi câte un indice pentru fiecare dimensiune.
4. La tablourile cu două dimensiuni, primul indice din adresă reprezintă linia, iar cel de-al doilea coloana tabloului.
5. Operaţiile care se repetă pentru fiecare element din tablou pot fi grupate în structuri repetitive cu contor. Contorul generează chiar indicele de adresă.

3. IMPLEMENTAREA TABLOURILOR BIDIMENSIONALE

Pentru memorarea şi prelucrarea datelor organizate ca tablouri bidimensionale, înainte de scrierea unui program trebuie să cunoaştem următoarele elemente:

– tipul elementelor din tablou (datele cu aceeaşi semnificaţie),

– numărul maxim de elemente din tablou (capacitatea tabloului).

capacitatea tabloului = nrmax _linii * nrmax _coloane

Aceste elemente rezultă din analiza problemei şi sunt folosite de compilator pentru determinarea zonei de memorie alocată tabloului.

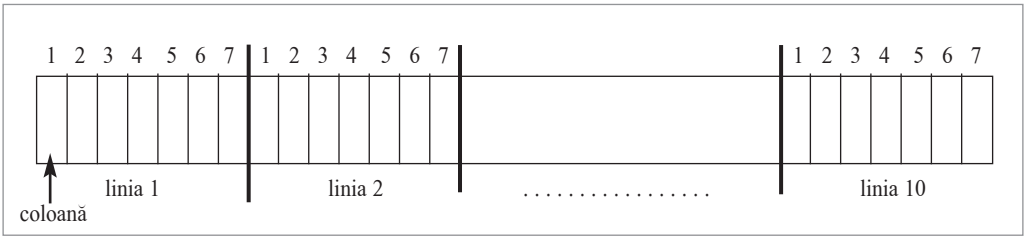


Figura 6 – Liniarizarea tabloului bidimensional

Tabloul ocupă în memoria calculatorului o „suprafaţă" (*array*) compusă din locaţii învecinate (adiacente). În fiecare locaţie este memorată valoarea unui element. Adresa locaţiilor începe de la o *valoare de referinţă* specifică limbajului de programare (corespunzătoare primei linii) şi se construieşte prin valori succesive (corespunzătoare coloanelor de pe linie), pentru fiecare element din tablou. În memoria calculatorului, tabloul este liniarizat: elementele sunt memorate în locaţii adiacente, în ordinea liniilor (fig. 6). Adresarea unui element se face printr-o pereche de indici corespunzători liniei şi coloanei pe care se află elementul respectiv.

În consecință, tipul variabilelor folosite ca indice de adresă trebuie să accepte valori în domeniile:

(1) [*valoare de referință, nrmax-linii*] pentru indicele de linie și

(2) [*valoare de referință, nrmax-coloane*] pentru indicele de coloană.

Adresarea elementelor se face, cel mai frecvent, prin două structuri repetitive cu contor, imbricate:

```
pentru  indice_linie = valoare-de referință la nr-linii execută
        pentru  indice_coloana = valoare-de referință la nr-coloane execută
          // prelucrarea elementelor din tablou în ordinea așezării pe linii
        sfârșit pentru
sfârșit pentru
```

Pentru indicele de linie sau indicele de coloană, se pot folosi tipuri diferite de date, ceea ce permite o referire asemănătoare cu cea întâlnită la jocul de șah; spre exemplu: T[2][C] reprezintă elementul de pe tabla de șah aflat pe linia 2 în coloana C (fig. 7).

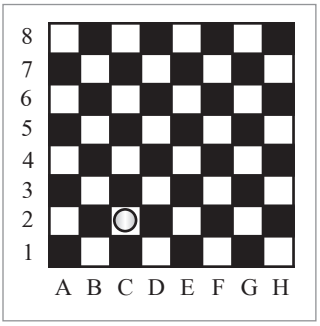


Figura 7

Elementele de sintaxă specifice tablourilor bidimensionale sunt prezentate în Tabelul 1.

***Tabelul* 1. Tablouri bidimensionale – elemente de sintaxă**

PASCAL	**C / C++**
Declararea tabloului	
var *tablou* **array** [1...*nl, 1..nc*] **of** *tipelement;*	*tipelement tablou*[*nl*][nc];
Declararea indicelui de adresă	
var *l,c:tipindice;*	*tipindice l,c;*
Adresarea unui element din tablou	
tablou [l,c] adresa de referință este 1.	*tablou* [*l*][*c*] adresa de referință este 0.
Exemplu: pentru tabloul *temperaturi* cu 100 de elemente de tip întreg distribuite pe 10 linii și 10 coloane: se inițializează cu zero elementele de pe primele 5 linii	
var *temperaturi* : **array** [1..10,1..10] **of integer**; l,c: **byte;** **begin** **for** l:=1 to 5 do **for** c:=1 to 10 do *temperaturi* [l,c] =0; **end**	**int** *temperaturi* [10] [10]; **short** l,c; { **for** (l=0; i< 5; i++) **for** (c=0; i< 10; i++) *temperaturi* [l][c]=0; }

Datele organizate în tablouri bidimensionale sunt prelucrate element cu element. Iată câteva dintre cele mai frecvente prelucrări:

- introducerea valorilor direct de la tastatură sau dintr-un fișier;
- afișarea valorilor pe ecran sau într-un fișier;
- verificarea unor proprietăți;
- compararea valorilor (aflarea elementului maxim sau minim);
- simularea unor situații reale.

Tabelul 2. **Operații la nivel de matrice**

PASCAL	C / C++
A. Introducerea valorilor în ordinea „pe linii”	
var A: array[1..10, 1..10] of integer; l,c,m,n : byte; begin write (‘ numarul de linii=’); readln (m); write (‘ numarul de coloane=’); readln (n); for l:=1 to m do for c:=1 to n do readln (A[l,c]); end;	int A [10] [10] ; int l,c,m,n ; { cout<<” numarul de linii=” ; cin>>m; cout<<” numarul de coloane=” ; cin>>n for (l=0; l< m; l++) for (c=0; c< n; c++) cin >>A[l][c]; }
B. Afișarea valorilor în ordinea „pe coloane” se păstrează declarațiile de la secvența A	
begin for c:=1 to n do begin writeln; for l:=1 to m do write (A[l,c], ‘ ’); end; end;	for (c=0; c< n; c++) { cout<<endl; for (l=0; l< m; l++) cout <<A[l][c]<< “ ”; }
C. Determinarea elementului minim de pe o linie oarecare, k se păstrează declarațiile de la secvența A	
var k:byte; min: integer; begin write (‘specificati linia=’); readln (k); min := A[k,1]; for c:=2 to n do if A[k,c] < min then min:= A[k,c]; write (‘min. de pe linia’, k, ‘=’, min); end;	int k, min; { cout<<” specificati linia=” ; cin>>k; min=A[k][0]; for (c=1; c< n; c++) if A[k][c] < min min= A[k][c]; cout<<”min. de pe linia”<<k<<”=”<<min; }

PASCAL	C / C++
D. Simularea unor situații reale. **Exemplu:** memorarea relațiilor de prietenie dintre membrii unui grup de n persoane (n<=10)	
var R : array [1..10, 1..10] of byte; l,c,i,j,d,n : byte; begin write (‘ numarul de persoane=’); readln (n); d:=1; repeat write (‘prietenie intre: ’); readln (i,j); {relatia de pritenie este reciproca} R[i,j] :=1; R[j,i] :=1; write (‘mai sunt prieteni? ’); readln (d); { da (d=1)/ nu (d=0)} until d=0; for l:=1 to n do begin writeln; for c:=1 to n do write (R[l,c], ‘ ’); end; end;	int R[10] [10] ; int l,c,i,j,d=1,n ; { cout<<” numarul de persoane=” ; cin>>n; while (d) { cout<<”prietenie intre:” ; cin>>i>>j; // relatia de prietenie este reciproca R[i][j]=1; R[j][i]=1; cout<<” mai sunt prieteni?” ; cin>>d; // da (d=1)/ nu (d=0) } for (l=0; l< n; l++) { cout<<endl; for (c=0; c< n; c++) cout>>R[l][c]<<” ”; } }	

Reluăm aplicația Floare de colț

Din analiza problemei, a reieșit necesitatea organizării datelor într-un tablou bidimensional. Continuăm rezolvarea problemei.

2. *Raționamentul problemei*

Pentru determinarea temperaturii minime și a temperaturii maxime înregistrate în cele zece zone, pe parcursul săptămânii, se prelucrează toate elementele din tablou.

Pentru determinarea temperaturii medii, **tmed**, se solicită zona și se prelucrează doar elementele de pe linia corespunzătoare.

Pentru determinarea zonei în care s-a atins temperatura minimă sau maximă, se solicită ziua și se prelucrează doar coloana corespunzătoare.

3. *Reprezentarea algoritmului*

```
început temperaturi2
alocă T[10, 7]
pentru zona = 1, 10 execută
    pentru zi = 1, 7 execută
        citește T[zona, zi]
    sfârșit pentru
sfârșit pentru
tmin ← T[1, 1]
tmax ← T[1, 1]
pentru zona = 1, 10 execută
    pentru zi = 1, 7 execută
        dacă T[zona, zi] < tmin
            atunci tmin ← T[zona, zi]
        sfârșit dacă
        dacă T[zona, zi] > tmax
            atunci tmax ← T[zona, zi]
        sfârșit dacă
    sfârșit pentru
sfârșit pentru
scrie tmin
scrie tmax
scrie "ce zonă vă interesează?"
citește zona
tmed ← 0
pentru zi = 1, 7 execută
    tmed ← tmed + T[zona, zi]
sfârșit pentru
tmed ← tmed/7
scrie zona, tmed
scrie "ce zi vă interesează?"
citește zi
tmin ← T[1, zi]
zmin ← 1
tmax ← T[1, zi]
zmax ← 1
pentru zona = 2, 10 execută
    bloc
        dacă T[zona, zi] < tmin
            atunci
                bloc
                    tmin ← T[zona, zi]
                    zmin ← zona
                sfârșit bloc
        sfârșit dacă
        dacă T[zona, zi] > tmax
            atunci
                bloc
                    tmax ← T[zona, zi]
                    zmax ← zona
                sfârșit bloc
        sfârșit dacă
    sfârșit bloc
sfârșit pentru
scrie zmin, tmin, zi
scrie zmax, tmax, zi
sfârșit temperaturi2
```

TEME

1. La cabinetul medical, se calculează înălțimea medie a tuturor băieților din școală. Precizați care este forma de organizare a datelor corespunzătoare acestei situații.

2. La cabinetul medical, se calculează înălțimea medie a tuturor băieților din școală, pe grupe de vârstă (ani de studiu). Precizați care este forma de organizare a datelor corespunzătoare acestei situații.

3. La cabinetul medical, se calculează înălțimea medie a tuturor băieților din școală pe grupe de vârstă (ani de studiu), pentru fiecare clasă în parte (9A,..., 12 C,...). Precizați care este forma de organizare a datelor corespunzătoare acestei situații.

4. Administratorul unui bloc cu 12 etaje și 20 de apartamente pe etaj calculează cheltuielile de întreținere, în funcție de numărul de persoane din fiecare apartament (toate apartamentele au același număr de camere).

Realizați un program care să răspundă următoarelor cerințe:

a) înregistrarea numărului de persoane din fiecare apartament, pe etaje, de la parter până la ultimul etaj;

b) afișarea numărului de persoane din fiecare apartament, pe etaje, de la ultimul etaj până la parter;

c) cunoscând numărul unui apartament, introdus de la tastatură, să se afișeze etajul la care se află apartamentul și numărul de persoane care locuiesc în apartamentul respectiv.

5. Se consideră tabloul **M** cu următoarele elemente:

1	2	3	4
5	6	7	8
9	10	11	12

Precizați ce valori afișează secvența de mai jos:

```
pentru c = 1, 3 execută
    pentru l = 1, 2 execută
        scrie M[l, c]
    sfârșit pentru
sfârșit pentru
```

a) 1, 5, 9, 2, 6, 10, 3, 7, 11; b) 1, 2, 3, 5, 6, 7; c) 1, 5, 2, 6, 3, 7.

6. Ce realizează următoarea secvență de operații:

```
alocă M[10, 10]
pentru i = 1, 10 execută
    M[i, i] ← i
sfârșit pentru
```

a) atribuie valori de la 1 la 10 elementelor din vectorul **M**;

b) atribuie fiecărui element de pe diagonala matricei **M** o valoare egală cu linia pe care aceasta se află;

c) atribuie valori de la 1 la 10 primelor 10 elemente din matricea **M**.

7. Care dintre următoarele secvențe de operații memorează în colțurile matricei **M** valori citite de la tastatură; matricea are 5 linii și 5 coloane.

a)
```
pentru i = 1, N execută
    pentru j = 1, N execută
        citește M[i, j]
    sfârșit pentru
sfârșit pentru
```

b)
```
pentru i = 1, 5 execută
    pentru j = 1, 5 execută
        dacă (i = 1) și (j = 5)
            atunci citește M[i, i]
            altfel citește M[j, j]
        sfârșit dacă
    sfârșit pentru
sfârșit pentru
```

c)
```
l ← 1
c ← 5
pentru x = 1, 4 execută
    citește M[l, c]
    c ← l
    l ← c
sfârșit pentru
```

d)
```
l ← 1
c ← 5
citește M[l, l], M[l, c], M[c, l], M[c, c]
```

4. TABLOURI BIDIMENSIONALE – CAZURI PARTICULARE

Matrice pătrată

Rezolvarea problemelor cu calculatorul necesită găsirea soluțiilor de organizare și memorare a datelor, astfel încât acestea să păstreze semnificațiile reale atât din punct de vedere al valorilor proprii, cât și al relațiilor cu alte date. Spre exemplu, dacă membrii unui grup (persoane) împrumută bani unii de la alții, interesează atât suma de bani primită/datorată, cât și cine/de la cine a împrumutat. În acest caz, dacă în grup sunt *n* persoane, un tablou bidimensional, *G*, cu *n* linii și *n* coloane, este suficient pentru păstrarea atât a valorilor împrumutate, cât și a relațiilor de împrumut (fig. 8).

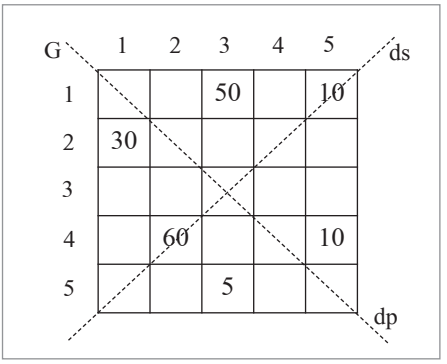


Figura 8

Fiecare element din tablou reprezintă valoarea unui împrumut; liniile și coloanele reprezintă persoanele din grup care dau bani unei alte persoane sau primesc bani de la altă persoană din grup. Fie *i* și *j* două persoane: *G[i,j]* reprezintă suma de bani pe care i-a împrumutat-o lui *j,* adică suma de bani pe care *j* a primit-o de la *i*. Pentru exemplul din figura 8, *G[4,2]* reprezintă suma de 60 lei pe care persoana 4 a împrumutat-o persoanei 2.

Tabloul folosit are o particularitate: numărul de linii este egal cu numărul de coloane, de aceea este numit tablou pătrat sau, mai simplu, *matrice pătrată.*

Într-o matrice pătrată deosebim următoarele elemente specifice (fig. 8):
– diagonala principală (*dp*);
– diagonala secundară (*ds*);
– triunghiurile formate de cele două diagonale;
– direcțiile paralele cu fiecare dintre cele două diagonale.

Matrice binară

Reluăm situația grupului de persoane care împrumută bani unii de la alții, dar urmărim numai relația de împrumut: cine/de la cine a primit bani. În acest caz, nu se mai păstrează valoarea împrumutului. Fie *i* și *j* două persoane: *G[i,j]* are semnificația *i* a împrumutat bani lui *j* echivalent cu *j* a primit bani de la *i* (fig. 9). Elementele matricei nu pot avea decât două valori alese convențional (spre exemplu 0 sau 1), cu semnificația:

G	1	2	3	4	5
1	0	0	1	0	1
2	1	0	0	0	0
3	0	0	0	0	0
4	0	1	0	0	1
5	0	0	1	0	0

Figura 9

$$G[i,j] = \begin{cases} 1 & \text{dacă } i \text{ împrumută bani lui } j \\ 0 & \text{dacă } i \text{ nu împrumută bani lui } j \end{cases}$$

Matricea ale cărei elemente au valori în mulțimea {0,1}, cu semnificații logice complementare, se numește *matrice binară.*

Matrice simetrică

Dacă în grupul de *n* persoane urmărim relațiile de prietenie, acestea pot fi memorate tot într-o matrice binară ale cărei elemente au următoarea semnificație:

$$P[i,j] = \begin{cases} 1 \text{ dacă } i \text{ este prieten cu } j \\ 0 \text{ dacă } i \text{ nu este prieten cu } j \end{cases}$$

Prietenia este o relație reciprocă; acest aspect se regăsește în proprietatea de *simetrie* a matricei binare asociată grupului de persoane: *P [i, j] = P [j, i]* (fig. 10).

P	1	2	3	4	5
1			1	1	
2			1		1
3	1	1		1	
4	1		1		1
5		1		1	

Figura 10

Matricea punctelor unui plan

Pe ecranul monitorului, în modul de lucru text, caracterele sunt afișate pe rânduri, de sus în jos, de la stânga la dreapta, pe fiecare rând. Se poate spune că ecranul monitorului este o suprafață plană ale cărei puncte sunt distribuite pe linii și coloane (cel mai frecvent, 24 de linii și 80 de coloane). În acest exemplu, regăsim modelul bidimensional de organizare a datelor: oricărei suprafețe plane îi poate fi asociată o *matrice a punctelor*. Valorile atribuite elementelor din matrice au semnificația specifică problemei modelate. Pentru exemplul suprafeței-ecran, dacă elementele matricei sunt de tip caracter, în matrice poate fi reținut textul de pe un ecran.

Un alt exemplu: o imagine – fotografie – este tot o reprezentare în plan. Punctele planului formează obiecte distincte, dacă sunt evidențiate diferit de la un obiect la altul, prin culoare. Dacă toate punctele unei imagini (suprafață) sunt colorate la fel, imaginea este formată dintr-un singur obiect.

Oricărei imagini îi poate fi asociată o matrice a punctelor; valorile atribuite elementelor din matrice au semnificația culorii fiecărui punct. În figura 11 este reprezentată matricea asociată unei imagini, în care s-au folosit 3 coduri de culori cu semnificația: 0 – alb, 1– negru, 2 – roșu.

Cu ajutorul matricelor de tip plan, pot fi modelate și situații de joc: așezarea pieselor pe tabla de șah, configurația unui labirint, „X și O", „avioane" și altele.

0	0	0	1	0	0	0	0
0	1	1	1	1	1	0	0
1	2	2	2	2	0	0	0
1	2	2	2	1	0	0	0
1	2	2	1	2	0	0	0
0	1	1	2	1	1	1	0
0	0	1	2	1	2	1	0
0	1	1	1	0	1	1	0

Figura 11

TEME

1. Determinați proprietățile elementelor aflate pe diagonala principală, într-o matrice pătrată. Scrieți un program pentru afișarea acestor elemente.

2. Determinați proprietățile elementelor aflate pe diagonala secundară, într-o matrice pătrată. Scrieți un program pentru afișarea acestor elemente.

3. Scrieți un program care să verifice dacă o matrice pătrată este simetrică.

4. Formulați un exemplu de problemă care să necesite organizarea datelor într-o matrice binară.

5. Formulați un exemplu de problemă care să necesite organizarea datelor într-o matrice de tip plan.

6. Determinați condițiile de amplasare pe tabla de șah a două piese de joc – dame –, astfel încât acestea să nu se atace. (Indicație: două piese de șah – dame – se atacă dacă sunt amplasate pe aceeași linie, pe aceeași coloană sau pe aceeași diagonală.)

7. Construiți tabloul de vecinătate pentru țările situate pe harta din figura 12.

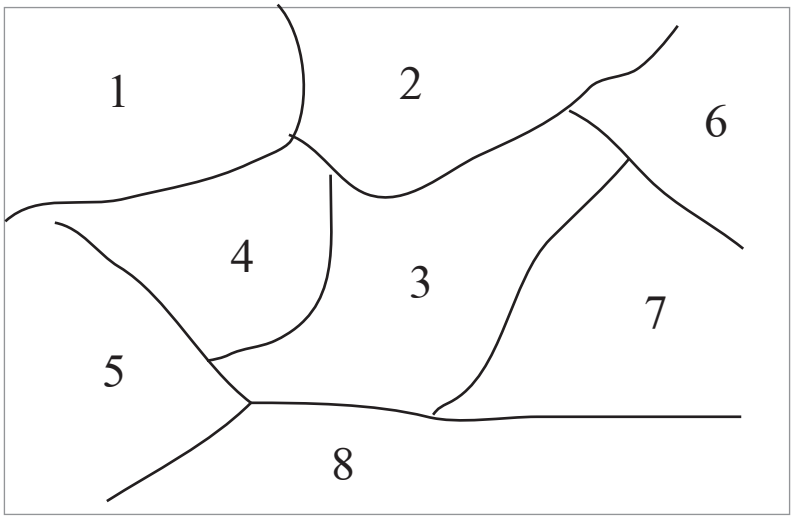


Figura 12

8. Să se verifice dacă o matrice pătrată este „tablou magic". Într-un tablou magic, suma elementelor de pe oricare linie este egală cu suma elementelor de pe oricare dintre coloane precum și cu suma elementelor de pe oricare dintre diagonale.

Exemplu:

3	2	7
8	4	0
1	6	5

5. PRELUCRAREA TABLOURILOR BIDIMENSIONALE

5.1. Localizarea elementelor cu aceeași proprietate

Formațiuni geografice

Se dorește determinarea configurației unui teren dreptunghiular după formațiunile geografice din Tabelul 3 și figura 13.

Analiza terenului se face prin secționarea acestuia pe orizontală și pe verticală.

Punctele aflate la intersecția dintre secțiunile orizontale și secțiunile verticale sunt cotate față de nivelul mării; cotele sunt valori numerice întregi și pozitive.

*Terenul este secționat prin **n** secțiuni orizontale și **m** secțiuni verticale.*

O formațiune geografică este formată din cel puțin trei puncte.

***Tabelul 3.* Formațiuni geografice**

FORMAȚIUNEA GEOGRAFICĂ	
Pantă	a)
Râpă	b)
Deal	c)
Vale	d)
Platou	e)
Teren denivelat	f)
Punct de tip *șa_xy*: punct aflat la cota maximă pe secțiunea orizontală *x* și la cota minimă pe secțiunea verticală *y*; se poate defini și punct de tip *șa_yx*	g)

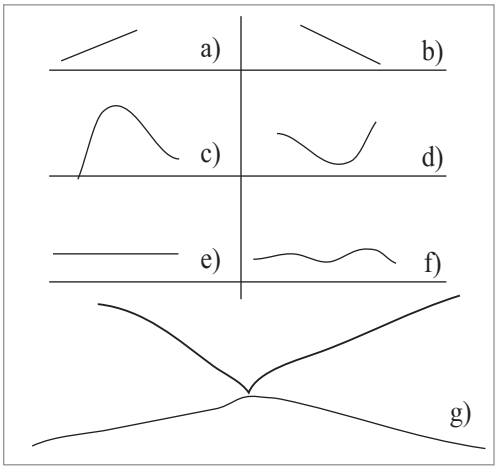


Figura 13

Exemplu:

În Tabelul 4 este reprezentat un teren pe care s-au făcut patru secțiuni orizontale și șapte secțiuni verticale.

***Tabelul 4.* Înregistrarea datelor într-un teren secționat**

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	25	72	69	69	69	73	40
2	20	40	50	56	30	19	100
3	15	30	35	40	39	20	10
4	10	55	0	60	42	50	19

Rezultatele analizei pe secțiuni sunt prezentate în tabelul Tabelul 5.

Tabelul 5. **Analiza pe secțiuni**

Secțiuni orizontale		**Secțiuni verticale**	
Numărul secțiunii	**Formațiunea geografică**	**Numărul secțiunii**	**Formațiunea geografică**
1	Platou la cota 69	1	Râpă
2	Deal cu vârf la cota 56	2	Vale cu punct minim la cota 30
3	Deal cu vârf la cota 40	3	Râpă
4	Teren denivelat	4	Vale cu punct minim la cota 40
		5	Teren denivelat
		6	Vale cu punct minim la cota 19
		7	Teren denivelat
Puncte de tip șa_xy: punctul de coordonate [3,4] la cota 40			

Sugestie de rezolvare

Pentru determinarea formațiunilor geografice, cotele vor fi înregistrate într-un tablou bidimensional cu semnificația: linii – secțiuni orizontale, coloane – secțiuni verticale.

Pentru fiecare secțiune (linie sau coloană) se va studia monotonia șirurilor de valori (cotele înregistrate pe o linie sau pe o coloană) – Tabelul 6.

Rezolvarea problemei conduce la determinarea elementelor cu aceeași proprietate dintr-un tablou bidimensional.

Tabelul 6. **Monotonia șirurilor de valori**

Formațiunea geografică	**Monotonia șirului de valori**
Pantă/ Râpă	Șir crescător/ descrescător
Deal	Există un punct de tip vârf, astfel încât toate punctele dispuse la stânga acestuia formează un șir monoton crescător, iar toate punctele dispuse la dreapta acestuia formează un șir monoton descrescător.
Vale	Există un punct de cota minimă, astfel încât toate punctele dispuse la stânga acestuia formează un șir monoton descrescător, iar toate punctele dispuse la dreapta acestuia formează un șir monoton crescător.
Platou	Există cel puțin trei puncte consecutive aflate la aceeași cotă.

Pentru exemplificarea rezolvării, prezentăm, în pseudocod, secvența prelucrărilor pentru determinarea formațiunii de tip *platou* și a punctelor de tip *șa_xy* .

Determinarea formațiunii de tip platou

început platou
// căutarea unei formațiuni platou pe linia *k*
//semnificația variabilelor de lucru
//*M*[100,100] tabloul cotelor cu m secțiuni verticale
// *lp* lungimea platoului
// *cp* cota platoului
// inițalizare lungime platou
lp←1
cp←*M*[k,1]
pentru c=2 **la** m **execută**
// se verifică dacă punctul *M*[k,c] aparține platoului
dacă M[k,c] =cp
atunci lp ← lp+1 // crește lungimea platoului
altfel
dacă lp>=3 // există platou la copta cp
atunci scrie *platou la cota* cp
sfârșit dacă
// inițializări pentru determinarea unui nou platou
lp←1
cp←M [k,c]
sfârșit dacă
sfârșit pentru
// se verifică dacă linia k se termină cu platou
dacă lp>=3
atunci scrie *platou la cota* cp
sfârșit dacă
sfârșit platou

Determinarea punctelor de tip șa_xy

- Raționamentul de rezolvare

Pasul 1

Se parcurge tabloul pe linii: pentru fiecare linie, se determină elementul maxim și se păstrează coloana acestuia în vectorul *max_linii*.

Pentru exemplul dat, vectorul *max_linii* are următoarele valori: 6, 7, 4, 4.

Pasul 2

Se parcurge tabloul pe coloane; pentru fiecare coloană, se determină elementul minim și se păstrează linia acestuia în vectorul *min_coloane*.

Pentru exemplul dat, vectorul *min_coloane* are următoarele valori: 4, 3, 4, 3, 2, 2, 3.

Pasul 3

Se parcurge vectorul *max_linii*: în variabila de lucru *cmax,* se reține valoarea unui element *max_linii[k]*

cmax = max_linii[k]

se verifică proprietatea de punct șa_xy:

min_colane[cmax] = k

Pentru exemplul dat, urmărim datele din Tabelul 7.

Există punct șa_xy pe linia 3, coloana 4.

Tabelul 7. **Determinarea punctului șa_xy**

linia	Cmax	min-coloane[cmax]
1	6	2
2	7	3
3	4	3
4	4	3

- Secvența pseudocod a prelucrărilor pentru determinarea punctelor de tip *șa_xy*:

```
începeut   punct_șa_xy
// căutarea unui punct șa_xy
//semnificația variabilelor de lucru
//M[100,100]  tabloul cotelor cu  m secțiuni orizontale  și n secțiuni verticale
// max_linii [100]  vector  cu m elemente în care se reține coloana  elementului maxim de pe
        fiecare linie
// min_coloane[100]  vector  cu n elemente în care se reține linia  elementului minim de pe
        fiecare coloană
//max    valoarea maximă   pe o linie;  cm coloana pe care se află  max
//min    valoarea minimă   pe o coloană;  lm linia  pe care se află  min

// se determină elementul maxim de pe fiecare linie
pentru k=1 la m execută
  max ← M[k,1]
  cm  ← 1
  pentru c=2  la  n execută
   dacă  M[k,c] >max
              atunci
                    bloc
                    max← M[k,c]
                    cm ← c
                   sfârșit bloc
    sfârșit dacă
 sfârșit pentru
```

```
// în linia k, elementul maxim se află pe coloana cm
max_linii[k] ← cm
sfârșit pentru

// se determină elementul minim de pe fiecare coloană
pentru c=1 la n execută
  min ← M[1,c]
  lm ← 1
  pentru k=2 la m execută
   dacă M[k,c] <min
            atunci
                 bloc
                 min ← M[k,c]
                 lm ← k
                sfârșit bloc
    sfârșit dacă
  sfârșit pentru
// în coloana c, elementul minim se află pe linia lm
min_coloane[c] ← lm
sfârșit pentru
// se parcurge vectorul max_linii
pentru k=1 la m execută
 cmax ← max_linii [k]
// se verifică proprietatea de punct șa_xy
 dacă min_coloane[cmax]=k
            atunci
                    scrie punct sa de coordonate k, cmax
  sfârșit dacă
sfârșit pentru
sfârșit punct _sa_xy
```

TEME

1. Construiți exemple numerice pentru următoarele formațiuni geografice:

a) platou pe coloana 3, c) râpă pentru coloana 5, e) punct șa_xy,
b) deal pentru linia 1, d) vale pentru linia 4, f) punct șa_yx.

2. Construiți expresii pentru relațiile de monotonie corespunzătoare fiecărei formațiuni geografice.

3. Codificați, în limbajul de programare studiat, secvența pseudocod pentru determinarea formațiunii geografice de tip platou.

4. Codificați, în limbajul de programare studiat, secvența pseudocod pentru determinarea punctelor de tip șa_xy.

5. Realizați, în limbajul de programare studiat, un program pentru determinarea punctelor de tip șa_yx.

6. Scrieți secvențele de instrucțiuni pentru determinarea următoarelor formațiuni geografice:
a) râpă, b) deal, c) vale.

7. Realizați și testați programul pentru derminarea configurației unui teren ale cărui coordonate și cote se citesc din fișierul *teren.in* cu următoarea structură:

– pe prima linie valorile *n* și *m* reprezentând: *n* numărul de secțiuni orizontale și *m* numărul de secțiuni verticale;

– pe următoarele *n* linii câte *m* valori reprezentând cotele aflate pe fiecare secțiune orizontală.

5.2. Prelucrarea elementelor distribuite pe aceleași direcții (linii, coloane, diagonale)

Aranjamente florale

Un grădinar are mai multe soiuri de plante pe care dorește să le planteze atât în aranjamente clasice, cât și în forme noi; spre exemplu, în locul rondurilor, grădinarul vrea să compună careuri florale. Întrucât timpul de creștere și înflorire al plantelor nu poate fi întârziat, grădinarul și-a propus să testeze modelele cu calculatorul.

Analiza problemei

Pentru rezolvarea cu calculatorul, soiurile de plante decorative vor fi codificate. În Tabelul 8 se prezintă un exemplu de codificare.

Tabelul 8. **Codificarea plantelor decorative**

Planta decorativă/culoare	Codul plantei
Iarbă/verde	1
Trandafir imperial/roșu	2
Narcise/albe	6
Narcise/galbene	7
Crizanteme/mov	8
Crizanteme/albe	9
Tuia/verde	11

Careul pe care va fi probat modelul este format din *n***m* sau *n***n* puncte; în fiecare punct, poate fi sădită o plantă. În Tabelul 9 sunt prezentate câteva modele după care se vor testa aranjamentele florale.

Tabelul 9. **Modele pentru aranjamente florale**

Denumirea modelului	Exemplu de model
Brazde paralele orizontale	Decor de primăvară cu narcise albe și galbene: 6 6 6 6 6 7 7 7 7 7 6 6 6 6 6
Triunghi	Decor de primăvară cu narcise albe și galbene; aleea centrală (diagonală) cu gazon: 1 7 7 7 7 6 1 7 7 7 6 6 1 7 7 6 6 6 1 7 6 6 6 6 1
Diagonale paralele cu diagonala principală	Decor cu trandafiri imperiali și tuia plantați pe direcții paralele cu aleea centrală (diagonală): 1 2 11 2 11 2 1 2 11 2 11 2 1 2 11 2 11 2 1 2 11 2 11 2 1
Spirală	Decor de toamnă cu crizanteme mov și albe: 8 8 8 8 8 8 8 9 9 9 9 8 8 9 8 8 9 8 8 9 8 8 9 8 8 9 9 9 9 8 8 8 8 8 8 8

Sugestie de rezolvare

Careurile florale vor fi generate într-un tablou bidimensional, G, cu *n* linii și *m* coloane, sau *n* linii și *n* coloane, în funcție de model. Pentru fiecare model, se determină adresele punctelor în care vor fi sădite plantele și se înregistrează la aceste adrese codul plantei corespunzător modelului.

Rezolvarea problemei conduce la umplerea matricei cu valori dispuse pe direcții specifice modelului: linii, coloane, triunghiuri, diagonale, spirală.

În Tabelul 10 sunt prezentate secvențele pseudocod pentru generarea adreselor de umplere corespunzătoare modelelor prezentate.

Tabelul 10. **Secvențe pseudocod pentru generarea adreselor de umplere**

Modelul floral	Direcțiile de umplere	Generarea adreselor
Brazde orizontale	Linii	// brazda de tip linie // linia i **pentru** c=1 la m **execută** G[i,c] ← cod **sfârșit pentru**
Triunghi	Triunghi stânga	//se lucrează cu o matrice pătrată **n*n** // pe linii **i** și coloane **j** **pentru** i=2 la n **execută** **pentru** j=1 la i-1 **execută** G[i,j] ← cod **sfârșit pentru** **sfârșit pentru**
Paralele la diagonala principală	Se pot forma n-2 direcții **(d)** paralele cu diagonala principală și situate deasupra acesteia	//se lucrează cu o matrice **n*n** // pe linii **i** și coloane **j** // pe diagonalele **d** **pentru** d=1 la n-2 **execută** **pentru** i=1 la n-d **execută** G[i,i+d] ← cod **sfârșit pentru** **sfârșit pentru**
Spirală	Se parcurg cadranele de la exterior – primul cadran-**p**, spre interior, ultimul cadran-**u**	u p←1 u←n **repetă** // se parcurge prima linie, **p**, din cadran, //de la stânga la dreapta **pentru** j=p **la** u **execută** G[p,j] ← cod **sfârșit pentru** // se parcurge ultima coloană, **u**, din cadran, // de sus în jos **pentru** i=p+1 **la** u **execută** G[i,u] ← cod **sfârșit pentru** //se parcurge ultima linie, **u**, din cadran

		//de la dreapta la stânga **pentru** j=u -1 **la** p **execută** G[u,j] ← cod **sfârșit pentru** // se parcurge prima coloană, **p**, din cadran, // de jos în sus **pentru** i=u - 1 **la** p - 1 **execută** G[i,p] ← cod **sfârșit pentru** // se pregătesc valorile **p** și **u** // pentru cadranul următor p← p+1 u←u-1 // se testează dacă se mai pot forma cadrane **până când** p>u

TEME

1. Construiți expresii pentru relațiile care definesc direcțiile de umplere pentru fiecare dintre modelele propuse.

2. Scrieți secvențele de instrucțiuni pentru generarea fiecărui model.

3. Realizați și testați programul pentru generarea fiecărui model floral (pentru fiecare model se va scrie un program).

4. Pentru atractivitatea prezentării, realizați un program care să afișeze modelul floral colorat, corespunzător culorii de cod. (Indicație: modul de lucru text permite setarea atributului de culoare atât pentru fond, cât și pentru text.)

5. Realizați un program care să permită utilizatorului (grădinarul) să aleagă, pe rând, oricare dintre modelele oferite (program cu meniu – fig. 14).

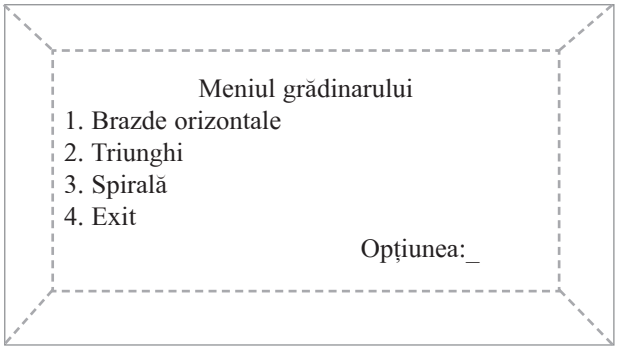


Figura 14

6. Rescrieți secvența umplerii în spirală astfel încât să folosiți cât mai puține structuri repetitive.

7. Compuneți un model floral nou și scrieți secvența de instrucțuni (programul) pentru generarea modelului propus.

8. Analizați următoarele secvențe pseudocod și determinați modelul de umplere generat:

a) **pentru** i=2 **la** n **execută** **pentru** j=n, **la** n+2-i **execută** G[i,j] ← cod **sfârșit pentru** **sfârșit pentru**	b) **pentru** i= 1 **la** [n/2] **execută** **pentru** j=i+1 **la** n-i **execută** G[i,j] ← cod **sfârșit pentru** **sfârșit pentru**
c) **pentru** i=n **la** [n/2]+1 **execută** **pentru** j=n **la** n+2-i **execută** G[i,j] ← cod **sfârșit pentru** **sfârșit pentru**	d) **pentru** i=1 **la** n **execută** **pentru** j=1, la **m execută** G[i,j] ← cod **sfârșit pentru** **sfârșit pentru**

5.3. Simularea unor situații reale

Tabloul de familie

Se consideră o familie formată din părinți (1) și copii (2). Fiecare copil are doi părinți între care există relație de căsătorie. Nu toți membrii familiei sunt căsătoriți; nu toți membrii familiei au copii.

Relațiile dintre membrii familiei sunt păstrate în tabloul de familie (fig. 15).

Figura 15. Tablou pentru o familie formată din 8 persoane

Membrul familiei	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***
1				*1*		*2*		*2*
2					*2*	*1*	*2*	
3					*1*			
4	*1*					*2*		*2*
5			*1*					
6		***1***			***2***		***2***	
7								
8								

Interpretarea tabloului de familie este prezentată în Tabelul 11.

Tabelul 11.

Membrul familiei	Căsătorit cu	Copii	Părinți
1	*4*	*6 și 8*	
2	*6*	*5 și 7*	
3	*5*		
4	*1*	*6 și 8*	
5	*3*		*2 și 6*
6	***2***	***5 și 7***	***1 și 4***
7			***2 și 6***
8			***1 și 4***

După cum rezultă din Tabelul 11, fiecare membru al familiei este complet caracterizat prin analiza înregistrărilor de tip linie și a înregistrărilor de tip coloană.

Analiza înregistrărilor de tip linie:

• familia are 8 membri; pentru fiecare membru (*i*) analizăm valorile de pe linia *i* din tablou, cu semnificația: (1) – *i* este căsătorit cu *j,* unde *j* este poziția valorii 1 pe linie,
(2) – *i* este părinte pentru *k,* unde *k* este poziția valorii 2 pe linie.

Analiza înregistrărilor de tip coloană:

• familia are 8 membri; pentru fiecare membru (*j*) analizăm valorile de pe coloana *j* din tablou cu semnificația: (1) – *j* este căsătorit cu *i,* unde *i* este poziția valorii 1 pe coloană,
(2) – *k* este părinte pentru *j,* unde *k* este poziția valorii 2 pe coloană.

Concluzii și restricții

- oricărei familii îi poate fi asociat un tablou de familie de tip matrice pătrată;
- pe o linie poate fi înregistrată cel mult o valoare 1;
- pe o coloană poate fi înregistrată cel mult o valoare 1;
- pe o coloană pot fi înregistrate cel mult două valori 2;
- dacă *i* este în relație de tip 1 cu *j,* și *j* este în relație de tip 1 cu *i;*
- dacă *i* este în relație de tip 1 cu *j,* și *i* este în relație de tip 2 cu *k,* atunci și *j* este în relație de tip 2 cu *k.*

TEME

1. Cunoscând relațiile de tip 1 sau 2 dintre cei *n* membri ai unei familii, să se construiască tabloul de familie.

2. Validarea tabloului de familie: fiind dat un tablou de familie, să se verifice corectitudinea înregistrărilor.

3. Fiind dat un tablou de familie validat, caracterizați fiecare membru al familiei.

4. Să se determine descendenții unui membru al familiei (pentru exemplul analizat, 1 are copii pe 6 și 8 și nepoți pe 5 și 7).

5. a) Realizați tabloul de familie pentru familia cu 10 membri din exemplul următor:

Membrul familiei	Căsătorit cu	Copii
2	3	1, 5, 7
5	8	1, 10
10	4	9

b) Analizați tabloul de familie obținut la cerința **a)** și precizați ce restricții au fost încălcate.

6. Cum procedăm pentru a determina toți descendenții unui membru al familiei?

7. Cum procedăm pentru a determina cel mai vârstnic ascendent al familiei?

5.4. Prelucrarea imaginilor

Aplicația 3. Virusuri într-o matrice

După ce au descoperit cele mai puternice virusuri, biologii și-au propus să le izoleze și să le studieze comportamentul în alte colonii de bacterii. După o vreme, au constatat că unele virusuri au rămas izolate, altele nu. Privită la microscop, colonia de bacterii pare formată dintr-o mulțime de puncte pe care biologii le-au reprezentat ca în figura de mai jos.

În această colonie există un singur virus izolat (cel încercuit). Biologii doresc să prelucreze aceste „imagini" cu calculatorul. Pentru început vor să localizeze și să numere virusurile izolate.

Cum rezolvăm problema?

1. Introducerea imaginii

De data aceasta, datele de intrare au aspectul unei „imagini alb-negru": punctele de interes sunt virusurile colorate în negru; restul suprafeței este colorată în alb. Pentru a transmite calculatorului „imaginea", ar trebui să-i spunem culoarea fiecărui punct. Sau, mai simplu, să-i spunem de la început că imaginea este „albă" și apoi să-i dăm adresele punctelor colorate în „negru". Imaginea descompusă în puncte seamănă atât de bine cu o matrice, încât ea poate fi memorată codificând albul cu zero și negrul cu unu.

	x				x
x			x	x	x
			x	x	
	(x)		x		

0	1	0	0	0	1
1	0	0	1	1	1
0	0	0	1	1	0
0	1	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Localizarea virusurilor izolate

2. Reprezentarea algoritmului

Secvența de operații pentru citirea imaginii:

```
început citire
    alocă M[10,10]
    citește NL, NC     //număr linii -NL
                       //număr coloane -NC
    pentru i = 1, NL execută
        pentru j = 1, NC execută
            M[i, j] ← 0
        sfârșit pentru
    sfârșit pentru
        citește p //numărul de puncte negre
        pentru k = 1,p execută
        bloc
            citește l,c
            M[l,c] ← 1
        sfârșit bloc
        sfârșit pentru
sfârșit citire
```

3. Prelucrarea imaginii. Bordajul

Urmează prelucrarea „imaginii": localizarea și numărarea virusurilor (punctele negre izolate).

Pentru rezolvarea acestei cerințe, se parcurge matricea și, pentru fiecare valoare de 1, se verifică vecinii acesteia. Un punct „negru" este izolat dacă toți vecinii săi sunt „albi".

Câți vecini are un punct? Majoritatea punctelor au câte opt vecini. Punctele de la „bariera" coloniei au mai puțini vecini.

Pentru a simplifica procedeul de numărare a vecinilor, vom înconjura colonia cu o zonă fără virusuri. Imaginea se lărgește. La fel și matricea: în urma acestui „bordaj", ea se va mări cu două linii și două coloane.

0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	1	1	1	0
0	0	0	0	1	1	0	0
0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Bordajul se pregătește înainte de citirea imaginii prin alocarea unui spațiu de memorie suficient, ținând seama că la numărul de linii și coloane necesar memorării imaginii se mai adaugă câte două pentru bordaj.

În matricea bordată, fiecare punct al imaginii are câte opt vecini. În figura 16 sunt prezentate adresele vecinilor punctului aflat la adresa [**l**, **c**].

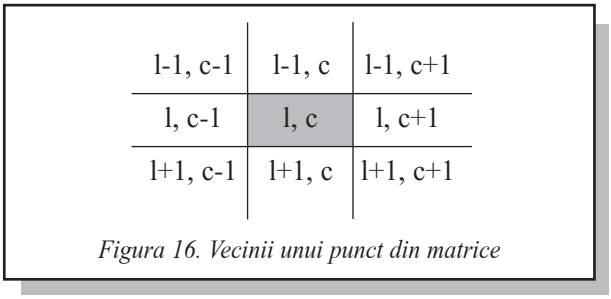


Figura 16. Vecinii unui punct din matrice

Dacă punctul aflat la adresa [**l**, **c**] are valoarea 1 și fiecare dintre cei opt vecini ai săi are valoarea zero, atunci punctul este izolat. Condiția o vom nota cu **cond**.

4. Reprezentarea algoritmului

• Secvența pentru localizarea și numărarea punctelor izolate:

```
început puncte_izolate
    nrp ← 0
    pentru l = 2,NL + 1 execută
        pentru c = 2,NC+1  execută
            dacă M[l,c] = 1 atunci
                dacă cond
                    atunci
                        bloc    scrie l - 1, c - 1
                                nrp ← nrp + 1
                        sfârșit bloc
                sfârșit dacă
            sfârșit dacă
        sfârșit pentru
    sfârșit pentru
    scrie nrp
sfârșit puncte_izolate
```

TEMĂ

Folosind cele două secvențe prezentate, realizați algoritmul complet și programul corespunzător.

De reținut!

1. Cu ajutorul matricelor pot fi rezolvate probleme care necesită descompunerea unei imagini în puncte sau probleme de orientare și deplasare în plan.

2. Pentru ușurința prelucrării punctelor se folosește tehnica bordajului.

3. Multe jocuri precum „X și zero", „Vaporașe", „Perspico", mutarea pieselor pe tabla de șah, „Război" pot fi simulate cu ajutorul matricelor.

PROBLEME PROPUSE

1. JOC

Se dau *n* jetoane numerotate de la 1 la *n*. Din fiecare tip de jeton, există *n* bucăți. Jetoanele trebuie așezate pe o grilă formată din *n***n* careuri, ca în exemplul din figura 17 a.

Figura 17. *Joc cu jetoane*

a)

1	2	3	4	5	6
2	3	4	5	6	1
3	4	5	6	1	2
4	5	6	1	2	3
5	6	1	2	3	4
6	1	2	3	4	5

b)

1	2	3	4	5
2	3	4	5	1
3	4	4	1	2
4	5	1	2	3
5	1	2	3	4

Cerințe:

1. Stabiliți regula după care sunt așezate jetoanele pe grila de joc.
2. Realizați un program care să așeze jetoanele pe grila de joc după regula stabilită la cerința precedentă.
3. Realizați un program care să verifice dacă jetoanele de pe grila de joc respectă regula de amplasare stabilită la cerința 1.

Spre exemplu, pentru grila de joc din figura 17 b, nu sunt îndeplinite următoarele proprietăți:
- pe diagonala secundară se află un jeton diferi de jetonul *n*;
- pe linia 3 nu se află jetoane distincte;
- pe coloana 3 nu se află jetoane distincte.

Indicație: se vor analiza jetoanele dispuse pe aceeași linie/coloană sau pe diagonale/paralele la diagonale.

2. TABELE MATEMATICE

a) TABLA ÎNMULȚIRII

Realizați un program pentru afișarea sub formă de tabel a tablei înmulțirii cu 1, 2, 3, până la 10. Spre exemplu, linia 3 va conține tabla înmulțirii cu 3:

3	6	9	12	15	18	21	24	27	30

b) TABLA ADUNĂRII

Realizați un program pentru afișarea sub forma de tabel a tablei adunării cu 1, 2, 3, până la 10. Spre exemplu, linia 7 va conține tabla adunării cu 7:

8	9	10	11	12	13	14	15	16	17

c) TABELA PITAGORA_n

Realizați un program pentru afișarea tabelei *Pitagora_n* sub formă de tabel cu *n* linii și trei coloane: *a, b, c*. Pentru fiecare linie, valorile din tabelă trebuie să respecte condiția:

*a*a = b*b + c*c*. **Exemplu:** tabela *Pitagora_2*

5	3	4
15	12	9

d) MATRICEA *n_PALINDROM*

Realizați un program care să verifice dacă o matrice este *n_Palindrom*. O matrice *n_Palindrom* are *n* linii; pe fiecare linie, *i*, se află câte *n* palindromuri distincte formate din *i* cifre. Un număr simetric se numește *palindrom.* Exemplu de matrice *5_Palindrom:*

1	9	7	6	3
22	33	66	4455	55
121	323	464	585	979
3223	8668	7337	8118	9009
56365	43234	11111	91219	37473

3. MATRICE RARĂ

O matrice cu *n* linii și *m* coloane se numește *matrice rară* dacă valorile 0 sunt majoritare. **Exemplu** pentru o matrice cu 3 linii și 5 coloane:

0	0	20	0	0
0	0	0	12	0
0	88	0	0	0

Într-o astfel de matrice, interesează valorile semnificative, de aceea este suficient să memorăm valorile diferite de zero și pozițiile lor în matrice. Poziția reprezintă în acest caz numărul de ordine de 1 la *n*m*.

Pentru exemplul prezentat, se vor memora perechile: (20, 3) , (12, 9) și (88,12).

Cerințe:

a) propuneți structurile de date necesare memorării valorilor semnificative dintr-o matrice rară;

b) se cunoaște numărul valorilor semnificative dintr-o matrice rară (fie *p* acest număr) și perechile *valoare*, *adresă* pentru fiecare număr semnificativ.

Scrieți un program care să genereze și să afișeze pe ecran matricea rară.

Exemplu: Date de intrare

n =4 m=4 p=2

23 4 9 14

Date de ieșire — matricea rară

0	0	0	23
0	0	0	0
0	0	0	0
0	9	0	0

6. ORGANIZAREA DATELOR ÎN STRUCTURI NEOMOGENE

6.1. Studiu de caz *Catalogul clasei*

Într-o clasă sunt 30 de elevi. Profesorul diriginte dorește să păstreze, într-un catalog electronic, următoarele informații (figura 18):

– numele elevului;
– media generală pe semestrul I;
– media generală pe semestrul II;
– media generală anuală;
– numărul de absențe.

Profesorul diriginte dorește să calculeze media generală a clasei la sfârșit de semestru sau an școlar și să afișeze:

– media generală a clasei;
– lista elevilor după media generală;
– lista elevilor după numărul de absențe.

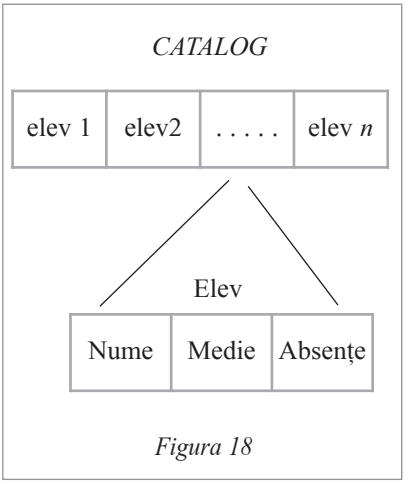


Figura 18

Analiza problemei

În catalogul electronic se vor păstra date despre elevi; se poate spune că în catalogul electronic se vor păstra date cu aceeași semnificație. Numărul înregistrărilor din catalog va fi egal cu numărul de elevi din clasă. Se va ține seama de numărul maxim de elevi ce pot fi înscriși într-o clasă (30 de elevi).

Catalogul poate fi organizat ca un tablou unidimensional – vector de elevi.

Pentru fiecare elev se rețin mai multe categorii de date, cu semnificații diferite – *date neomogene*. Fiecare informație are un tip specific, spre exemplu:

– medie de tip *real*;
– număr de absențe de tip *întreg* cu valori pozitive.

Soluția de organizare a datelor

Ansamblul datelor despre *elev* formează o structură de date cu tipuri diferite (structură neomogenă) numită *articol* sau *înregistrare*.

Catalogul electronic al clasei va fi un *vector de articole*.

Prelucrarea datelor

Pentru rezolvarea cerințelor formulate de profesorul diriginte, sunt necesare următoarele prelucrări:

– înregistrarea datelor pentru fiecare elev;
– afișarea datelor despre elevi;
– determinarea și afișarea mediei generale a clasei la sfârșit de semestru;

– afișarea elevilor în ordine descrescătoare, după media generală semestrială;
– determinarea mediei generale, anuale, pentru fiecare elev;
– afișarea elevilor în ordine descrescătoare, după media generală anuală;
– determinarea și afișarea mediei generale a clasei la sfârșit de an;
– afișarea elevilor în ordine crescătoare, după numărul de absențe.

6.2. Definirea structurilor neomogene de date-articole

Categoriile distincte de informații reținute într-un articol se numesc *câmpuri*.
Un articol se caracterizează prin următoarele elemente ce formează *macheta articolului*:
– numele articolului;
– două sau mai multe *câmpuri*; pentru fiecare *câmp* se precizează:
– numele câmpului;
– tipul câmpului (stabilit în funcție de semnificația reală a datelor). Câmpurile pot fi date elementare sau grupate: *vectori*, *articole*.

Exemplul 1:
Pentru exemplul analizat – catalogul clasei –, construim o machetă simplificată (cu mai puține câmpuri) corespunzătoare articolului *elev* (fig. 19a).

Elev	
Numele câmpului	Tipul câmpului
MEDIE	Real
ABSENȚE	Natural

Figura 19, a)

Exemplul 2:
Completăm macheta articolului *elev* cu câmpul note, un tablou unidimensional (*vector)* cu 4 elemente de tip natural (fig. 19b).

Elev	
Numele câmpului	Tipul câmpului
NOTE	Vector (4)
MEDIE	Real
ABSENȚE	Natural

Figura 19, b)

Fiecare limbaj de programare dispune de cuvinte cheie pentru declararea structurilor de date de tip articol (Tabelul 12).

Tabelul 12. **Definirea articolelor**

PASCAL	C/C++
{cuvântul cheie pentru tipul articol este ***record****}* *Type* **articol = record** camp1:*tip1*; camp2:*tip2*; _____ end;	*// cuvântul cheie pentru tipul articol* *// este* ***struct*** *Typedef* struct { *tip1* camp1; *tip2* camp2; _____- } **articol**;
{definirea variabilei cu tipul asociat articolului} *var* variabila : **articol**;	//definirea variabilei cu tipul asociat articolului **articol** variabila;
Exemplu pentru articolul elev	
{ se definește articolul elev } *Type elev*=record …… medie:*real*; absente:*byte*; . . . . . . end; {se definește variabila cu tipul elev} *var* e: *elev*;	// se definește articolul elev *Typedef* struct { …… *float* medie; int:absente; . . . . . . } elev; // se definește variabila cu tipul elev *elev e*;

TEME

1. Realizați macheta articolului *data calendaristică* în care să fie reținută data în forma *zi, lună, an*.

2. Realizați macheta articolului *elev* care să conțină și câmpul *data nașterii* sub forma grupului de date *data calendaristică*.

3. Realizați macheta articolului *elev* care să conțină câmpul *medii* sub forma unui vector.

6.3. Prelucrarea datelor organizate în structuri neomogene

Prelucrarea datelor înregistrate într-o structură de tip articol se face la nivel de câmp. Pentru accesul la informațiile unui câmp, referirea acestuia se face prin *adresare punctuală* după sintaxa: *nume-variabilă-de-tip-articol.nume-câmp.*

Exemplu:

– referirea câmpului medie din articolul *elev* **e.medie**

– referirea câmpului absențe din articolul *elev* **e.absente**

– referirea unui element, i, al câmpului medii din articolul *elev* **e.medii[i]**

Prelucrarea acestor date începe cu introducerea şi memorarea lor în zona de memorie rezervată la definirea variabilei de tip articol; pentru verificare, datele memorate vor fi afişate. Fiecare câmp poate fi prelucrat prin operații specifice tipului corespunzător (Tabelul 13).

Tabelul 13. **Prelucrarea articolelor**

PASCAL	**C / C++**
Exemplu pentru articolul elev	
1. Introducerea şi afişarea datelor	
Program exemplu; { se defineste articolul elev } *Type elev*=record . . . . . . medie1, medie2, medie_an:*real*; absente:*byte*; . . . . . . end; {se defineste variabila cu tipul elev} *var* e: *elev*; begin {se introduc datele unui elev} write (’medie1:’); readln (e.medie1); write (’medie2:’); readln (e.medie2); write (’absente:’); readln (e.absente); . . . . . . {se afiseaza datele elevului} writeln (’medie1:’, e.medie1); writeln (’medie:2’, e.medie2); writeln (’absente:’, e.absente); end.	#include <iostream.h> void main () {// se defineste articolul elev *Typedef* struct { . . . . . . *float* medie1, medie2, medie_an; unsigned:absente; . . . . . . } elev; // se defineste variabila cu tipul elev *elev e*; // se introduc datele unui elev cout<< ” medie1:”; cin>> e.medie1; cout<< ” medie2:”; cin>> e.medie2; cout<< ” absente:”; cin>> e.absente; . . . . . . // se afiseaza datele elevului cout<< ” medie1:”<<e.medie1<<endl; cout<< ” medie2:”<<e.medie2<<endl; cout<< ” absente:”<< e.absente<<endl; }

2. Operații asupra datelor	
{ se calculează media anuală} e.medie_an := (e.medie1 +e.medie2)/2; {se afiseaza media anuala} writeln('medie anuala: ', e.medie_an);	// se calculează media anuală e.medie_an= (e.medie1 +e.medie2)/2; // se afiseaza media anuala cout<<:"medie anuala:"<< e.medie_an;
3. Introducerea și afișarea datelor în câmpuri grupate	
Type elev=record . . . . . . medii : array [1..3] of *real*; . . . . . . end; {se defineste variabila cu tipul elev} *var* e: *elev*; i:byte; begin {se introduc mediile unui elev} for i:=1 to 3 do begin write ('media:',i); readln (e.medii[i]); end; end	*Typedef* struct { . . . . . . *float* medii [3]; . . . . . . } elev; // se defineste variabila cu tipul elev *elev e*; int i; { // se introduc mediile unui elev for (i=0; i<3;i++) {cout<< " media:"<<i; cin>> e.medii[i]; } }

TEME

1. Descrieți sub formă de articol următoarele entități:
a) carte; b) mașină; c) calculator; d) persoană.

2. Definiți, în limbajul de programare studiat, tipurile de date și variabilele corespunzătoare articolelor descrise la cerința precedentă.

3. Realizați, în limbajul de programare studiat, un program care să determine trimestrul calendaristic al zilei curente. Se va folosi un articol cu câmpurile: *zi, lună, an*.
Exemplu: pentru data curentă: ziua 29 luna 08 anul 2007, se va afișa: trimestrul 3.

4. Realizați, în limbajul de programare studiat, un program care să determine dacă două puncte *A* și *B* îndeplinesc una dintre următoarele condiții:
a) punctele A și B se află pe o dreaptă paralelă cu axa OX;
b) punctele A și B se află pe o dreaptă paralelă cu axa OY;
c) punctele A și B se află pe o dreaptă egal depărtată de axele OX si OY.
Pentru fiecare punct se cunosc coordonatele *x* și *y*. Fiecare punct va fi descris printr-un articol.
Exemplu: Pentru punctul A de coordonate x=3, y=10 și punctul B de coordonate x=35, y=10 se va afișa mesajul: *punctele se află pe o dreaptă paralelă cu axa OX*.

6.4. Gruparea datelor organizate în structuri neomogene

Datele cu semnificații diferite care necesită gruparea în structuri neomogene reprezintă, în cele mai frecvente cazuri, caracteristici sau atribute ale unei situații din realitate (entitate): persoană, produs, carte, mașină etc. Memorarea datelor prin care poate fi descrisă o entitate se face într-o singură zonă de memorie (variabilă); dacă în problemă sunt prelucrate date despre mai multe *instanțe* ale unei *entități* (mai multe persoane, mai multe produse), se vor păstra întotdeauna datele ultimei instanțe introduse (ultima persoană, ultimul produs).

Exemplu:

*La un magazin, se vând într-o zi **n** produse. Se cunosc cantitatea și prețul fiecărui produs; se dorește calcularea și afișarea valorii totale a produselor vândute într-o zi.*

Rezolvarea problemei necesită introducerea, pe rând, a datelor despre fiecare produs. Pentru fiecare produs introdus, se calculează valoarea:

*valoare = cantitate * pret.*

Valoarea calculată se însumează la *total:*

total= total + valoare.

La sfârșit, se afișează *totalul.* (Tabelul 14)

Tabelul 14. **Exemplu *Magazin***

PASCAL	C/C++
program produse; { se defineste articolul produs } *Type produs*=record cantitate:*byte*; pret:*real*; end; {se defineste variabila cu tipul produs} *var* p: *produs*; {se definesc variabilele de lucru} var valoare, total: real; n, i :integer; begin {se introduce numarul de produse} Write('numarul de produse:'); readln (n); total:=0; for i:= 1 to n do begin	#include <iostream.h> void main () { // se defineste articolul produs *Typedef* struct {*unsigned* cantitate *float* pret; } produs; // se defineste variabila cu tipul produs *produs p*; //se definesc variabilele de lucru float valoare, total=0; int n, i ; //se introduce numarul de produse cout<< " numarul de produse:"; cin>> n; for (i=1; 1<n; i++) { //se introduc datele unui produs

{se introduc datele unui produs} write('cantitate:'); readln (p.cantitate); write ('pret:'); readln (p.pret); {se calculeaza valoarea produsului} valoare:= p.cantitate * p.pret; {se insumează valoarea la total} total:= total + valoare; end; {se afiseaza valoarea totala} Writeln ('valoarea totala:', total); end	cout<< " cantitate:"; cin>>p.cantitate; cout<< " pret:" cin>>p.pret; //se calculeaza valoarea produsului valoare = p.cantitate * p.pret; //se insumează valoarea la total} total= total + valoare; } //se afiseaza valoarea totala cout<<"valoarea totala:"<<total; }

În cele mai multe situații reale, se păstrează datele fiecărei instanțe (fiecare persoană, fiecare produs) pentru prelucrări la nivel de grup: sortări, selecții. În *Catalogul clasei*, se păstrează datele despre toți elevii.

Gruparea instanțelor unei entități (toate persoanele, toate produsele, toți elevii) se face prin memorarea valorilor corespunzătoare în tablouri unidimensionale care, în acest caz, devin *vectori de articole*. Un element al vectorului reprezintă o instanță (o persoană, un produs, un elev) și este de tip articol.

Câmpul unui articol dintr-un vector de articole se adresează punctual, cu precizarea că, în acest caz, numele articolului este înlocuit cu adresa acestuia în vector (Tabelul 15).

Tabelul 15. **Vectori de articole**

PASCAL	C/C++
{definirea tipului de date articol} *Type* articol=record camp1:*tip1*; camp2:*tip2*; . . . . . . end; {definirea vectorului cu n elemente de tipul asociat articolului} *var* vector:array [1..10] of articol; var i : byte; {adresarea unui camp dintr-un articol oarecare,i} vector [i].camp	*// definirea tipului de date articol* *Typedef* struct { *tip1* camp1; *tip2* camp2; . . . . . . } articol; // definirea vectorului cu n elemente //de tipul asociat articolului *articol* vector [10]; int i; //adresarea unui camp dintr-un articol //oarecare,i} vector [i].camp

Exemplul 1 *prelucrarea a **n** produse*

PASCAL

```
program produse;
{ se defineste articolul  produs }
Type produs=record
    cantitate:byte;
    pret:real;
   end;

{se defineste vectorul p cu  100 de elemente
de  tip produs}
var p:array [1..100] of  produs;

{se definesc variabilele de lucru}
var valoare, total: real;
    n, i :integer;
begin

{se introduce numarul de produse}
write('numarul de produse:');
readln (n);
total:=0;
for i:= 1 to n do
begin

{se introduc  si se memoreaza  datele fiecarui
produs}
write('cantitate:');
readln (p[i].cantitate);
write ('pret:'); readln (p[i].pret);

{se calculeaza valoarea produsului}
valoare:= p[i].cantitate * p[i].pret;
{se însumează valoarea la total}
total:= total + valoare;
end;
{se afiseaza valoarea totala}
writeln ('valoarea totala:', total);
end.
```

C/C++

```
#include <iostream.h>
void main ()
{
// se defineste articolul  produs
Typedef struct
    {unsigned cantitate
     float pret;
     } produs;

// se defineste vectorul p cu 100
//elemente de tip produs
produs  p[100];

//se definesc variabilele de lucru
float  valoare, total=0;
int    n, i ;

//se introduce numarul de produse
cout<< " numarul de produse:";
cin>> n;
for (i=0; i<n; i++)
{

//se introduc si se memoreaza  datele  //
fiecarui produs
cout<< " cantitate:";
cin>>p[i].cantitate;
cout<< " pret:"
cin>>p[i].pret;

//se calculeaza valoarea produsului
valoare = p[i].cantitate * p[i].pret;
{se însumează valoarea la total}
total= total + valoare;
}
//se afiseaza valoarea totala
cout<<"valoarea totala:"<<total;
}
```

Exemplul 2 sortarea a ***n*** produse - metoda *Bubble sort*

```
var aux: produs; s:byte;
begin
{sortare crescatoare dupa campul pret}
repeat
 s:=0;
for i:=1 to n-1 do
  begin
     if p[i].pret> p[i+1].pret
       then
{interschimbarea se face la nivel de articol }
          begin
            aux:=p[i];
             p[i]:= p[i+1];
             p[i+1]:= aux;
             s:=1;
          end;
until s=0;
writeln (' lista preturilor');
for i:=1 to n do
writeln (p[i].pret);
end
```

```
{produs aux; unsigned s;
//sortare crescatoare dupa campul pret
do
{
   s=0;
   for (i=0; i< n-1; i++)
   if p[i].pret> p[i+1].pret

//interschimbarea se face la nivel de articol
          {
            aux=p[i];
             p[i]= p[i+1];
             p[i+1]= aux;
             s=1;
          }
}
while (s);
cout<<"lista preturilor"<<endl;
for (i=0; i< n; i++)
cout<<p[i].pret<<endl;
}
```

TEME

1. Formulați exemple care să necesite organizarea datelor în vectori de articole.

2. Prezentați o situație reală care să necesite ordonarea datelor organizate în vectori de articole.

3. Alcătuiți o listă cu prelucrări specifice datelor grupate în tablouri unidimensionale; pentru fiecare prelucrare, construiți câte un exemplu care să necesite gruparea articolelor în vectori de articole.

De reținut! ARTICOL

• structură de date necesară pentru înregistrarea informațiilor despre un aspect al realității (obiect, persoană, activitate) cu mai multe caracteristici;

• fiecare caracteristică formează un câmp al articolului;

• fiecare câmp poate avea un tip propriu; acest aspect determină proprietatea de structură neomogenă;

• memorarea și prelucrarea datelor organizate în structuri neomegene se face la nivel de câmp;

• referirea unui câmp se face prin adresare punctuală:
– nume_articol.nume_câmp;

• articolele pot fi grupate în structuri omogene: vectori de articole;

• referirea unui câmp de articol dintr-un vector se face prin adresare punctuală, înlocuindu-se numele articolului cu adresa acestuia în vector.

PROBLEME PROPUSE

1. Colecție

Se dorește înregistrarea următoarelor date despre obiectele dintr-o colecție: denumire, anul achiziției, valoare.

Cerințe:

a) realizați un program pentru introducerea și afișarea datelor despre un obiect din colecție;

b) realizați un program pentru introducerea și afișarea datelor despre mai multe obiecte din colecție (max. 100);

c) realizați un program care să determine cel mai vechi obiect din colecție;

d) realizați un program care să afișeze obiectele din colecție în ordinea descrescătoare a vechimii acestora.

2. Concurs

Se dorește înregistrarea următoarelor date despre candidații înscriși la un concurs: numele, data nașterii (zi, lună, an), are carnet de conducere (da/nu).

Cerințe:

a) realizați un program pentru introducerea și afișarea datelor despre un singur candidat;

b) realizați un program pentru introducerea și afișarea datelor despre mai mulți candidați (max. 100);

c) realizați un program care să determine candidații cu aceeași vârstă, *v*, introdusă de la tastatură;

d) realizați un program care să afișeze, în ordinea vârstei, candidații cu carnet de conducere.

3. Catalogul clasei

Realizați un program cu meniu care să rezolve următoarele cerințe ale profesorului diriginte:

– înregistrarea datelor pentru fiecare elev;
– afișarea datelor despre elevi;
– determinarea și afișarea mediei generale a clasei la sfârșit de semestru;
– afișarea elevilor în ordine descrescătoare, după media generală semestrială;
– determinarea mediei generale, anuale, pentru fiecare elev;
– afișarea elevilor în ordine descrescătoare, după media generală anuală;
– determinarea și afișarea mediei generale a clasei la sfârșit de an;
– afișarea elevilor în ordine crescătoare, după numărul de absențe.

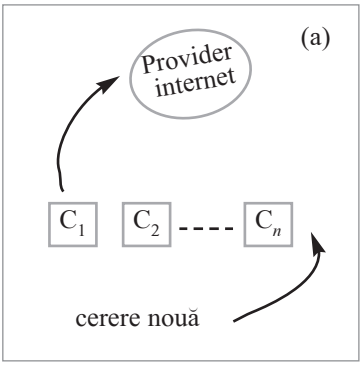
7. ORGANIZAREA DATELOR ÎN STRUCTURI DINAMICE

7.1. Modele de structuri dinamice

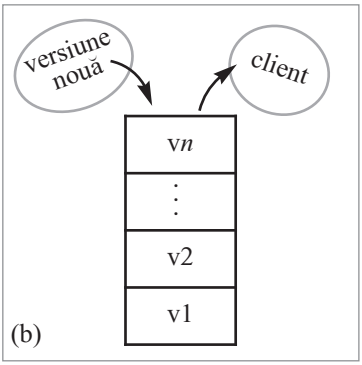
În foarte multe situații reale, există relații sau reguli care trebuie modelate astfel încât soluția de organizare a datelor să respecte atât semnificația, cât și restricțiile de comportament specifice.

Exemple:

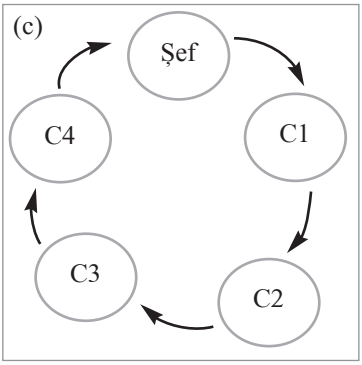
Exemplul 1. Persoanele care solicită un serviciu – conectarea la Internet printr-un provider autorizat – se înscriu pe o listă de aşteptare. O cerere nouă este aşezată, întotdeauna, ultima în listă. Serviciul de conectare este acordat, întotdeauna, primului solicitant din listă; după acordarea serviciului, cererea este eliminată din listă (fig. 20 a).



Exemplul 2. O firmă de software a realizat un program antivirus; programul este îmbunătăţit continuu, prin tratarea de noi viruşi; oferta de piaţă a firmei este organizată astfel încât clienţii să aibă acces la program, începând întotdeauna cu ultima versiune a acestuia (fig. 20 b).



Exemplul 3. Pentru a comunica rapid şi sigur, „şeful" unui grup de copii a întocmit o listă astfel încât un mesaj să poată fi transmis întregului grup, din copil în copil; „şeful" transmite mesajul primului copil din listă; ultimul copil comunică „şefului" că mesajul a ajuns la el (fig. 20 c).



4. Fiecare persoană are exact doi părinţi; fiecare părinte este o persoană care la rândul ei are exact doi părinţi. Pentru a păstra atât datele despre persoane, cât şi relaţiile directe de rudenie copil-părinţi, se construieşte arborele de familie – arborele genealogic (fig. 20 d).

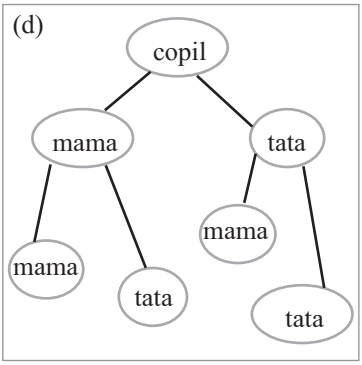


Figura 20

Fiecare dintre exemplele propuse necesită organizarea datelor după un model propriu: (1) *modelul firului de așteptare*; (2) *modelul stivei*; (3) *modelul listei circulare*; (4) *modelul arborescent*.

Implementarea acestor modele într-un limbaj de programare necesită soluționarea următoarelor probleme:

– organizarea datelor după semnificația acestora, cel mai frecvent în structuri omogene: tablouri unidimensionale;

– așezarea (intrarea) unui element din structură după regula specifică modelului;

– scoaterea (ieșirea) unui element din structură după regula specifică modelului;

– accesul la elementele structurii (numărarea/listarea elementelor) după regula specifică modelului.

În fiecare dintre situațiile reale din exemplele prezentate, numărul elementelor variază în timp: oricând poate să apară un solicitant pentru serviciul Internet sau o versiune nouă a programului antivirus; și în grupul de copii poate intra sau poate pleca un copil; în orice familie, copiii devin la rândul lor părinți, și arborele genealogic crește.

Întrucât numărul elementelor nu este constant și nici nu poate fi precizat în timp, structurile de date folosite pentru implementarea acestor modele se numesc *structuri dinamice*. Variația în timp a numărului de elemente (aspectul dinamic al structurii) respectă relațiile și disciplina (regulile de comportament) specifice modelului. Implementarea structurilor dinamice prin memorarea acestora în tablouri unidimensionale folosește *alocarea statică* a memoriei (mecanism de alocarea a memoriei din segmentul de date prin care zona de memorie maxim alocată – corespunzător capacității tabloului – rămâne la dispoziția programului pe toată durata de execuție a acestuia).

Aspectul dinamic al structurilor de date este pus și mai bine în evidență în modul de *alocare dinamică* a memoriei: mecanism de alocarea a memoriei de tip *Heap* prin care zonele de memorie pot fi solicitate și eliberate chiar în timpul execuției programului.

TEME

1. Formulați un exemplu real care să necesite organizarea datelor într-un model de tip *fir de așteptare*. Puneți în evidență aspectul dinamic al structurii.

2. Formulați un exemplu real care să necesite organizarea datelor într-un model de tip stivă. Puneți în evidență aspectul dinamic al structurii.

3. Formulați un exemplu real care să necesite organizarea datelor într-un model de tip *listă circulară*. Puneți în evidență aspectul dinamic al structurii.

4. Formulați un exemplu real care să necesite organizarea datelor într-un model de tip *arborescent*. Puneți în evidență aspectul dinamic al structurii.

5. Asociați modelul dinamic corespunzător fiecăreia dintre următoarele situații:

a) organizarea calculatoarelor într-o rețea locală de tip inel;

b) organizarea aplicațiilor deschise de un utilizator în sistemul de operare Windows;

c) organizarea informațiilor pe discul sistem;

d) organizarea cererilor de listare la imprimantă;

e) organizarea instrucțiunilor unui program.

ARBORELE GENEALOGIC
temă de compoziţie

Realizați arborele genealogic personal printr-o prezentare cât mai atractivă care să pună în evidența personalitatea fiecărui membru al familiei.

Se poate lucra în oricare dintre aplicațiile cu efecte grafice cunoscute.

Sugestie de rezolvare:

– prezentarea acestei teme de către elevi, în laborator, conduce la o activitate foarte atractivă; se pot face „clasamente": cel mai „înalt" arbore; cel mai „vârstnic" arbore, cel mai „stufos" arbore.

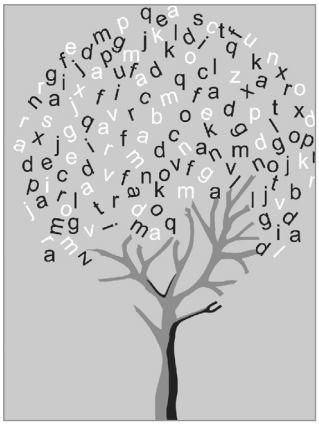
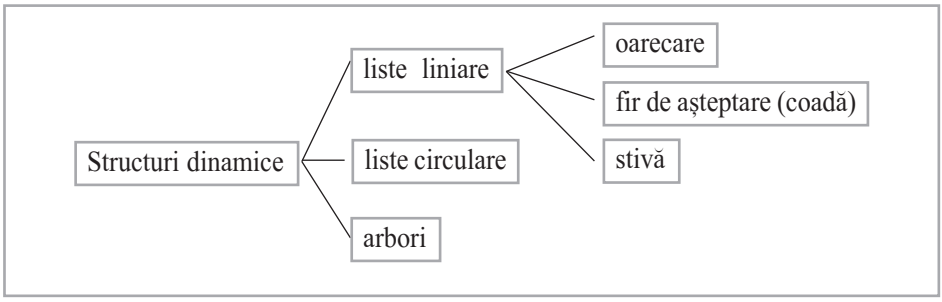


Figura 21

7.2. Clasificarea structurilor dinamice

Relațiile dintre elemente determină mai multe categorii de structuri dinamice prezentate în clasificarea din figura 22.

Figura 22. Clasificarea structurilor dinamice



Fiecare dintre categoriile de structuri dinamice din această clasificare are proprietăți specifice (Tabelul 16).

Tabelul 16. **Proprietățile structurilor dinamice**

STRUCTURA DINAMICĂ	PROPRIETĂȚI
Listă liniară	Relațiile dintre elementele structurii sunt de tip succesor – predecesor; există un singur element fără predecesor, capul listei, și un singur element fără succesor, ultimul element din listă.
Listă oarecare	Nu există nicio regulă pentru intrarea și ieșirea elementelor din structură.
Fir de așteptare	Intrarea și ieșirea elementelor din structură se face după regula FIFO (First Input First Output = *primul intrat, primul ieșit*).

Stivă	Intrarea și ieșirea elementelor din structură se face după regula LIFO (Last Input First Output = *ultimul intrat, primul ieșit*).
Listă circulară	Relațiile dintre elementele structurii sunt de tip succesor – predecesor; nu există niciun element fără predecesor sau fără succesor.
Arbori	Relațiile dintre elementele structurii sunt de tip ascendent – descendent; există un singur element fără ascendent – rădăcina arborelui – și unul sau mai multe elemente fără descendenți – elementele terminale sau frunze.

TEME

1. Identificați tipul listei liniare care poate fi asociat următoarelor situații:

a) Profesorul diriginte întocmește lista cu elevii care vor participa la o excursie cu număr limitat de locuri (mai puține decât efectivul clasei).

b) Întrucât numărul elevilor care doresc să meargă în excursie este mult mai mare decât numărul de locuri, profesorul diriginte reface lista pentru a-i elimina mai ușor pe cei care s-au înscris mai târziu.

c) La ora de Educație Fizică, elevii intră pe rând din vestiar în sala de sport; fiecare elev trebuie să se alinieze, ocupându-și locul astfel încât, în fiecare moment, șirul elevilor prezenți în sală să fie ordonat descrescător după înălțime.

2. Se consideră o listă liniară oarecare cu *n* elemente (n>10).

Determinați valoarea următoarelor expresii:

a) succesor (element 4) = b) predecesor (element 9) =
c) succesor (element n) = d) predecesor (element n) =
e) succesor (element 1) = f) predecesor (element 1) =
g) succesor (predecesor (n)) = h) predecesor (predecesor (3)) =

3. Stabiliți relația prin care o listă liniară oarecare cu *n* elemente poate fi transformată într-o listă circulară.

4. 4.1. Pentru fiecare membru din arborele genealogic personal, determinați:

a) numărul descendenților;

b) numărul ascendenților;

c) numărul elementelor terminale (fără descendenți).

4.2. Precizați care este semnificația elementelor terminale din arborele genealogic personal.

4.3. Care este semnificația elementului rădăcină din arborele genealogic personal?

4.4. Cum ar trebui construit arborele genealogic personal, astfel încât autorul să fie un element terminal?

7.3. Prelucrări specifice structurilor dinamice liniare

Pentru organizarea datelor în structuri dinamice liniare, sunt necesare următoarele prelucrări:

● **Crearea** structurii dinamice: această prelucrare corespunde memorării datelor pentru primul element din structură.

● **Parcurgerea** structurii dinamice: această prelucrare permite localizarea fiecărui element din structură, respectându-se regulile de ordine specifice modelului de structură dinamică.

● **Actualizarea** structurii dinamice: această prelucrare permite modificarea numărului de elemente din structură prin adăugare sau inserare de elemente noi sau prin eliminarea unor elemente; tot prin actualizare, se pot modifica informațiile specifice unui element.

În Tabelul 17 sunt descrise operațiile necesare implementării acestor prelucrări.

Tabelul 17. **Structuri dinamice – prelucrări și operații specifice**

<table>
<tr><th colspan="2" rowspan="2">PRELUCRARE</th><th rowspan="2">OPERAȚII SPECIFICE</th><th colspan="2">STAREA STRUCTURII
(numărul n de elemente)</th></tr>
<tr><th>ÎNAINTE DE PRELUCRARE</th><th>DUPĂ PRELUCRARE</th></tr>
<tr><td colspan="2">CREARE</td><td>– se verifică dacă structura este vidă
– se memorează datele pentru primul element din structură</td><td>dacă n=0
(structura este vidă)</td><td>atunci
n=1
altfel
operație fără sens</td></tr>
<tr><td rowspan="2">PARCURGERE</td><td>Totală</td><td>– localizarea tuturor elementelor</td><td colspan="2" rowspan="2">Nu se modifică nici numărul, nici valorile elementelor.</td></tr>
<tr><td>Parțială</td><td>– localizarea elementelor care îndeplinesc o condiție specificată</td></tr>
<tr><td rowspan="3">ACTUALIZARE</td><td>Adăugare / Inserare</td><td>– se verifică dacă s-a completat capacitatea structurii
– se memorează datele pentru un element nou care intră în structură respectând regula de intrare specifică modelului.</td><td>dacă
n=capacitatea structurii</td><td>atunci
operație imposibilă
altfel
n+1 elemente</td></tr>
<tr><td>Ștergere</td><td>– se verifică dacă structura este vidă
– se elimină din structură un element respectând regula de ieșire specifică modelului.</td><td>dacă
n= 0
structura este vidă</td><td>atunci
operație imposibilă
altfel
n-1 elemente</td></tr>
<tr><td>Modificare</td><td>Se localizează (prin parcurgere parțială) elementul ale cărui valori trebuie modificate; se modifică valorile elementului localizat.</td><td colspan="2">Nu se modifică numărul de elemente din structură; se modifică valorile pentru unul sau mai multe elemente.</td></tr>
</table>

TEME

1. Precizați de ce este necesară cunoașterea capacității unei structuri dinamice în varianta implementării prin alocarea statică a memoriei.

2. Identificați situațiile în care pot fi date următoarele mesaje:
a) *operație fără sens, listă existentă;*
b*) operație imposibilă, listă vidă;*
c) operație imposibilă, element inexistent;
d*) nu este permisă operația de inserare.*

3. Alcătuiți câte o secvență de operații elementare, necesară fiecăreia dintre următoarele prelucrări:
a) afișarea numărului de elemente dintr-o listă oarecare;
b) intrarea unui element nou într-un fir de așteptare;
c) afișarea numărului de ordine al elementelor care respectă o condiție specificată;
d) verificarea existenței în listă a unui element care respectă o condiție specificată;
e) ieșirea unui element dintr-o stivă.

7.4. Implementarea structurilor dinamice liniare

7.4.1. FIRUL DE AȘTEPTARE (COADA)

Disciplina structurii dinamice *fir de așteptare* sau *coadă* este de tip FIFO (First Input First Output = *primul intrat, primul ieșit*). Implementarea într-un limbaj de programare a acestui model de structură dinamică revine la controlul operațiilor de intrare/ieșire în/din structură, astfel încât să fie respectată disciplina FIFO.

În acest scop, structura va fi controlată prin:

(1) doi marcatori (indici) de poziție pe care îi vom numi *primul* și *ultimul* (*fig. 22).*

Figura 22

(2) *capacitatea structurii* (numărul maxim de elemente alocate pe care îl vom nota cu *n*).

Cazuri particulare

Dacă firul de așteptare nu conține niciun element, firul este gol (*fir vid*).

Dacă a fost ocupată toată capacitatea structurii, *firul este plin*.

Prelucrările specifice firului de așteptare

- **crearea** are sens doar dacă firul este gol (fir vid);
- **intrarea** unui element nou:

– un element poate intra în structură, doar dacă nu a fost completată capacitatea structurii („mai sunt locuri libere”);

– întotdeauna, noul element se așază la sfârșitul structurii, el devine *ultimul*;

- **ieșirea** unui element (întrucât firele de așteptare/coada se formează pentru satisfacerea unor cereri, „servicii”, spunem că ieșirea din fir are loc când *primul* element a fost *servit*):

– un element poate ieși din structură, doar dacă firul nu este vid (există cel puțin un element);

– întotdeauna, după ieșirea unui element din structură, toate elementele se deplasează „în față”; coada „avansează” și elementul ajuns pe poziția *primul* poate fi *servit (fig. 23);*

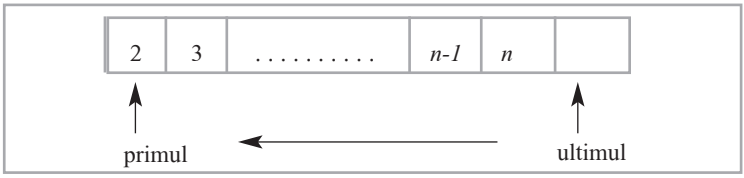


Figura 23

- **parcurgerea** se poate efectua doar dacă firul nu este vid:

– parcurgerea se poate face pentru determinarea numărului de elemente „așezate la coadă”, pentru afișarea elementelor sau pentru determinarea unor proprietăți;

– parcurgerea se face întotdeauna de la *primul* la *ultimul* element.

Elementele firului de așteptare pot fi persoane, obiecte, procese care așteaptă satisfacerea unor cereri; exemple:

– persoanele înscrise pe o listă de așteptare pentru rezervări de bilete, acceptarea la un interviu etc.;

– persoanele care așteaptă la rând pentru efecturea unor plăți, cumpărarea de produse etc.;

– mașinile aflate într-un spațiu de parcare la o benzinărie;

– evenimentele planificate într-o locație (concerte, concursuri etc.).

În cele mai frecvente situații, se înregistrează mai multe informații despre elementele firului de așteptare. Fiecare element din fir este descris ca o structură eterogenă – articol; firul de așteptare va fi memorat într-un vector de articole.

Pentru început, vom considera că fiecare element este descris prin numărul său de ordine (vector cu valori întregi, pozitive).

În continuare, se prezintă secvența pseudocod a operațiilor necesare implementării unui fir de așteptare.

```
început fir _de_ așteptare_1
// se lucrează  cu vectorul F pentru care se alocă 100 de elemente
//  variabile de lucru:
// n numărul maxim de elemente  din fir (capacitatea firului )
// primul    ultimul  marcatori de poziție
//  i indicele de adresă pentru F
// nr  numărul de elemente din fir

// secvența de inițializare
// secvența poate fi completată cu validarea lui n fața de numărul de elemente alocate (100)
scrie introduceti valoare pentru capacitatea firului de asteptare, n=
citește n
pentru i=1 la n execută
F[i] ← 0
sfârșit pentru
 primul←0     // fir vid; se consideră că adresarea elementelor din vector începe de la 1
 ultimul←0
// sfârșit secvența de inițializare

// secvența pentru creare
dacă  primul = 0
atunci
  bloc
            primul←1
            scrie introduceti valoare pentru primul element F[primul]=
            citește F[primul]
            ultimul← primul
   sfârșit  bloc
altfel
       scrie    operatie fara sens: firul nu este vid
sfârșit dacă
//sfârșit  secvența creare

// secvența pentru intrarea unui element nou
ultimul ← ultimul+1
dacă ultimul < =n
       atunci
                    bloc
             scrie introduceti valoare pentru noul  element F[ultimul]=
citește F[ultimul]
             sfârșit  bloc

           altfel
                     scrie    operatie imposibila:  firul este plin
sfârșit dacă
//sfârșit  secvența  pentru intrarea unui element nou
```

```
// secvența pentru ieșirea (servirea) unui element
dacă primul =o
      atunci
                  scrie   operatie imposibila:  firul este gol
         altfel
               bloc
    scrie   este servit elementul F[primul]

//  coada avansează – secvența A          |  secvența B
pentru i= primul  la ultimul-1 execută    |  dacă primul < ultimul
        F [i] ← F[i+1]                    |     atunci
 sfârșit   pentru                         |  F[primul] ← 0
F[ultimul] ← 0                            |  primul ← primul +1
ultimul ← ultimul -1                      |  sfârșit  dacă
              sfârșit  bloc
sfârșit dacă
//sfârsit  secvența  pentru ieșirea unui element

//secvența pentru parcurgerea firului: se afișează elementele din șir și lungimea firului
dacă primul =o
      atunci
                  scrie   operatie imposibila:  firul este gol
         altfel
            bloc
                  nr ← 0
                  pentru i= primul  la ultimul execută
                        bloc
                              scrie  F [i]
                              nr ←  nr + 1
                         sfârșit  bloc
                  sfârșit pentru
                 scrie  firul contine nr elemente
             sfârșit  bloc
sfârșit dacă
//sfârșit  secvența  pentru  parcurgerea firului de așteptare

sfârșit fir _de_ așteptare_1
```

Pentru simularea aspectului dinamic al firului de așteptare, sugerăm implementarea prelucrărilor specifice printr-un program cu meniu (fig. 24).

```
început   fir _de_ așteptare_2
//se afișează opțiunile din meniu
optiune ← 0
repetă
// se sterge ecranul
// afișare meniu
     repetă
scrie     fir de asteptare –prelucrări specifice
scrie  1 creare
scrie  2 intrarea unui  element nou
scrie  3 iesirea unui element
scrie  4 parcurgere
scrie   5 sfarsit program

scrie  introduceti optiunea (1,2,3,4,5):
citeste optiune
      până când  optiune <=5    // se validează optiunea
   selectează  optiune
optiune=1
              // secvență creare
optiune=2
             // secvență  intrare element nou
optiune=3
            // secvență ieșire element
optiune=4
           // secvență parcurgere
optiune=5
         scrie    sfarsit program
    sfârșit selectează
până când optiune=5

sfârșit fir _de_ așteptare_2
```

Meniu *FIFO*

1. Creare
2. Intrare
3. Servire
4. Parcurgere
5. Exit

Optiune _

Figura 24

Printr-o singură rulare a programului, utilizatorul poate să aleagă din meniu, de mai multe ori, opțiunea corespunzătoare simulării firului de așteptare.

Spre exemplu, pentru simularea intrării primului element urmată de intrarea a încă trei elemente și ieșirea a două elemente, utilizatorul poate dirija execuția programului prin următoarea secvență de opțiuni: 1, 4, 2, 4, 2, 4, 2 ,4, 3, 4, 3, 4, 5.

După fiecare opțiune care simulează dinamica structurii (creare, intrare, ieșire), s-a ales parcurgerea (opțiunea 4) prin care se afișează elementele din fir. În acest mod, utilizatorul poate controla dacă programul modelează corect firul de așteptare.

TEME

1. Codificați, în limbajul de programare studiat, fiecare dintre secvențele pseudocod propuse pentru simularea unui fir de așteptare.

2. Precizați care este efectul următoarei secvențe de opțiuni :
a) crearea și parcurgerea firului de așteptare;
b) crearea, intrarea unui nou element și parcurgerea firului de așteptare;
c) crearea, ieșrea unui element și parcurgerea firului de așteptare;
d) ieșirea unui element din firul de așteptare.

3. Care este efectul eliminării din program a secvenței de instrucțiuni pentru crearea firului de așteptare?

4. Compuneți un program nou care să nu conțină secvența de creare.

5. Pentru ieșirea unui element din coadă, au fost propuse două secvențe care simulează „avansul": secvența A și secvența B. Urmăriți secvența B și răspundeți la următoarele întrebări:
a) Care este semnificația momentului *primul = ultimul*?
b) Cum pot fi utilizate locațiile de memorie eliberate prin ieșirea unui element?

6. Realizați un program cu meniu pentru modelarea următoarei situații:
Mai multe persoane se înscriu pentru rezervarea de bilete la un spectacol la care au acces doar elevii. Se cunosc numărul de bilete și anul nașterii fiecărei persoane înscrisă pe listă. Organizatorii doresc să afle numărul de persoane înscrise, câți elevi s-au înscris, câte bilete disponibile mai sunt.

7. Propuneți o situație reală a cărei rezolvare cu calculatorul să necesite modelarea datelor prin fire de așteptare.

7.4.2. STIVA

Disciplina structurii dinamice numită *stivă* este de tip LIFO (Last Input First Output =*ultimul intrat, primul ieșit*). Implementarea într-un limbaj de programare a acestui model de structură dinamică revine la controlul operațiilor de intrare/ieșire în/din structură, astfel încât să fie respectată disciplina LIFO.

În acest scop, structura va fi controlată prin:

♦ doi marcatori (indici) de poziție pe care îi vom numi *bază* și *vârf* (fig. 25);

♦ *capacitatea structurii* (numărul maxim de elemente alocate pe care îl vom nota cu *n*).

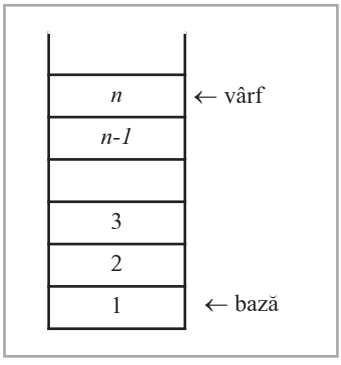


Figura 25

Cazuri particulare

- Dacă stiva nu conține niciun element, stiva este goală (*stivă vidă*).
- Dacă a fost ocupată toată capacitatea structurii, *stiva este plină*.

Prelucrările specifice stivei

▸ **crearea** are sens doar dacă stiva este goală (stivă vidă);

▸ **intrarea** unui element nou:

– un element poate intra în structură doar dacă nu a fost completată capacitatea structurii („mai sunt locuri libere");

– întotdeauna, noul element se așază peste celelalte elemente, în vârful stivei;

– după intrarea unui element nou în stivă, spunem că „stiva crește";

▸ **ieșirea** unui element:

– un element poate ieși din structură, doar dacă stiva nu este vidă (există cel puțin un element);

– întotdeauna, iese din structură elementul aflat în vârful stivei;

– după ieșirea unui element din stivă, spunem că „stiva scade ";

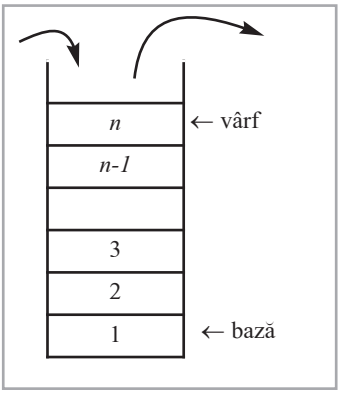


Figura 26

▸ **parcurgerea** se poate efectua doar dacă stiva nu este vidă:

– parcurgerea se poate face pentru determinarea numărului de elemente „stivuite", pentru afișarea elementelor sau pentru determinarea unor proprietăți;

– parcurgerea se face întotdeauna de la *vârf* spre *bază*.

Elementele stivei pot fi persoane, obiecte, procese care așteaptă rezolvarea unei solicitări după regula LIFO; exemple:

– persoanele propuse pentru disponibilizare de la un loc de muncă al cărui manager ține seama de vechimea angajaților;

– vagoanele de tren aflate în probă pe o linie de manevră;

– programele activate de un utilizator *Windows*.

În cele mai frecvente situații, se înregistrează mai multe informații despre elementele aflate în stivă. Fiecare element din stivă este descris ca o structură eterogenă – articol; pentru memorarea elementelor din stivă se va folosi un vector de articole.

Pentru început, vom considera că fiecare element este descris prin numărul său de ordine (vector cu valori întregi, pozitive). În continuare, se prezintă secvența pseudocod a operațiilor necesare implementării stivei.

```
început stiva1
// se lucrează cu vectorul S pentru care se alocă 100 de elemente
//  variabile de lucru:
// n numărul maxim de elemente din stivă (capacitatea stivei )
// baza  vârf  marcatori de poziție
//  i indicele de adresă pentru S
// nr  numărul de elemente din stivă

// secvența de inițializare
// secvența poate fi completată cu validarea lui n față de numărul de elemente alocate (100)

scrie introduceti valoare pentru capacitatea stivei, n=
citește n
pentru i=1 la n execută
S[i] ← 0
sfârșit pentru
 //   stiva  este vidă; se consideră că adresarea elementelor din vector începe de la 1
vârf←0
 baza←0
//  sfârșit secvența de inițializare

// secvența pentru creare
dacă  vârf = 0
atunci
  bloc
            vârf←1
scrie introduceti valoare pentru elementul din varful stivei S[vârf]=
citește S[vârf]
   sfârșit bloc
altfel
       scrie    operatie fara sens:stiva  nu este vida
sfârșit dacă
//sfârșit  secvența creare
```

```
// secvența pentru intrarea unui element nou
vârf ← vârf+1   //  stiva crește
dacă vârf < =n
      atunci
              bloc
          scrie introduceti valoare pentru elementul din varful stivei S[vârf]=
           citește S[vârf]
          sfârșit bloc

         altfel
                  scrie    operatie imposibila: stiva este plină
sfârșit dacă
//sfârșit  secvența  pentru intrarea unui element nou
// secvența pentru ieșirea unui element
dacă vârf =0
      atunci
                  scrie    operatie imposibila: stiva  este goala
         altfel
               bloc
 scrie   iese elementul din varful stivei S[varf]
                  //  stiva scade
S[varf] ←0
varf ← varf – 1
              sfârșit bloc
sfârșit dacă
//sfârșit  secvența  pentru ieșirea unui element

//secvența pentru parcurgerea stivei:
//se afișează elementele din stivă și  înălțimea stivei
dacă vârf =o
      atunci
                  scrie    operatie imposibila: stiva este goala
         altfel
            bloc
                  nr ← 0
                  pentru i= vârf  la baza  execută
                          bloc
                              scrie  S [i]
                                nr ← nr + 1
                           sfârșit  bloc
                  sfârșit pentru
                 scrie  stiva  contine nr elemente
             sfârșit  bloc
sfârșit dacă
//sfârșit  secvența  pentru  parcurgerea stivei
sfârșit stiva1
```

Pentru simularea aspectului dinamic al stivei, sugerăm implementarea prelucrărilor specifice printr-un program cu meniu (fig. 27).

```
început   stiva2
//se afișează opțiunile din meniu
optiune←0

repetă
// se șterge ecranul
// afișare meniu
     repetă
scrie    operatii asupra stivei
scrie  1 creare
scrie  2 intrare element nou
scrie  3 iesirea unui element
scrie  4 parcurgere
scrie   5 sfarsit program

scrie  introduceti optiunea (1,2,3,4,5):
citește optiune
      până când  optiune <=5    // se validează optiunea

   selectează  optiune
optiune=1
              // secventa creare
optiune=2
             // secventa  intrare element nou
optiune=3
            // secventa iesire element
optiune=4
           // secventa parcurgere
optiune=5
         scrie    sfarsit program
    sfârșit selectează
până când optiune=5
sfârșit stiva2
```

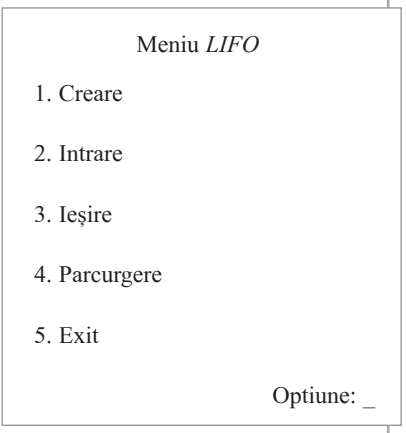


Figura 27

- Printr-o singură rulare a programului, utilizatorul poate să aleagă din meniu, de mai multe ori, opțiunea corespunzătoare simulării stivei.
- Spre exemplu, pentru simularea intrării primului element urmată de intrarea a încă trei elemente și ieșirea a două elemente, utilizatorul poate dirija execuția programului prin următoarea secvență de opțiuni: 1, 4, 2, 4, 2, 4, 2, 4, 3, 4, 3, 4, 5.

• După fiecare opțiune care simulează dinamica structurii (creare, intrare, ieșire) s-a ales parcurgerea (opțiunea 4) prin care se afișează elementele stivei. În acest mod, utilizatorul poate controla dacă programul modelează corect stiva.

TEME

1. Codificați, în limbajul de programare studiat, fiecare dintre secvențele pseudocod propuse pentru simularea stivei.

2. Precizați care este efectul următoarei secvențe de opțiuni:
a) crearea și parcurgerea stivei;
b) crearea, intrarea unui nou element și parcurgerea stivei;
c) crearea, ieșirea unui element și parcurgerea stivei;
d) ieșirea unui element din stivă.

3. Precizați care este efectul eliminării din program a secvenței de instrucțiuni pentru crearea stivei.

4. Compuneți un program nou care să nu conțină secvența de creare a stivei.

5. Care este rolul marcatorului de poziție *baza* stivei?

6. Compuneți un program nou în care marcatorul *baza* să aibă valoarea *n* (capacitatea stivei).

7. Realizați un program cu meniu pentru modelarea următoarei situații:
La biblioteca școlii, cărțile de Informatică sunt atât de solicitate, încât bibliotecara nu le mai pune la loc în raft decât la sfârșitul zilei; în rest, le așază una peste alta pe masă; s-a constatat că nici elevii nu selectează cărțile și împrumută de fiecare dată cartea aflată „la vedere" (deasupra celorlalte cărți). Teancul de cărți nu poate depăși *n* exemplare ($n \leq 25$). Se cunoaște autorul, anul editării și prețul fiecărei cărți. Dorim să aflăm, la cerere, numărul cărților, valoarea acestora precum și numărul cărților împrumutate într-o zi. (Execuția programului va simula activitatea de restituire și împrumut pe parcursul unei zile.)

8. Propuneți o situație reală a cărei rezolvare cu calculatorul să necesite modelarea datelor prin structuri dinamice de tip stivă.

PROBLEME PROPUSE

1. La un cabinet medical, pacienții intră la consultație în ordinea sosirii. Despre fiecare pacient se cunoaște anul nașterii, genul (M/F), înălțimea și greutatea. Realizați un program care să simuleze înregistrarea pacienților în lista de așteptare, modificarea listei după fiecare consultație și determinarea următoarelor informații:
a) numărul pacienților care așteaptă pentru consultație;
b) numărul pacienților bărbați care așteaptă pentru consultație;
c) numărul pacienților supraponderali care așteaptă pentru consultație (greutatea ≥ 80 Kg și înălțimea ≤1,70m).

2. Pentru a rezolva mai repede solicitările unor clienți nemulțumiți de serviciile primite, *Oficiul pentru protecția consumatorilor* a decis să înființeze, pe lângă ghișeul de lucru *G1*, un ghișeu nou, *G2*. Coada formată la ghișeul *G1* este reorganizată astfel: clienții cu număr de ordine *par* trec la ghișeul *G2*.

Realizați un program pentru simularea celor două cozi: *G1* și *G2* pornind de la o coadă inițială, *G1*. Stabiliți singuri datele memorate pentru fiecare client.

3. Realizați un program pentru adunarea a două numere foarte mari. Justificați structura dinamică folosită.

4. Realizați un program pentru afișarea în ordine inversă a unui text introdus de la tastatură (exemplu: textul introdus este *mai mult ca perfectul* textul afișat este *lutcefrep ac tlum iam*). Justificați structura dinamică folosită.

5. Realizați un program pentru transformarea unui număr din baza 10 în baza 2. Justificați structura dinamică folosită.

6. Se consideră formate două cozi C1și C2; în cele două cozi sunt memorate valori numerice. Să se formeze o coadă nouă, C3, prin așezarea elementelor din coada C2 după elementele cozii C1.

Exemplu:

Coada C1 este: 7, 3, 6, 5, 8, 9, 2.
Coada C2 este: 6, 8, 1, 2, 5, 1, 3, 5, 9.
Se obține coada C3: 7, 3, 6, 5, 8, 9, 2, 6, 8, 1, 2, 5, 1, 3, 5, 9.

7. Se consideră formate două stive S1 și S2; în cele două stive sunt memorate valori numerice. Să se formeze o stivă nouă, S3, prin așezarea elementelor din stiva S2 peste elementele stivei S1.

Exemplu:

Stiva S1 este: 7, 3, 6, 5, 8, 9, 2.
Stiva S2 este: 6, 8, 1, 2, 5, 1, 3, 5, 9.
Se obține stiva S3: 6, 8, 1, 2, 5, 1, 3, 5, 9, 7, 3, 6, 5, 8, 9, 2.

8. Realizați un program care să prelucreze o secvență de comenzi pentru un calculator de buzunar; prima comandă executată este prima comandă din secvență. Fiecare comandă are următoarea structură:

operator1 operand operator2

unde: *operator1, operator2* sunt valori numerice reale iar *operand* poate avea valorile *S*-sumă, *D*–diferență *P*–produs, *E*–sfârșitul secvenței de comenzi (*sfârșit prelucrare*).

Exemplu:

Secvența de comenzi	rezultate
3.4 S 3 25.5 D 20.5 12 P 4 E	6.4 5 48 *sfârșit prelucrare*

9. Un șir de caractere conține duplicate în serie (caractere de același fel care se repetă unul după altul). Se dorește rafinarea șirului prin eliminarea duplicatelor în serie. Șirul vid este șir rafinat.

Exemplu:
Șirul inițial este: 912222223444355aa16777777800095.
Șirul final este: 96895.

10. Într-un grup de *n* copii există relații de prietenie; fiecare copil este identificat prin numărul său de ordine în grup; fiecare copil poate fi prieten cu mai mulți copii din grup. Profesorul diriginte dorește să așeze copiii într-o ordine care să pună în evidență relațiile de prietenie, și anume: începând cu un copil oarecare *i*, acesta își strigă toți prietenii – în ordinea crescătoare a numărului; prietenii copilului *i* se așază în rând, unul după altul. Primul copil care urmează în șir după copilul *i* își strigă toți prietenii – în ordinea crescătoare a numărului; aceștia se așază și ei în rând, unul după altul (dacă un copil a fost strigat înainte, el își păstrează locul în rând).

Exemplul 1: în grup sunt 6 copii.
Relațiile de prietenie dintre copii:

Figura 28

a)

	1		1		1
1		1		1	
	1		1	1	
1		1			
	1	1			
1					

Pentru i=2, ordinea copiilor în șir este: 2, 1, 3, 5, 4, 6.

Exemplul 2: în grup sunt 6 copii.
Relațiile de prietenie dintre copii:

b)

	1				1
1		1			
	1			1	
		1	1		
1					

Pentru i=3, ordinea copiilor în șir este: 3, 2, 5, 1, 4, 6.

Cerințe

a) Specificați structurile necesare organizării datelor din această problemă.
b) Determinați situațiile particulare care pot fi întâlnite în grupul de copii.
c) Formulați exemple numerice care să pună în evidență situațiile particulare determinate.
d) Realizați un program care să afișeze aranjarea copiilor din grup în ordinea dorită de profesorul diriginte.

MINIPROIECT ÎN ECHIPĂ. COMPANIA EFICIENT

Etapa: *Analiză* (identificarea datelor și a prelucrărilor)
Cerințe:
1. Analizați datele și cerințele prezentate în studiul de caz *Compania Eficient* (pag. 3); justificați formele de organizare a datelor prin analiza comparativă după eficiența prelucrării acestora cu calculatorul.
2. Alcătuiți documentația de proiect corespunzătoare acestei etape.

Capitolul 2

SUBPROGRAME

1. UN EXEMPLU DE MODULARIZARE

Modularizarea ca metodă de rezolvare a problemelor s-a dovedit foarte utilă nu numai în programare, ci și în alte domenii ale științei și chiar în viața de zi cu zi.

Rezolvarea acestei „probleme" prin modularizare prezintă următoarele avantaje:

A. ORGANIZAREA FESTIVITĂȚII DE ABSOLVIRE A LICEULUI

Să presupunem că avem de organizat festivitatea de absolvire a liceului.

Pare mult mai greu decât în cazul organizării unei onomastici; de aceea, vom încerca să descompunem „problema" în probleme mai „mici":

(**p1**) cum vom contacta invitații?

(**p2**) cum vom organiza partea distractivă?

(**p3**) cum vom organiza partea culinară?

Problema (**p1**) se poate și ea descompune în alte trei probleme:

(**p1.1**) lista celor care pot fi anunțați telefonic;

(**p1.2**) lista celor care pot fi anunțați prin e-mail;

(**p1.3**) lista celor care pot fi anunțați prin poștă.

Problema (**p2**) se poate descompune în două subprobleme:

(**p2.1**) stabilirea scenariului, alegerea formațiilor și a soliștilor;

(**p2.2**) anunțarea formațiilor și a soliștilor invitați.

Problema (**p3**) se poate descompune la rândul ei în următoarele subprobleme:

(**p3.1**) alcătuirea meniului;

(**p3.2**) contactarea furnizorilor.

- problemele (**p1.1**), (**p1.2**) , … , (**p3.2**) sunt mult mai simple și mai ușor de rezolvat decât problema inițială; ele pot fi rezolvate și verificate – independent;
- soluția găsită pentru o anumită subproblemă poate fi folosită și pentru alte subprobleme; de exemplu: problemele (**p2.2**) și (**p2.3**) se pot rezolva la fel ca și problema (**p1**), (fig. 29); acest fapt determină o mare economie de efort și de timp de lucru;

- soluția găsită pentru o anumită subproblemă poate fi adaptată pentru rezolvarea altor probleme: de exemplu, soluția problemei (**p2.2**) poate fi adaptată și poate conduce la rezolvarea problemei (**p3.2**). Și acest fapt determină o mare economie de efort și de timp de lucru;
- având de rezolvat mai multe probleme independente – mai simple – în loc de una singură – mai complicată – soluționarea acestora se poate face independent de către diverse persoane; acest fapt determină, de asemenea, o mare economie de timp de lucru.

Figura 29 ilustrează structurarea și descompunerea problemei de mai sus în subprobleme (blocurile corespunzătoare problemelor care admit aceeași soluție au fost hașurate identic).

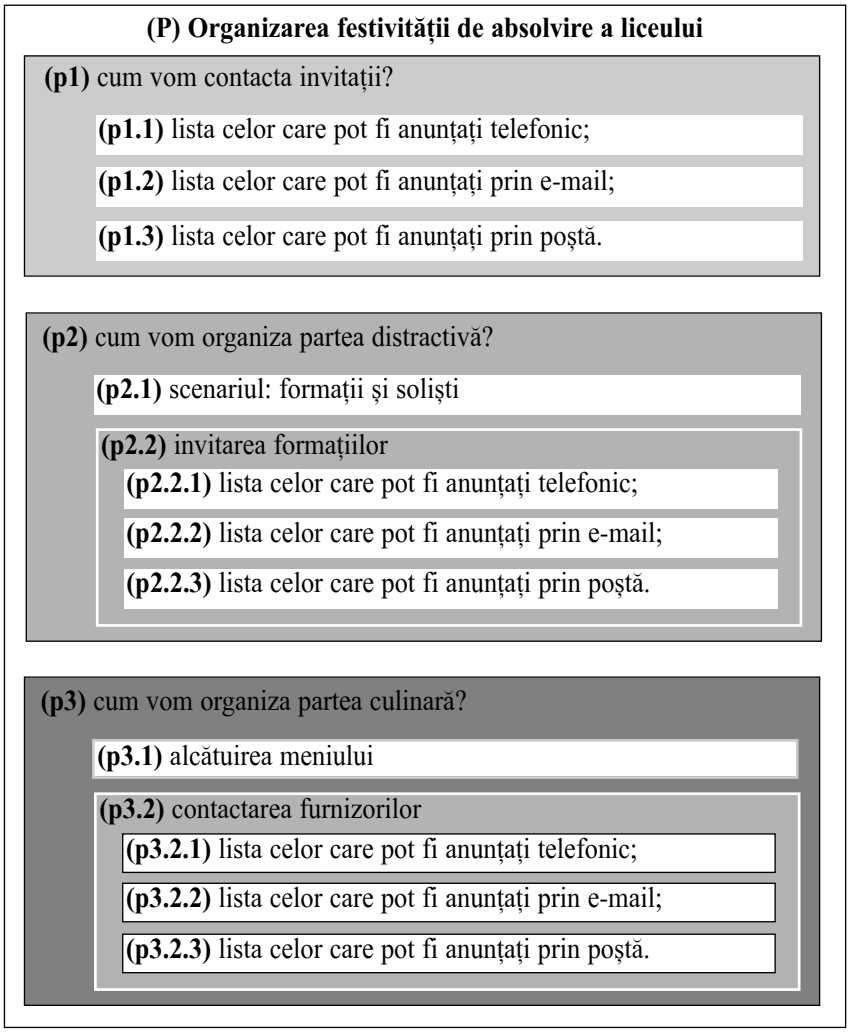


Figura 29. Descompunerea unei probleme în subprobleme

B. CUM FOLOSIM MODULARIZAREA LA NIVELUL UNUI PROGRAM?

🕮 Definiție

Un **modul de program** este o unitate de prelucrare care face parte dintr-un program.

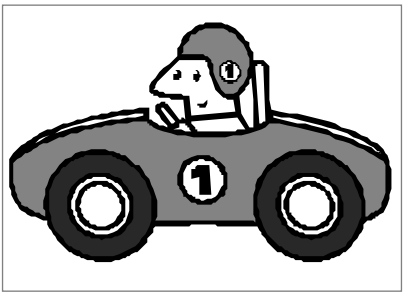
Orice program poate fi considerat o funcție în sensul matematic al cuvântului:

– programul primește date de intrare (argumente; de ex. numerele: 1500; 19,44; 13,89; 25);

– programul aplică datelor de intrare un șir de prelucrări (de ex.: înmulțirea, ridicarea la pătrat, împărțirea la 2);

– programul furnizează rezultate (în cazul nostru, valorile: 283.435; 144.699; 468.750).

„Programul" calculează energia cinetică dezvoltată de un corp de masă **m** (aici: un autoturism de 1,5 tone), atunci când viteza sa este egală cu 70, 50, respectiv 90 km/h[1] (adică atunci când circulă în localități, în afara lor, respectiv pe autostrăzi). Cu alte cuvinte, avem de a face cu calcularea energiei cinetice după relația:



$$E_C = \frac{m \cdot v^2}{2}$$ sau, cu o funcție $$f : R_+ \times R_+ \to R_+ ,\ f(m,v) = \frac{m \cdot v^2}{2}$$

Să simplificăm puțin lucrurile și să presupunem că masa corpului pentru care trebuie calculată energia cinetică este constantă, variind doar viteza. Obținem o funcție cu un singur argument:

$$f_1 : R_+ \to R_+ ,\ f_1(v) = \frac{m}{2} \cdot v^2$$

Dacă examinăm expresia funcției f_1, observăm că aceasta rezultă din compunerea a două funcții: (1) funcția h pentru ridicarea la pătrat:

$$h : R \to R ,\ h(v) = v^2$$

(2) funcția g pentru înmulțirea cu o constantă:

$$g : R \to R ,\ g(w) = k \cdot w ,\ unde\ \ k = \frac{m}{2}$$ adică:

$$f_1 : R \to R ,\ f_1 = g \circ h ,\ f_1(v) = g(h(v))$$

[1] 70km/h ≈ 19,44 m/s²; 50km/h ≈ 13,89 m/s²; 90km/h ≈ 25 m/s².

În matematică, putem obține funcții oricât de complexe prin **compunerea** funcțiilor elementare (funcția polinomială, funcția rațională, funcția de ridicare la putere sau de extragere de radical etc.).

În programare, putem construi programe oricât de complexe prin tehnica de modularizare, adică prin **înlănțuirea logică** a unor „unități elementare" de program: citirea datelor de intrare, prelucrarea lor, afișarea rezultatelor obținute în urma prelucrării datelor. Un modul de prelucrare poate rezulta din înlănțuirea logică a mai multor module: calcularea valorii minime dintr-un set de valori, calcularea valorii maxime dintr-un set de valori, calcularea mediei aritmetice a acestora etc.

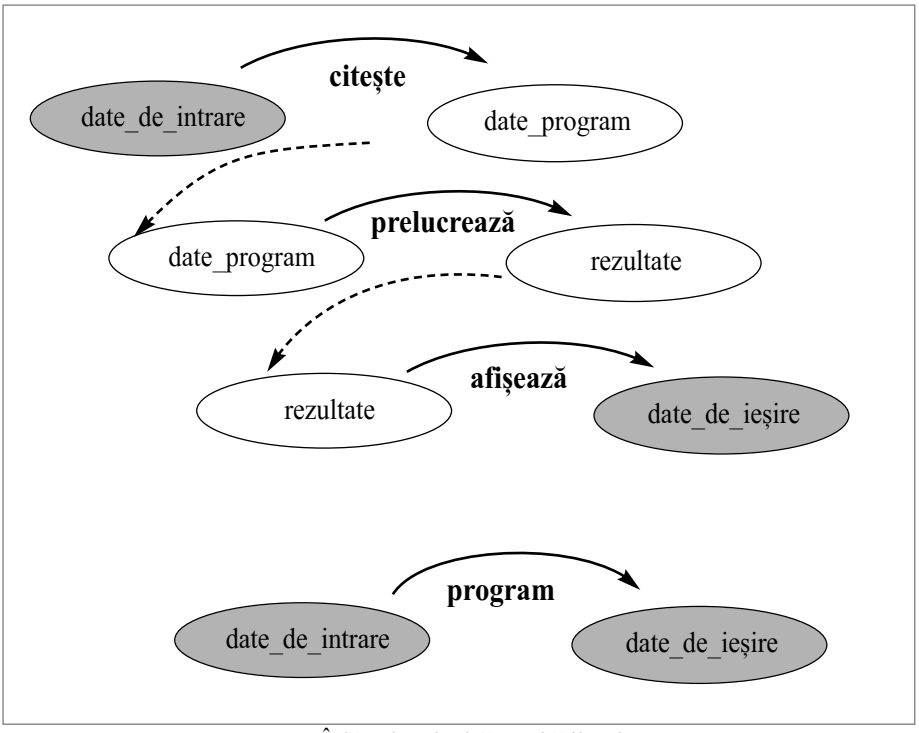


Figura 30. Înlănțuirea logică a unităților de program

Înlănțuirea logică a unităților de program, după **modelul compunerii** funcțiilor matematice, pune în evidență mulțimile de valori specifice rezolvării problemelor cu calculatorul:

Program (date_de_intrare → date_de_ieșire)
citește (date_de_intrare → date_program)
prelucrează (date_program → rezultate)
afișează (rezultate → date_de_ieșire).

2. MODULARIZAREA PROGRAMELOR

A. TIPURI DE PROBLEME (FACULTATIV)

Rezolvarea unei probleme cu ajutorul unui program de calculator este etapa finală a unui proces care depinde de gradul în care cunoaștem datele de intrare, tipul rezultatelor, metoda de rezolvare. Ținând seama de aceste aspecte, problemele cu care suntem zilnic confruntați se pot clasifica astfel:

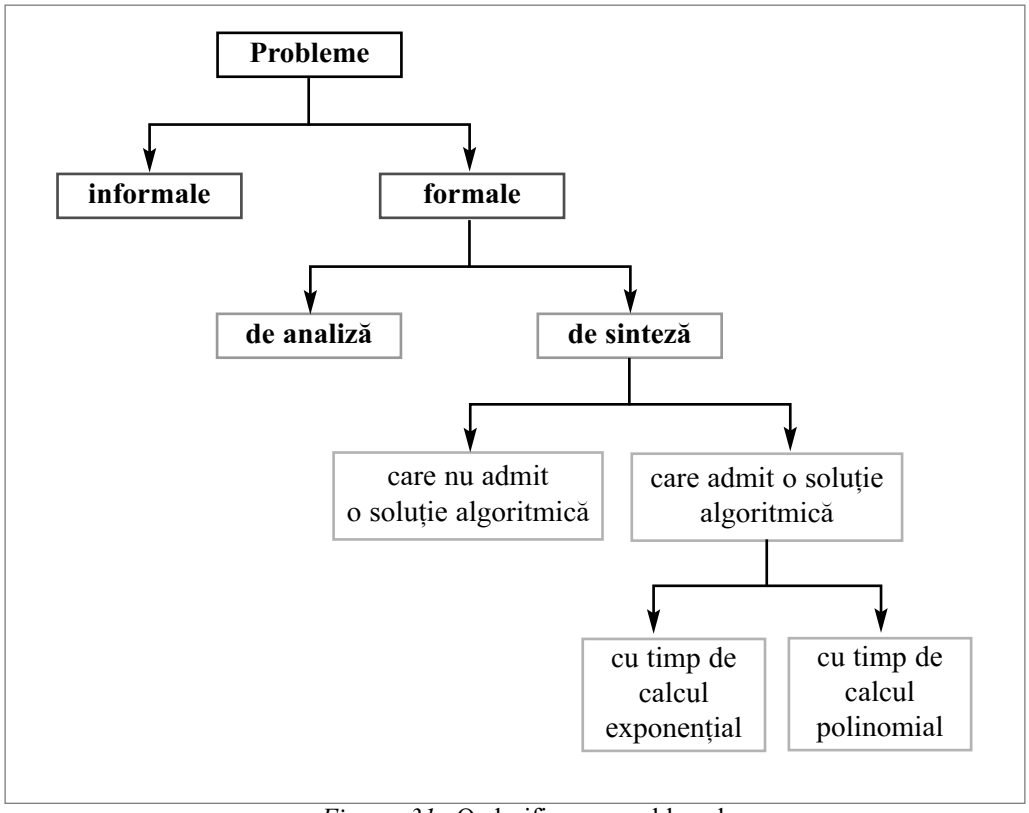


Figura 31. O clasificare a problemelor

Exemple:

• O *problemă informală* este o problemă pentru care nu știm exact de ce date dispunem, nu cunoaștem metoda de rezolvare, nu știm nici măcar cum ar trebui să arate rezultatele. O problemă informală poate fi formulată astfel: *Cum să procedez pentru a fi apreciat de cei din jur?*

• O *problemă formală de analiză* este o problemă pentru care cunoaștem mulțimea datelor inițiale și forma rezultatelor, dar nu cunoaștem metoda de rezolvare; de exemplu: ne aflăm într-un orășel portuar străin, nu cunoaștem nici măcar grafia limbii respective și dorim să ajungem de la hotel în port. Analizând problema, constatăm că nu putem apela la localnici – nici măcar la taximetriști sau hotelieri –, dar, de la lecțiile de geografie, știm că portul se află

în partea sudică a orașului respectiv. Presupunând că avem parte de o zi însorită, ne putem totuși orienta să găsim, astfel, drumul.

• O *problemă formală de sinteză* este o problemă pentru care cunoaștem mulțimea datelor inițiale și metoda de rezolvare. (Foarte multe probleme de matematică, fizică, chimie fac parte din această categorie.)

• O *problemă de sinteză care nu poate fi rezolvată algoritmic* este mai greu de explicat intuitiv. De aceea, vom recurge la prezentarea *problemei corespondenței*[1] *Post*[2]. Aceasta este o problemă de decizie nerezolvabilă algoritmic care, informal, poate fi enunțată astfel: *Fie un dicționar care conține perechi de fraze și fie două fraze oarecare; se poate decide dacă cele două fraze au același înțeles în ambele limbi?*

În continuare, vom considera o variantă mai simplă a problemei lui Post. Fie două mulțimi finite de numere scrise în baza 10, $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$. Se cere să se determine o secvență de indici $i_1, i_2, \ldots, i_k$, astfel încât numerele $x_{i_1} x_{i_2} x_{i_k}$ și $y_{i_1} y_{i_2} y_{i_k}$, obținute prin concatenare, să fie identice. Se poate demonstra că nu există un algoritm care să determine o astfel de secvență de indici, dându-se oricare două mulțimi X și Y. Spunem că problema este nedecidabilă (algoritmic). Există cazuri particulare ale problemelor nedecidabile care pot fi rezolvate cu algoritmi particulari, în funcție de exemplul tratat; schimbarea datelor de intrare implică găsirea unei alte soluții (deci a unui alt algoritm!) așa cum vom arăta imediat.

Fie mulțimile $X_1 = \{44, 48, 84, 88\}$ și $Y_1 = \{4444, 4448, 4884, 8448\}$. Problema lui Post nu are soluție pentru aceste mulțimi deoarece oricare dintre numerele din Y_1 este mai lung (are mai multe cifre) decât oricare dintre numerele din X_1.

Fie acum mulțimile $X_2 = \{121, 21, 22, 1221\}$, $Y_2 = \{1121, 12, 112, 122\}$. Problema lui Post are soluție pentru aceste mulțimi: secvența de indici este 4,2,2,1. Într-adevăr:

$$x_4x_2x_2x_1 = 12212121121 = 12212121121 = y_4y_2y_2y_1$$

$$\begin{array}{l} \overbrace{122}^{x_4}\overbrace{12}^{x_2}\overbrace{12}^{x_2}\overbrace{121}^{x_1} \\ \underbrace{122}_{y_4}\underbrace{12}_{y_2}\underbrace{12}_{y_2}\underbrace{1121}_{y_1} \end{array}$$

Ce se întâmplă cu mulțimile $X_3 = \{23, 2333, 323\}$ și $Y_3 = \{22, 32323, 33\}$? Dar cu $X_4 = \{22, 2223, 333\}$ și $Y_4 = \{22, 22233, 32\}$? Dar cu…?!

---

[1] Această problemă a fost formulată în anul 1946 și publicată în articolul "A variant of a recursively unsolvable problem", în volumul 52 al Bulletin of the American Mathematical Society. Fiind mai simplă decât problema opririi programelor (a se vedea Anexa 1), *problema corespondenței Post* este frecvent folosită pentru demonstrarea – prin reducere – a nedecidabilității unor probleme din informatică.

[2] Emil Leon POST (1897 – 1954): matematician de origine poloneză, profesor la City College of New York. În teza sa de doctorat, susținută în 1920, Post a demonstrat completitudinea și noncontradicția calculului cu propoziții – descris de B. Russell și A.N. Whitehead în "Principia Mathematica" – folosind pentru prima dată metoda tabelelor de adevăr. Este considerat părintele teoriei demonstrației, un precursor al lui Kurt Gödel (prin lucrările sale privind mulțimile recursiv enumerabile, gradele de nedecidabilitate) și John von Neumann (printr-un model matematic de mașină de calcul foarte asemănător celui descris de von Neumann în celebrul său articol din 1946).

• O *problemă formală de sinteză, care admite o soluție algoritmică dar al cărei ordin de mărime este exponențial,* este problema determinării combinărilor de **n** elemente luate câte **m**, **0 ≤ m ≤ n** folosind bine-cunoscuta proprietate a combinărilor.

• O *problemă formală de sinteză, care admite o soluție algoritmică și al cărei ordin de mărime este polinomial* (chiar liniar), este problema căutării secvențiale. Pentru un șir ordonat – dacă utilizăm metoda căutării binare, studiată în clasa a IX-a – complexitatea algoritmului de căutare devine $O(\log_2 n)$.

Transformarea unei probleme informale într-o problemă formală – mai întâi de analiză și apoi de sinteză – și, mai departe, într-o problemă rezolvabilă algoritmic este – uneori – extrem de dificilă. Nici găsirea unui algoritm de rezolvare cu complexitate polinomială, liniară sau logaritmică nu este întotdeauna… rezolvabilă algoritmic!

TEMĂ

Exemplificări

a) În ce categorie de probleme se încadrează următoarele enunțuri:
– calcularea titlului unui aliaj;
– construirea unui aeromodel;
– verificare primalității unui număr natural dat;
– acordarea unui împrumut;
– sortarea crescătoare a unui șir de numere reale;
– calculul sumei primelor ***n*** numere naturale;
– intrarea în cartea recordurilor;
– calcularea dobânzii primite pentru banii depuși la bancă;
– ordonarea elevilor din clasă după înălțime și greutate.

b) Căutați exemple din domeniile de interes școlar sau din situații reale pentru fiecare dintre categoriile de probleme prezentate.

B. MODULARIZAREA REZOLVĂRII PROBLEMELOR

În practică suntem confruntați, cel mai adesea, fie cu probleme care admit soluții algoritmice și au un grad de complexitate rezonabil, fie numai cu cazuri particulare ale unor probleme nedecidabile, pentru care putem găsi soluții convenabile. În rezolvarea acestor probleme cu ajutorul unui program, folosim modularizarea atunci când căutăm o soluție pentru fiecare dintre următoarele subprobleme (fig. 32):

(M1.) introducerea datelor de intrare (**modulul de citire a datelor**);

(M2.) prelucrarea datelor (**modulul de prelucrare**);

(M3.) transmiterea datelor de ieșire (**modulul de afișare a rezultatelor**).

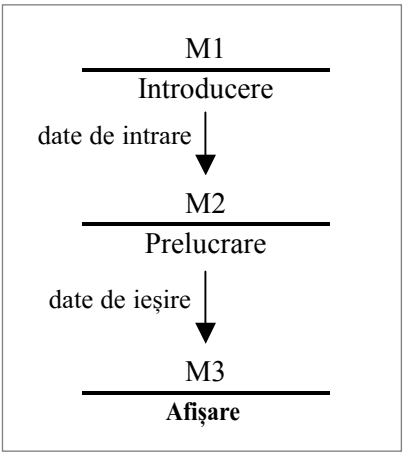


Figura 32. Modularizarea rezolvării unei probleme

Noțiunea de modul este independentă față de limbajul de programare. Un algoritm poate fi format din mai multe module (subalgoritmi). În continuare, vom folosi termenul de program, fără a implica prin aceasta folosirea unui anumit limbaj de programare.

APLICAȚIE

Problema traiectoriilor

Un tun execută ***n*** *aruncări de proiectile. Pentru fiecare aruncare se cunosc viteza inițială,* ***v****, și unghiul de tragere* α. *Trebuie determinate:*

(1) – distanța maximă,

– viteza inițială,

– unghiul de tragere pentru care distanța pe orizontală atinsă de proiectil după ***t*** *secunde este maximă;*

(2) – înălțimea minimă,

– viteza inițială,

– unghiul de tragere pentru care înălțimea pe verticală atinsă de proiectil după ***t*** *secunde este minimă.*

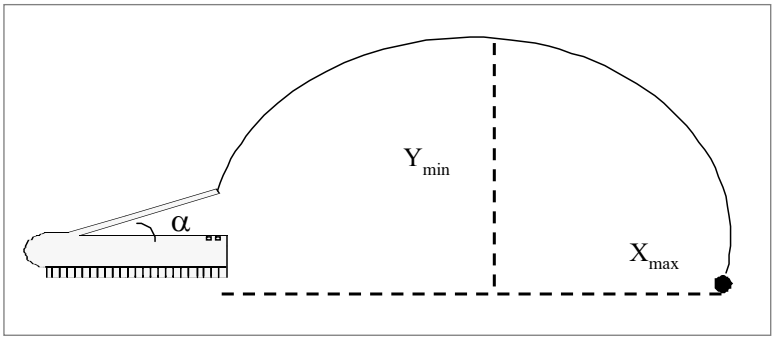


Figura 33. Aruncarea unui proiectil

1. Analiza problemei

• Date de intrare:

– numărul de aruncări (**n>1**);

– n viteze inițiale (**V**);

– n unghiuri de tragere (**A**).

3	3	5	2	4	6	2	7		

15	30	15	20	45	60	20	75		

• Date de ieșire:

– cea mai mare distanță la care a ajuns un proiectil după **t** secunde ($\mathbf{X_{max}}$);

– viteza inițială pentru care, după **t** secunde, proiectilul a atins distanța maximă ($\mathbf{V_k}$);

– unghiul de aruncare pentru care, după **t** secunde, proiectilul a atins distanța maximă ($\mathbf{A_k}$);

– cea mai mică înălțime la care a ajuns un proiectil după **t** secunde ($\mathbf{Y_{min}}$);

– viteza inițială pentru care, după **t** secunde, proiectilul a atins înălțimea minimă ($\mathbf{V_j}$);

– unghiul de aruncare pentru care, după **t** secunde, proiectilul a atins înălțimea minimă ($\mathbf{A_j}$).

• Condiții și relații importante:

Se cunoaște **t** = 10 secunde.

Se utilizează formulele de calcul pentru aflarea distanței (1) și a înălțimii (2) din traiectoria unui proiectil:

(1) $x = v \cdot t \cdot \cos\alpha,$

(2) $y = v \cdot t \cdot \sin\alpha - \frac{g \cdot t^2}{2}$

2. Raționamentul problemei

Metoda I: căutăm un algoritm cât mai eficient pentru rezolvarea problemei. Aceasta înseamnă soluționarea concomitentă a ambelor cerințe.

Pasul 1. Se citește numărul de aruncări **n.**

Pasul 2. Se citesc cele **n** perechi de viteze inițiale și unghiuri de tragere **(V_1, A_1), (V_2, A_2),…, (V_n, A_n).**

Pasul 3. Se inițializează constantele **t** = numărul de secunde și **g** = accelerația gravitațională.

Pasul 4. Se calculează distanța X_1 și înălțimea Y_1 conform formulelor (1) și (2).

Pasul 5. $X_{max} \leftarrow X_1$, **k** $\leftarrow$ **1**, $Y_{min} \leftarrow Y_1$, **j** $\leftarrow$ **1.**

Pasul 6. Se calculează o nouă pereche (X_i,Y_i).

Pasul 7. Dacă $X_i > X_{max}$ atunci $X_{max} \leftarrow X_i$, **k** $\leftarrow$ **i.**

Pasul 8. Dacă $Y_i < Y_{min}$ atunci $Y_{min} \leftarrow Y_i$, **j** $\leftarrow$ **i.**

Pasul 9. Se repetă pașii **P6, P7, P8** pentru toate perechile de valori (V_i, A_i).

Pasul 10. Se afișează U, distanța maximă X_{max} și înălțimea minimă Y_{min}, viteza V_k și unghiul de tragere A_k, viteza V_j și unghiul de tragere A_j.

3. Reprezentarea algoritmului

```
început traiectorii1
      variabile n, V, A, X, Y, Xmax, Ymin, k, j, i
t = 10
g = 9,8
citește n
      pentru  i =1 la, n  execută
                citește Vi, Ai
      sfârșit pentru
      pentru  i =1 la, n  execută
                Xi  ← Vi * t * cos(Ai)
                Yi  ← Vi * t * sin(Ai) – g * t * t / 2
      sfârșit pentru
      Xmax  ← X1
      k  ← 1
```

```
        Ymin ← Y1
        j ← 1
    pentru  i =2 la, n  execută
                    dacă  Xi > Xmax  atunci
                            bloc
                              Xmax ← Xi
                              k ← i
                            sfârșit bloc
                    sfârșit dacă
                    dacă  Yi < Ymin  atunci
                            bloc
                              Ymin ← Yi
                              j ← i
                            sfârșit bloc
                    sfârșit dacă
        sfârșit pentru
        scrie  Xmax, Vk, Ak, Ymin, Vj, Aj
    sfârșit traiectorii1
```

Metoda II: rezolvarea prin modularizare

Căutăm să descompunem problema în probleme mai simple, pentru care – eventual – avem soluții, chiar sub formă de subalgoritmi:

(M1) modulul de citire a datelor.

(M1.1) modulul de inițializare a constantelor: **t = 10** și **g = 9,8**;

(M1.2) modulul de citire a variabilelor: numărul de aruncări **n**, cele **n** viteze inițiale $\mathbf{V_1}$, $\mathbf{V_2}$,…, $\mathbf{V_n}$ și cele **n** unghiuri de tragere $\mathbf{A_1}$, $\mathbf{A_2}$,…, $\mathbf{A_n}$;

(M2) modulul de prelucrare

(M2.1) modulul de calculare a distanțelor $\mathbf{X_1}$, $\mathbf{X_2}$,…, $\mathbf{X_n}$ și a înălțimilor $\mathbf{Y_1}$, $\mathbf{Y_2}$,…, $\mathbf{Y_n}$ prin aplicarea formulelor (1) și (2);

(M2.2) modulul de calculare a valorii maxime $\mathbf{X_{max}}$ dintre $\mathbf{X_1}$, $\mathbf{X_2}$,…, $\mathbf{X_n}$ și reținere a indexului corespunzător **k**;

(M2.3) modulul de calculare a valorii minime $\mathbf{Y_{min}}$ dintre $\mathbf{Y_1}$, $\mathbf{Y_2}$,…, $\mathbf{Y_n}$ și reținere a indexului corespunzător **j**;

(M3) modulul de afișare a rezultatelor

afișarea distanței maxime $\mathbf{X_{max}}$, a vitezei inițiale și a unghiului de tragere corespunzător $\mathbf{V_k}$, $\mathbf{A_k}$; afișarea înălțimii minime $\mathbf{Y_{min}}$, a vitezei inițiale și a unghiului de tragere corespunzător $\mathbf{V_j}$, $\mathbf{A_j}$.

Raționamentul modularizat este prezentat în figura 34.

(P) Calcularea distanței maxime și a înălțimii minime atinse de un proiectil din mai multe aruncări

(M1) citirea datelor de intrare

(M 1.1) modulul de inițializare a constantelor:
t = 10
g = 9,8

(M 1.2) modulul de citire a variabilelor: numărul de aruncări **n**, cele **n** viteze inițiale $V_1, V_2, ..., V_n$ și cele **n** unghiuri de tragere $A_1, A_2, ..., A_n$;

(M2) prelucrarea datelor

(M 2.1) modulul de calculare a distanțelor: $X_1, X_2, ..., X_n$ și a înălțimilor $Y_1, Y_2, ..., Y_n$ prin aplicarea formulelor (1) și (2);

(M 2.2) modulul de calculare a valorii maxime X_{max} dintre $X_1, X_2, ..., X_n$ și reținere a indexului corespunzător **k**;

(M 2.2) modulul de calculare a valorii minime Y_{min} dintre $Y_1, Y_2, ..., Y_n$ și reținere a indexului corespunzător **j**.

(M3) afișarea rezultatelor

distanța maximă X_{max}, viteza V_k și unghiul A_k pentru care a fost atinsă;
înălțimea minimă Y_{min}, viteza V_j și unghiul A_j pentru care a fost atinsă.

Figura 34. Problema traiectoriilor – rezolvare modularizată

3. Reprezentarea algoritmului

```
început traiectorii2
    variabile n, V, A, X, Y, Xmax, Ymin, k, j, i
        t = 10
        g = 9,8
        citește n
        pentru i = 1 la n execută
                citește Vi
        sfârșit pentru
        pentru i ← 1 , n execută
                citește Ai
        sfârșit pentru
        pentru i ← 1, n execută
          Xi ← Vi*t*cos(Ai)
          Yi ← Vi*t*sin(Ai)–9,8*t*t/2
        sfârșit pentru
        Xmax ← X1
        k ← 1
        pentru i ← 2, n execută
          dacă Xi > Xmax atunci
            bloc
              Xmax ← Xi
              k ← i
            sfârșit bloc
          sfârșit dacă
        sfârșit pentru
        Ymin ← Y1
        j ← 1
        pentru i ← 2, n execută
          dacă Yi < Ymin atunci
            bloc
              Ymin ← Yi
              j ← i
            sfârșit bloc
          sfârșit dacă
        sfârșit pentru
        scrie Xmax, Vk, Ak
        scrie Ymin, Vj, Aj
sfârșit traiectorii2
```

Atenție

Programul *traiectorii 1* este realizat printr-o secvență de operații ce poate fi folosită doar pentru rezolvarea acestei probleme.

Programul *traiectorii 2*

– poate folosi module deja scrise, și anume:

- **modulul pentru determinarea valorii maxime dintr-o secvență de valori și reținere a poziției pe care apare acest maxim,**
- **modulul pentru determinarea valorii minime dintr-o secvență de valori și reținere a poziției pe care apare acest minim;**

– se poate adapta mai ușor;

– prezintă un grad mai mare de generalitate.

Următoarele exemple pun în evidență flexibilitatea oferită de modularizare.

Exemplul 1: *modulul de determinare a valorii minime dintr-o secvență de valori se poate obține din cel de determinare a valorii maxime (și reciproc) prin modificarea operatorului relațional din instrucțiunea de test*:

Calcularea minimului	Calcularea maximului
început modul minim **variabile** Z, VAL, i VAL ← Z1 **pentru** i = 2 la n **execută** **dacă** Zi < VAL **atunci** VAL←Zi **sfârșit dacă** **sfârșit pentru** **returnează** VAL **sfârșit** minim	**început modul** maxim **variabile** Z, VAL, i VAL ← Z1 **pentru** i = 2 la n **execută** **dacă** Zi > VAL **atunci** VAL ← Zi **sfârșit dacă** **sfârșit pentru** **returnează** VAL **sfârșit** maxim

Exemplul 2: *modulul de determinare a poziției valorii maxime (minime) dintr-o secvență de valori se poate obține din modulul de determinare a valorii maxime (minime) prin adăugarea a două instrucțiuni de atribuire*:

Calcularea poziției	Calcularea valorii
început modul poziție_maxim **variabile** Z, VAL, INDEX, i VAL ← Z1 INDEX ← 1 **pentru**i = 2 la n **execută** **dacă** Zi > VAL **atunci** **bloc** VAL ← Zi INDEX ← i	**început modul** maxim **variabile** Z, VAL, i VAL ← Z1 **pentru** i = 2 la n **execută** **dacă** Zi > VAL **atunci** VAL←Zi

sfârșit bloc **sfârșit dacă** **sfârșit pentru** **returnează** VAL, INDEX **sfârșit** poziție_maxim	**sfârșit dacă** **sfârșit pentru** **returnează** VAL **sfârșit** maxim

TEME

Utilizarea modulelor

1. Scrieți un modul de program pentru citirea datelor din problema traiectoriilor, folosind, în loc de tablouri unidimensionale, variabile independente. Condiția $n > 1$ este eliminată. Semnalați încheierea operației de citire prin:
   a) citirea numărului de aruncări n;
   b) utilizarea unei valori-semafor care să indice oprirea sau continuarea citirii.

2. Scrieți un modul de program care să numere elementele nenule dintr-o secvență de n numere date, $n \in N$ fixat.

3. Scrieți un modul de program pentru calcularea mediei aritmetice a unor numere întregi nenule, care să poată fi folosit într-un alt modul de program.

4. Se citesc $n \geq 3$ numere întregi strict pozitive mai mari decât 1000. Să se scrie un modul de program care să determine cifra unităților, a sutelor, a miilor și – dacă este cazul – a milioanelor; cifrele căutate vor fi afișate într-un alt modul de program.

5. Formulați exemple de prelucrări care pot fi organizate în module (subalgoritmi) și utilizate în alte prelucrări (algoritmi).

C. TEHNICI DE MODULARIZARE

Pentru modularizarea programelor putem folosi două tehnici frecvente în proiectare și programare:

(**T1.**) tehnica *top-down* (de sus în jos, de la complex la simplu; tehnică specifică operației de analiză);

(**T2.**) tehnica *bottom-up* (de jos în sus, de la simplu la complex; tehnică specifică operației de sinteză).

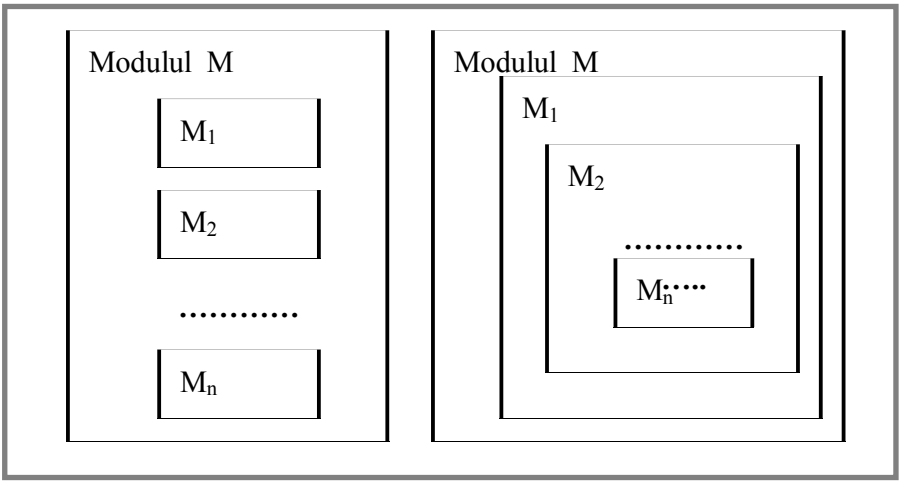
Tehnica *top-down* constă în descompunerea modulelor în submodule până la izolarea celor mai simple unități de lucru (este calea pe care am urmat-o în exemplele prezentate).

Tehnica *bottom-up* procedează exact invers: pornește de la unități de prelucrare elementare și – prin compunerea lor – ajunge la soluția problemei. Acest demers este intuitiv asemănător operației matematice de compunere a funcțiilor.

Două module $\mathbf{M_1}$ și $\mathbf{M_2}$ din cadrul unui program se pot afla unul față de altul în una dintre următoarele relații (ilustrate în figura 35):

(**R1**) $\mathbf{M_2}$ succede lui $\mathbf{M_1}$; prin urmare, $\mathbf{M_1}$ și $\mathbf{M_2}$ se numesc **module independente**;

(**R2**) $\mathbf{M_2}$ face parte din $\mathbf{M_1}$; prin urmare $\mathbf{M_2}$ se numește **submodul** (subalgoritm, subprogram) și îi este subordonat modulului (algoritmului, programului) $\mathbf{M_1}$.



Module independente (**R1**) Module subordonate (**R2**)

Figura 35. Subordonarea modulelor

În ambele cazuri, $\mathbf{M_1}$ îi transferă controlul lui $\mathbf{M_2}$. Acest transfer **are loc necondiționat,** numai dacă execuția modulului $\mathbf{M_1}$ **s-a încheiat corect**.

Cu alte cuvinte, **modul $\mathbf{M_2}$ se poate executa pentru că modulul $\mathbf{M_1}$ a pregătit toate datele necesare**. Aceste date constituie interfața prin intermediul căreia cele două module comunică și pot fi:

– date de intrare, pentru modului $\mathbf{M_2}$;

– date de ieșire, pentru modului $\mathbf{M_1}$.

În cazul (**R1**), interfața trebuie să asigure numai comunicarea de la modulul $\mathbf{M_1}$ la modulul $\mathbf{M_2}$: datele de ieșire ale modulului $\mathbf{M_1}$ trebuie transmise modulului $\mathbf{M_2}$ și devin date de intrare pentru acesta.

În cazul (**R2**), interfața trebuie să asigure atât comunicarea de la modulul ($\mathbf{M_1}$) la submodulul ($\mathbf{M_2}$), cât și de la submodulul **M2** către modulul **M1**:

– elementul de interfață, care asigură comunicarea de la modul spre submodul, se introduce prin cuvântul pseudocod **apelează** urmat de numele submodulului și mulțimea datelor de intrare;

apelează modul(date_de_intrare)

– elementul de interfață, care asigură comunicarea de la submodul spre modul, se introduce prin cuvântul poseudocod **returnează,** urmat de mulțimea datelor de ieșire**.**

returnează(date_de_ieșire)

Figura 36 ilustrează modul de comunicare într-un program compus din mai multe module, aflate fie în relația (**R1**), fie în relația (**R2**).

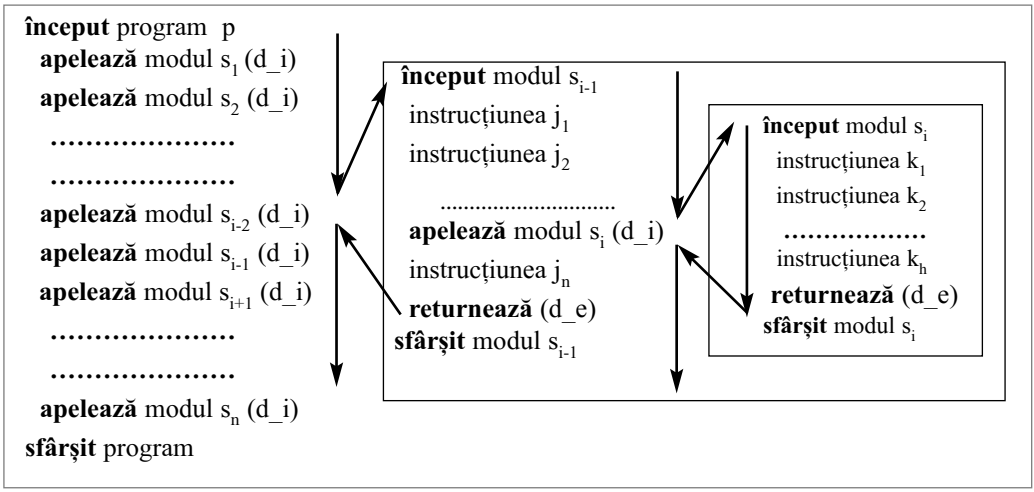


Figura 36. Comunicarea între module

TEME

Comunicarea între module

1. Proiectați un modul de program care să numere zilele care au trecut de la începutul anului până în ziua curentă. Ce date de intrare primește modulul? Ce date de ieșire returnează modulul?

2. Se citesc ***n*** perechi de numere întregi. Proiectați un program modularizat pentru:

a) determinarea mediei aritmetice a fiecărei perechi de numere;

b) determinarea perechilor de numere pentru care media aritmetică are valoarea maximă.

Câte module poate avea programul? De câte ori poate fi apelat fiecare modul?

D. IMPLEMENTAREA MODULARIZĂRII. STIVA SISTEM

Fiecare apel de modul determină o întrerupere a prelucrărilor din modulul apelant. Pentru reluarea acestora după revenirea în modul, adresa operației la care s-a făcut întreruperea este reținută într-o zonă specială de memorie, la dispoziția procesorului, numită **stiva sistem**. Pe același nivel al stivei sistem sunt memorate și datele de intrare în modul. Cu fiecare apel/întrerupere, stiva sistem crește; la revenirea din modulul apelat, stiva sistem scade.

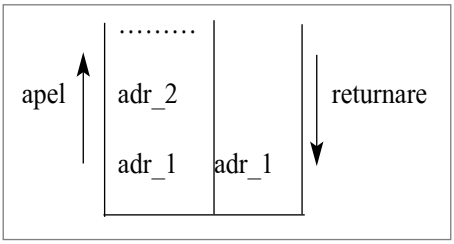


Figura 37. Stiva sistem

Exemplu: Intervalul de lungime maximă.

Fie două intervale de numere reale [A, B] și [C, D]. Se cere să se determine intervalul de lungime maximă.

1. Analiza problemei

● Date de intrare:

capetele celor două intervale (A,B,C,D ∈ R).

● Date de ieşire:

un mesaj corespunzător.

2. Raţionamentul de rezolvare

● pentru fiecare pereche de numere se verifică dacă formează interval şi, în caz negativ, se interschimbă „capetele" intervalului;

● se calculează lungimea fiecărui interval;

● se compară cele două lungimi;

● se tipăreşte mesajul corespunzător: *[A, B] are lungimea cea mai mare sau [C, D] are lungimea ce mai mare.*

3. Modularizarea rezolvării

● se foloseşte un submodul pentru calcularea lungimii unui interval şi – dacă e cazul – pentru interschimbarea „capetelor" intervalului.

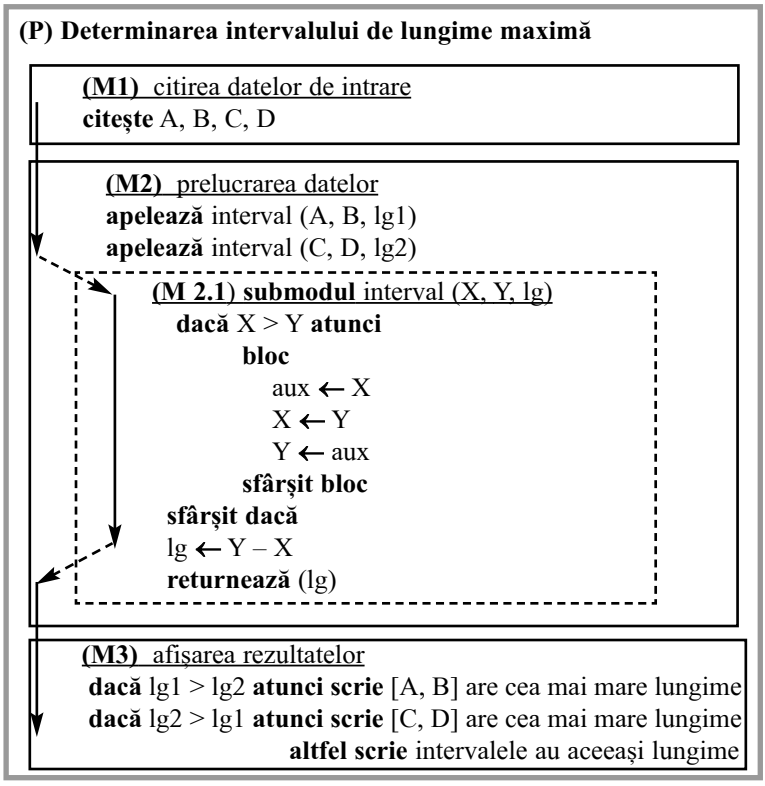


Figura 38. Raţionamentul problemei

4. Descrierea fluxului de control

● se intră în modulul de citire a datelor de intrare (**M1**) ⇒

– se citesc cele patru numerele reale **A, B, C, D**;

- se intră în modulul de prelucrare (**M2**) ⇒
  – se apelează submodulul (**M 2.1**) pentru perechea **A** și **B** ⇒
  – se memorează adresa de întrerupere în stiva sistem;
  – adresele valorilor **A** și **B** sunt depuse în stiva sistem;
  – se execută submodulul (**M 2.1**) pentru perechea **A** și **B** ⇒
  – lungimea intervalului [A, B], este transmisă modulului (**M2**),
- stiva sistem scade;
  – se apelează submodulul (**M 2.1**) pentru perechea **C** și **D** ⇒
  – se memorează adresa de întrerupere în stiva sistem,
  – adresele valorilor **C** și **D** sunt depuse în stiva sistem;
  – se execută submodulul (**M 2.1**) pentru perechea **C** și **D** ⇒
- lungimea intervalului [C, D], este transmisă modulului (**M2**),
- se revine în modulul **(M2)**
- stiva sistem scade;
- se intră în modulul de afișare a rezultatelor (**M3**) ⇒
- se compară cele două lungimi de interval **lg1** și **lg2** și se tipărește mesajul corespunzător.

TEME

Stiva sistem

1. Explicați rolul stivei sistem în implementarea fluxului de control modul-submodul atunci când modulele se află în relația ***(R1)***, respectiv ***(R2)***.

2. În ce situație crește stiva sistem?

3. Ce efecte are ieșirea dintr-un modul asupra stivei sistem?

4. Ce semnificație are stiva sistem goală?

5. În ce împrejurări zona de memorie ***stiva sistem*** nu mai are spațiu liber iar procesorul transmite mesajul <<stack overflow>> („stiva se revarsă")?

6. Scrieți un program modularizat pentru calcularea mediei valorilor energiei cinetice dezvoltate de un automobil, un autocamion și o motocicletă pentru un număr dat **n** de viteze de rulare $v_1, v_2, ..., v_n$. Cum se modifică stiva sistem?

3. LUCRUL CU SUBPROGRAME ÎN PSEUDOCOD

A. STRUCTURA SUBPROGRAMELOR

Structura unui subprogram nu diferă de structura unui program. În oricare dintre aceste module, întâlnim următoarele elemente de structură:

- un antet;
- o secțiune (opțională) de declarații (constante, variabile, alte subprograme);
- o secțiune de instrucțiuni executabile (simple sau structurate).

B. DEFINIREA SUBPROGRAMELOR

În cele mai multe dintre limbajele de programare, subprogramele sunt plasate „înaintea" secțiunii de instrucțiuni a programului propriu-zis (numit, de obicei, **program principal**). Definirea unui subprogram înseamnă:

– alegerea unui nume;

– stabilirea datelor de intrare și a datelor de ieșire proprii subprogramului;

– scrierea instrucțiunilor prin care se realizează prelucrarea datelor de intrare în vederea obținerii datelor de ieșire din subprogram.

> **Atenție**
>
> **Mulțimea variabilelor care constituie datele de intrare și datele de ieșire ale subprogramului formează lista parametrilor formali ai subprogramului.**
>
> **Acești parametri apar în interiorul unei perechi de paranteze rotunde (tocmai pentru a sublinia similaritatea dintre subprograme și funcțiile matematice) și sunt separați prin punct și virgulă.**

C. DECLARAREA SUBPROGRAMELOR

Declararea unui subprogram înseamnă precizarea prin cuvinte rezervate a elementelor de interfață care permit comunicarea între subprogram și programul apelant. Aceste elemente sunt descrise în **antetul** subprogramului prin:

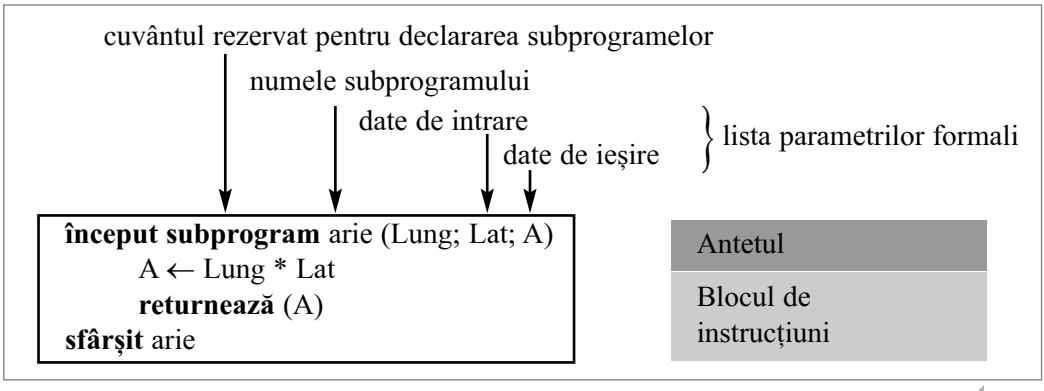
- cuvântul rezervat **subprogram**;
- numele dat de programator subprogramului;
- lista parametrilor formali.

subprogram nume(par_formal$_1$;par_formal$_2$;…;par_formal$_n$)

Exemplu: Subprogramul de calculare a ariei unui dreptunghi:

început subprogram arie (Lung; Lat; A)
    A ← Lung * Lat
    returnează (A)
sfârșit arie

Figura 39.
Declararea unui subprogram în pseudocod



TEME

Declararea subprogramelor

Declarați câte un subprogram pentru rezolvarea următoarelor cerințe:

a) calcularea ariei și volumului unei sfere de rază dată;

b) calcularea perimetrului și ariei unui triunghi, cunoscându-i laturile;

c) calcularea greutății unui obiect de masă dată;

d) calcularea densității unui lichid, cunoscându-i masa și volumul.

Explicați semnificația parametrilor pentru fiecare dintre subprograme.

D. APELAREA SUBPROGRAMELOR

Orice program care apelează un subprogram se numește **program apelant**, deoarece nu numai programul principal admite subprograme, ci și un subprogram oarecare poate avea, la rândul său, subprograme (figurile 35 și 36).

Prin **apelarea unui subprogram S de către un program apelant P** înțelegem o comandă pe care programul apelant o trimite subprogramului și prin care îi cere acestuia să execute prelucrarea din blocul său de instrucțiuni. Prin această comandă, programul apelant **P** trebuie:

(1) să-i furnizeze subprogramului **S** datele de intrare necesare prelucrării;

(2) să-i indice datele de ieșire în care subprogramul **S** trebuie să-i furnizeze rezultatul – sau rezultatele – prelucrărilor sale.

Toate aceste informații sunt transmise subprogramului cu ajutorul listei de parametri din apelul subprogramului. Acești parametri se numesc **parametri actuali**. Elementele comenzii de apel sunt:

– cuvântul rezervat **apelează**;

– numele subprogramului;

– lista parametrilor actuali.

apelează nume(par_actual$_1$;par_actual$_2$;…;par_actual$_n$)

Atenție

Parametrii actuali din comanda de apel trebuie să corespundă – ca tip, număr și ordine de enumerare – parametrilor formali din antetul subprogramului.

Exemplu: Apelarea subprogramului pentru calcularea ariei unui dreptunghi
apelează arie(1;5;suprafața)

TEME

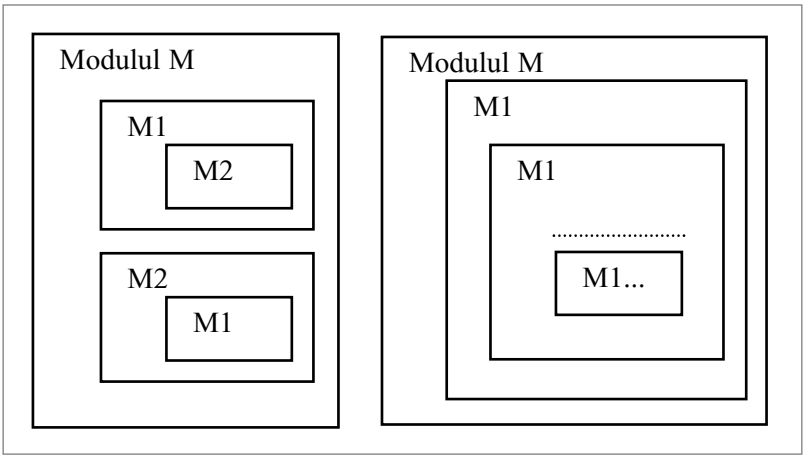
Apelarea subprogramelor

Formulați comenzile de apel pentru fiecare dintre subprogramele definite în cadrul Temei anterioare – **Declararea subprogramelor**.

Atenție

De ce este nevoie de două liste de parametri: o listă de parametri formali și o listă de parametri actuali? Dacă ne-am limita la o singură listă (lista parametrilor formali), aceasta ar trebui să apară – evident! – în antetul subprogramului în momentul definirii sale, iar la apel, toate programele apelante ar trebui să cunoască și să utilizeze exact această listă. Existența celei de-a doua liste, lista parametrilor actuali – cu respectarea convențiilor enumerate mai sus – asigură independența subprogamului apelat față de oricare dintre programele apelante.

Figura 40 ilustrează două cazuri particulare de apelare a subprogramelor.



(a) Module care se apelează unul pe celălalt

(b) Modul care se apelează pe el însuși

Figura 40. Apelarea subprogramelor – cazuri particulare

E. RETURNAREA VALORILOR CĂTRE PROGRAMUL APELANT

Un subprogram trebuie să producă un rezultat – prin prelucrările efectuate asupra datelor de intrare. De fapt, un program apelează subprogramul atunci când are nevoie de acel rezultat. Transmiterea rezultatului (sau rezultatelor) către programul apelant necesită:

(1) includerea în lista parametrilor a unui număr de parametri egal cu numărul de rezultate care trebuie transmise;

(2) includerea în corpul subprogramului – în poziția determinată de fluxul controlului – a unei comenzi care să asigure comunicarea rezultatelor.

Această comandă conține:

– cuvântul rezervat **returnează**,

– lista de variabile care conțin valorile ce trebuie transmise.

returnează(rez_1,rez_2,…,rez_k)

Exemplu: Returnarea valorii ariei dreptunghiului către programul apelant
returnează (A)

F. TRANSFERUL PARAMETRILOR LA APEL

Partea cea mai importantă din instrucțiunea de apelare a subprogramelor este lista parametrilor actuali. În scrierea ei, trebuie să pornim de la lista parametrilor formali și să respectăm:

(1) corespondența dintre parametrii formali și parametrii actuali ca tip, număr și ordine;

(2) modul de transmitere a parametrilor: prin valoare sau prin adresă.

Condiția (1) este legată de forma apelului. În apel apar variabile de program (simple sau structurate), caracterizate aici prin *nume* și *tip*. Un parametru actual poate avea același nume cu parametrul formal corespunzător sau poate avea un nume diferit. În general, este recomandabilă redenumirea parametrilor formali în lista parametrilor actuali doar acolo unde – altfel – s-ar crea confuzii. Tipul de date al parametrului formal trebuie să fie identic sau mai cuprinzător decât tipul de date al parametrului actual corespunzător. Rezultă de aici necesitatea păstrării ordinii la nivelul ambelor liste. Revenind la subprogramul pentru determinarea ariei unui dreptunghi, iată un exemplu și două contraexemple de apelare:

Exemplu	*Contraexemple*
	apelează arie(5;suprafața;12)
apelează arie(12;5;suprafața)	**apelează** arie(12;suprafața)

Condiția (2) este legată de apel. În lista parametrilor trebuie specificat tipul acestora:

– parametri care transmit subprogramului datele de intrare;

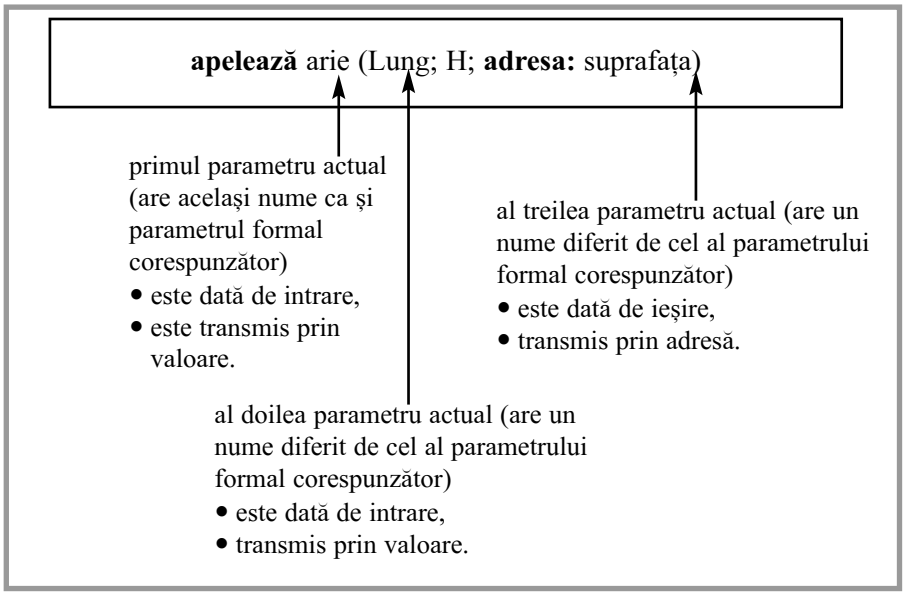
– parametri care transmit programului apelant rezultatele.

Uneori, un același parametru poate fi și dată de intrare, necesară subprogramului pentru efectuarea prelucrărilor, și dată de ieșire, prin care să transmită programului apelant rezultatul prelucrării.

Informația depusă într-un parametru-dată de ieșire sau într-un parametru-dată de intrare-ieșire trebuie să fie accesibilă atât pentru subprogram, cât și pentru programul apelant (care o poate prelua în vederea unor noi prelucrări). Din această cauză, este necesar să comunicăm compilatorului că trebuie să aloce **aceeași locație de memorie** atât variabilei-parametru formal, cât și variabilei-parametru actual – chiar dacă ele au nume diferite. Pentru ca zona de memorie să fie „vizibilă" (accesibilă), subprogramul trebuie să cunoască adresa acestei zone. În acest caz, parametrul va fi transmis **prin adresă**, lucru semnalat – numai în lista parametrilor actuali – printr-un cuvânt rezervat, **adresă**, care precede numele parametrului în listă. Spunem că **transmitem** acel **parametru** (dată de ieșire sau dată de intrare-ieșire) **prin adresă** (sau prin referință).

Un parametru care este dată de intrare pentru subprogram trebuie să fie accesibil acestuia (este principala modalitate de primire a datelor de către subprogram), dar trebuie să fie protejat împotriva unor modificări prin prelucrările din subprogram. Din această cauză, este necesar să comunicăm compilatorului că trebuie să aloce acelei variabile **două locații de memorie**: una accesibilă programului – definită în segmentul de date – pe care subprogramul nu o poate modifica, și una accesibilă subprogramului – definită pe un nivel de stivă – pe care subprogramul o poate modifica oricum. Spunem că **transmitem** acel **parametru** (dată de intrare) **prin valoare**.

Exemplu: Transmiterea parametrilor la apelul subprogramului pentru determinarea ariei unui dreptunghi:



Transmiterea incorectă a parametrilor la apel poate genera erori greu de depistat.

Exemplu: Calcularea lungimii unui interval

Fie un interval de numere reale [A,B]. Se cere să se determine lungimea acestui interval.

1. Analiza problemei

• Date de intrare:

capetele intervalului (A,B ∈ R).

• Date de ieșire:

lungimea intervalului (lg ∈ R).

2. Raționamentul problemei

• se verifică dacă numerele A și B formează interval și, dacă este cazul, se interschimbă aceste valori;

• se calculează lungimea intervalului;

• se folosește un submodul pentru interschimbarea "capetelor" intervalului.

3. Reprezentarea algoritmului

```
început program lungime_interval
 variabile A,B,lg
 început subprogram interschimbare(adresă:X;adresă:Y)
  variabile aux
  aux ← X
```

```
 X ← Y
 Y ← aux
 returnează (X;Y)
sfârșit interschimbare
citește A,B
dacă A>B atunci
 apelează interschimbare(A;B)
sfârșit dacă
lg ← B-A
scrie [A, B] are lungimea lg
sfârșit lungime_interval
```

4. Verificarea algoritmului

Să presupunem că rulăm programul pentru valorile **A**=5, **B**=2

	A	*B*	*lg*
dacă A>B **atunci** **apelează** interschimbare(A;B) **sfârșit dacă**	**5** **2**	**2** **5**	
lg ← B-A			**3**

Rezultatele obținute sunt corecte deoarece am transmis parametrii subprogramului *interschimbare* prin adresă (ambii sunt date de intrare-ieșire).

Să presupunem că transmitem acești parametri prin valoare, adică antetul subprogramului este:

început subprogram interschimbare(X;Y)

În acest caz, rezultatul obținut va fi –3 deoarece programul apelant nu "vede" zona de memorie în care subprogramul a efectuat interschimbarea și calculează lg ← 2 – 5.

	Stiva sistem		***Segmentul de date***		
	A	*B*	*A*	*B*	*lg*
citește A,B			**5**	**2**	
dacă A>B **atunci** **apelează** interschimbare(A;B) **sfârșit dacă**	**5** **2**	**2** **5**			
lg ← B-A					**– 3**
scrie [A, B] are lungimea lg					

TEME

Transmiterea parametrilor prin adresă

1. Explicați de ce un parametru transmis prin valoare nu poate îndeplini rolul de dată de ieșire.

2. Modificați antetul din definiția subprogramelor din cadrul **Temelor** de la paginile 78 și 82 astfel încât să puneți în evidență parametrii transmiși prin adresă.

G. VARIABILE LOCALE ȘI VARIABILE GLOBALE

Orice modul de program prelucrează date proprii, definite ca **variabile locale**. Acest aspect conferă subprogramelor independență față de modulul apelant. Variabilele locale sunt zone de memorie alocate pe nivelul de stivă sistem corespunzător subprogramului; de aceea nu sunt vizibile din modulul apelant.

Pentru ca o variabilă să fie bine definită, trebuie specificate:

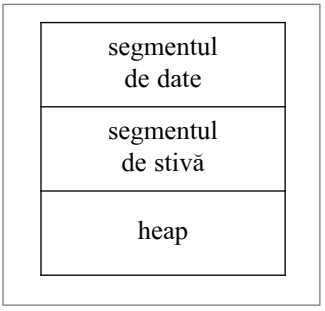
- numele;
- tipul de date pe care le va păstra;
- zona de memorie;
- durata de viață;
- domeniul de vizibilitate.

Până acum, definirea variabilelor folosite într-un program s-a limitat la precizarea numelui și a tipului lor.

Prin **zona de memorie** a unei variabile înțelegem tipul de memorie internă în care variabila este memorată (unde se află locația atribuită ei). Variabilele oricărui program pot fi memorate în una dintre următoarele zone de memorie internă:

- segmentul de date;
- segmentul de stivă;
- *heap*-ul;

sau într-un registru dedicat al microprocesorului.



Prin **durata de viață** a unei variabile înțelegem intervalul de timp în care variabila dispune de locația de memorie atribuită ei de compilator (la sfârșitul intervalului, locația de memorie va fi eliberată, deci ultima valoare a variabilei se pierde). După acest criteriu, varibilele se clasifică în:

- **variabile statice**: locațiile de memorie se alocă în momentul compilării și sunt disponibile pe toată perioada execuției programului (alocare statică în segmentul de date);
- **variabile dinamice**: locațiile de memorie se alocă în momentul execuției (alocare dinamică în *heap*).

Prin **domeniul de vizibilitate** al unei variabile înțelegem segmentul de program (blocul de instrucțiuni) în care variabila poate fi utilizată. În raport cu segmentul de program în care sunt definite, variabilele pot fi:

- **variabile locale** (definite în subprogramul apelat);
- **variabile globale** (definite în programul apelant).

Exemplu: Variabile locale și globale

Fie o matrice pătratică cu elemente numere reale; se cere:

a) *să se înlocuiască fiecare element de pe diagonala principală a matricei cu suma elementelor de pe linia corespunzătoare (inclusiv elementul de pe diagonală);*

b) *să se afișeze indexul liniei pentru care suma elementelor este minimă, precum și valoarea acestei sume.*

1. Analiza problemei

● Date de intrare:

– numărul de linii și coloane (**n**),

– matricea (**A**).

● Date de ieșire:

– valoarea celei mai mici sume (Val),

– indexul liniei corespunzătoare acestei sume (pMin).

2. Modularizarea rezolvării

Vom folosi:

– un subprogram pentru calcularea sumei elementelor de pe liniile matricei și depunerea acestor sume pe diagonala principală;

– un subprogram de determinare a minimului unui vector cu **k** elemente și a poziției sale.

3. Raționamentul problemei

Pasul 1. Se citesc datele de intrare: **n**, **A.**

Pasul 2. Se apelează subprogramul de calculare a sumei pe linii și înlocuire a elementului de pe diagonala principală cu suma respectivă.

Pasul 3. Se depun elementele de pe diagonala principală a matricei într-un vector.

Pasul 4. Se apelează subprogramul de determinare a elementului minim dintr-un vector și a poziției sale.

Pasul 5. Se afișează suma minimă și indexul liniei pe care se află aceasta.

4. Reprezentarea programului cu subprograme

```
început program sumaMinimă
  variabile n, A, Val,pMin,V,i    //          globale
  început subprogram diag(adresa:mat)
    variabile suma,i,j              //          locale
    pentru i=1 la n execută
        suma ← mat(i,1)
        pentru j=2 la n execută
          suma ← suma+mat(i,j)
        sfârșit pentru
        mat(i,i) ← suma
    sfârșit pentru
    returnează mat
```

```
  sfârșit diag
  început subprogram pozMinim(V;k; adresa:pMin)
    variabile i                          //        locale
    Val ← V(1)
    pMin ← 1
    pentru i=2 la k execută
       dacă V(i) < Val
       atunci
          Val ← V(i)
          pMin ← i
       sfârșit dacă
    sfârșit pentru
    returnează pMin
  sfârșit pozMinim
  citește n,A
  apelează diag(A)
  pentru i=1 la n execută
    V(i) ← A(i,i)
  sfârșit pentru
  apelează pozMinim (V;n;pMin)
  scrie Val,pMin
sfârșit sumaMinimă
```

5. Analiza programului

Examinând acest program, observăm:

● lista de parametri a subprogramului *diag* constă dintr-un singur element: o matrice. Aceasta reprezintă atât datele de intrare (elementele matricei trebuie transmise subprogramului pentru a permite calcularea sumelor), cât și datele de ieșire pentru subprogram (matricea este returnată după ce a fost modificată prin depunerea sumelor pe diagonala principală). Ca urmare, matricea este transmisă prin adresă nu prin valoare;

● n = numărul de linii din matrice este o variabilă globală; de aceea această variabilă nu trebuie transmisă în lista de parametri a subprogramelor. Totuși, *n* a fost transmis ca dată de intrare subprogramului *pozMin* deoarece am folosit forma generală a subprogramului de determinare a poziției minimului, în care trebuie să apară numărul de elemente pentru care se calculează minimul și poziția acestuia;

● variabila *suma* din subprogramul *diag* este necesară numai pentru prelucrările din acest subprogram. Această variabilă a fost declarată în subprogram și este variabilă locală pentru acesta;

● dacă valoarea sumei minime nu trebuia afișată, atunci variabila *Val* putea fi declarată în interiorul subprogramului *pozMin* (deci ar fi fost o variabilă locală pentru el, la fel ca variabila *i*). Fiind declarată în programul principal (apelant) ca variabilă globală, valoarea ei s-a păstrat și a putut fi afișată;

● variabilele care apar în lista parametrilor actuali trebuie declarate în programul apelant; dar variabilele care apar în lista parametrilor formali nu trebuie declarate. Totuși, în programul

sumaMinimă variabila *V*, parametru formal pentru subprogramul *pozMinim* a fost declarată în programul apelant (principal). Motivul: ea este folosită și ca parametru actual pentru apelarea subprogramului *pozMinnim*.

TEME

1. Variabile locale și variabile globale

Fie programul de mai jos:

```
început program test
  variabile a, b, c
  început subprogram  S1 (adresa y)
    ……..
    returnează y
    ……..
  sfârșit S1
  început subprogram S2 (x, adresa t)
    variabile a, y
    început subprogram  S3 (adresa y)
      variabile b
      ……..
      returnează y
      ……..
    sfârșit S3
      ……..
    returnează t
      ……..
  sfârșit S2
      ……..
sfârșit test
```

Presupunem că plasăm următoarele instrucțiuni în blocul de instrucțiuni executabile al subprogramului menționat. Precizați care dintre instrucțiuni va fi corect/incorect plasată și de ce:

a) în subprogramul S3, instrucțiunea a ← b
b) în programul test, instrucțiunea **returnează** S3(3.14)
c) în programul test, instrucțiunea a ← S2(10)
d) în subprogramul S1, **apelează** S3(d)
e) în subprogramul S1, c ← S2(c)
f) în programul test, instrucțiunea **apelează** S1(z)
g) în programul test, instrucțiunea **apelează** S1(S2(3))
h) în programul test, instrucțiunea **apelează** S2(c)
i) în subprogramul S2, **apelează** S1(y)
j) în subprogramul S1, y ← S2(c)

2. Variabile locale și variabile globale

Care dintre afirmațiile de mai jos este corectă? Corectați afirmațiile greșite:

a) nu este permisă utilizarea variabilelor globale în interiorul unui subprogram;

b) nu este recomandabilă utilizarea variabilelor globale în interiorul unui subprogram;

c) variabilele globale pot fi folosite exclusiv pentru citirea datelor de intrare în program;

d) variabilele globale pot fi folosite exclusiv pentru citirea datelor de intrare în subprogram;

e) datele de intrare ale subprogramelor nu pot fi transmise decât prin variabile globale.

3. Reguli de lucru cu subprogramele

Care dintre afirmațiile de mai jos este corectă/greșită; justificați răspunsul:

a) fie S1 un subprogram din programul principal P și S2 un subprogram definit în interiorul lui S1; atunci S2 nu poate conține la rândul său alt subprogram;

b) constantele utilizate într-un program cu subprograme urmează aceleași reguli privind domeniul lor de vizibilitate ca și variabilele;

c) într-un subprogram putem declara și o variabilă și o constantă sub același nume;

d) într-un program cu subprograme putem declara și o variabilă și o constantă sub același nume, numai dacă variabila este globală iar constanta este locală.

4. Corectarea erorilor în subprograme

Fie programul de mai jos:

```
început program test2
  variabile a, b, c
  început subprogram S1(x, adresa w)
    variabile a, y
    început subprogram S2(adresa y)
      variabile b
      ……
      returnează y
      ……
    sfârșit S3
    ……
    returnează w
    ……
  sfârșit S1
  început subprogram S3 (adresa y)
      ……
    returnează y
      ……
  sfârșit S3
      ……
sfârșit test
```

Presupunem că plasăm următoarele instrucțiuni în blocul de instrucțiuni executabile al subprogramului menționat. Precizați care dintre instrucțiuni va fi corect/incorect plasată și de ce (atenție la domeniul de vizibilitate al variabilelor):

a) în subprogramul S2, instrucțiunea a ← b
b) în programul test, instrucțiunea **apelează** S2(3.14)
c) în programul test, instrucțiunea a ← S1(10)
d) în subprogramul S3, **apelează** S2(3.14)
e) în subprogramul S3, c ← S1(c)
f) în programul test, instrucțiunea **apelează** S3(a)
g) în programul test, instrucțiunea **apelează** S1(S3(z))
h) în programul test, instrucțiunea **apelează** S1(c)
i) în subprogramul S1, instrucțiunea **apelează** S3(y)
j) în subprogramul S2, instrucțiunea y ← S1(c)

APLICAȚIE

TAPETAREA APARTAMENTULUI

Să se calculeze suprafața de tapet necesară pentru renovarea unei sufragerii, a unui dormitor și a unei bucătării (toate de formă paralelipipedică).

1. Analiza problemei

● Date de intrare:
– lungimile pereților sufrageriei (**S1**, **S2**), dormitorului (**D1**, **D2**) și ai bucătăriei (**B1**, **B2**),
– înălțimea pereților (**H**),
– lățimea sulului de tapet (**T**).
● Date de ieșire:
– numărul de metri de tapet necesari pentru sufragerie (**NS**), dormitor (**ND**), bucătărie (**NB**),
– numărul total (**NR**).
● Condiții și relații importante

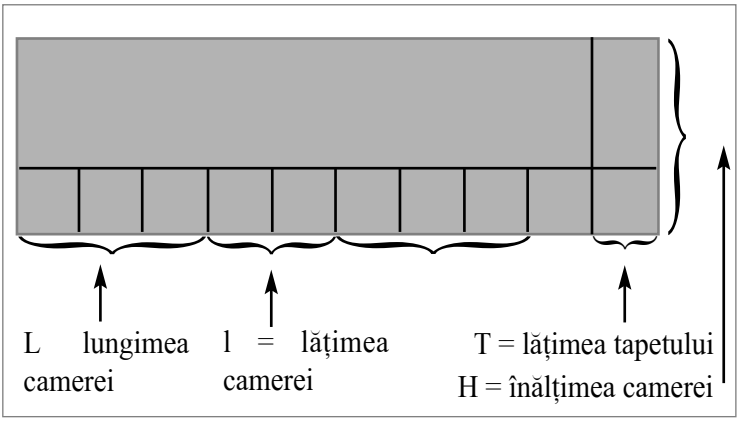


Figura 41. Acoperirea pereților camerei cu tapet

Începem cu sufrageria: este suficient să calculăm P = 2*(S1+S2) = perimetrul camerei și să-l împărțim la T = lățimea sulului de tapet pentru a afla m = P / T = câte bucăți de tapet de lungime egală cu înălțimea camerei sunt necesare. Apoi, NS = m * H = numărul de metri de tapet necesari pentru sufragerie. Analog obținem numărul de metri de tapet necesari pentru dormitor și pentru bucătărie. În final, calculăm numărul total NR = NS + ND + NB.

2. Raționamentul problemei

Pasul 1. Se citesc datele de intrare: S1, S2, D1, D2, B1, B2, H, T.
Pasul 2. Se calculează P ← 2 * (S1 + S2); m ← P / T; NS ← m * H.
Pasul 3. Se calculează P ← 2 * (D1 + D2); m ← P / T; ND ← m * H.
Pasul 4. Se calculează P ← 2 * (B1 + B2); m ← P / T; NB ← m * H.
Pasul 5. Se calculează NR ← NS + ND + NB.
Pasul 6. Se afișează NS, ND, NB, NR.

3. Reprezentarea algoritmului

Metoda I Un program fără module

```
începeput tapet1
variabile S1,S2,D1,D2,B1,B2,H,T,NS,ND,NB,NR,P,m
citește S1,S2,D1,D2,B1,B2,H,T
P ← 2*(S1 + S2)
m ← P/T
NS ← m*H
P ← 2*(D1 +D2)
m ← P/T
ND ← m*H
P ← 2*(B1 + B2)
m ← P/T
NB ← m*H
NR ← NS+ND+NB
scrie  NS,ND,NB,NR
sfârșit tapet1
```

Metoda II Un program modularizat

```
începeput tapet2
variabile S1,S2,D1,D2,B1,B2,H,T,NS,ND,NB,NR
început subprogram calcul(L;l;adresa:N)
variabile P,m
P ← 2*(L+l)
m ← P/T
N ← m*H
returnează N
sfârșit calcul
citește S1,S2,D1,D2,B1,B2,H,T
apelează calcul(S1;S2;NS)
```

apelează calcul(D1;D2;ND)
apelează calcul(B1;B2;NB)
NR ← NS+ND+NB
scrie NR,NS,ND,NB
sfârșit tapet2

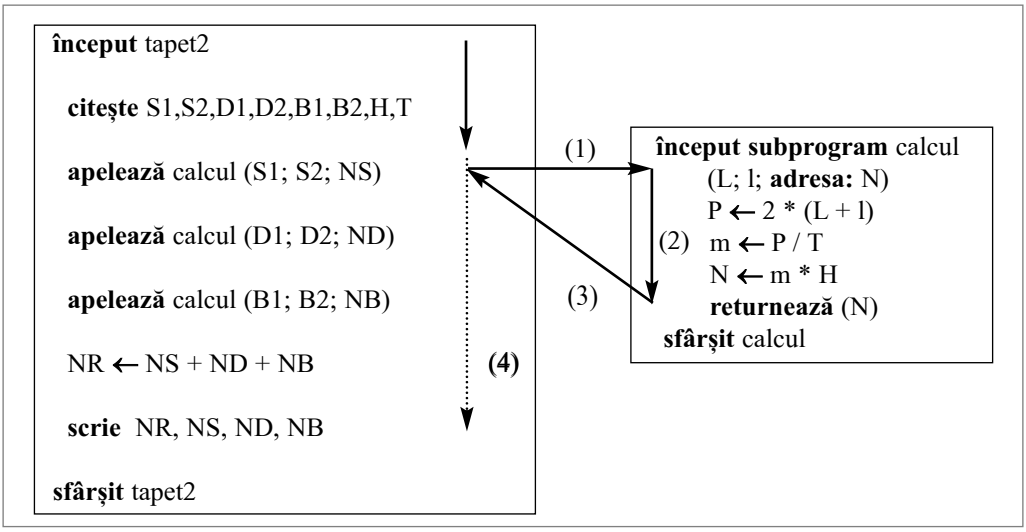


Figura 42. Programul tapet2 – fluxul de execuție

În figura 42 este evidențiat fluxul de execuție a programului:

(1) controlul este transferat subprogramului;

(2) se execută instrucțiunile subprogramului în același mod în care s-ar executa dacă ar fi vorba de un program propriu-zis;

(3) după parcurgerea întregului flux de execuție al subprogramului, controlul (și rezultatele) se transferă programului apelant; execuția acestuia continuă cu instrucțiunea aflată imediat după instrucțiunea care apelează subprogramul;

(4) fiecare apel determină reluarea fluxului (1), (2), (3).

TEME

1. Fluxul de execuție

Reprezentați stiva sistem corespunzător fluxului de execuție program – subprogram din figura 42.

2. Rezolvarea problemelor cu ajutorul subprogramelor

Scrieți câte un program cu subprograme pentru rezolvarea fiecăreia dintre următoarele cerințe:

a) Să se determine numărul *p* de numere prime dintr-un șir de *n* numere naturale introduse de la tastatură. Să se verifice dacă *p* este număr prim și să se afișeze un mesaj corespunzător.

Exemplu: pentru n = 5, șirul de numere este: 25, 3, 42, 29, 7. Se determină p = 3 și se afișează mesajul: *3 este număr prim.*

b) Să se calculeze numerele „mistice“ mai mici decât un număr $n \in N$ dat *(*$x \in N$ se numește număr „mistic“ dacă suma cifrelor care îl compun este egală cu produsul lor; exemplu: 22, 13131).

c) Să se calculeze divizorii primi ai unui număr $n \in N$ *dat.*

d) Să se calculeze numerele perfecte mai mici decât un număr $n \in N$ dat *(*$x \in N$ se numește număr perfect dacă suma divizorilor săi primi este egală cu produsul lor; exemplu: 6 este un număr perfect deoarece 1+2+3=2*3).

e) Fie M o matrice cu *n* linii și *m* coloane. Să se afișeze în ordine crescătoare elementele de pe liniile pare.

f) Fie M o matrice cu *n* linii și *m* coloane. Pentru fiecare linie, să se afișeze cel mai mare divizor comun dintre *max* și *min,* unde *max* reprezintă cea mai mare valoare de pe linie iar *min* reprezintă cea mai mică valoare de pe linie.

MINIPROIECT ÎN ECHIPĂ. COMPANIA EFICIENT

Etapa: *Proiectare* (stabilirea unităților de program)

Cerințe:

1. Organizați unitățile de program necesare rezolvării modulare a prelucrărilor identificate în etapa de analiză.

2. Completați documentația proiectului cu descrierea modulelor și reprezentarea relațiilor dintre acestea.

4. IMPLEMENTAREA SUBPROGRAMELOR ÎN LIMBAJELE DE PROGRAMARE

A. SUBPROGRAME ÎN LIMBAJUL PASCAL: FUNCȚII ȘI PROCEDURI

În limbajul ***Pascal*** se utilizează două categorii de subprograme:

- funcții,
- proceduri.

Vom prezenta asemănările și deosebirile dintre aceste două categorii de subprograme cu ajutorul programului de calculare a ariei unui dreptunghi

🕮 Definiție

Funcție =

= un subprogram care primește oricâte valori ca date de intrare dar returnează ca rezultat o valoare și numai una.

Declarare:

function nume(par_formal$_1$,…,par_formal$_n$:tip$_1$;…; par_formal$_l$,…,par_formal$_m$:tip$_k$): tip;

Apelare:
x:= nume(par_actual$_1$;par_actual$_2$;…;par_actual$_w$);
sau:
write(nume(par_actual$_1$;par_actual$_2$;…;par_actual$_w$)**);**

🕮 Definiție

Procedură =

= un subprogram care primește oricâte valori ca date de intrare și poate returna ca rezultat o valoare, mai multe valori sau niciuna (poate efectua o prelucrare care să nu aibă ca efect obținerea unei valori).

Declarare:
procedure nume([***var***] par_formal$_1$,…,par_formal$_n$:tip$_1$;…; [***var***] par_formal$_1$,…, par_formal$_m$:tip$_k$);
Apelare:
nume(par_actual$_1$;par_actual$_2$;…;par_actual$_w$);

🔔 Atenție

Spre deosebire de pseudocod, în limbajul ***Pascal:***

● cuvântul rezervat pentru transmiterea parametrilor prin adresă este *var*;

● apelarea subprogramelor (ca și returnarea rezultatelor) nu necesită utilizarea niciunui cuvânt rezervat (altfel spus, cuvintele rezervate **apelează** și **returnează** folosite în pseudocod nu au un corespondent în limbajul ***Pascal***).

	Funcții		***Proceduri***
Declarare	**function** A (L:**Real**; l:**Real**): **Real;**		**procedure** arie(L:**Real**; l:**Real**; **var** A: **Real**); eventual
Apelare	v:=A (Lung;H);	write (A(Lung;H));	arie(Lung;H; suprafata);

Observații

● Pentru identificarea fiecărui tip de subprogram se utilizează cuvinte rezervate diferite: **function** și respectiv **procedure**.

● Declarația *var* este indusă între paranteze drepte deoarece este opțională.

● Lista de parametri formali a procedurii trebuie să includă un număr de parametri formali transmiși prin adresă egal cu numărul de rezultate returnate de procedură către programul apelant. Lista de parametri formali a funcției include parametrii transmiși prin valoare (datele de intrare) sau prin referință (dacă există date de intrare-ieșire); valoarea determinată – rezultatul – va fi returnată prin *numele* funcției. De aceea:

– „funcția are tip" (și anume un tip simplu, nestructurat);

– în blocul de instrucțiuni executabile al funcției trebuie să existe cel puțin o instrucțiune de atribuire având ca membru stâng **numele** funcției.

● Funcția se apelează prin nume și listă de parametri actuali în interiorul unei expresii.

● Procedura se apelează prin nume și listă de parametri actuali într-o instrucțiune de sine stătătoare.

Exemplu: Funcții și proceduri în limbajul ***Pascal***

Fiind dat numărul real a și numărul întreg n, să se calculeze a^n.

1. Analiza problemei

● Date de intrare:

baza (*a*) și exponentul (*n*).

Date de ieșire:

valoarea a^n.

2. Modularizarea rezolvării

Vom utiliza procedura *putere1* și funcția *putere2* (pentru a ilustra transformarea unei proceduri într-o funcție și reciproc).

Vom valida datele de intrare, tratând cazul 0^0 într-un mod particular: $0^0=0$ (cazul bazei egale cu 1 nu va fi tratat separat!).

3. Raționamentul problemei

Pasul 1. Se citesc datele de intrare: a, n.
Pasul 2. Dacă a = 0 atunci r ← 0 și salt la **Pasul 9**.
Pasul 3. Dacă n = 0 atunci r ← 1 și salt la **Pasul 9**
Pasul 4. nn ← n; dacă nn < 0 atunci nn ← – nn
Pasul 5. i ← 1.
Pasul 6. r ← r * a și i ← i + 1
Pasul 7. Dacă i ≤ nn atunci se reia **Pasul 6**.
Pasul 8. Dacă n < 0 atunci r ← 1/ r
Pasul 9. Se scrie r.

4. Implementarea algoritmului în limbajul *Pascal*

```
program ridicareLaPutere;
 var a,r:real;
      n:integer;
 procedure putere1(baza:real; expo:integer; var rez:real);
{ rez – parametru transmis prin adresă }
  var e,i:integer;
```

```
 begin
  if (baza = 0.0) then rez:=0.0
   else
    if (expo=0) then rez:=1.0
     else
      begin
      rez:= baza;
      if expo<0 then e:=-expo else e:=expo;
      for i:=2 to e do rez:=rez*baza;
      if expo<0 then rez:=1/rez;
      end;
 end; {procedura putere1}
function putere2(baza:real; expo:integer):real; {funcție de tip real}
 var e,i:integer;
    rez:real;
 begin
  if (baza = 0.0) then putere2:=0.0
   else
    if (expo=0) then putere2:=1.0
     else
      begin
      rez:= baza;
      if expo<0 then e:=-expo else e:=expo;
      for i:=2 to e do rez:=rez*baza;
      if expo<0 then rez:=1/rez;
      putere2:= rez;
      end;
 end; {functia putere2}
 begin {program principal}
  write('a ='); readln(a);
  write('n ='); readln(n);
  putere1(a,n,r);
  write('Calculul puterii cu procedura putere1: ');
  writeln(a:4:2,' la puterea ',n:2,' este ',r:8:3);
  write('Calculul puterii cu  functia  putere2: ');
  writeln(a:4:2,'la puterea ',n:2,' este',putere2(a,n):8:3)
end.
```

5. Analiza programului

Identificatorul *putere2* **nu** desemnează o variabilă (având același nume și tip ca și funcția), ci este numele funcției și **acestuia** i se atribuie valoarea variabilei **rez.** Acesta este motivul pentru care, în blocul de instrucțiuni executabile al funcției **care nu se apelează pe ea însăși**, numele funcției nu poate apărea în membrul drept al unei atribuiri: este necesară utilizarea unei

variabile locale, auxiliare. Tabelul de mai jos prezintă funcțiile *putere2* și *putere3* ca exemplu și contraexemplu de subprograme de tip funcție.

Exemplu de funcție Pascal: putere2	**Contraexemplu de funcție Pascal: putere3**
function putere2(baza:**real**; expo:**integer**):**real**; **var** e,i:**integer**; rez:**real**; **begin** if (baza=0.0) then putere2:=0.0 **else** **if** (expo=0) **then** putere2:=1.0 **else** **begin** rez:= baza; **if** expo<0 **then** e:=-expo **else** e:=expo; **for** i:=2 **to** e **do** rez:=rez*baza; **if** expo<0 **then** rez:=1/rez; putere2:= rez; **end**; **end**; {functia putere2}	**function** putere3(baza:**real**, expo:**integer**):**real**; **var** e,i:**integer**; **begin** **if** (baza=0.0) **then** putere3:=0.0 **else** **if** (expo=0) **then** putere3:=1.0 **else** **begin** putere3:=baza; **if** expo<0 **then** e:=–expo **else** e:= expo; for i:=2 to e do putere3:= putere3*baza if expo<0 then putere3:=1/putere3; **end** **end**; {functia putere3}

TEMĂ

De la procedură la funcție

Transformați următoarea procedură într-o funcție:

```
procedure Verificare (err,toleranta:real; var accept:boolean);
  begin
  if err<toleranta then
     accept:=true
     else
     accept:=false;
end;
```

Observație

Tipul unui parametru formal nu poate fi un tip anonim:

Exemplu:	***type*** matrice=***array***[1..10,1..20] ***of real***; ***function*** maxim(n,m:***integer***; A:matrice):***real***;
Contraexemplu	***function*** maxim(n,m:***integer***; A:***array***[1..10,1..20] ***of real***):***real***;

TEME

Proceduri sau funcții?

1. Fie două numere raționale r1 și r2; se cere să se afișeze: suma, produsul și câtul celor două numere. *Scrieți și analizați cele două soluții posibile: un program cu subprograme de tip funcție, respectiv cu subprograme de tip procedură (pentru fiecare operație).*

2. Fie **M** o matrice pătratică de ordin **n** > 1 cu elemente numere naturale; se cere să se afișeze numerele prime (și poziția lor în matrice) aflate ca valoare între cel mai mic număr de pe diagonala principală și cel mai mare număr de pe diagonala secundară. *Care dintre module poate fi transformat numai într-o funcție/numai într-o procedură/fie într-o procedură/fie într-o funcție. Argumentați.*

3. Fie vectorul **V** cu **n** > 1 componente, numere naturale; *se cere să se afișeze:*

a) numere pare care se găsesc pe poziții de index impar în **V**;
b) numerele prime din **V;**
c) cifrele numărului de pe ultima poziție din vector, în ordine inversă.

Domeniul de vizibilitate al variabilelor

4. *Examinați programul **Pascal** de mai jos din punct de vedere al domeniului de vizibilitate al variabilelor. Completați (după modelul oferit pe prima linie) toate liniile din tabelul furnizat, indicând tipul de date al variabilelor (prin inițiala tipului: I pentru Integer, R pentru real etc.) și valorile constantelor, în conformitate cu cerințele fiecărui bloc de instrucțiuni executabile. Dacă un identificator nu este definit într-un anumit bloc, atunci introduceți în celula respectivă codul N/A.*

```
program domeniu;
  const i=10;
  var a,b:integer;
     c,f:real;
     d,e:char;
       g:boolean;
  procedure P1(a,e:real; c:boolean);
    const k=2.1;
    var   f:boolean;
    begin {P1}
    ……..
    end; {P1}
  procedure P2(c:integer; g:real);
    const b=10;
    var   a:boolean;
          e:char;
    procedure P3(a:char; b:real);
```

```
    const d=3.14;
    var i,f:integer;
    begin {P3}
    ……..
    end; {P3}
  procedure P4(a:real);
    const e='a';
    var g,k:integer;
    begin {P4}
    ……..
    end; {P4}
  begin {P2}
    ……..
  end; {P2}
procedure P5(e,g: integer; d:boolean);
  const k=40;
  var   i:real;
  begin {P5}
    ……..
  end; {P5}
begin {program principal}
    ……..
end.
```

Domeniul	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***i***	***k***
Program princ.	I	I	R	C	C	R	B	10	N/A
P1									
P2									
P3									
P4									
P5									

B. SUBPROGRAME ÎN LIMBAJUL C/C++

Limbajul ***C/C++*** admite un singur tip de subprograme: subprograme de tip funcție (însuși programul principal este o funcție cu un nume rezervat: **main**).

Declarare:

tip nume(tip1 [***&***] par_formal$_1$,…, tip$_m$ [***&***] par_formal$_m$);

Declarația & este inclusă între paranteze drepte deoarece este opțională.

Apelare

v=nume(par_actual$_1$,…, par_actual$_m$); sau

nume(par_actual$_1$,…, par_actual$_m$);

În limbajul C/C++ există două categorii de funcții:

• funcții „cu tip" – acestea transmit către programul apelant o valoare atribuită numelui funcției; dacă tipul expresiei din instrucțiunea

return expresie;

nu este identic cu cel al funcției, atunci se face automat conversia. O astfel de funcție se apelează într-o expresie;

• funcții „fără tip" – acestea nu transmit către programul apelant nicio valoare; în acest caz, tipul funcției este **void**, rezultatele prelucrărilor sunt depuse în variabile globale (și astfel programul apelant are acces la ele) iar revenirea în programul apelant se face fie la întâlnirea instrucțiunii

return;

fie după executarea tuturor instrucțiunilor din corpul funcției. O astfel de funcție se apelează printr-o instrucțiune de sine stătătoare.

În descrierea şi apelul funcțiilor trebuie respectate următoarele reguli:

• numele funcției este precedat de tipul funcției;

• numele fiecărui parametru formal este precedat de tipul parametrului;

• transmiterea parametrilor prin adresă este precizată prin operatorul ***&*** care precede numele parametrului formal, de exemplu:

double arie(**double** L, **double** l, **double &** A));

• în blocul de instrucțiuni al funcției există o instrucțiune specială pentru returnarea rezultatului:
return expresie; sau **return**;
return A;

Exemplul 1: Verificarea parității sumei a două numere întregi.

1. Analiza problemei

• Date de intrare:
numerele întregi **a, b.**

• Date de ieșire:
un mesaj care indică dacă suma celor două numere este pară sau nu.

2. Modularizarea rezolvării

Varianta I

Vom folosi o funcție care returnează o valoare (restul împărțirii sumei la 2).

Varianta II

Vom folosi o funcție care nu returnează nicio valoare (variabila în care se calculează restul este declarată variabilă globală):

3. Raționamentul problemei

Pasul 1. Se citesc datele de intrare: a, b.

Pasul 2. Se apelează subprogramul de determinare a parității sumei celor două numere.

Pasul 3. Dacă restul returnat de subprogram este zero atunci se afișează mesajul "Suma este pară", altfel se afișează mesajul "Suma este impară".

4. Implementarea algoritmului

I. Program cu funcție care returnează o valoare	II. Program cu o funcție care nu returnează nicio valoare
#**include**<stdlib.h> #**include**<conio.h> **int** par(**int** a, **int** b) {**int** S; S=(a+b)%2; **return** S;} **void main (void)** {**int** a,b,S; **cout**<<"Dați întregii a și b"; **cin**>>a>>b; S=par(a,b); **if** (S == 0) **cout**<<"Suma celor doua numere este para"; **else cout**<<"Suma celor doua numere este impara";}	**#include<stdlib.h>** **#include<conio.h>** **int** S; **void** par (**int** a, **int** b) {S=(a+b)%2;} **void main (void)** {**int** a,b; **cout**<<"Se introduc a și b"; **cin**>>a>>b; par(a,b); **if** (S==0) **cout**<<"Suma celor doua numere este para"; **else cout**<<"Suma celor doua numere este impara";}

Exemplul 2: Interschimbarea a două valori *a* și *b*.

În foarte multe prelucrări, este necesară interschimbarea valorilor reținute în două locații de memorie, fie acestea *a* și *b*. De aceea, este util să realizăm un subprogram care să primească valorile celor două locații de memorie și să transmită programului principal noile valori rezultate prin interschimbare. Subprogramul necesită doi parametri care reprezintă adresele celor două locații de memorie.

Funcție cu parametri de tip adresă

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

void interschimb (int &x, int &y)
 {int aux;
   aux=x; x=y; y=aux;}
void main (void)
 {int a,b;
  cout<< "se introduc valorile";
//  are loc interschimbarea valorilor
    interschimb (a, b);
    cout << "se afișează noile valori";
    cout << a << " " << b;
    }
```

TEME

Utilizarea subprogramelor în limbajul C/C++

1. Fie două numere raționale r1 și r2; se cere să se afișeze: suma, produsul și câtul celor două numere. Scrieți și analizați un program cu subprograme (pentru fiecare operație câte un subprogram).

2. Fie M o matrice pătratică de ordin n > 1 cu elemente numere naturale; se cere să se afișeze numerele prime (și poziția lor în matrice) aflate în intervalul dintre cel mai mic număr de pe diagonala principală și cel mai mare număr de pe diagonala secundară. Realizați un program cu subprograme. Descrieți și argumentați tipul fiecărui subprogram.

3. Examinați programul C/C++ de mai jos din punct de vedere al domeniului de vizibilitate al variabilelor.

```
#include<iostream.h>
char a,b; int c; float d;
void P1 (float y)
{float b;
————————————}
void P2(char y)
{————————————-}
char F1(int x)
{float a; char y;
P1(y);
return y;}
void main (void)
{————————————}
```

Presupunem că plasăm următoarele instrucțiuni în blocul de instrucțiuni executabile al subprogramului menționat. *Indicați care dintre instrucțiuni va fi corect/incorect plasată și de ce:*

a) în program, P2(F1(c));
b) în subprogramul F1, P1(a);
c) în subprogramul P2, P1(d);
d) în program, a=F1(c);
e) în subprogramul P1, P2(b).

C. EXERCIȚII CU SUBPROGRAME

1. Codificați în limbajul de programare studiat următoarele secvențe și completați tabelul de variație pentru a determina valoarea variabilelor pe parcursul execuției.

a)

```
început program calcul
 variabile a,b,c,d,e
 început subprogram P1(x;y; adresa:z;u;v)
  variabile a
  a←1; b←7; x←y; z←a+x; v←z+b
  returnează z,u,v
 sfârșit P1;
 a←4; b←5; c←12; d←b; e←c
 apelează P1(a;b;c;d;e)
 scrie a,b,c,d,e
 apelează P1(1;2;a;b;c)
 scrie a,b,c,d,e
sfârșit calcul
```

a	b	c	d	e

b)

```
început program opuse
 variabile x,y
 început subprogram F1(x)
  variabile i,s
  s ← 0;
  pentru i = 1la y execută
   s ← s+i+x
   x ← -1
  sfârșit pentru
  F1 ← s
 sfârșit F1
  x ← 0; y ← 1
  cât timp y <= 2
   scrie x,y
   x ← x–F1(y);
   y ← y+1;
  sfârșit cât timp
sfârșit opuse
```

x	y	s	i

c)

```
început program verificare
    început subprogram verificăValori(date de intrare: v1; v2;v3; date de ieșire:v3)
     dacă v1>v2
         atunci
           scrie v1,v2
           v3 ← v3+1
         altfel
           scrie v2,v1
```

```
        v3 ← v3-1
    sfârșit dacă
sfârșit verificăValori
contor←0
citește index
pentru i=1 la index execută
 citește v1,v2,v3,v4
 contor←contor+1
 citește v5
 cât timp v3<v5
    v3←v3+2
 sfârșit cât timp
 v5←v5+v3
 dacă v3 este par
      atunci
       apelează verificăValori(v2;v4;v5)
      altfel
       apelează verificăValori(v1;v3;v5)
 sfârșit dacă
sfârșit pentru
scrie v1,v2,v3,v4,v5
sfârșit program
```

v1	v2	v3	v4	v5

2. Corectarea erorilor

– Eliminați erorile din programele de mai jos:

a)

VARIANTA *PASCAL*	VARIANTA *C/C++*
program erori; **const** limita=10; **var** index:**integer**; **function** Func(cheie:**char**): **boolean**; **var** sem:**boolean**; cod:**char**; **begin** {Func}	**#include<iostream.h>** **#include<conio.h>** **int** limita=10; **int** Func(**char** cheie) { **int** sem,cod; **cout**<<"Introduceți codul ";

```
  sem:=false;
  write('Introduceți codul");
  read (cod); writeln;
  if cod=cheie then        semaf:=true
  Func:=sem
 end {Func}
begin {program principal}
 for index:=1 to limita
  if (func(chr(index)) then
     writeln ('Da');
  end; {if}
end. {program principal}
```

```
 cin>>cod; cout<<endl;
 if(cod==cheie)
    semaf=1;
 return sem;
}

void main (void)
for (int index=1; index<=limita;index++)
{if (func(char(index)))
cout<<"Da";
}
getch();
}
```

b)

VARIANTA PASCAL	VARIANTA C/C++
program erori2; **var** a,b,i:**integer**; **procedure** P1(**var** x:**integer**, y:**integer**); **begin** {P1} x=y **end** {P1} **begin** {program principal} **set** s **to false** **while not** s **begin** **readln**(a **and** b); **if** (a>b) **then** S1 a, b); **else** S1(b,a); **for** i=1 **to** b **do** **begin** a:=a–(b+i); **writeln**('a+b=,a+b) **end for**; **end**. {program principal}	**#include<iostream.h>** **int** a,b,i; **void** P1(**int** x, **int** y) {x=y} **void main (void)** {s=0; **while** (!s) {**cin**>>a>>b; **if**(a>b) S1(a,b); **else** S1(b, a) **for** (i=1;i<=b,i++) {a=a-(b+i); **cout**<<'a+b= '<<a+b;}}

3. Stive și cozi

a) Realizați un program cu subprograme pentru simularea unei stive.

b) Realizați un program cu subprograme pentru simularea unui fir de așteptare.

Utilizarea funcțiilor predefinite

4. Scrieți câte un program care să utilizeze funcțiile și procedurile standard enumerate în Anexa 2/Anexa 4.

5. Identificați funcțiile predefinite ce se pot folosi în programul de calculare a energiei cinetice și în problema traiectoriilor (pag. 65 și pag. 70).

6. Se consideră un automobil într-o curbă de rază r; dându-se unghiurile pantelor α și φ, se cere să se scrie un program care să afișeze viteza maximă și viteza minimă cu care poate ataca automobilul curba în condiții de siguranță. Se cunosc următoarele formule:

$$v_M = \sqrt{r \cdot g \cdot tg(\alpha + \varphi)} \quad , \quad v_m = \sqrt{r \cdot g \cdot tg(\alpha - \varphi)}$$

Ce subprograme predefinite se pot folosi?

5. ȘIRURI DE CARACTERE

A. PARTICULARITĂȚI DE MEMORARE A ȘIRURILOR DE CARACTERE

Cum procedăm pentru a citi și afișa, pe rând, numele inventatorilor primelor calculatoare mecanice, electromecanice, electronice: John Napier, Wilhelm Schickard, Blaise Pascal, Gottfried Wilhelm Leibnitz, Charles Babbage, Konrad Zuse, S.P. Eckert, J.W. Mauchly ș.a.

Să începem cu C. Babbage: o primă soluție constă în utilizarea unui vector cu cel puțin 10 componente de tipul predefinit **char**. Putem citi, prelucra și afișa tot vectorul sau numai anumite elemente ale sale (a se vedea figura 43).

indice în vectorul s	0	1	2	3	4	5	6	7	8	9
caracter tastat		**C**	**.**	**B**	**a**	**b**	**b**	**a**	**g**	**e**
valoare în memorie	9	067	046	066	097	098	098	097	103	101

(a) varianta ***Pascal***

indice în vectorul **s**	0	1	2	3	4	5	6	7	8	9
caracter tastat	**C**	**.**	**B**	**a**	**b**	**b**	**a**	**g**	**e**	**null**
valoare în memorie	067	046	066	097	098	098	097	103	101	000

(a) varianta ***C/C++***

Figura 43. Memorarea șirurilor de caractere

Se observă că în locațiile din memoria internă asociate componentelor vectorului sunt depuse codurile *ASCII* (American Standard Code for Information Interchange (ASCII) este codul universal acceptat pentru memorarea internă a caracterelor) ale caracterelor din șir: s[1] conține valoarea 067 = codul *ASCII* al caracterului 'C', s[3] conține valoarea 066 = codul *ASCII* al caracterului 'B', s[5] și s[6] conțin valoarea 098 = codul *ASCII* al caracterului 'b' etc. (Anexa 7)

ÎN PASCAL	ÎN C/C++
în primul octet (atenție: s[0] și nu s[1] este prima componentă a vectorului!) este depusă automat lungimea efectivă a șirului (aici: 9).	ultimul octet (s[9], pentru că indexarea se face de la 0!) conține, conform convenției, valoarea 0 = codul *ASCII* al caracterului null

LIMBAJUL PASCAL	LIMBAJUL C/C++
program sir; **var** s:**array**[1..50] of **char**; n,i: **integer**; **begin** **writeln**('n='); **readln**(n); **for** i:=1 **to** n **do read** (s[i]); **writeln**('Sirul citit este:'); **write** (s); **end**.	**#include<iostream.h>** **void main (void)** {**char** s[50]; **int** n, i; **cout**<<"n="; **cin**>>n; **cout**<<"Introduceti sirul."<<**endl**; **for** (i=0; i<n; i++) **cin**>>s[i]; s[n]=0; **cout**<<"Sirul este:"<<**endl**; **for** (i=0;i<n;i++)**cout**<<s[i];}

Atenție

În limbajul *Pascal* am putut afișa vectorul de caractere ca pe o variabilă independentă (nu am mai recurs la indexare, așa cum am făcut la citire).

În limbajul *C/C++*, dacă vectorul este inițializat prin declarare (char s[10]="C.Babbage"), atunci caracterul null este memorat automat în ultima componentă; dacă este inițializat prin citire (ca mai sus), atunci trebuie inclusă în program o instrucțiune explicită de atribuire (s[n]=0;).

B. FACILITĂȚI PENTRU PRELUCRAREA ȘIRURILOR DE CARACTERE ÎN LIMBAJUL PASCAL

Limbajul ***Pascal*** oferă un tip de date standard special: tipul **string**. O variabilă de tip **string** poate reține până la 255 caractere și poate fi declarată în două moduri

1.

type sir=**string**;

var s:sir;

în acest caz se rezervă în memorie 256 de octeți – indiferent câte caractere are la citire șirul *s* – primul octet conține lungimea efectivă a șirului citit pentru exemplul șirului "C.Babbage" lungimea șirului este 9) – ;

2.

type sir=**string[n];**

var s:sir**;**

în acest caz se rezervă în memorie *n* de octeți – primul octet conține lungimea efectivă a șirului citit; – se pot citi șiruri cu maximum *n-1* caractere.

Atenție

Programatorul nu trebuie să se preocupe de depunerea lungimii efective a șirului în primul octet.

Variabila *s* de tipul string va fi privită ca o variabilă independentă dar "componentele ei" vor putea fi accesate prin indexare, la fel ca și componentele oricărui vector!

TEME

1. Ce afișează următorul program?

```
program sir2;
  var s: string;
begin
  writeln('Introduceti sirul: ');
  readln (s);
  writeln('Sirul',s,'are',ord(s[0]),'  caractere');
  writeln('s[3]=',s[3],'s[5]=',s[5],'s[6]=',s[6]);
  writeln
end.
```

Operația de atribuire pentru șiruri

2. Variabilele de tip **string** pot fi inițializate – ca orice variabile – prin citire sau atribuire.

Fie variabilele g și k inițializate cu șirurile 'Leibnitz' și respectiv 'Zuse'. Care dintre următoarele moduri de declarare permite efectuarea corectă (fără trunchiere la dreapta) a oricăreia dintre atribuirile: a ← b , b ← a:

```
(1) var a:string;          (2) var a:string[4];     (3) var a:string[8];
        b:string;                  b:string[4];             b:string[8];
(4) var a:string;          (5) var a:string[15];    (6) var a:string[20];
    b:string[4];               b:string[13];            b:string[20];
```

C. FACILITĂȚI PENTRU PRELUCRAREA ȘIRURILOR DE CARACTERE ÎN LIMBAJUL C/C++

Limbajul ***C/C++*** permite citirea și scrierea șirurilor de caractere, fără a mai preciza adresa elementelor. Astfel, primul program care exemplifică prelucrarea șirurilor de caractere devine:

```
#include<iostream.h>
void main (void)
{ char s[50];
  cout<<"Introduceti sirul fara spatii"<<endl;
```

```
  cin>>s;
  cout<<"Sirul introdus:"<<endl;
  cout<<s;
}
```

Această metodă prezintă un dezavantaj: șirul citit nu poate conține mai multe spații (nu poate fi citită o "frază" în care "cuvintele" să fie separate prin spații).

Pentru înlăturarea acestui dezavantaj, vom folosi o funcție specială care permite citirea șirurilor de caractere de orice lungime care conțin orice caracter, inclusiv spații.

Să presupunem că trebuie să citim și să prelucrăm următorul – text – șir de caractere:

'John Napier a inventat nu numai logaritmii naturali dar și primul calculator mecanic, în stare să reducă efectuarea înmulțirilor și împărțirilor la efectuarea de adunări și scăderi.'

Acest șir are 182 de caractere, deci poate fi declarat astfel:
char secv[183];
Prezentăm mai jos câteva exemple de citire a acestui șir cu ajutorul funcției **cin.get**.
(ex1) cin.get(secv,183,'.');
(ex2) cin.get(secv,183);
(ex3) cin.get(secv,183,',');
(ex4) cin.get(secv,86);
În primele două cazuri, este citit întregul șir; în ultimele două cazuri este citit numai subșirul *'John Napier a inventat nu numai logaritmii naturali dar și primul calculator mecanic,'*. Această funcție predefinită are următorul antet:

🕮 Definiție

cin.get(șir, **int** nr, **char**='\n')

Semnificația parametrilor:

- primul parametru actual trebuie să fie o variabilă de tip șir de caractere, declarată anterior, de exemplu, prin **char** v[n];
- al doilea parametru actual trebuie să fie un număr cuprins între **0** și **n**, care să indice numărul de caractere din șir care trebuie citite;
- al treilea parametru actual este opțional; dacă nu apare, atunci se subînțelege că este '\n'.

Efectele apelării funcției (în oricare dintre variante) sunt:

- se citesc nr-1 caractere (iată de ce am folosit 183 deși șirul nostru are 182 caractere), inclusiv spațiile;
- caracterul *null* este inserat automat după ultimul caracter din șir;
- atunci când apare, caracterul transmis ca al treilea parametru actual forțează sfârșitul șirului (de aceea în exemplul **(ex3)** de mai sus, citirea se oprește la întâlnirea virgulei).

Atenție

Pentru a citi numai un prefix al șirului, nu întreg șirul, putem folosi:

- varianta **(ex3)**: se citește șirul *secv* până la întâlnirea primului caracter ‘,‘;
- varianta **(ex4)**: se citesc primele 85 componente din șirul *secv*.

Pentru a citi un singur caracter, fie că este spațiu sau nu, putem folosi funcția **cin.get** fără parametri: **cin.get();** acest apel este folosit frecvent la citirea succesivă a mai multor șiruri de caractere pentru preluarea separatorului de sfârșit de șir.

D. SUBPROGRAME PREDEFINITE PENTRU PRELUCRAREA ȘIRURILOR DE CARACTERE

- **Atribuirea de valori**

LIMBAJUL PASCAL	LIMBAJUL C/C++
operatorul **:=**	funcția **strcpy**
program atribuire1; **var** s1,s2,s3:**string**; **begin** s1:=’Charles’; s2:=’Babbage’; **writeln**(‘Sirurile citite:’); **writeln**(s1,’ ‘,s2); s3:=s2; s2:=s1; s1:=s3; **writeln**(‘Sirurile dupa interschimbare:’); **writeln**(s1,’ ‘,s2); **end**.	**#include<iostream.h>** **#include<string.h>** **#include<conio.h>** **void main (void)** **{clrscr();** **char** s1[10]=”Charles”, s2[10]=”Babbage”, s3[10]; **cout**<<s1<<s2<<**endl**; **strcpy**(s3,s2); **strcpy**(s2,s1); **strcpy**(s1,s3); **cout**<<”Sirurile dupa interschimbare:”<<**endl**; **cout**<<s1<<s2<<**endl**; **getch();**}
	Definiția funcției: **char *strcpy**(**char ***destinație, **char ***sursa); Apelarea: **strcpy**(s1,s2); Ca urmare, șirul s2 este depus în șirul s1; șirul s2 rămâne nemodificat; șirul s1 va conține rezultatul operației și va fi returnat către programul apelant.
Ambele programe afișează secvențele Charles Babbage și Babbage Charles	

Atenție (C/C++)

Caracterul * care precede numele funcției și al parametrilor formali indică faptul că aceștia nu reprezintă valori ci adrese. Cu alte cuvinte, ***sursa** reprezintă adresa primei locații din zona de memorie atribuită acestui parametru al funcției **strcpy**.

• Compararea șirurilor de caractere

Șirurilor de caractere li se pot aplica operatorii relaționali (<, >, = etc). Cum putem însă compara litera *a* cu cifra *7* sau cu paranteza deschisă?! Dar șirul 'aX' cu șirul 'Ax'?! Metoda constă în compararea codurilor ASCII prin care se reprezintă în memorie caracterele din șir (așa cum am observat în reprezentarea șirului 'C.Babbage', codul literei a, 097, este predecesorul în sens aritmetic al codului literei b, 098).

Fie două variabile **x** și **y** de lungime **n**, respectiv **m**, care trebuie comparate. Distingem următoarele cazuri:

• codul primului caracter din **x** este mai mare decât codul primului caracter din **y**; atunci x>y (de exemplu: 'bit' > 'Octet' deoarece *ASCII*('b') = 098 > 079 = *ASCII*('O'));

• codul primului caracter din **x** este mai mic decât codul primului caracter din **y**; atunci x<y (de exemplu: 'Bit' < 'octet' deoarece *ASCII*('B') = 066 < 111 = *ASCII*('o'));

• codul primului caracter din **x** este egal cu codul primului caracter din **y** (este practic același caracter); distingem următoarele subcazuri:

♦ n=m=1; atunci x=y;

♦ n>m=1; atunci x>y (de exemplu: 'if'>'i');

♦ m>n=1; atunci y>x;

♦ n≥m>1; atunci se continuă cu compararea codurilor caracterelor de pe pozițiile următoare din **x** și **y**. Să presupunem că pentru toate primele *h* poziții codurile au fost egale:

– dacă h=m=n atunci x=y;

– dacă h=m<n atunci x>y;

– dacă h<m atunci relația dintre **x** și **y** este relația dintre codurile *ASCII* ale caracterelor aflate în **x** și **y** pe poziția h+1.

♦ m≥n>1; se procedează analog.

Compararea în Limbajul Pascal	Compararea în Limbajul C/C++
operatorii relaționali	funcția **strcmp**
program comparare1; **var** s1,s2:**string**; **begin** **writeln**('s1:');**readln**(s1); **writeln**('s2:');**readln**(s2); **if**(s1<s2) **then** **writeln**(s1,'<',s2)	**#include<iostream.h>** **#include<string.h>** **void main(void)** {**char** s1[20],s2[20]; **int** v; **cout**<<"\n s1:";**cin.get**(s1,20); **cin.get(); cout**<<"\n s2:";**cin.get**(s2,20);

else **if**(s1>s2) **then** **writeln**(s1,'>',s2) **else** **writeln**(s1,'=',s2); **end**.	**cout**<<**endl**; v=**strcmp**(s1, s2); **if** (v < 0) **cout**<<s1<<"<"<<s2; **else** **if** (v >0) **cout**<<s1<<">"<<s2; **else cout**<<s1<<"="<<s2; }
	Definiția funcției: **int strcmp(const char** *sir1**, const char** *sir2**);** Apelarea: v=**strcmp(**s1,s2**);** Funcția compară sirurile s1 și s2 și returnează către programul apelant o valoare întreagă care este: < 0 , dacă s1 < s2, = 0 , dacă s1 = s2, > 0 , dacă s1 > s2.
Pentru s1 = "abc" și s2 = "aac", ambele programe afișează abc > aac	

- **Determinarea lungimii unui șir de caractere**

LIMBAJUL PASCAL	LIMBAJUL C/C++
funcția **length** sau valoarea conținută în primul octet din șir	funcția **strlen**
program length1; **var** s1:**string**; **begin** s1:='Acest sir are 27 caractere.'; **writeln**('Acest sir are ',**ord**(s1[0]),' caractere.'); **writeln**('Acest sir are ',**length**(s1),' caractere.'); **end**.	**#include<iostream.h>** **#include<string.h>** **void main (void)** {**char** s1[28]="Acest sir are 27 caractere."; **cout**<<"Acest sir are "<< **strlen**(s1)<<" caractere."; }
Definiția funcției: **function length** (sursa:**string**): :**Integer**; Apelarea: n:= **length**(s); Funcția returnează către programul apelant numărul de caractere din șirul s (primul octet, rezervat automat numărului de caractere din șir, nu este numărat).	Definiția funcției: size_t **strlen** (**char** *sursa); Apelarea: n = **strlen** (s); Funcția returnează către programul apelant numărul de caractere din șirul s (ultimul octet, rezervat automat caracterului null, nu este numărat).
Ambele programe afișează textul „Acest șir are 27 caractere."	

- **Concatenarea şirurilor de caractere**

LIMBAJUL PASCAL	LIMBAJUL C/C++
funcția **concat** sau operatorul +	funcția **strcat**
program concat1; **var** s1,s2,r1,r2:**string**; **begin** s1:='Turbo';s2:='Pascal'; r1:=s1+' '+s2; r2:=**concat**(s1,' ',s2); **writeln**('Cu operatorul + : ',r1); **writeln**('Cu functia concat: ',r2); **end**.	**#include<iostream.h>** **#include<string.h>** **void main (void)** {**char** s1[10]="Borland",s2[10]="C"; **strcat** (s1, " "); **strcat** (s1, s2); **cout**<<"**\n**"<<s1; }
Definiția funcției: **function concat**(s1[,s2,…,sn]:**string**):**string;** Apelarea: s:=**concat**(a1,a2,…,ak); Ca urmare, şirurile a1, a2, …, ak sunt concatenate în această ordine și de la stânga la dreapta. Dacă şirul rezultat are mai mult de 255 caractere, atunci el este trunchiat la dreapta.	Definiția funcției: **char *strcat** (**char ***destinație, **char ***sursa); Apelarea: **strcat** (a1, a2); Ca urmare, şirul a1 este concatenat la dreapta cu șirul a2; șirul s2 rămâne nemodificat; şirul s1 va conține rezultatul operației și va fi returnat către programul apelant. Pentru a concatena n șiruri, funcția trebuie apelată de n-1 ori: **strcat**(a1,a2); **strcat**(a1,a3); . . .; **strcat**(a1,an);
Programul afişează secvența „Turbo Pascal“ (de două ori!)	Programul afişează secvența „Borland C“ (o singură dată)

Atenție

Operația de concatenare a șirurilor NU este comutativă.

- **Conversia literelor mici în majuscule**

LIMBAJUL PASCAL	LIMBAJUL C/C++
funcția **upcase**	funcția **strupr**
program upcase1; **var** s1:**string**; i:**integer**; **begin** s1:='Clasa a XI-a'; **writeln**('Sirul initial: ', s1); **for** i:=1 **to** length(s1) **do** s1[i]:= **upcase**(s1[i]); **writeln**('Sirul cu majuscule: ', s1); **end**.	**#include<iostream.h>** **#include<string.h>** **void main (void)** {**char** s1[13]="Clasa a XI-a"; **cout**<<"**\n** Sirul initial: "<<s1<<**endl**; **strupr** (s1); **cout**<<"Sirul cu majuscule: "<<s1; }
Definiția funcției: **function upcase (**car**: Char):Char;** Apelarea: w:= **upcase** (c); Funcția returnează majuscula corespunzătoare literei c primită ca parametru actual; dacă acest parametru nu se află în domeniul a..z atunci el rămâne neschimbat.	Definiția funcției: **char *strupr(char ***sursa**);** Apelarea: adr=**strupr** (s1); Funcția returnează șirul s1, în care toate caracterele – aflate în domeniul a..z – au fost transformate în majuscule. Cele din afara acestui domeniu rămân neschimbate.
Ambele programe afișează Clasa a XI-a și, apoi, CLASA A XI-A	

- **Căutarea unui subșir într-un șir de caractere**

LIMBAJUL PASCAL	LIMBAJUL C/C++
funcția **pos**	funcția **strstr**
program pos1; **var** s1,s2:**string**; n:**integer**; **begin** s1:='imprimanta'; **writeln**(s1); n:=**pos**('prim',s1); **writeln**('"prim" incepe din pozitia:',n); **writeln**('"rima" incepe din pozitia:',**pos**('rima',s1)); n:=**pos**('prima',s1); **writeln**('"prima" incepe din pozitia:',n); **writeln**('"manta" incepe din	**#include<iostream.h>** **#include<string.h>** **void main (void)** {**char** s1[11]="imprimanta", s2[5]="prim",s3[5]="rima", s4[6]="prima"; **char** s5[6]="manta",s7[5]="mira", s6[8]= "imprima", *adr; adr=**strstr**(s1,s2); **if** (adr) **cout**<<adr-s1+1<<**endl**; **else cout**<<"Nu apare in sir" <<**endl**; adr=**strstr**(s1,s3); **if** (adr) **cout**<<adr-s1+1 <<**endl**;

pozitia:',**pos**('manta',s1)); **writeln**('"imprima" incepe din pozitia:',**pos**('imprima',s1)); n:= **pos**('mira',s1); **writeln**('"mira" incepe din pozitia:',n,', adica nu se afla in sir'); **end**.	**else cout**<<"Nu apare in sir"<<**endl**; adr=**strstr**(s1,s4); **if** (adr) **cout**<<adr-s1+1<<**endl**; **else cout**<<"Nu apare in sir"<<**endl**; adr=**strstr**(s1, s5); **if** (adr) **cout**<<adr-s1+1<<**endl**; **else cout**<<"Nu apare in sir"<<**endl**; adr = **strstr**(s1, s6); **if** (adr) **cout**<<adr-s1+1<<**endl**; **else cout**<<"Nu apare in sir"<<**endl**; adr = **strstr**(s1, s7); **if** (adr) **cout**<<adr-s1+1<<**endl**; **else cout**<<"Nu apare in sir"<<**endl**;}
Definiția funcției: **function pos(**subsir**: String;** sir**: String): Byte;** Apelarea: n:=**pos**(s1,s2); Funcția verifică (de la stânga la dreapta) dacă secvența s1 apare – în întregime – în secvența s2. În caz afirmativ, returnează indicele caracterului din s2 care este și primul caracter din s1; altfel returnează valoarea 0.	Definiția funcției: **char *strstr(char ***sir**, char ***subsir**);** Apelarea: adr=**strcstr**(s1,s2); Funcția verifică (de la stânga la dreapta) dacă secvența s2 apare – în întregime – în secvența s1. În caz afirmativ, returnează adresa caracterului din s1 care este și primul caracter din s2; altfel returnează valoarea *null*.
Programele afișează numerele 3, 4, 3, 6, 1, 0	

- **Conversia unei valori numerice într-o secvență de caractere**

LIMBAJUL PASCAL	LIMBAJUL C/C++
procedura **str**	funcțiile **itoa**
program str1; **var** s1:**string**; a:**integer**; **begin** a:= 2006; **str**(a:4,s1); **writeln**('Numarul ',a:4); **writeln**('Sirul ',s1); **end**.	**#include<iostream.h>** **#include<stdlib.h>** **void main (void)** {**int** a=2006; **char** s1[100]; **itoa**(a,s1,10); **cout**<<"Numarul:"<<a; **cout**<<"Sirul:"<<s1<<**endl**; }

Definiția procedurii: **procedure str(**numar:i; **var** sursa:**string**); Apelarea: **str**(nr:n,s); Procedura returnează șirul s obținut prin convertirea numărului nr de la tipul întreg la tipul **string**. Parametrul **n** indică numărul total de caractere pe care se face conversia (numărul de octeți ai **s**).	Definiția funcțiilor: **char *itoa(int** valoare, **char ***sir, **int** baza**);** Apelarea: **itoa**(v1, s1, b1); Funcția **itoa** returnează șirul s1 obținut prin conversia numărului întreg v1 din baza b1 la tipul **char** (dacă b1=10 atunci se reține și semnul lui v1). Există două funcții similare funcției **itoa**:**ltoa** și **ultoa**. prima convertește în șir o valoare de tip **long int**, a doua convertește o valoare de tip **unsigned long.**
Ambele programe afișează secvența "2006".	

- **Conversia unei secvențe de caractere într-o valoare numerică**

LIMBAJUL PASCAL	LIMBAJUL C/C++
procedura **val**	funcțiile **atol, atof**
program val1; **var** s1:**string**; i,c:**integer**; r:**real**; **begin** s1:='2006'; **writeln**('Sirul: ',s1); **val**(s1,i,c); **if** c=0 then **writeln**('Numarul:',i:4) **else** **writeln**('Eroare: s1= ',s1); s1:= '-1.23'; **writeln**('Sirul: ',s1); **val**(s1, r, c); **if** c=0 **then writeln**('Numarul:',r:5:2) **else** **writeln**('Eroare: s1= ',s1); s1:='17 ani'; **writeln**('Sirul: ',s1); **val**(s1,r,c); **if** c=0 **then writeln**('Numarul:',r:5:2)	**#include<iostream.h>** **#include<stdlib.h>** **void main (void)** {**long** l; **float** f; **char** s1[5]="2006"; **char** s2[6]="-1.23"; l=**atol**(s1); **cout**<<"s1: "<<s1<<";l: "<<l<<**endl**; l=**atol**("17ani"); **cout**<<" l: "<<l<<**endl**; f=**atof**(s2); **cout**<<"s2: "<<s2<<"; f: "<<f<<**endl**; f=**atof**("3.02*7"); **cout**<<" f: "<<f<<**endl**; }

else **writeln**('Eroare la conversie.'); **end**.	
Definiția procedurii: **procedure val(**sursa**; var** v:**integer; var** cod**:integer);** Apelarea: **val**(s1,nr,c); Procedura returnează numărul nr (întreg sau real) la care a fost convertit șirul s1 și valoarea întreagă c, indicând modul în care s-a realizat conversia: dacă s1 a putut fi convertit (nu a conținut decât cifre, virgulă, punct zecimal, semnul + sau -) atunci c = 0; altfel c ? 0 (conversia nu s-a realizat).	Definiția funcțiilor: **long atol(char ***sursa**);** **double atof(char ***sursa**);** Apelarea: n = **atol**(s1); n = **atof**(s1); Funcția **atol** (**atof**) convertește șirul s1 la un întreg de tip **long** (la un real de tip **float**); dacă s1 nu se poate converti, atunci funcția returnează valoarea 0. Observație Există și funcțiile **atoi, strtol, strtoul, strtod** care convertesc un șir la: un întreg, un întreg de tip **long**, un întreg de tip **long** fără semn, respectiv la un real de tip **double**.
Programul afișează - pe 3 rânduri - 2006, -1.23 și Eroare: s1= 17 ani	Programul afișează - pe 4 rânduri - 2006, 17, -1.23 și 3.02

PROBLEME PROPUSE

Realizați, în limbajul de programare studiat, câte un program pentru rezolvarea fiecăreia dintre următoarele probleme:

1. Pentru fiecare elev din clasă se citesc, pe rând, numele și prenumele (scrise cu litere mici și/sau mari).

Să se afișeze (pentru fiecare elev), pe un singur rând, numele (cu majuscule) și prenumele (doar prima literă să fie majusculă).

Să se afișeze numele elevului cu cel mai lung prenume și prenumele elevului cu cel mai lung nume.

Să se ordoneze crescător elevii după numele de familie.

Să se ordoneze descrescător elevii după prenume.

2. *Să se verifice dacă o frază este de tip palindrom (exemple: Au o nava noua. Icre, pui, ciuperci. Ele fac cafele. Sa nu iei un as. O rama maro.)* În acest scop se vor elimina spațiile (codul ASCII al spațiului este 032).

3. Se citește o "frază". *Se cere să se afișeze:*

- numărul "cuvintelor" care apar în "frază";
- cel mai lung și cel mai scurt cuvânt precum și poziția începând cu care apar acestea în "frază";
- să se șteargă din "frază" primul "cuvânt" care începe cu litera I;
- să se insereze în "frază", după fiecare "cuvânt", același "cuvânt" dar scris cu majuscule.

4. Se citește o "frază". *Se cere să se afișeze:*

- *toate perechile de "cuvinte" adiacente care "rimează";*
- *toate perechile de "cuvinte" neadiacente care "rimează", împreună cu poziția primei lor litere în frază (considerăm că două cuvinte rimează dacă cel puțin ultimele lor două litere coincid).*

5. Se citește un șir format numai din litere și spații. *Se cere să se afișeze șirul, după ce toate vocalele consecutive au fost înlocuite cu majuscule.*

6. Se citește un șir de cifre zecimale. *Se cere să se afișeze șirul după ce toate cifrele pare* ***p*** *care apar au fost înlocuite cu cifrele corespunzătoare (9–p) iar cifrele impare* ***i*** *cu cifrele corespunzătoare (i-1). Exemplu: pentru șirul 143265 se va afișa șirul 052734.*

7. Statisticile arată că fiecare utilizator de e-mail are 1,8 (!!) căsuțe de poștă electronică. *Să se citească toate adresele e-mail ale unei persoane și să se afișeze separat numele-utilizator și adresa serverului (codul ASCII al caracterului @ este 064).*

8. Se citesc mai multe adrese **Web**. *Să se afișeze separat subșirul care reprezintă: schema (http// , ftp:// etc.), numele serverului, calea, numele fișierului care trebuie încărcat.*

9. Se citesc două șiruri. *Se cere să se afișeze un subșir comun de lungime maximă.*

10. Se citește un șir. *Se cere să se afișeze cel mai lung subșir format din litere ordonate alfabetic (ab, abbc, afgk etc.).*

MINIPROIECT ÎN ECHIPĂ. COMPANIA EFICIENT

Etapa: *Realizare și testare* (implementarea unităților de program în limbajul de programare studiat)

Cerințe:

1. Realizarea și testarea individuală a programelor.
2. Integrarea programelor într-o aplicație unitară.
3. Completarea documentației cu specificațiile tehnice de programare.

6. RECURSIVITATEA

A. DEFINIRE. EXEMPLIFICARE

Toți algoritmii sunt reprezentați printr-o succesiune logică și finită de pași descriși prin structuri specifice: structuri secvențiale, structuri decizionale sau structuri repetitive.

Un algoritm care conține structuri repetitive poate fi reprezentat: *iterativ* sau *recursiv.*

Reprezentarea iterativă a unui algoritm repetitiv implică reluarea parcurgerii secvenței de operații până când este îndeplinită condiția de oprire. În limbajele de programare cunoscute, structurile repetitive sunt implementate iterativ cu ajutorul instrucțiunilor corespunzătoare.

LIMBAJUL PASCAL	LIMBAJUL C/C++
for i:=1 **to** n **do** **begin** secvența de operații **end**; **for** i:=n **to** downto 1 **do** **begin** secvența de operații **end**;	**for** (i=1; i<=n; i++) { secvența de operații } **for** (i=n; i>=1; i—) { secvența de operații }
while (cond_logică) **do** **begin** secvența de operații **end**;	**while** (cond_logică) **do** { secvența de operații}
repeat secvența de operații **until** (cond_logică);	**do** {}**while**(cond_logică)

Repetarea unei secvențe de instrucțiuni (S) poate fi determinată și prin organizarea modulară a programului astfel încât S să aparțină unui modul care se autoapelează (cap. 2, fig. 40b). Autoapelul este în acest caz comanda de execuție a secvenței S pentru prelucrarea unor alte date de intrare. Ieșirea din secvența de autoapeluri este controlată printr-o condiție de oprire.

Definirea în matematică a unor relații recurente (șiruri recurente) și prelucrarea acestora au stat la baza construirii mecanismului recursivității pentru reprezentarea algoritmilor.

Exemplu:

Se consideră un număr ***n*** natural nenul. Să se calculeze şi să se afişeze n! (factorialul numărului natural *n*).

Varianta iterativă:

Pentru implementarea iterativă se foloseşte relaţia:

*n! = 1*2*...*n , unde 0!=1*

Se apelează din programul principal, respectiv din funcţia principală, funcţia *fact* care returnează valoarea lui ***n!***. În corpul funcţiei, variabila ***p*** se iniţializează cu valoarea 1 şi apoi într-o instrucţiune repetitivă cu contor se repetă de ***n*** ori operaţia p=p*i, i=1,n. Funcţia va returna valoarea finală a produsului ***p***.

IMPLEMENTAREA ITERATIVĂ	
LIMBAJUL PASCAL	**LIMBAJUL C/C++**
uses crt; var n:integer; function fact(n:integer):longint; var i:integer; p:longint; begin p:=1; for i:=1 to n do p:=p*i; fact:= p; end; begin; clrscr; write(‘ n = ‘);readln(n); writeln(‘ Factorialul ‘,n,’! = ‘, fact(n)); end.	#include<iostream.h> #include<conio.h> long fact(int n) { int i; long p=1; for (i=1;i<=n;i++) p=p*i; return p; } void main() { int n; clrscr(); cout<<” n = “;cin>>n; cout<<” Factorialul “<<n<<”! = “<<fact(n); }

Varianta recursivă:

Se foloseşte umătoarea relaţie de recurenţă:

$$fact(n) = \begin{cases} 1, \text{ pentru } n = 0 \\ n * \text{fact}(n-1) \end{cases}$$

Exemplu: Pentru n=4, atunci n! se determină astfel:

Se apelează din programul principal, respectiv din funcţia principală funcţia *fact* care returnează valoarea lui ***n!***. Funcţia *fact* are ca parametru formal numărul natural ***n***. Funcţia este

construită pe baza relației de recurență, definită în enunțul problemei. După apel, funcția se autoapelează cu parametrul formal (n-1) multiplicat cu *n* adică: *n*fact(n-1)*. Dacă n>0 atunci se continuă autoapelul (n → n-1 , iar parametrul funcției are valoarea n-2). Șirul de autoapeluri se încheie când este îndeplinită *condiția de oprire*.

Conform relației de recurență a factorialului deducem *condiția de oprire: n=0.*

Autoapelurile au loc de sus în jos, (apelul principal → condiția de oprire) iar construcția soluției se face de jos în sus (condiția de oprire → apelul principal), adică:

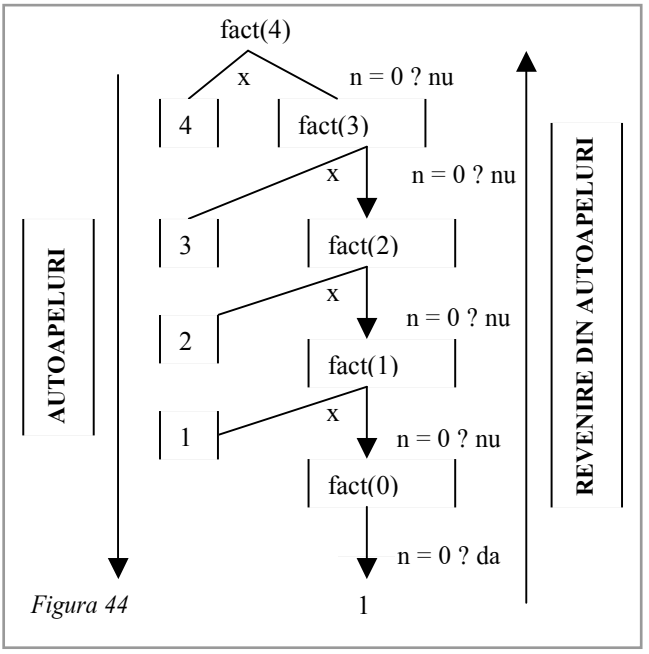


Figura 44

4!= fact(4)=4*fact(3)=4*(3*fact(2))=4*(3*(2*fact(1)))=4*(3*(2*(1*fact(0))))=

→

Autoapeluri ale subprogramului fact până la întâlnirea condiției de oprire n =0

= 4*(3*(2*(1*1))) = 4*(3*(2*1))= 4*(3*2)=4*6=24

→

Construirea soluției după întâlnirea condiției de oprire

IMPLEMENTAREA RECURSIVĂ

LIMBAJUL PASCAL

```
uses crt;
 var n:integer;

function fact(n:integer):longint;
  begin
   p:=1;
   if n=0 then fact:=1
          else fact:=n*fact(n-1)
  end;
```

LIMBAJUL C/C++

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

long  fact(int n)
 { int i;
   if (!n) return 1;
   else
       return n*fact(n-1);
 }
```

```
begin;
  clrscr; _
  write(‘ n = ‘);readln(n);
  writeln(‘ Factorialul ‘,n,’! = ‘, fact(n));
 end.
```

```
void main()
{ int n;
  clrscr();
  cout<<” n = “;cin>>n;
  cout<<” Factorialul “<<n<<”! = “<<fact(n);
  }
```

Un *algoritm recursiv* conține un bloc procedural care se autoapelează, altfel spus se apelează pe el însuși din interiorul său. Din afara blocului procedural se face un prim apel al acestuia, după care blocul se autoapelează de un anumit număr de ori; la fiecare nouă autoapelare se execută secvența de instrucțiuni din codul său, eventual cu alte date, realizându-se un așa-numit “*lanț de apeluri recursive*”.

Pentru a defini corect recursivitatea trebuie urmărite următoarele aspecte:

1. Condiția de oprire: În corpul subprogramului recursiv există unul sau mai multe teste prin care se verifică condițiile de ieșire din autoapelul recusiv. Aceste teste de ieșire formează c*ondiția de oprire*.

În subprogramul ***fact*** *condiția de oprire* este n=0.

2. Prelucrarea progresivă: Esența definiției recursive constă în rezolvarea cazului curent apelând la cazul precedent. În acest fel, celelalte cazuri ale autoapelului recursiv, diferite de condiția de oprire, trebuie rezolvate prin tratarea lor astfel încât procesul apelului recursiv să le orienteze către una dintre condițiile de oprire.

În exemplul prezentat, valoarea parametrului formal ***n*** este decrementată până când atinge valoarea 0 (*condiția de oprire*).

TEME

1. Stabiliți care dintre următoarele afirmații sunt adevărate sau false, prin marcarea căsuței corespunzătoare:

Nr. crt.	**Afirmații**	**Răspuns**	
		A	F
1	Un algoritm care nu conține structuri repetitive poate fi reprezentat atât iterativ, cât și recursiv.		
2	Executarea unor secvențe de operații, de un număr cunoscut de ori, este tratată iterativ prin intermediul unor structuri repetive corespunzătoare.		
3	Executarea unor secvențe de operații, de un număr cunoscut de ori, este tratată recursiv prin intermediul autoapelurilor aceluiași subalgoritm de un număr finit de ori.		
4	Subalgoritmul se autoapelează până când este îndeplinită condiția de oprire.		

2. Se consideră următoarea definiție (relație de recurență):

$$f(n) = \begin{cases} 1, \text{ dacă } n = 1 \\ \dfrac{f(n-1)}{a + f(n-1)}, \text{ dacă } n > 1,\ a \in R^* \end{cases}$$

a) Stabiliți care este condiția de oprire din autoapel.

b) Pentru n=4 construiți toate autoapelurile realizate.

3. Analizați următorul enunț și puneți în evidență aspectele recurente:

Succesorul oricărui număr natural este un număr natural: zero este număr natural.

B. MECANISMUL DE IMPLEMENTARE A RECURSIVITĂȚII

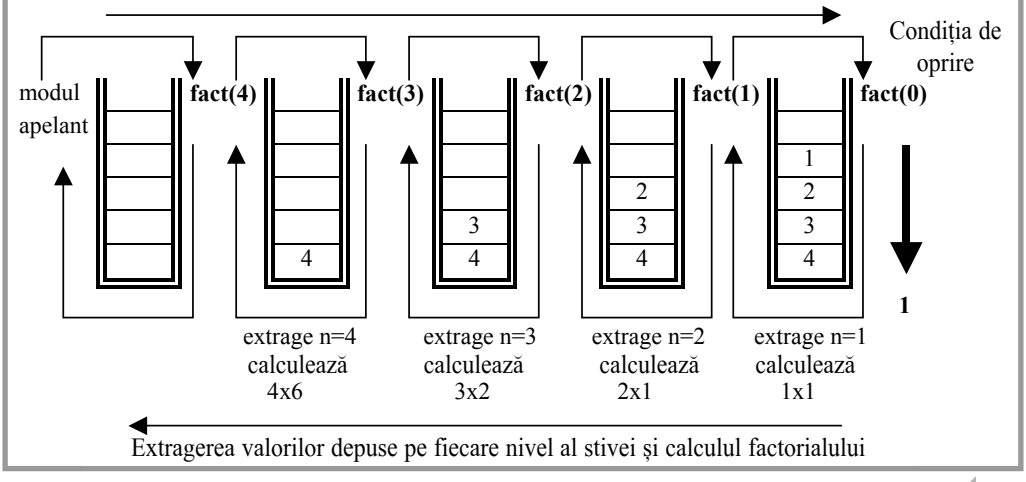
În capitolul *'Subprograme'* s-a precizat că orice apel de subprogram are ca efect salvarea într-o zonă de memorie denumită stivă (segment de stivă) a adresei de revenire de după apel, a valorilor parametrilor formali transmiși prin valoare și a adresei parametrilor transmiși prin adresă, precum și alocarea de spațiu pe stiva sistemului pentru variabilele locale ale subprogramului.

La revenirea în subprogramul apelant, se extrag în ordine inversă valorile salvate pe stivă (atât ale variabilelor locale, cât și ale parametrilor).

În cazul subprogramelor recursive, la apelul subprogramului se depun pe primul nivel al stivei sistemului, în ordine, parametrii formali transmiși, se alocă spațiu pentru variabilele locale (dacă este cazul) și adresa de revenire după apel. La fiecare autoapel al subprogramului, se urcă cu câte un nivel în stivă; pe acest nivel se depun noile valori ale parametrilor transmiși, se alocă spațiu pentru variabilele locale. Când se întâlnește condiția de oprire se încheie autoapelul subprogramului, se extrag din stivă (prin vârful stivei, de sus în jos) în ordine inversă valorile depuse și se efectuează prelucrările impuse în secvența de operații executată la fiecare autoapel. Când se revine în modulul apelant, stiva sistemului este vidă.

Exemplificarea mecanismului recursivității pentru determinarea valorii funcției 4!=fact(4).

Figura 45.



TEME

1. Stabiliți care dintre următoarele afirmații sunt adevărate sau false, prin marcarea căsuței corespunzătoare:

Nr. crt.	Afirmații	Răspuns	
		A	F
1	Prin apelul unui subprogram se salvează în *segmentul de date* adresa de revenire de după apel, valorile parametrilor formali transmiși prin valoare și adresele parametrilor transmiși prin adresă și variabilele locale (dacă este cazul) ale subprogramului.		
2	La fiecare autoapel al subprogramului se urcă cu câte un nivel în stivă pe care se depun noile valori ale parametrilor formali și se alocă spațiu pentru variabilele locale.		
3	Când se întâlnește condiția de oprire se încheie autoapelul subprogramului, se extrage din stiva sistemului în *ordinea depunerii* lanțul de valori depuse și se efectuează prelucrările impuse.		
4	În modulul apelant se revine din subprogramul recursiv când s-au extras toate valorile depuse în stiva sistemului (stiva este vidă).		

2. Reprezentați conținutul stivei pe durata executării apelului principal, al autoapelurilor extragerii valorilor din stivă, construirea valorii returnate și revenirea în subprogramul apelant pentru următoarea definiție (relație de recurență):

$$f(n) = \begin{cases} 1, \text{ dacă } n = 1 \\ \dfrac{f(n-1)}{a + f(n-1)}, \text{ dacă } n > 1,\ a \in R^* \end{cases}$$

C. TIPURI DE RECURSIVITATE

Există două tipuri de recursivitate (fig. 46):

a) Recursivitate directă

Dacă un subprogram se autoapelează până la întâlnirea condiției de oprire se spune că recursivitatea este directă. În acest caz se execută următorii pași:

Pasul 1. Apelul subprogramului recursiv.

Pasul 2. Autoapelul subprogramului până la îndeplinirea condiției de oprire.

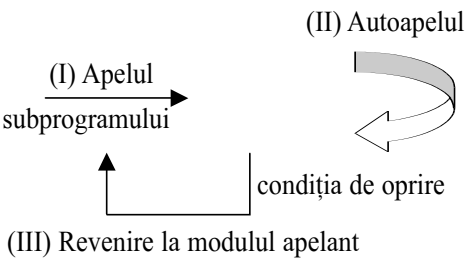
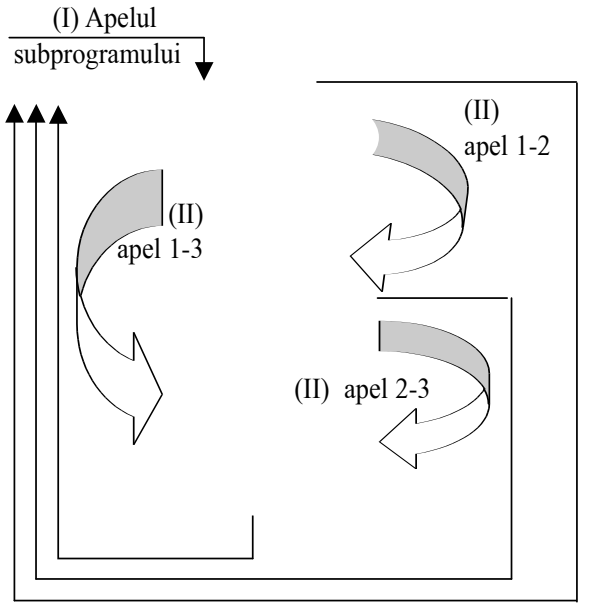
Pasul 3. Revenirea la modulul apelant.

b) Recursivitate indirectă

Dacă există două sau mai multe subprograme care se apelează reciproc până când se întâlnește condiția de oprire în cel puțin unul dintre ele, recursivitatea este indirectă sau încrucișată. În acest caz se execută următorii pași:

Pasul 1. Apelul subprogramului.
Pasul 2. Apeluri încrucișate până la întâlnirea condiției de oprire.
Pasul 3. Revenirea la modulul apelant.

Figura 46

Reprezentarea recursivității directe	Reprezentarea recursivității indirecte
	

Implementarea recursivității indirecte în limbajele de programare are anumite particularități:

LIMBAJUL PASCAL	LIMBAJUL C/C ++
În limbajul Pascal se poate apela un subprogram care nu a fost declarat, folosind o declarație fictivă (convențională) care este formată din **antet** (numele subprogramului urmat eventual de lista parametrilor formali) și cuvântul rezervat **FORWARD**, care este separate de antet prin '**;**'. Această declarație de funcție sau procedură permite referirea la ea fără a fi definită în momentul apelării. Corpul subprogramului se va scrie fiind precedat din nou de antetul acestuia fără a mai fi necesar declararea parametrilor formali.	În limbajul C/C++ se poate apela o funcție care nu a fost declarată, folosind o declarație fictivă (convențională) care este formată din **antet** (numele subprogramului urmat eventual de lista parametrilor formali). Această declarație de funcție permite referirea la ea fără a fi definită în momentul apelării. Corpul funcției se va scrie fiind precedat din nou de antetul acesteia.

Exemplu *de implementare a recursivității indirecte:*

Să se afișeze toate numerele naturale cuprinse în intervalul dat [a, b].

De pe prima linie a fișierului text ***rec_ind.in*** se citesc numerele naturale **a** și **b** (a<b) separate prin câte un spațiu. Numerele naturale din intervalul [a, b] sunt afișate în fișierului text ***rec_ind.out***, separate prin câte un spațiu.

Rezolvare:

Din programul principal, respectiv din funcția principală se apelează subprogramul ***increment(a)*** cu parametrul efectiv ***a***. Acest subprogram are rolul de a genera toate numerele naturale consecutive, începând cu ***a***. Din acest subprogram se apelează subprogramul ***scrie(n)***, care are rolul de a scrie în fișerul ***rec_ind.out*** valoarea parametrului formal ***n*** transmis și apoi apelează subprogramul ***increment(a)*** până când este îndeplinită condiția de oprire (n<b).

Implementarea problemei:

LIMBAJUL PASCAL	LIMBAJUL C/C++
var a,b:integer; f,g:text; **procedure increment(n:integer); forward**; procedure scrie(n:integer); begin write(g,n,' '); if(n<b) then increment(n+1); end;	#include<fstream.h> ifstream f ("rec_ind.in"); ofstream g ("rec_ind.out"); int a,b; **void increment(int n);** void scrie(int n) { g<<n<<" "; if(n<b)

LIMBAJUL PASCAL	LIMBAJUL C/C++
procedure increment(n:integer); begin scrie(n); end; begin assign(f,'rec_ind.in');reset(f); assign(g,'rec_ind.out');rewrite(g); read(f,a,b); increment(a); close(f);close(g); end.	increment(n+1); } void increment(int n) { scrie(n); } void main() {f>>a>>b; increment(a); f.close();g.close(); }
Rec_ind.in 12 20	***Rec_ind.out*** 12 13 14 15 16 17 18 19 20

D. APLICAȚII IMPLEMENTATE RECURSIV

Recomandări pentru elaborarea unui subprogram recursiv:

1. Înainte de a trece la implementarea unui subprogram recursiv este necesar să se identifice corect relația recursivă; pot exista două situații:
   - relația recursivă este o formulă de recurență matematică;
   - relația de recurență trebuie determinată din enunțul problemei.
2. Necesitatea existenței în subprogram a unei condiții corecte de oprire din autoapeluri pentru a se evita autoapelarea la infinit.
3. *Reducerea listei parametrilor formali ai subprogramului* poate elimina riscul unei recursivități dezavantajoase. Pot fi eliminate din lista parametrilor formali acele argumente care nu trebuie stocate în stivă, declarându-le ca variabile globale.
4. Greșeli frecvente în elaborarea unui program recursiv:
   - dacă subprogramul recursiv conține parametrii formali transmiși prin valoare sau prin referință (prin adresă) și/sau variabile locale care ocupă spațiu mare de memorie, stiva sistemului se va umple foarte repede, ajungându-se la depășirea spațiului rezervat acesteia chiar și pentru un număr mic de autoapeluri;
   - un autoapel incorect conduce la un rezultat eronat;
   - condiția de ieșire din autoapel greșit formulată conduce la autoapel infinit și umplerea stivei sistem (mesajul: *Stack overflow)*.

Concluzie:

Pentru anumiți algoritmi este posibil să se transforme reprezentarea recursivă într-o reprezentare simplă, iterativă.

✓ *Recursivitatea reduce complexitatea unor algoritmi.*

✓ *Deși programele recursive sunt mai dificil de elaborat, totuși ele sunt foarte puternice, formând baza limbajelor inteligenței artificiale.*

✓ *La alegerea între metoda recursivă și iterativă în elaborarea unui program, trebuie să se țină seama de eficiența oferită programului de către fiecare dintre variante, urmărind timpul de rulare și spațiul de memorie necesar. Nu în ultimul rând, trebuie urmărită claritatea programului.*

1. ȘIRUL LUI FIBONACCI

Să se afișeze al ***n***-lea termen al șirului lui Fibonacci, folosind un program care să respecte următoarea definiție:

$$fibo(n) = \begin{cases} 1, \text{ dacă } n < 2 \\ \text{fibo}(n-1) + \text{fibo}(n-2), \text{ altfel} \end{cases}$$

Soluție:

Se scrie o funcție recursivă *fibo(n)*, care va returna al ***n***-lea termen al șirului. Conform definiției, acesta este egal cu predecesorul său f_{n-1} pe care l-ar returna apelul *fibo(n-1)* plus anteprecedentul său f_{n-2} pe care l-ar returna apelul *fibo(n-2).* Deci în interiorul apelului fibo(n) apar două apeluri recursive. Condiția de oprire din lanțul de apeluri recursive este „*n=1* sau *n=2*“.

LIMBAJUL PASCAL	LIMBAJUL C/C++
var n:integer; function fibo(n:integer):integer; begin if n>1 then fibo:=fibo(n-1)+fibo(n-2) else fibo :=1; end; begin; write(‘ n = ‘);readln(n); writeln(‘ Elementul ‘,n,’ = ‘, fibo(n)); readln end.	#include<iostream.h> int fibo(int n) { if (n>1) return (fibo(n-1)+fibo(n-2)); else return 1; } void main() { int n; cout<<” n = “; cin>>n; cout<<” Elementul “<<n<<” = “<<fibo(n); }

2. CEL MAI MARE DIVIZOR COMUN

Dându-se două numere întregi ***a*** și ***b*** să se determine cel mai mare divizor comun al celor două numere, folosind:

a) algoritmul lui Euclid (se aplică algoritmul lui Euclid, cunoscut la matematică);

b) algoritmul lui Nicomachus, descris de următoarea relație:

$$cmmdc(a,b) = \begin{cases} a, \text{ dacă } a = b \\ cmmdc((a-b),b), \text{ dacă } a > b \\ cmmdc(a, (b-a)), \text{ în rest} \end{cases}$$

a) Algoritmul lui Euclid

Se construiește funcția recursivă *cmmdc(a,b)* care va returna valoarea ultimului rest diferit de zero, respectiv *b* sau *r*. Algoritmul prelucrează perechi de date întregi, de orice semn și cu orice relație de ordine între ele. Se observă că la fiecare autoapel funcția va executa aceeași secvență de instrucțiuni (un "pas" al algoritmului), dar cu alte valori ale lui ***a*** și ***b***. Oprirea lanțului de apeluri recursive are loc când restul împărțirii întregi al celor doi parametri este zero.

LIMBAJUL PASCAL	LIMBAJUL C/C++
var a,b:integer; function cmmdc(a,b:integer):integer; begin if a mod b <>0 then cmmdc:=cmmdc(b, a mod b) else cmmdc:=b; end; begin; write(' a = ');readln(a); write(' b = ');readln(b); writeln(' (',a,' , ',b,') = ', cmmdc(a,b)); end.	#include<iostream.h> int cmmdc(int a, int b) { if ((a%b)!= 0) return (cmmdc(b,a%b)); else return b; } void main() { int a,b; cout<<" a = ";cin>>a; cout<<" b = ";cin>>b; cout<<" ("<<a<<", "<<b<<") = "<<cmmdc(a,b); }

b) Algoritmul lui Nicomachus

Se construiește funcția recursivă *cmmdc(a,b),* la fiecare autoapel funcția va executa aceeași secvență de instrucțiuni (un "pas" al algoritmului), dar cu alte valori ale lui ***a*** și ***b***. Oprirea lanțului de apeluri recursive are loc când cei doi parametri au devenit egali. Pe baza acestor observații, definiția recursivă a algoritmului este evidentă, iar funcția recursivă transpune efectiv într-o instrucțiune ***if*** această definiție. Algoritmul prelucrează perechi de date întregi, de orice semn și cu orice relație de ordine între ele, dar la apelul din programul principal sau din funcția principală parametrii efectivi trebuie să fie în valoare absolută.

LIMBAJUL PASCAL	LIMBAJUL C/C++
var a,b:integer; function cmmdc(a,b:integer):integer; begin if (a = b) then cmmdc:=a else if (a>b) then cmmdc:= cmmdc(a-b,b) else cmmdc:=cmmdc(a,b-a); end; begin; clrscr; write(‘ a = ‘);readln(a); write(‘ b = ‘);readln(b); writeln(‘ (‘,a, ‘,’,b,’) = ‘, cmmdc(abs(a),abs(b))); end.	#include<iostream.h> #include<math.h> int cmmdc(int a, int b) { if (a==b) return a; else if (a>b) return (cmmdc(a-b,b)); else return (cmmdc(a,b-a)); } void main() { int a,b; cout<<” a = “;cin>>a; cout<<” b = “;cin>>b; cout<<” (“<<a<<”,”<<b<<”) = “; cout<<cmmdc(abs(a),abs(b)); }

3. CITIREA ELEMENTELOR UNUI TABLOU UNIDIMENSIONAL

Să se scrie un program pentru citirea și afișarea elementelor unui șir ***y*** cu ***m*** elemente reale. Dimensiunea șirului și elementele acestuia se citesc de la tastatură, iar afișarea se va face pe monitor.

Rezolvare:

Pentru citirea elementelor șirului de numere se utilizează subprogramul recursiv citire care are ca paramnetru indicele primului element citit. Acest indice este incrementat la fiecare autoapel, până când devine egal cu dimensiunea **n** a șirului. Condiția de oprire este i=n. În mod asemănător se construiește și subprogramul recursiv de afișare a elementelor tabloului.

LIMBAJUL PASCAL	LIMBAJUL C/C++
var y:array[1..20] of real; m: integer; procedure citeste(i : integer); begin if(i<=m) then begin read(y[i]); citire(i+1) end; end; begin read(m) ; citeste(1); end.	#include<iostream.h> float y[20]; int m; void citire(int i) { if(i<=m) { cin>>y[i]; citire(i+1); } void main() { cin>>m; citire(1); }

APLICAȚII DE LABORATOR

1. Scrieți un program pentru completarea elementelor unui șir ***y*** de numere naturale și afișarea acestora, utilizând subprograme recursive. Numărul de elemente ***t*** este citit de la tastură, fiecare element al șirului este generat aleatoriu (0<y[i]< 80, i=1,***t***). Elementele șirului se vor afișa pe monitor (separate prin câte un spațiu).

2. Stabiliți care este efectul programului următor. Formulați un enunț al unei aplicații care se poate rezolva cu ajutorul acestui program.

LIMBAJUL PASCAL	LIMBAJUL C/C++
var a:array[1..20,1..20] of integer; m,n:integer; f,g:text; procedure citire(i,m,j,n:integer); begin if(i<=m) then if(j<=n) then begin read(f,a[i][j]); citire(i,m,j+1,n); end else citire(i+1,m,1,n); end; procedure afisare(i,m,j,n>integer); begin if(i<=m) then if(j<=n) then begin write(g,a[i][j],' '); afisare(i,m,j+1,n); end; end; begin assign(f,'cit_mat.in');reset(f); assign(g,'cit_mat.out');rewrite(g); read(f,m,n); citire(1,m,1,n); afisare(1,m,1,n); close(f);close(g); end.	#include<fstream.h> ifstream f("cit_mat.in"); ofstream g("cit_mat.out"); int a[20][20]; void citire(int i,int m,int j,int n) { if(i<=m) if(j<=n) { f>>a[i][j]; citire(i,m,j+1,n); } else citire(i+1,m,1,n); } void afisare(int i, int m, int j, int n) { if(i<=m) if(j<=n) { g<<a[i][j]<<" "; afisare(i,m,j,+1,n); } else { g<<endl; afisare(i+1,m,1,n); } void main() { int m,n; f>>m>>n; citire(1,m,1,n); afisare(1,m,1,n); g.close(); f.close(); }

3. Din fișierul ***matrice.in*** se citește o matrice ***a*** cu ***m*** *x* ***n*** elemente întregi, astfel: de pe prima linie a fișierului se citesc numerele naturale nenule ***m*** și ***n*** separate prin câte un spațiu, iar de pe următoarele ***m*** linii se citesc ***n*** numere întregi separate prin câte un spațiu.

Datele de ieșire se vor afișa, pe linii distincte, în fișierul ***matrice.out.***

Să se scrie un program pentru :

a) citirea elementelor matricei din fișierul ***matrice.in***

b) afișarea elementelor matricei în fișierul ***matrice.out***

4. SUMA ELEMENTELOR UNUI ȘIR

Se consideră un șir ***x*** cu ***n*** numere reale. Să se calculeze suma celor **n** elemente ale tabloului.

Datele de intrare se citesc din fișierul ***suma.in***, astfel: de pe prima linie se citește dimensiunea ***n*** a tabloului numeric (***n*** ∈ N, 1<n<100), iar de pe următoarea linie se citesc ***n*** numere reale separate prin câte un spațiu.

În fișierul ***suma.out*** se va afișa șirul dat, pe prima linie, iar următoarea linie un număr real, cu trei zecimale, reprezentând suma elementelor.

LIMBAJUL PASCAL

```
var x:array[1..20] of real;        n:integer;
    f,g:text;
procedure citire(i:integer);
begin
  if(i<=n) then
    begin
      read(f,x[i]);        citire(i+1);
    end;
end;
procedure afisare(i:integer);
begin
  if(i<=n) then
    begin
      write(g,x[i]:2:3,' ');         afisare(i+1);
    end;
end;
function suma(i:integer):real;
 begin
  if(i<=n) then
     suma:= x[i]+suma(i+1)
   else
      suma:=0;
 end;
 begin
  assign(f,'suma.in');reset(f);
  assign(g,'suma.out');rewrite(g);
  read(f,n);   citire(1);   afisare(1); writeln(g);
  writeln(g,' suma = ', suma(1):2:3);
  close(f); close(g);
 end.
```

LIMBAJUL C/C++

```
#include<fstream.h>
#include<iomanip.h>
ifstream f("suma.in");
ofstream g("suma.out");
float x[20];     int n;
void citire(int i)
{
  if(i<=n)
    { f>>x[i];        citire(i+1);     }
}
void afisare(int i)
{
  if(i<=n)
    { g<<setprecision(3)<<x[i]<<" ";
      afisare(i+1);       }
}
float suma(int i)
{
  if(i<=n)
     return x[i]+suma(i+1);
   else
    return 0;
}
void main()
{ f>>n;
  citire(1); afisare(1);
  g<<" suma = "<<setprecision(3)<<suma(1);
  g.close();f.close();
}
```

APLICAȚIE DE LABORATOR

Se consideră un șir ***x*** cu ***n*** elemente întregi. Datele de intrare se citesc din fișierul ***min_max.in*** astfel: de pe prima linie un număr ***n*** natural nenul (care reprezintă numărul de elemente din șir), iar de pe următoarea linie ***n*** valori întregi separate prin câte un spațiu (care reprezintă elementele șirului **x**) .

Să se se scrie câte un program pentru fiecare cerință :

a) Determinarea simultană (în același subprogram) a minimului și a maximului.

Maximul și minimul se vor afișa în fișierul ***min_max.out***.

Exemplu: ***min_max.in***	***min_max.out***
5	Minimul = -3
6 2 1 7 -3	Maximul = 7

b) Modificați programul construit la punctul a) (salvați-l cu un alt nume) astfel încât să determinați simultan (în același subprogram) minimul și poziția acestuia în șir. Minimul și poziția acestuia se vor afișa în fișierul ***min_poz.out***.

Exemplu: ***min_max.in***	***min_poz.out***
5	Minimul = -3
6 2 1 7 -3	Poziția = 5

c) Modificați programul construit la punctul a) sau b) (salvați-l cu un alt nume) astfel încât să determinați simultan (în același subprogram) minimul, poziția minimului în șir, maximul și poziția maximului în șir. Minimul, poziția minimului, maximului și poziția maximului în șir se vor afișa în fișierul ***min_max3.out***.

Exemplu: ***min_max.in***	***mip_map.out***
5	Minimul = -3 pe poziția 5
6 2 1 7 -3	Maximul = 7 pe poziția 4

5. PRODUSUL SCALAR

Se consideră două șiruri de ***n*** numere întregi ***x*** și ***y***, având amândouă același număr de elemente ***n***, să se calculeze produsul lor scalar.

Produsul scalar se definește ca fiind un număr rezultat prin însumarea produselor elementelor cu aceiași indici din cele două tablouri.

Pentru datele problemei, produsul scalar = $x_1y_1 + x_2y_2 + \ldots + x_ny_n$.

Datele de intrare se citesc din fișierul ***scalar.in*** astfel: de pe prima linie un număr natural nenul ***n*** (dimensiunea celor două tablouri), de pe următoarea linie câte ***n*** valori întregi, separate prin câte un spațiu reprezentând elementele tabloului ***x***, iar de pe ultima linie ***n*** valori întregi, separate prin câte un spațiu reprezentând elementele tabloului ***y***. Valoarea produsului scalar se va afișa pe monitor.

Observație: Problema are scopul de a exemplifica un proces recursiv în care participă mai multe tipuri de date.

Rezolvare:

Funcția recursivă primește ca parametru indicele curent al elementelor din cele două tablouri, al căror produs este depus pe nivelul curent al stivei sistem. Condiția de oprire este îndeplinită atunci când indicele este egal cu *n* . Tablourile au fost declarate global.

LIMBAJUL PASCAL	LIMBAJUL C/C++
type vect=array[1..20]of integer; var n:integer; x,y: vect; f:text ; function prod_sc(i:integer):integer; begin if (i<=n) then prod_sc:= x[i]*y[i] + prod_sc(i+1) else prod_sc:= 0 ; end; procedure citeste(n : integer; var x:vect); begin for i:=1 to n do read(f, x[i]) end; end; begin assign('scalar.in'); reset(f); read(f,n); citeste(n,x); citeste(n,y); write(' produsul scalar = ', prod_sc(1)); close(f) end.	include<fstream.h> ifstream f("scalar.in"); int n, vect x[20] ,y[20]; int prod_sc(int i) { if (i<=n) return (x[i]*y[i] + prod_sc(i+1)); else return 0 ; } void citeste(int n, vect x) {int i; for (i=1;i<=n;i++) f>>x[i]; } void main() { f>>n; citeste(n,x); citeste(n,y); cout<<" Produsul scalar = "<<prod_sc(1); f.close(); }

APLICAȚII DE LABORATOR

1. Se consideră un șir ***x*** cu ***n*** (0<n<10) elemente reale. Datele de intrare se citesc din fișierul ***operatii.in*** astfel: de pe prima linie un număr ***n*** natural nenul (care reprezintă numărul de elemente din șir), iar de pe următoarea linie ***n*** valori întregi separate prin câte un spațiu (care reprezintă elementele șirului **x**).

Datele de ieșire se vor afișa, pe linii distincte, în fișierul ***operatii.out.***

Să se scrie un program:

a) citirea elementelor şirului din fişierul ***operatii.in***

b) afişarea elementelor şirului în fişierul ***operatii.out***

c) determinarea şi afişarea sumei elementelor şirului (cu exact trei zecimale);

d) determinarea şi afişarea produsului elementelor şirului (cu exact trei zecimale);

e) determinarea şi afişarea numărului de elemente strict negative din şir (cu exact două zecimale);

f) determinarea şi afişarea mediei aritmetice a elementelor strict pozitive din şir (cu exact patru zecimale).

Exemplu:

operatii.in	***operatii.out***
6 12 3 154 23 2 4	3 154 23 2 4 Suma = 198 Produsul = 1020096 Numarul de elemente impare = 2 Suma ultimelor cifre = 18

2. Se consideră o matrice $A_{n,n}$ cu elemente întregi. Datele de intrare se citesc din fişierul ***matrice.in*** astfel: de pe prima linie se citeşte numărul natural ***n*** nenul, iar de pe următoarele ***n*** linii se citesc **n** numere întregi separate prin câte un spaţiu. Datele de ieşire se vor afişa în fişierul ***matrice.out***. Cerinţe:

a) Citirea elementelor matricei din fişier.

b) Afişarea elementelor matricei în fişierul de ieşire.

c) Determinarea şi afişarea sumei elementelor de pe diagonala principală.

d) Determinarea şi afişarea produsului elementelor de pe diagonala secundară.

e) Afişarea elementelor situate deasupra diagonalei principale.

f) Afişarea elementelor situate sub diagonala secundară.

g) După interschimbarea liniei 1 cu linia n să se afişeze noua matrice.

Pentru fiecare cerinţă scrieţi câte un subprogram recursiv.

Exemplu:

matrice.in				
4 2 3 1 5 4 6 0 7 9 5 8 1 3 4 1 0	2 3 1 5 4 6 0 7 9 5 8 1 3 4 1 0 Suma Dp= 16 Produsul Ds= 0	Elementele situate deasupra diagonalei principale 3 1 5 0 7 1	Elementele situate sub diagonala secundară 7 8 1 4 1 0	Matricea după interschimbare 3 4 1 0 4 6 0 7 9 5 8 1 2 3 1 5

6. ORDONAREA ELEMENTELOR UNUI ȘIR

Se consideră un șir **x** cu ***n*** numere întregi. Să se ordoneze crescător elementele șirului.

Dimensiunea șirului și cele ***n*** elemente ale sale se citesc de la tastatură, iar șirul ordonat se afișează pe ecran.

Rezolvare:

Subprogramul recursiv *ordon(a,n)* va returna șirul ordonat crescător, prin metoda interschimbării. Autoapelurile se încheie când este îndeplinită condiția de oprire *i=n*.

LIMBAJUL PASCAL

```
type vect=array[1..20]of integer;

 var n,i:integer;
    a:vect;

 procedure ordon(var x:vect; n,i:integer)
  var aux:integer;
   begin
     if i<n then
           if x[i]>x[i+1] then
                              begin
                              aux:=x[i];
                               x[i]:=x[i+1];
                               x[i+1]:=aux;
                               ordon(x,n,1)
                              end
                            else ordon(x,n,i+1)
  end;

 procedure citeste(n,x);
  begin
   for i:=1 to n do
    begin
     write(‘ elementul ‘,i,’ = ‘); readln(x[i])
    end;
  end;

 begin;
  write(‘ n = ‘);readln(n);
  writeln(‘ Elementele sirului a ‘);writeln;
  citeste(n,a);writeln;
  ordon(a,n,1);
  writeln(‘ sirul ordonat crescator : ‘);
  for i:=1 to n do write(a[i],’ ‘)
 end.
```

LIMBAJUL C/C++

```
#include<iostream.h>

 int vect[20],n;

 void  ordon(int x,int n, int i)
  { int aux;
    if(i<n)
     if (x[i]>x[i+1])
        {  aux=x[i];
           x[i]=x[i+1];
           x[i+1]=aux;
           ordon(x,n,1);
        }
     else ordon(x,n,i+1);
   }

 void citeste( int n, int x)
  {int i;
   for (i=1;i<=n;i++)
    {
      cout<<” elementul “<<i<<” “;
      cin>>x[i];
    }
  }

  void main()
  { cout<<” n = “;cin>>n;
     cout<<” citirea sirului a “<<endl;
    citeste(n,a);
    cout<<endl;
    ordon(a,n,1);
    cout<<” Sirul ordonat = “<<endl;
    for(i=1;i<=n;i++) cout<<a[i]<<” “;
   }
```

7. SCRIEREA UNUI NUMĂR ÎNTR-O BAZĂ

Se citește un număr natural nenul ***n***. Să se transforme în baza ***b***, unde b=2..9.

Rezolvare:

Subprogramul recursiv *baza(n)* memorează pe fiecare nivel al stivei restul împărțirii lui **n** la **b** și câtul împărțirii lui **n** la **b** (care reprezintă noua valoare a lui **n**). Condiția de oprire este n=0 (ultimul cât este 0).

LIMBAJUL PASCAL	LIMBAJUL C/C++
uses crt; var n,b:integer procedure baza(n:integer; b:integer); var r:integer; begin r:=n mod b; if (n>=b) baza(n div b,b); write(r); end; begin clrscr; write(‘ n = ‘);readln(n); repeat write(‘ baza = ‘);readln(b); until(b<2 and b>10); write(n,’→’, baza(n,b); end.	#include<iostream.h> #include<conio.h> void baza(int n, int b) { int r=n%b; if (n>=b) baza(n/b,b); cout<<r; } void main() { int n, b; clrscr(); cout<<” n = “;cin>>n; do { cout<<” baza = “;cin>>b; }while((b<2) && (b>10)); cout<<n<< “→“<<baza(n,b); }

8. INVERSAREA UNUI CUVÂNT

Să se inverseze caracterele unui cuvânt citit de la tastatură, literă cu literă până la întâlnirea caracterului *spațiu*.

Rezolvare:

Se scrie un subprogram fără parametri formali, *invers()/invers*. La fiecare autoapel, se citește un nou caracter ***c***. Condiția de oprire a lanțului de apeluri este îndeplinită când caracterul citit este *spațiu*. Afișarea caracterelor în ordine inversă este realizată printr-o operație de scriere (*write*/*cout*). Când este îndeplinită condiția de oprire, caracterele depuse în stivă la fiecare autoapel sunt extrase și afișate (în ordine inversă).

LIMBAJUL PASCAL	LIMBAJUL C/C++
procedure inver; var c:char; begin readln(c); if (c<>' ') then inver; write(c); end; begin inver; end.	#include<iostream.h> void inver() { char c; cin>>c; if (c!='0') {inver(); cout<<c; } } void main() { inver(); }

PROBLEME PROPUSE

1. Se consideră următorul program :

LIMBAJUL PASCAL	LIMBAJUL C/C++
var x,y:integer; procedure t(x:integer; y:integer); begin if x>0 then begin x:=x-1; y:=y+1; t(x,y); end; end; begin x:=3;y:=1; write(x,' ',y,' '); t(x,y); writeln(x,' ',y); end.	#include <iostream.h> int x,y:integer; void t(int x, int y) { if (x>0) { x = x-1 ; y = y+1; t(x,y); } } void main() { x = 3; y = 1; cout<<x<<" "<<y<<" "; t(x,y); cout<<x<<" "<<y<<" "<<endl; }

Precizați ce va tipări programul de mai sus:

a) 3 1 1 1; **b)** 3 1 0 3; **c)** 3 1 0 2; **d)** 3 1 3 1; **e)** 3 1 0 1.

Scrieți modificările în program astfel încât după execuție să se tipărească 3 1 0 1.

Scrieți modificările în program astfel încât după execuție să se tipărească 3 1 0 4.

2. Scrieți subprograme recursive pentru calculul valorii următoarelor polinoame care verifică relațiile de recurență, într-un punct ***x*** real, cunoscut.

1. Polinomul Cebâșev de speța I

$$T_n(x) = \begin{cases} 1\ , & dacă\ n = 0 \\ x\ , & dacă\ n = 1 \\ 2xT_{n-1}(x) - T_{n-2}(x)\ , \end{cases}$$

2. Polinomul Cebâșev de speța II

$$U_n(x) = \begin{cases} 1\ , & dacă\ \ n = 0 \\ 2x\ , & dacă\ \ n = 1 \\ 2xU_{n-1}(x) - U_{n-2}(x)\ , \end{cases}$$

3. Polinomul lui Legendre

$$(n+1)P_{n+1}(x) = \begin{cases} 1\ , & dacă\ \ n = 0 \\ x\ , & dacă\ \ n = 1 \\ (2n+1)xP_n(x) - nP_{n-1}(x)\ , & dacă\ \ n = 0 \end{cases}$$

3. a) Să se stabilească care este efectul următorului subprogram recursiv pentru apelul s(4):

VARIANTA PASCAL/BORLAND PASCAL	VARIANTA C++/C
procedure s(y: integer) ; begin if (y>0) then begin writeln('***'); s(y-1); writeln('aaa'); end end;	void s(int y) { if (y>0) { cout<<"***"<<endl; s(y-1) ; cout<<"aaa"<<endl; } }

b) Modificați subprogramul și precizați apelul din programul principal/funcția principală astfel încât să se afișeze :

**aa
**aa
**aa
**aa

4. Se dă următoarea funcție recursivă care calculează valoarea polinomului lui Hermite în punctul ***x*** real. Cerințe:

a) Scrieți expresia după care se calculează valoarea polinomului, punând în evidență recurența matematică.

VARIANTA PASCAL/BORLAND PASCAL	VARIANTA C++/C
function her(x:real; n: integer) : real; begin if (n=0) then her:=1 else if (n=1) then her :=2*x else her:=2*x*her(x,n-1)- 2*(n-1)*her(x,n-2); end;	float her(float x, int n) { if (n==0) return 1; else if (n==1) return 2*x; else return 2*x*her(x,n-1) - 2*(n-1)*her(x,n-2); }

b) Care este valoarea afișată pentru apelul her(1,3):

(i) –4; (ii) 2; (iii) 0; (iv) 8; (v) 4.

c) Să se scrie un subprogram iterativ echivalent cu cel dat.

5. Se consideră următorul subprogram:

Varianta Pascal/Borland Pascal	Varianta C++/C
a) procedure u(k: integer); var d,p:integer; begin if k>1 then begin d:=s+1; p:=0; while (k mod d =0) then begin k:=k div d; p:=p+1 end; write(d,' ',p,' '); s:=d; u(k) end else writeln(k); end;	void u(int k) { int d,p; if (k>1) { d =s+1; p=0; while (k % d = =0) { k =k div d; p =p+1 } cout<<d<<" "<<p<<" "; s = d; u(k); } else cout<<k<<endl; }

a) Ce se va afișa pentru n=18 și s=1 ?

(i) 2 1 9 3 2 1; (ii) 2 1 3 2 1; (iii) 18 2 1 3 2 1; (iv) 2 1 9 3 1; (v) 2 1 3 1 1.

b) Scrieți instrucțiunea sau instrucțiunile necesare apelului acestui subprogram, precizând limbajul utilizat.

c) Formulați un enunț pe baza căruia să poată fi aplicat acest subprogram.

6. Fie șirul de numere definit astfel:

$$a_{n+1} = \begin{cases} m, dacă\ n = 0 & m \in R \\ \frac{a_n}{4}, \text{ dacă n este par} \\ 4a_n - 1, \text{ altfel} \end{cases}$$

a) Să se scrie un subprogram recursiv pentru calculul termenului $n \in N$, n>0, pentru o valoare cunoscută a parametrului real ***m***.

b) Care este valoarea termenului a_5 , dacă m=2:

(i) 5; (ii) 6; (iii) 12; (iv) 7; (v) 2.

7. Fie următorul subprogram recursiv:

VARIANTA PASCAL	VARIANTA C++
procedure **u**(a:integer) begin if a **div** 8 > 0 then u(a **div** 8); write(a **mod** 8) end;	void **u**(int a) { if (a/8 >0) u(a div 8); cout<<a%8; }

a) Pentru apelul **u**(39) precizați : numărul de autoapeluri ce se va afișa la revenirea din subprogram:

(i) 2 / 74; (ii) 1 / 74; (iii) 2 / 47; (iv) 1 / 47; (v) 3 / 47.

b) Transformați subprogramul recursiv într-un subprogram iterativ, precizându-se varianta de limbaj utilizată.

8. Se dă următorul subprogram recursiv pentru calculul valorilor unei funcții:

VARIANTA PASCAL	VARIANTA C++
function f(x:integer):integer; begin if x >= 15 then f:=x-2 else f:=f(x+4) end;	int **f**(int x) { if (x >15) return x-2; else return f(x+4) ; }

a) Să se identifice care este varianta corectă a conținutului stivei pentru calculul valorii funcției în punctul x=5 și care este valoarea funcției.

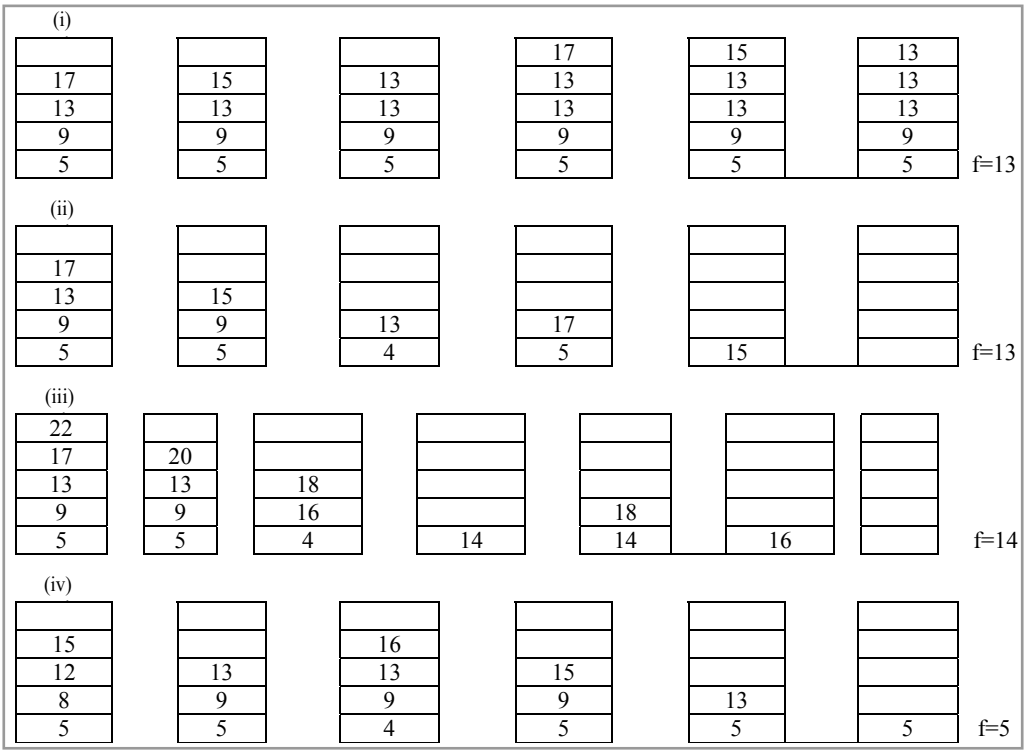


Figura 47.

b) Care sunt valorile posibile pe care le poate lua **f** ?
(i) {13,14,15,16}; (ii) {12,13,15,16}; (iii) {12, 14,16,18}; (iv) {14, 15, 16,18}; (v) o infinitate de valori.

9. Implementați recursiv preluările specifice structurii dinamice de tip *coadă*.
10. Implementați recursiv prelucrările specifice structurii dinamice de tip *stivă.*
11. Explicați aspectul recurent al structurilor dinamice arborescente (arborele de familie).

Capitolul 3

METODE DE PROGRAMARE

I. DIVIDE ET IMPERA

1. STUDIU DE CAZ – *CAMPIONAT DE BASCHET*

Anual are loc Campionatul Național de baschet al liceenilor. La acest concurs participă echipele clasate pe locul I din fiecare județ și echipa câștigătoare din municipiul București. La înscriere fiecare echipă primește un număr de înregistrare, începând de la 1 la n.

Toate echipele sunt împărțite în două grupe. Fiecare grupă se împarte în alte două subgrupe – câștigătorul unei grupe se stabilește în urma meciului dintre echipa câștigătoare din prima semigrupă și echipa câștigătoare din semigrupa a doua. La rândul ei fiecare semigrupă este împărțită în subgrupe, până când se obțin semigrupe formate din două echipe sau dintr-o singură echipă. Echipa care este singură într-o semigrupă va juca cu echipa câștigătoare din semigrupa corespunzătoare.

Finala are loc între echipele câștigătoare ale fiecărei grupe.

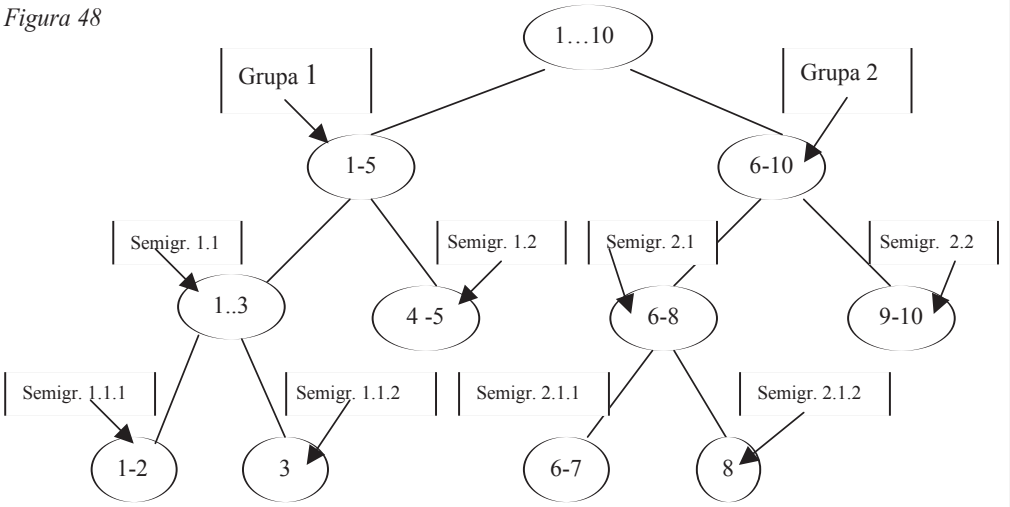
Să presupunem că în anul acesta s-au înscris 10 echipe, înregistrate cu numere de la 1 la 10. Prima grupă este formată din echipele numerotate de la 1 la 5, iar grupa a doua din echipele cu numere de înregistrare de la 6 la 10.

Organizatorii afișează, pe site-ul concursului, planificarea meciurilor.

Formarea semigrupelor și desfășurarea meciurilor din concurs pot fi urmărite și pe diagrama de mai jos.

Planificarea desfășurării meciurilor			
Grupa 1		Grupa 2	
Nr. meci	Echipele care joacă meciul	Nr. meci	Echipele care joacă meciul
1.1	echipa 1 - echipa 2	2.1.	echipa 6 – echipa 7
1.2	echipa câștigătoare din meciul 1.1. – echipa 3	2.2.	echipa câștigătoare din meciul 2.1. – echipa 8
1.3	echipa 4 - echipa 5	2.3.	echipa 9 joacă – echipa 10
1.4	echipa câștigătoare din meciul 1.2. – echipa câștigătoare din meciul 1.3.	2.4.	echipa câștigătoare din meciul 2.2. – echipa câștigătoare din meciul 2.3.
1-2	Finala are loc între echipa câștigătoare din meciul 1.4. și echipa câștigătoare din meciul 2.4.		
3-4	Semifinala are loc între echipele care au pierdut în meciurile 1.4. și 2.4.		

Figura 48



2. DESCRIEREA GENERALĂ A METODEI *DIVIDE ET IMPERA*

Pentru organizarea campionatului s-au folosit elemente specifice metodei *Divide et Impera*. Această metodă se aplică în rezolvarea problemelor care pot fi descompuse în subprobleme independente, asemănătoare problemei inițiale, de dimensiuni mai mici și care pot fi rezolvate mai ușor. În studiul de caz, problema inițială este *planificarea meciurilor* dintre cele 10 echipe. Prin descompunere, se ajunge la subprobleme din ce în ce mai mici, până la planificarea meciurilor dintre doar două echipe. După fiecare meci, echipa câștigătoare este desemnată să susțină meciul cu echipa din semigrupa corespunzătoare. În acest mod, prin compunerea rezultatelor, se ajunge la meciul final și desemnarea echipei campioane.

În rezolvarea problemei s-au parcurs două etape:

Etapa 1: descompunerea problemei este o etapă analitică prin care problema inițială este redusă – prin divizare – până la problema elementară.

Etapa 2: compunerea soluțiilor este o etapă de sinteză prin care soluțiile problemelor parțiale conduc la obținerea soluției finale.

Generalizare

Fie ***P(n)*** problema inițială care necesită prelucrarea a ***n*** elemente:

Etapa I – *Divide* (*Împarte*): Descompunerea problemei inițiale *P(n)* de dimensiune ***n*** într-un număr de subprobleme de același tip, independente, dar de dimensiuni mai mici, $P1(n_1)$, $P2(n_2)$,…, $Pk(n_k)$. Prin dimensiunea problemei se înțelege numărul de elemente prelucrate pentru rezolvarea acesteia.

Etapa II – *Impera* (*Stăpânește*):

- Rezolvarea subproblemelor P1, P2,…, Pk și obținerea soluțiilor parțiale S1, S2,…, Sk.
- Combinarea soluțiilor parțiale S1, S2,…, Sk pentru obținerea soluției S a problemei inițiale.

Observații:

- În funcție de dimensiunile rezultate, subproblemele P1, P2,…, Pk se rezolvă direct sau se descompun la rândul lor în alte subprobleme.
- Descompunerea în subprobleme se oprește atunci când se ajunge la subprobleme suficient de simple pentru a putea fi rezolvate direct, prin metode elementare (pentru exemplul dat, condiția de oprire a descompunerii în subgrupe este cazul în care subgrupa este formată din două sau o singură echipă).

3. ALGORITMUL METODEI

Repetarea divizării unei probleme în subprobleme de același fel, care se rezolvă pe domenii (intervale de valori) din ce în ce mai mici, este un procedeu recursiv cu condiția de oprire la nivelul subproblemelor elementare.

Se consideră un șir de valori $X=(x_1, x_2, \ldots, x_n)$ asupra căruia se aplică o prelucrare impusă de cerințele problemei P. Se poate demonstra că pentru orice ***p*** și ***u*** $\in \{1, 2, \ldots, n\}$, ***p*** < ***u***, există o valoare ***poz*** (***p***<***poz***<***u***) astfel încât prin prelucrarea subșirurilor $\{x_p, x_{p+1}, \ldots, x_{poz}\}$ și $\{x_{poz+1}, x_{poz+2}, \ldots, x_u\}$ se obțin soluțiile corespunzătoare. Prin combinarea acestor soluții se obține soluția prelucrării secvenței parțiale $\{x_p, x_{p+1}, \ldots, x_u\}$. Valoarea ***poz***, numită și ***pivot***, este, în cele mai frecvente cazuri, poziția elementului median – aflat la jumătatea intervalului [p,u].

Algoritmul metodei folosește un subprogram recursiv cu parametri de intrare ***p*** și ***u***; la primul apel, ***p*** = 1 și ***u*** = n. Soluția fiecărei subprobleme se transmite prin parametrul de ieșire, *sol.*

ALGORITMUL *DIVIDE et IMPERA*

```
algoritm divide_et_impera(p,u, sol)
   dacă (p<u) atunci
                  împarte(p,u,poz) // determină poziția pivotului
                  divide_et_impera(p, poz, sol1);  // se aplică algoritmul
                  divide_et_impera(poz+1, u, sol2); // pentru fiecare subproblemă
                  sol = combină(sol1, sol2); // formarea soluției
              altfel
                 rezolvă(sol)          //rezolvă subproblema elementară
   sfârșitdacă
stop algoritm divide_et_impera
```

ÎNTREBĂRI:

1. Stabiliți care dintre următoarele afirmații sunt adevărate sau false, prin marcarea căsuței corespunzătoare:

Nr. crt.	Afirmații	Răspuns	
		A	F
1	Metoda *Divide et Impera* este o metodă generală de programare care poate fi aplicată problemelor care se pot rezolva prin descompunerea problemei inițiale în subprobleme diferite de problema inițială, de dimensiuni mai mici și care pot fi rezolvate mai ușor.		
2	Subproblemele în care a fost divizată problema inițială sunt de dimensiuni mai mari decât cele ale problemei date.		
3	Descompunerea în subprobleme se oprește atunci când se ajunge la subprobleme suficient de simple pentru a putea fi rezolvate direct, prin metode elementare.		
4	Soluția finală se obține prin combinarea soluțiilor parțiale ale subproblemelor elementare.		

2. Justificați aspectul recursiv al raționamentului de rezolvare a problemelor prin metoda *Divide et Impera.*

3. Aplicați algoritmul metodei *Divide et Impera* pentru rezolvarea următoarelor probleme:

a) Căutarea unui abonat în cartea de telefoane.

b) Căutarea unui punct într-o suprafață plană.

c) Determinarea celui mai mare divizor comun a 4 numere. Generalizare.

Exemplu: cmmdc (6, 18, 12, 15) = 3.

4. IMPLEMENTAREA METODEI *DIVIDE ET IMPERA*

Implementarea metodei depinde de limbajul de programare și de specificul problemei. Aspectul recursiv al metodei se regăsește în subprogramul recursiv care reprezintă nucleul programului. Subprogramul recursiv are, obligatoriu, cei doi parametri de intrare ***p*** și ***u*** care reprezintă adresele primului și ultimului element din domeniul pe care se rezolvă, la un moment dat, problema. Dacă subprogramul este de tip funcție, parametrul de ieșire pentru soluție nu mai este necesar, soluția e returnată prin chiar numele funcției.

Implementarea metodei va fi exemplificată prin următoarele aplicații.

APLICAȚIA 1

Determinarea elementului minim

1. Să se determine valoarea minimă dintr-un șir ***x*** cu ***n*** elemente întregi, utilizând metoda *Divide et Impera*.

Datele de intrare se citesc de la tastatură sau din fișierul ***minim.in*** astfel: de pe prima linie numărul ***n*** natural nenul, iar de pe următoarea linie ***n*** valori întregi separate prin câte un spațiu. Minimul determinat se va afișa pe ecran.

Exemplu: ***minim.in*** Pe ecran se va afișa -3

5

6 2 1 7 -3

Rezolvare:

Pentru determinarea valorii minime din șirul de ***n*** numere, *descompunerea problemei* revine la împărțirea șirului în două subșiruri: subșirul (x_1, x_2, …, x_{poz}) și subșirul (x_{poz+1}, x_{poz+2}, …, x_n), unde **poz** este indicele elementului din mijloc: **poz**=[(1+n)/2]. Pentru determinarea minimului din fiecare dintre cele două subșiruri, se va împărți fiecare subșir în alte două subșiruri, și la rândul lor, cele două subșiruri în altele, până când se obțin subșiruri de dimensiune 1 (cu un singur element). Se obțin astfel, soluțiile elementare *min1* și *min2*. *Combinarea rezultatelor* se face prin compararea valorilor *min1* și *min2* obținute din fiecare pereche de subșiruri formate prin divizare. Dacă *min1* este mai mic decât *min2*, atunci minimul este *min1*, altfel minimul este *min2*.

Justificarea corectitudinii soluției:

- Dacă ***minim=min1*** atunci orice altă valoare din subșirul (x_1, x_2, …, x_{poz}) este mai mare sau egală cu ***min1.***
- Dacă ***minim=min2*** atunci orice altă valoare din subșirul (x_{poz+1}, x_{poz+2}, …, x_n) este mai mare sau egală cu ***min2.***
- Dacă ***min1<min2*** atunci ***min1***<x_k pentru k∈{poz+1, …, n} și ***min1*** determinat prin rezolvarea subproblemelor este cel mai mic element din subșirul (x_1, x_2, …, x_{poz}). În acest caz, ***min1*** este cea mai mică valoare din șirul dat.
- Dacă ***min1>min2*** atunci ***min2***<x_k pentru k∈{1, …, poz} și ***min2*** determinat prin rezolvarea subproblemelor este cel mai mic element din subșirul (x_{poz+1}, x_{poz+2}, …, x_n). În acest caz, ***min2*** este cea mai mică valoare din șirul dat.

În diagrama de mai jos sunt reprezentate toate subșirurile obținute prin divizarea șirului din exemplul numeric.

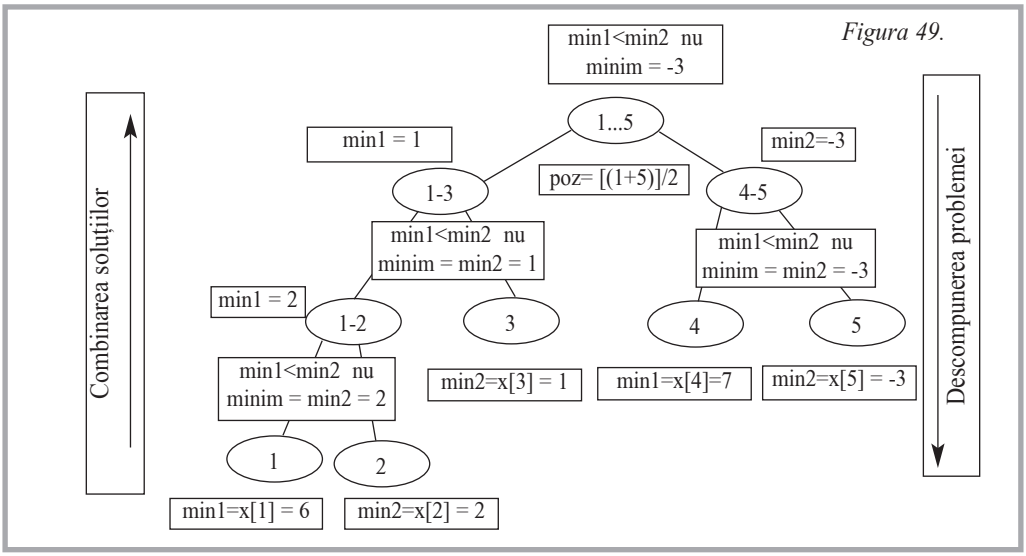


Figura 49.

Implementarea algoritmului

LIMBAJUL PASCAL	LIMBAJUL C/C++
var x: array[1..20] of integer; n, i:integer; f:text; procedure citire; {citirea din fişier} begin read(f,n); for i:=1 to n do read(f,x[i]); close(f); end; function divide_et_impera(p:integer; u:integer): integer; var poz,min1,min2:integer; begin if(p<u) then begin poz:=(p+u) div 2; min1:=divide_et_impera(p,poz); min2:=divide_et_impera(poz+1,u); if(min1<min2) then divide_et_impera:= min1 else divide_et_impera:= min2; end else divide_et_impera:=x[p]; end; begin assign(f,'minim.in');reset(f); citire; writeln('minim = ',divide_et_impera(1,n)); end.	#include<fstream.h> ifstream f("minim.in"); int x[20],n; void citire() // citirea din fişier { f>>n; for(int i=1;i<=n;i++) f>>x[i]; f.close(); } int divide_et_impera(int p, int u) { int poz,min1,min2; if(p<u) { poz=(p+u)/2; min1=divide_et_impera(p,poz); min2=divide_et_impera(poz+1,u); if(min1<min2) return min1; else return min2; } else return x[p]; } void main() { citire(); cout<<" minim = "<<divide_et_impera(1,n); g.close(); }

TEME

1. Modificaţi subprogramul ***divide_et_impera*** din programul prezentat mai sus astfel încât, programul să afişeze valoarea maximă dintr-un şir.

2. Se consideră un şir ***x*** cu ***n*** elemente întregi. Datele de intrare se citesc din fişierul ***min_max.in*** astfel: de pe prima linie un număr ***n*** natural nenul (care reprezintă numărul de elemente din şir), iar de pe următoarea linie ***n*** valori întregi separate prin câte un spaţiu (care reprezintă elementele şirului **x**) .

Cerinţe:

a) Determinaţi în acelaşi subprogram valoarea minimă şi valoarea maximă.

Exemplu: ***min_max.in***

5	Minimul = -3
6 2 1 7 -3	Maximul = 7

b) Modificați programul construit la punctul a) astfel încât să determinați simultan (în același subprogram) elementul minim și poziția acestuia în șir.

Exemplu: ***min_max.in***

5	Minimul = -3
6 2 1 7 -3	Poziția = 5

c) Modificați programul construit la punctul a) sau b) astfel încât să determinați simultan (în același subprogram) elementul minim și poziția sa în șir, precum și elementul maxim și poziția sa în șir.

Exemplu: ***min_max.in***

5	Minimul = -3 pe poziția 5
6 2 1 7 -3	Maximul = 7 pe poziția 4

EXERCIȚII

1. Se consideră un șir ***x*** cu ***n*** numere întregi.

a) Ce va afișa următorul program pentru ***n***=7 și valorile șirului ***x***: 12, -4, 13, 7, 8, 11, 2?

b) Formulați un enunț pentru o problemă care se poate rezolva cu ajutorul acestui program.

LIMBAJUL PASCAL	LIMBAJUL C/C++
var x: array[1..20] of integer; n:integer; …….. function s(p:integer; u:integer):integer; var poz :integer; begin if(p=u) then if (x[p] mod 2 = 0) then s:=1 else s:=0 else begin poz:=(p+u) div 2; s:= s(p,poz)+ s(poz+1,u); end end; begin …….. writeln(s(1,n)); …….. end.	#include<iostream.h> int x[20],n; …….. int s(int p, int u) { int poz; if(p==u) if(x[p]%2==0) return 1; else return 0; else { poz=(p+u)/2; return s(p, poz)+ s(poz+1,u); } } void main() {……. cout<<s(1,n); …….. }

2. Se consideră un șir ***x*** cu ***n*** numere întregi.

a) Ce va afișa următorul program pentru ***n***=8 și valorile șirului ***x***: 21, -14, 3, 5, 18, 11, 12, 8?

b) Formulați un enunț al unei probleme care se poate rezolva cu ajutorul acestui program.

LIMBAJUL PASCAL	LIMBAJUL C/C++
var x: array[1..20] of integer; n:integer; …….. function s(p:integer; u:integer):integer; var poz :integer; begin if(p=u) then if (x[u] mod 3 = 0) then s:=x[u] else s:=0 else begin poz:=(p+u) div 2; s:= s(p,poz)+ s(poz+1,u); end end; begin …….. writeln(s(1,n)); …….. end.	#include<iostream.h> int x[20],n; …….. int s(int p, int u) { int poz; if(p==u) if(x[u]%3==0) return x[u]; else return 0; else { poz=(p+u)/2; return s(p, poz)+s (poz+1,u); } } void main() {……. cout<<s(1,n); …….. }

3. Se citesc de la tastatură un șir ***y*** cu ***p*** elemente întregi și un număr natural nenul ***r***.

a) Să se determine câte elemente din șir sunt numere perfecte. Un *număr perfect* este egal cu suma divizorilor săi (fără el însuși).

b) Să se determine suma elementelor din șir care sunt divizibile cu ***r*** ($r \in N$).

c) Să se determine câte numere sunt prime.

d) Să se determine produsul $p= c_1*c_2*\ldots*c_p$ unde $(c_i)i=1,p$ este ultima cifră a numărului y_i din șirul dat.

Justificați necesitatea metodei *Divide et impera* în rezolvarea acestei probleme.

APLICAȚIA 2

Căutarea unei valori într-un șir ordonat – Căutare binară

Se consideră un șir ***x*** cu ***n*** numere întregi. Să se verifice dacă o valoare ***c*** (număr întreg) se găsește în șir.

Datele de intrare se citesc de la tastatură sau din fișierul ***caut.in***, astfel: de pe prima linie se citesc dimensiunea ***n*** a tabloului numeric (***n*** număr natural nenul) și valoarea ***c*** căutată, iar de pe următoarea linie se citesc cele ***n*** numere întregi separate prin câte un spațiu.

Pe ecran se va afișa un mesaj corespunzător rezultatului căutării.

Exemplul 1:

caut.in	Pe ecran se va afișa
7 3 1 3 5 23 30 42 67	Valoarea 3 există în șir

Exemplul 2:

caut.in	Pe ecran se va afișa
7 10 1 3 5 23 30 42 67	Valoarea 10 nu există în șir

Rezolvare:

Căutarea unei valori într-un șir ordonat se realizează după modelul căutării în dicționar a unui cuvânt. Pagina în care trebuie să identificăm cuvântul se determină printr-un proces de reducere a domeniului de căutare, în funcție de litera cu care începe cuvântul și apoi de ordinea lexicografică a literelor cuvântului și de ordonarea alfabetică a cuvintelor din dicționar.

Reducerea domeniului de căutare prin înjumătățirea acestuia conduce la rezolvarea problemei printr-un algoritm de complexitate logaritmică numit *căutare rapidă* sau *căutare binară*.

Pentru a verifica dacă valoarea ***c*** există în șirul ***x*** de numere ordonat (presupunem că este ordonat crescător) *descompunerea problemei* revine la a împărți șirul în două subșiruri. Șirul ***x*** are ***n*** elemente și el este împărțit în subșirul (x_1, x_2, …, x_{poz}) și subșirul (x_{poz+1}, x_{poz+2}, …, x_n), unde ***poz***=[(1+n)/2] este indicele elementului din mijloc.

Dacă valoarea ***c*** este mai mică decât elementul din mijlocul șirului (c< x[poz]) atunci căutarea continuă în subșirul (x_1, x_2, …, x_{poz-1}); se continuă procesul de divizare în acest subșir. Dacă valoarea ***c*** este mai mare decât elementul din mijlocul șirului (c> x[poz]) atunci căutarea continuă în subșirul (x_{poz+1}, x_{poz+2}, …, x_n); se continuă procesul de divizare în acest subșir.

Dacă valoarea ***c*** este egală cu elementul din mijlocul șirului (c = x[poz]), atunci căutarea se oprește; elementul căutat a fost găsit.

Căutarea se încheie fără succces (elementul căutat nu este în șir) atunci când subșirul în care se face căutarea curentă este vid (p>u). Inițial, p=1 și u=n.

În figura 50 sunt reprezentate subșirurile obținute prin divizare în procesul de căutare binară pentru exemplul 1.

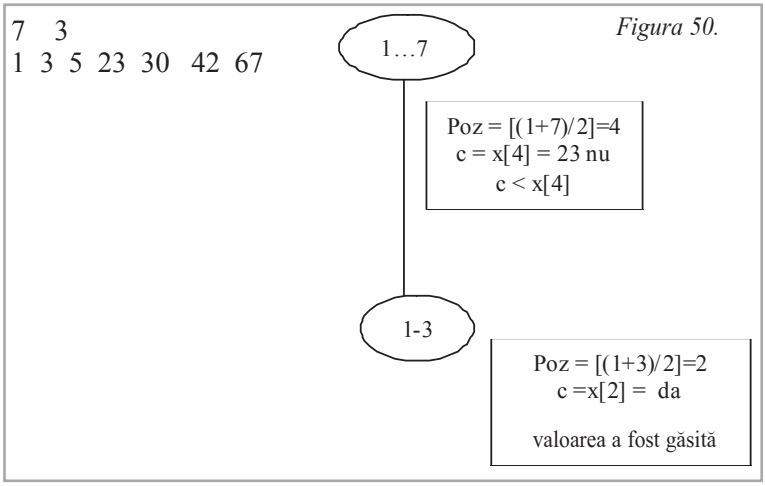


Figura 50.

În continuare este prezentat doar subprogramul prin care se face căutarea binară și apelul acestuia (șirul se presupune ordonat crescător):

LIMBAJUL PASCAL	LIMBAJUL C/C++
var x:array[1..20]of integer; n,c:integer; f:text; …….. function cautare_binara(p:integer;u:integer):integer; var poz:integer; begin poz:=(p+u) div 2; if(c=x[poz]) then cautare_binara:=1 else if(p<u) then if(c<x[poz]) then cautare_binara:=cautare_binara(p,poz-1) else cautare_binara:=cautare_binara(poz+1,u); cautare_binara:=0; end; begin …….. if(cautare_binara(1,n)<>0) then writeln(' Valoarea ',c,' exista in sir') else writeln(' Valoarea ',c,' exista in sir'); …….. end.	#include<fstream.h> ifstream f("caut_bin.in"); int x[20],n,c; …….. int cautare_binara(int p,int u) {int poz=(p+u)/2; if(c==x[poz]) return 1; else if(p<u) if(k<x[poz]) return cautare_binara(p,poz-1); else return cautare_binara(poz+1,u); return 0; } void main() {…… if(cautare_binara(1,n)) cout<<" Valoarea "<<c<<" exista in sir"; else cout<<" Valoarea "<<c<<" nu exista in sir"; …… }

TEME

1. Rescrieți subprogramul ***cautare_binara*** astfel încât acesta să returneze **poziția** în șir a valorii căutate (când valoarea căutată este în șir) sau un mesaj corespunzător în caz contrar.

2. Justificați eficiența căutării unei valori într-un șir ordonat, prin metoda căutării binare.

PROBLEME PROPUSE

1. ***Punct fix***.

Se consideră un șir ***a*** ordonat crescător cu ***n*** numere întregi distincte. Să se determine un indice ***m***, (***1≤ m≤ n),*** astfel încât **a[m] = m**. Datele de intrare se citesc de la tastatură sau din fișierul ***fix.in*** astfel: de pe prima linie un număr natural nenul ***n***, iar de pe următoarea linie ***n*** valori naturale, separate prin câte un spațiu.

Pe ecran se va afișa un mesaj corespunzător rezultatului căutării indicelui ***m***.

Exemplu: pentru valorile: -2, 0, 3, 6 se va afișa *m* =3;
pentru valorile: -2, 0, 1, 2 se va afișa mesajul „nu există soluție”.

2. ***Ghicirea unui număr ascuns***.

Ștefan îi propune colegului său Andrei să ghicească un număr natural ***g*** cuprins între 1 și ***n*** (1≤ ***n***≤ 1000). Andrei poate să adreseze lui Ștefan următoarele întrebări:

– numărul…… este egal cu numărul ales?
– numărul …… este mai mare decât numărul ales?
– numărul …… este mai mic decât numărul ales?

Andrei poate răspunde doar prin *da* sau *nu*.

Scrieți un program pentru simularea acestui joc.

Răspunsurile lui Ștefan sunt date de calculator. Numărul ***n*** se citește de la tastatură, numărul ***g*** (1≤ g≤ n) este generat aleator de calculator. După fiecare încercare de ghicire, să se afișeze pe ecran unul dintre mesaje: ‘*numărul este prea mare*’, sau ‘*numărul este prea mic*’, sau ‘*ai ghicit*’. Să se afișeze și numărul de încercări după care s-a ghicit numărul. (Pentru generarea aleatoare se vor folosi subprogramele specifice limbajului de programare.)

APLICAȚIA 3

Citirea elementelor unui șir

Să se scrie un program pentru citirea elementelor unui șir ***y*** cu ***m*** elemente reale, aplicând metoda *Divide et Impera*. Dimensiunea șirului și elementele acestuia se citesc de la tastatură.

Rezolvare:

Rezolvarea acestei probleme se reduce la citirea element cu element a valorilor unui șir; în termenii specifici metodei *Divide et Impera*, citirea unui element este chiar rezolvarea problemei elementare la care se ajunge prin divizare.

Descompunerea problemei revine la a împărți șirul în două subșiruri. Șirul ***y*** are ***m*** elemente și el este împărțit în subșirul (y_1, y_2, …, y_{poz}) și subșirul (y_{poz+1}, y_{poz+2}, …, y_m), unde **poz** = [(1+m)/2] este indicele elementului din mijlocul șirului.

Procesul de divizare se aplică fiecărui subșir, până când se obțin subșiruri de dimensiune 1 (cu un singur element): pentru acest subșir valoarea elementului este citită de la tastatură.

După citirea fiecărui element al subșirurilor de lungime 1, *combinarea rezultatelor* va conduce la completarea tuturor elementelor șirului.

Exemplu: dimensiunea șirului **m** = 5;
șirul de valori citite de la tastatură 16.24 21.34 7.5 -3.2 14.30.

În figura 51 sunt reprezentate toate subșirurile obținute prin divizare:

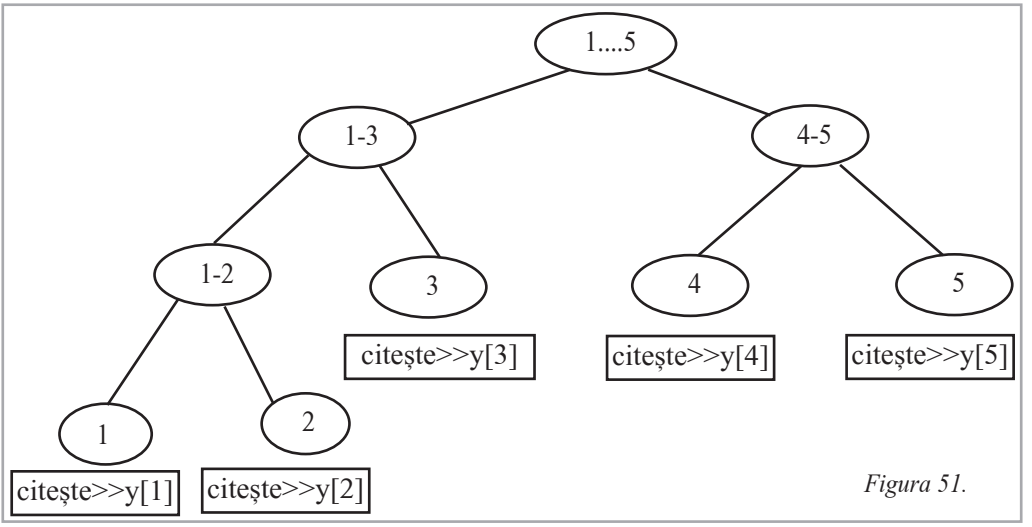


Figura 51.

LIMBAJUL PASCAL	LIMBAJUL C/C++
var y:array[1..20] of float; m:integer; procedure citire(p:integer;u:integer); var p:integer; begin if(p<u) then begin poz=(p+u)/2; citire(p,poz); citire(poz+1,u); end else read(y[p]); end ; begin read(m); citire(1,m); end.	#include<iostream.h> float y[20]; int m; void citire(int p, int u) { int poz; if(p<u) { poz=(p+u)/2; citire(p,poz); citire(poz+1,u); } else cin>>y[p]; } void main() { cin>>m; citire(1,m);}

TEME

1. Realizați un subprogram pentru afișarea elementelor unui șir folosind metoda *Divide et Impera*.

2. Justificați eficiența metodei *Divide et Impera* din punct de vedere al folosirii resurselor calculatorului (timp de lucru, memorie), pentru rezolvarea problemelor de citire/afișare a unor șiruri de valori.

APLICAȚIA 4

Suma elementelor unui șir

Se consideră un șir ***x*** cu ***n*** numere reale. Să se calculeze produsul celor ***n*** elemente ale șirului folosind metoda *Divide et Impera*.

Datele de intrare se citesc de la tastatură sau din fișierul ***suma.in***, astfel: de pe prima linie se citește dimensiunea ***n*** a tabloului numeric (***n*** ∈ N, 1≤ n ≤100), iar de pe următoarea linie se citesc ***n*** numere reale separate prin câte un spațiu.

În fișierul ***suma.out*** se va afișa, pe prima linie, șirul dat, iar pe următoarea linie un număr real, cu trei zecimale, reprezentând suma elementelor.

suma.in	***suma.out***
7 12 0.33 15.4 2.3 0.2 5.442	12 0.33 15.4 2.3 0.2 5.442 Suma = 35.672

Rezolvare:

În figura 52 sunt reprezentate toate subșirurile obținute prin divizarea șirului inițial până la obținerea subșirului cu un singur element.

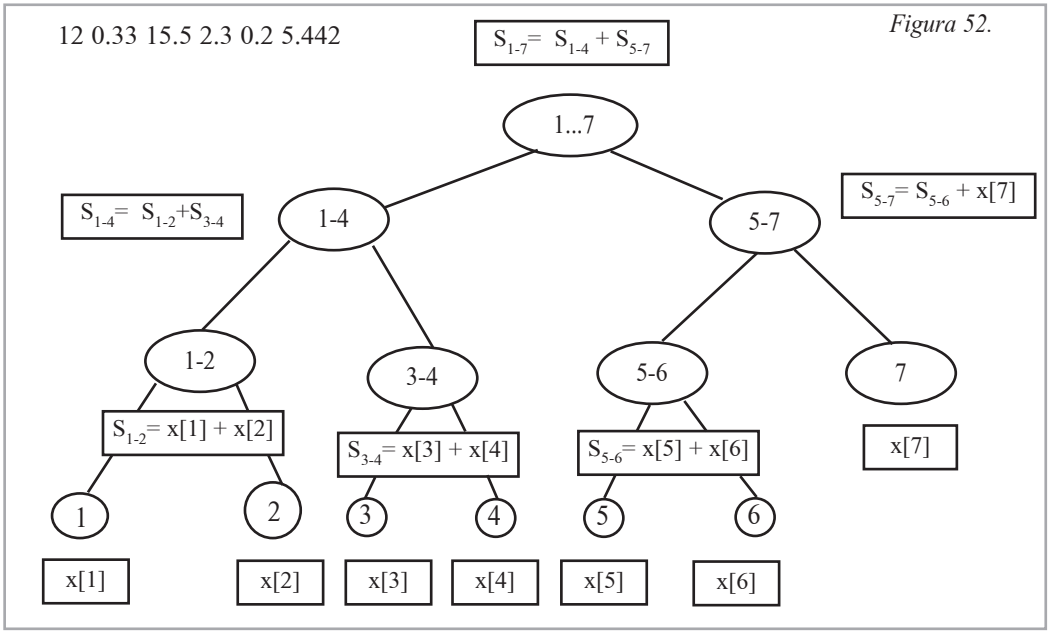


Figura 52.

Subalgoritmul pentru determinarea sumei elementelor şirului :

```
subalgoritm suma(p,u)
  dacă p=u atunci
        suma← x[p]
  altfel
        poz← [(p+u)/2]
        s1← suma(p,poz)
        s2← suma(poz+1,u)
        suma ← s1+s2
  sfârşit dacă
sfârşit subalgoritm.
```

TEME

1. Descrieți semnificația etapelor metodei *Divide et Impera* pentru rezolvarea acestei probleme.

2. Precizați numărul și tipul subprogramelor necesare pentru implementarea algoritmului de rezolvare a problemei.

3. Implementați algoritmul descris în limbajul de programare studiat.

EXERCIȚII ȘI PROBLEME PROPUSE

1. Stabiliți ce realizează următorul program. Formulați un enunț de problemă care se poate rezolva cu ajutorul acestui program.

LIMBAJUL PASCAL	LIMBAJUL C/C++
var a:array[1..20,1..20] of integer; m,n:integer; procedure sub1(pl,ul,pc,uc:integer); var poz:integer; begin if(pl=ul) then if(pc=uc) then read(a[pl,pc]) else begin poz:=(pc+uc) div 2; sub1(pl,ul,pc,poz); sub1(pl,ul,poz+1,uc); end else begin poz:=(pl+ul) div 2; sub1(pl,poz,pc,uc);	#include<iostream.h> int a[20][20],m,n; void sub1(int pl,int ul,int pc,int uc) { int poz; if(pl==ul) if(pc==uc) cin>>a[pl][pc]; else { poz=(pc+uc)/2; sub1(pl,ul,pc,poz); sub1(pl,ul,poz+1,uc); } else { poz=(pl+ul)/2; sub1(pl,poz,pc,uc); sub1(poz+1,ul,pc,uc); }

```
      sub1(poz+1,ul,pc,uc);
   end;
end;
begin
  read(m,n);     citire(1,m,1,n);
end.
```

```
}

void main()
{ cin>>m>>n;
  sub1(1,m,1,n);
 }
```

2. Se citește de la tastatură un număr ***n*** natural nenul. Să se scrie un program pentru calculul și afișarea pe ecran a următoarei sume:

$S = 1 \cdot 2 + 2 \cdot 3 + \ldots n \cdot (n+1)$.

Exemplu:

Pentru n=8, se va afișa S = 240.

3. Se citește de la tastatură un număr ***n*** natural nenul. Să se scrie un program pentru calculul și afișarea pe ecran a următoarei sume (cu o precizie de patru zecimale):

$$S = \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \ldots + \frac{1}{n \cdot (n+1)}$$

Exemplu:

Pentru n=8, se va afișa S = 0.8888.

4. Se citește de la tastatură un cuvânt format din cel mult 250 de caractere. Să se verifice dacă acest cuvânt are proprietatea de a fi *palindrom*.

Un cuvânt este *palindrom* dacă prin inversarea lui se obține un cuvânt identic.

Exemplu: Cuvântul inițial: *rotor*.
Cuvântul este palindrom.

5. Se consideră un șir ***z*** cu ***t*** numere întregi. Să se verifice dacă șirul este *simetric*. Datele de intrare se citesc din fișierul ***palin.in*** astfel: de pe prima linie un număr natural nenul care reprezintă dimensiunea ***t*** a șirului, iar de pe următoarea linie se citesc ***t*** valori întregi separate prin câte un spațiu. Mesajul prin care se arată dacă șirul are sau nu are proprietatea cerută se va afișa pe ecran.

Un șir are proprietatea de a fi *simetric* dacă:

$x[i] = x[n-i+1]$, pentru $\forall\, i \in \{1, 2, \ldots, [(n+1)/2]\}$

Exemplu:

palin.in

```
7
5 7 2 9 2 7 5
```

Se va afișa:

Șirul este simetric.

6. Fie matricea ***a*** cu ***n*** ($n \in N^*$) linii și ***n*** coloane cu elemente reale și un număr natural ***k*** dat ($1 \leq k \leq n$).

a) Precizați ce realizează următorul subprogram:

LIMBAJUL PASCAL	LIMBAJUL C/C++
…….. procedure s (p, u, k:integer); var poz:integer; aux:float; begin if(p<=u) then begin poz=(p+u)/2; aux=a[k][poz];a[k][poz]:=a[poz][k]; a[poz][k]:=aux; s (p,poz-1,k); s (poz+1,u,k); end end; begin …….. s (1,n,k); …… end.	…….. void s (int p, int u, int k) { int poz; float aux; if(p<=u) { poz=(p+u)/2; aux=a[k][poz];a[k][poz]=a[poz][k]; a[poz][k]=aux; s (p,poz-1,k); s (poz+1,u,k); } } void main() { …….. s (1,n,k); .....}

b) Încercuiți varianta corespunzătoare prelucrărilor realizate în matricea ***a*** după apelul subprogramului ***s***.

n = 4 k = 2 matricea inițială 3 4 5 6 8 2 1 0 9 7 2 4 9 5 2 1	Matricea după apelul s(1,n,k)			
	a) 3 4 9 6 8 2 7 0 5 1 2 2 9 5 4 1	b) 3 8 9 9 4 2 1 0 5 7 2 4 6 5 2 1	c) 3 8 5 6 4 2 7 5 9 1 2 4 9 0 2 1	d) 3 4 5 9 8 2 1 5 9 7 2 2 6 0 4 1

c) Formulați un enunț de problemă care se rezolvă cu ajutorul subprogramului ***s***.

APLICAȚIA 5

Determinarea celui mai mare divizor comun

Fie șirul ***y*** cu ***m*** numere naturale. Să se determine cel mai mare divizor comun al elementelor acestui șir (cmmmdc(y_1, y_2, …, y_m)) utilizând metoda *Divide et Impera.* Dimensiunea șirului și elementele acestuia se citesc de la tastatură. Cel mai mare divizor comun se va afișa pe ecran.

Exemplu:

m=5 12 102 54 36 93	Se va afișa pe monitor Cmmdc = 3

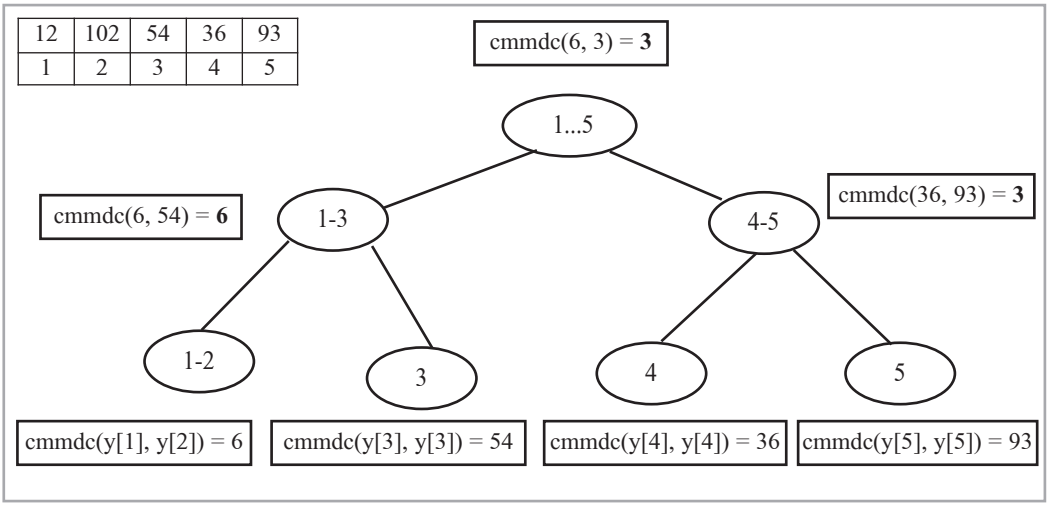
Rezolvare:

Șirul inițial se împarte în două subșiruri, primul subșir este (y_1, y_2, …, y_{poz}) și subșirul (y_{poz+1}, y_{poz+2}, …, y_m), unde ***poz***=[(1+p)/2] este indicele elementului din mijloc. La rândul lor aceste subșiruri sunt divizate până când se obțin subșiruri de lungime 1 (elementul acesta este chiar *cmmmdc* din subșirul generat) sau de lungime 2 (se determină *cmmdc* dintre cele două elemente).

După determinarea *cmmdc* al subșirurilor formate dintr-un singur element sau din două elemente, *combinarea rezultatelor* va conduce la determinarea ***cmmdc*** al tuturor elementelor șirului dat.

În figura 53 se prezintă determinarea *cmmdc* pentru exemplul dat.

Figura 53.



LIMBAJUL PASCAL	LIMBAJUL C/C++
var y:array[1..20]of integer; n:integer; procedure citire(p,u:integer); var poz:integer; begin if(p<u) then begin poz:=(p+u)div 2; citire(p,poz); citire(poz+1,u); end else read(y[p]); end;	#include<iostream.h> int y[20],m; void citire(int p, int u) { int poz; if(p<u) { poz=(p+u)/2; citire(p,poz); citire(poz+1,u); } else cin>>y[p]; }

```
function cmmdc(a,b:integer):integer;
 var r:integer;
  begin
   if(a mod b <>0) then     cmmdc:=cmmdc(b,a mod b)
                       else      cmmdc:=b;
end;

function divide_et_impera(p,u:integer):integer;
var poz:integer;
begin
 if((u-p)<=1) then
                    divide_et_impera:=cmmdc(y[p],y[u])
 else
  begin
   poz:=(p+u) div 2;
divide_et_impera:=cmmdc(divide_et_impera(p,poz),
                                divide_et_impera(poz+1,u));
  end
end;
begin
  read(n);   citire(1,n);
  writeln(' cmmdc = ',divide_et_impera(1,n));
 end.
```

```
int cmmdc (int a, int b)
{ int r;
  if(a%b)
    return cmmdc(b,a%b);
  else
   return b;
}

int divide_et_impera(int p,int u)
{int poz;
 if((u-p)<=1)
   return cmmdc(y[p], y[u]);
 else
  { poz=(p+u)/2;
    return cmmdc(divide_et_impera(p,poz),
                          divide_et_impera(poz+1,u));
   }
}

void main()
{ cin>>m;
  citire(1,m);
  cout<<" cmmdc = "<<divide_et_impera(1, m);

}
```

TEMĂ

Analizați programul prezentat pentru rezolvarea acestei probleme și răspundeți următoarelor cerințe:

a) Justificați aplicarea metodei *Divide et Impera* în rezolvarea acestei probleme.

b) Identificați subprogramele recursive și descrieți rolul acestora în rezolvarea problemei.

c) Apreciați eficiența fiecărui subprogram din punct de vedere al timpului de execuție.

d) Transformați subprogramele recursive – apreciate ca ineficiente – în subprograme iterative.

APLICAȚIA 6

Sortarea prin interclasare a elementelor unui șir

Se consideră un șir ***x*** cu ***n*** elemente întregi. Să se ordoneze șirul crescător prin metoda interclasării. Dimensiunea șirului și elementele acestuia se citesc din fișierul **interclas.in.** Șirul inițial și cel ordonat crescător se vor afișa pe linii distincte în fișierul **interclas.out.**

Exemplu:

interclas.in	*interclas.out*
7 17 8 12 -7 5 42 2	Șirul inițial:17 8 12 -7 5 42 2 Șirul ordonat: -7 2 5 8 12 17 42

Observație:

Prin ***interclasarea*** a două sau mai multe șiruri, ordonate după același criteriu, se obține un șir ordonat, format din reuniunea elementelor șirurilor inițiale (a se revedea din manualul de clasa a X-a interclasarea a două șiruri).

Rezolvare:

Se împarte șirul inițial în două subșiruri, primul subșir este (x_1, x_2, …, x_{poz}) și subșirul (x_{poz+1}, x_{poz+2}, …, x_n), unde ***poz***=[(1+n)/2] este indicele elementului din mijloc. La rândul lor aceste subșiruri sunt divizate până când se obțin subșiruri de lungime 1 (ordonate) sau de lungime 2, ordonate. Subșirurile ordonate se interclasează succesiv până când se obține șirul final, cu elementele în ordinea cerută.

În figura 54, sunt reprezentate toate subșirurile obținute prin divizare în procesul de interclasare, pentru exemplul numeric.

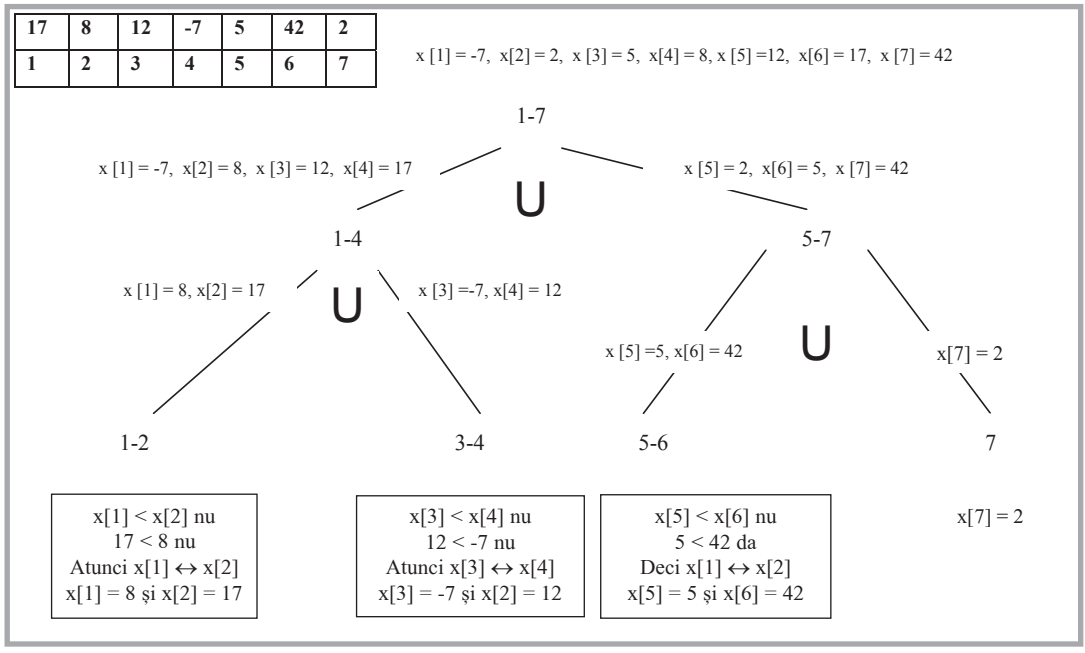


Figura 54.

TEME

1. Realizați interclasarea pentru șirurile de valori ***s1***: 1, 7, 10, 15, 35 și ***s2***: 5, 8, 9, 13, 23, 44.
2. Justificați cerința ca șirurile să fie ordonate după același criteriu.
3. Justificați rezolvarea acestei probleme prin metoda *Divide et Impera*.
4. Realizați programul pentru rezolvarea acestei probleme.

LIMBAJUL PASCAL	LIMBAJUL C/C++
procedure divide_et_impera(p,u:integer); var poz:integer; begin if((u-p)<=1)then ordon(p,u) else begin poz:=(p+u)div 2; divide_et_impera(p,poz); divide_et_impera(poz+1,u); interclasare(p,u,poz); end end; begin end.	void divide_et_impera(int p,int u) {int poz; if((u-p)<=1) ordon(p,u); else { poz=(p+u)/2; divide_et_impera(p,poz); divide_et_impera(poz+1,u); interclasare(p,u,poz); }} void main() { divide_et_impera(1,4); }

APLICAȚIA 7

Sortarea rapidă (quick-sort). *(Facultativ)*

Se consideră un șir ***x*** cu ***n*** elemente întregi. Să se ordoneze șirul crescător prin metoda de sortare rapidă. Dimensiunea șirului și elementele acestuia se citesc din fișierul **quick.in.** Șirul inițial și cel ordonat crescător se vor afișa pe linii distincte în fișierul **quick.out.**

Exemplu:

quick.in	***quick.out***
7 7 8 2 -7 42 5 12	Șirul inițial:7 8 2 -7 42 5 12 Șirul ordonat: -7 2 5 7 8 12 42

Rezolvare:

Algoritmul de sortarea rapidă a fost elaborat în anul 1962 de C.A.R. Hoare și se bazează pe mecanismul metodei *Divide et Impera*.

Pentru început, vom aplica direct algoritmul pentru șirul de valori din exemplul numeric.

Primul element se compară cu ultimul element din șir.

7 8 2 -7 42 5 **12.**

Întrucât x[1] este mai mic decât x[7] (se respectă relația de ordine cerută), atunci se compară x[1] cu x[6]:

7 8 2 -7 42 **5** 12.

x[1] este mai mare decât x[6] atunci cele două elemente se interschimbă, și șirul devine:

5 8 2 -7 42 **7** 12.

După ce a avut loc o interschimbare următoarea comparare se realizează între x[6] și x[2] (următorul element situat după elementul x[1] care a participat la interschimbare):

5 **8** 2 -7 42 **7** 12.

Întrucât x[2]>x[6] cele două valori se interschimbă:

5 **7** 2 -7 42 **8** 12.

După ce a avut loc o interschimbare, următoarea comparare se realizează între x[2] și x[5] (următorul element situat după elementul x[1] care a participat la interschimbare):

5 **7** 2 -7 **42** 8 12.

Întrucât x[2] este mai mic decât x[5] (se respectă relația de ordine cerută) atunci se compară x[2] cu x[4]:

5 **7** 2 **-7** 42 8 12.

Dar x[2]>x[4] cele două valori se interschimbă:

5 **-7** 2 **7** 42 8 12.

Ultima comparare are loc între x[3]și x[4]:

5 -7 **2** **7** 42 8 12.

Se observă că x[3] < x[4] (se respectă relația de ordine cerută):

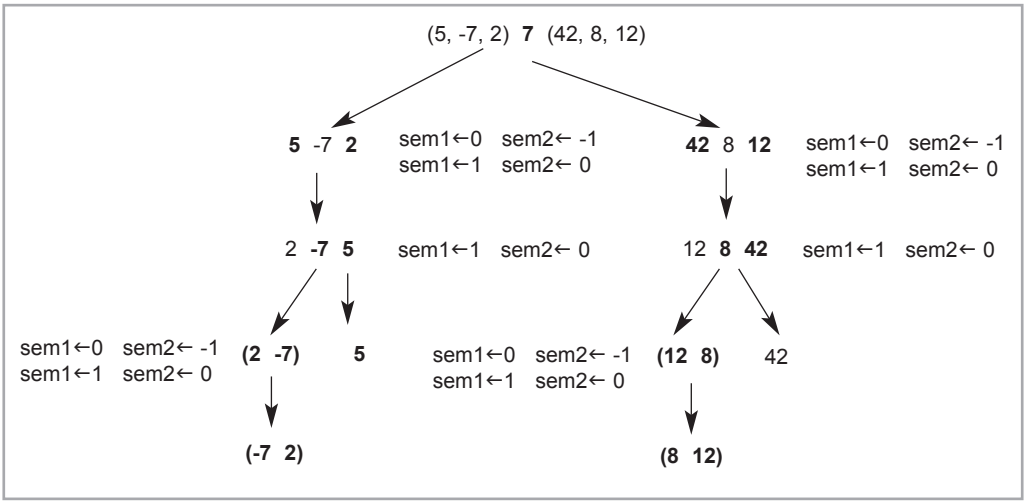
5 -7 2 **7** 42 8 12.

După această parcurgere numărul 7 se află pe poziția sa finală în șirul ordonat (**poz** = 4). Toate elementele din subșirul din stânga lui 7 sunt mai mici decât 7 și toate cele din subșirul din dreapta lui 7 sunt mai mari decât el.

Se obțin două subșiruri care vor fi ordonate separat, pe baza aceluiași raționament:

(5 -7 2) **7** (42 8 12).

Figura 55.



Metoda de sortare rapidă folosește împărțirea șirului în două subșiruri, astfel încât toate elementele din primul subșir să fie mai mici decât toate elementele din al doilea subșir.

Împărțirea șirului în subșiruri se oprește când se obțin subșiruri de lungime 1 (care sunt ordonate). Se observă că subșirurile se ordonează direct și combinarea rezultatelor parțiale nu se realizează.

Algoritmul ***quick_sort***:

```
algoritm quick_sort(p, u)
  dacă p<u atunci
                    divide(p,u, poz)
                    quick_sort(p, poz-1)
                    quick_sort(poz+1, u)
  sfârșit dacă
sfârșit algoritm
```

Cea mai importantă componentă a algoritmului este apelul subprogramului *divide* prin care se determină pivotul (***poz***) cu proprietatea: toate elementele din subșirul (x_p, x_2,…, x_{poz-1}) sunt mai mici decât x[poz] și toate elementele din subșirul (x_{poz+1}, x_{poz+2},…, x_u) sunt mai mari decât x[poz]. Indicele ***poz*** reprezintă poziția finală a elementului x[poz] în șirul ordonat.

Din exemplul numeric prezentat, se observă că la fiecare interschimbare se modifică direcția de deplasare în șir:

```
subalgoritm divide(p,u,poz)
  i←p
  j←u
  sem1←0
  sem2←-1
  cât timp i<j execută
     dacă xi>xj atunci
                    aux← xi
                    xi ← xj
                    xj ← aux
                    aux← sem1
                    sem1 ← -sem2
                    sem2 ← -aux
      sfârșit dacă
        i←i + sem1
        j←j + sem2
  sfârșit cât timp
      poz←i
sfârșit subalgoritm
```

Pentru controlul direcției se pot folosi două variabile cu rol de semafor:

(1) *sem1* controlează deplasarea în șir, de la stânga la dreapta;

(2) *sem2* controlează deplasarea în șir, de la dreapta la stânga.

O variabilă semafor are doar două stări: *activ* și *blocat.* Pentru a diferenția cele două variabile semafor, acestea sunt definite astfel:

$$\text{sem1} = \begin{cases} 0 \text{ blocat} \\ 1 \text{ activ} \end{cases} \qquad \text{sem2} = \begin{cases} 0 \text{ blocat} \\ -1 \text{ activ} \end{cases}$$

La un moment dat este activ un singur semafor.

Inițial, am ales să fie activ semaforul *sem2*.

Pentru adresarea elementelor care se compară, se folosesc doi indici *i* și *j*. Inițial, i = p și j = u.

Cât timp indicele *i* este mai mic decât indicele *j*, se compară elementele x[i] și x[j]; dacă relația de ordine corespunzătoare criteriului de sortare nu este îndeplinită (pentru sortarea crescătoare x[i] < x[j]), se interschimbă cele două valori și se modifică starea semafoarelor. La prima interschimbare, semaforul *sem2* devine *blocat* iar semaforul *sem1* devine *activ*. Prin acest procedeu, indicele *i* și indicele *j* își modifică alternativ valorile (*i* crește când *sem1* este activ; *j* scade când *sem2* este activ).

Când indicii *i* și *j* devin egali, s-a determinat elementul aflat pe poziția finală în șirul ordonat: x[poz], unde *poz* = i.

Execuția subprogramului *divide* pentru primul apel, *divide* (1, 7, **poz**), din exemplul numeric, poate fi urmărită cu ajutorul tabelului de variație.

Tabel de variație

i	*j*	*sem1*	*sem2*	*interschimb*	*poz*
1	7	0	-1	–	
	6			da	
2		1	0	da	
	5	0	-1	–	
	4			da	
3		1	0		
4				–	4

Turnurile din Hanoi. *(Facultativ)*

Varianta 1:

Se consideră trei tije verticale 1, 2, și 3. Pe tija 1 se găsesc ***n*** discuri de diametre diferite, perforate central, aranjate în ordine descrescătoare a diametrelor discurilor, de la bază către vârf. Să se mute toate discurile de pe tija 1 pe tija 2, folosind ca tijă auxiliară tija 3, respectând următoarele reguli:

– la fiecare pas se va muta un singur disc (cel care se află deasupra celorlalte discuri pe o tijă);

– un disc poate fi așezat pe o tijă doar peste un alt disc având diametru mai mare decât al său;

Numărul ***n*** (0<n≤10) de discuri se citește de la tastatură. În fișierul de ieșire ***hanoi.out*** se vor scrie, pe linii distincte, mutările efectuate prin perechi de două numere astfel: prima cifră reprezintă tija de pe care se ia discul, iar a doua cifră reprezintă tija pe care se așază discul respectiv.

Exemplu:

n = 3	***hanoi.out***
	1→ 2 1 → 3 2 → 3 1 → 2 3 → 1 3 → 2 1 → 2

Rezolvare:

O mutare se va nota astfel i→j, unde i, j ∈ {1, 2, 3} și i ≠ j, cu semnificația '*discul de pe tija i se mută pe tija j'*.

Dacă n=1 se execută o unică mutare discul de pe tija 1 se mută direct pe tija 2 (problema elementară).

Dacă n>1, se împarte problema în subprobleme astfel:

- se mută primele n-1 discuri de pe tija 1 pe tija 3;
- se mută ultimul disc (cel cu diametrul cel mai mare) de pe tija 1 pe tija 2;
- se mută cele n-1 discuri de pe tija 3 pe tija 2.

În acest moment s-au generat două subprobleme mai simple și anume mutarea a n-1 discuri de pe tija 1 pe tija 3, și mutarea celor n-1 discuri de pe tija 3 pe tija 2. La rândul lor aceste subprobleme se pot împărți în subprobleme până când rămâne de mutat un singur disc. Rezolvarea acestei probleme se poate implementa cu ajutorul metodei *Divide et Impera.*

Pentru a muta discul de diametru maxim de pe tija **i** , trebuie să fie mutate cele n-1 discuri situate peste acest disc pe tija auxiliară. Cele trei tije sunt numerotate distinct cu valori de la 1 la 3, suma acestor numere este 6; tija auxiliară are valoarea 6-i-j. Când discul de diametru maxim poate fi mutat pe tija **j,** se poate trece la mutarea celor n-1 discuri de pe tija 6-i-j pe tija **j**, folosind tija **i** (care nu mai are niciun disc pe ea) drept tijă auxiliară. Șirul de mutări necesare pentru a deplasa ***n*** discuri de pe tija ***i*** pe tija ***j*** se notează prin *hanoi(n, i, j)* și poate fi definit astfel:

$$hanoi(n,i,j)=\begin{cases} i \to j, \text{ dacă } n=1 \\ hanoi(n-1,i,6-i-j) \quad i \to j \;\; hanoi(n-1,\; 6-i-j, j), \quad \text{dacă } n>1 \end{cases}$$

În continuare este prezentat algoritmul de generare a mutărilor posibile:

algoritm *hanoi (n, i, j)*
  dacă n=1 **atunci**
    mută disc de pe tija i pe tija j
  altfel
    hanoi(n-1, i, 6-i-j)
    mută disc de pe tija i pe tija j
    hanoi(n-1, 6-i-j, j)
  sfârşit dacă
stop algoritm

Urmează subprogramul corespunzător algoritmului şi apelul acestuia din programul principal, respectiv din funcția principală.

LIMBAJUL PASCAL	LIMBAJUL C/C++
…… procedure hanoi(n, i, j:integer); var poz:integer; begin if(n=1) then writeln(g, i ,' → ', j) else begin hanoi(n-1, 6- i - j , j); writeln(g, i ,' → ', j) ; hanoi(n-1, i, 6 – i – j); end; end; begin …… hanoi(n, 1, 2); …… end.	…… void hanoi(int n, int i, int j) {int poz; if(n==1) g<<i<<" → "<<j<<endl; else { hanoi(n-1, 6- i-j, j); g<<i<<" → "<<j<<endl; hanoi(n-1, i, 6- i-j); } } void main() { …… hanoi(n, 1, 2); …… }

Varianta 2:

Se consideră trei tije verticale ***A***, ***B***, și ***C***. Pe tija ***A*** se găsesc ***n*** discuri de diametre diferite, perforate central, aranjate în ordine descrescătoare a diametrelor discurilor, de la bază către vârf. Să se mute toate discurile de pe tija ***A*** pe tija ***B***, folosind ca tijă auxiliară tija ***C***, respectând următoarele reguli:

– la fiecare pas se va muta un singur disc (cel care se află deasupra celorlalte discuri pe o tijă);

– un disc poate fi așezat pe o tijă doar peste un alt disc având diametru mai mare decât al său.

Numărul ***n*** ($0<n\leq10$) de discuri se citește de la tastatură. În fișierul de ieșire ***hanoi.out*** se vor scrie, pe linii distincte, mutările efectuate prin perechi de două litere astfel: prima literă reprezintă tija de pe care se ia discul, iar a doua literă reprezintă tija pe care se așază discul respectiv.

Exemplu:

n = 3	***hanoi.out***
	A → B A → C B → C A → B C → A C → B A → B

O mutare se va nota astfel A→B, cu semnificația '*discul de pe tija A se mută pe tija B'*.

Dacă n=1, se execută o unică mutare: discul de pe tija A se mută direct pe tija B (problema elementară).

Dacă n>1, se împarte problema în subprobleme astfel:

- se mută primele n-1 discuri de pe tija **A** pe tija C;
- se mută ultimul disc (cel cu diametrul cel mai mare) de pe tija **A** pe tija **B;**
- se mută cele n-1 discuri de pe tija C pe tija **B**.

În acest moment s-au generat două subprobleme mai simple, și anume mutarea a **n-1** discuri de pe tija **A** pe tija **C**, respectiv cele n-1 discuri de pe tija **C** pe tija **B**. La rândul lor aceste subprobleme se pot împărți în subprobleme până când rămâne de mutat un singur disc. Deci rezolvarea problemei se poate implementa cu ajutorul metodei *Divide et Impera.*

În continuare este prezentat algoritmul de generare a mutărilor posibile:

```
algoritm hanoi (n, A, B, C )
  dacă n=1 atunci
                mută disc de pe tija A pe tija B
              altfel
                hanoi(n-1, A,C, B)
                mută disc de pe tija A pe tija B
                hanoi(n-1, C, B, A)
  sfârşit dacă
stop algoritm
```

TEME

1. Rezolvați numeric problema turnurilor din Hanoi pentru ***n*** = 3 discuri și ***n*** = 4 discuri. Rețineți numărul de mutări necesare pentru rezolvarea problemei.

2. Care este numărul de mutări pentru un număr oarecare ***n*** de discuri?

3. Formulați problema elementară la care se ajunge prin aplicarea metodei *Divide et Impera*.

4. Justificați aplicarea metodei în rezolvarea acestei probleme.

APLICAȚII DE LABORATOR

1. Să se scrie programul complet pentru rezolvarea problemei turnurilor din Hanoi – varianta 1. Numărul natural nenul ***n*** ($1 \le n \le 10$) de discuri se citește de la tastatură. Mutările se vor afișa în fișierul ***hanoi1.out***, pe linii distincte.
Să se afișeze pe ultima linie câte mutări s-au executat

2. Să se scrie programul complet pentru rezolvarea problemei turnurilor din Hanoi – varianta 2. Numărul natural nenul ***n*** ($1 \le n \le 10$) de discuri se citește de la tastatură. Mutările se vor afișa în fișierul ***hanoi2.out***, pe linii distincte.
Să se afișeze pe ultima linie câte mutări s-au executat.

II. *BACKTRACKING*

1. STUDIU DE CAZ – PLANIFICAREA EXAMENELOR

Pentru îmbunătățirea pregătirii în domeniul 'Programării paginilor Web', un elev se înscrie la cursurile on-line organizate de o firmă atestată. Acest curs este format din mai multe module. Pentru a obține Certificatul de absolvire al unui modul, elevul trebuie să susțină on-line **n** ***teste finale în termen de*** **m** ***zile consecutive (n≤[m/2]). Examenele sunt numerotate de la*** **1** ***la*** **n** ***(în această ordine trebuie să fie susținute). Elevul poate da testele în ce zi dorește, cu condiția de a nu susține două teste în zile consecutive. Elevul încearcă să realizeze mai multe planificări și apoi să o aleagă pe cea mai avantajoasă. Pentru primul modul, în 4 zile, elevul trebuie să susțină două teste.***

Rezolvare:

Să vedem cum își planifică elevul susținerea testelor pentru cazul în care $n = 2$ și $m = 4$.

În prima zi, elevul planifică exemenul 1. În a doua zi, nu mai poate să planifice niciun examen. Ultimul examen este planificat în ziua a treia.

S-a obținut prima programare: în ziua 1 examenul 1,
în ziua 3 examenul 2.

Elevul observă că poate să-și planifice cel de-al doilea test și în ziua a patra. Se obține:

Programarea 2: în ziua 1 examenul 1,
în ziua 4 examenul 2.

Întrucât nu mai sunt zile în care să-și mute ultimul test, elevul se gândește să reprogrameze susținerea primului test în ziua 2. Testul 2 poate fi susținut doar în ziua a patra.

Programarea 3: în ziua 2 examenul 1,
în ziua 4 examenul 2.

Examenul 2 nu mai poate fi programat în altă zi. Elevul încearcă să programeze examenul 1 în ziua a treia; în acest caz, testul 2 ar putea fi programat doar în ziua a patra, însă condiția de a nu susține examenele în zile consecutive nu este respectată. Elevul încearcă să programeze testul 1 în ziua a patra, dar testul 2 nu poate fi susținut în aceeași zi și nici nu mai sunt zile disposibile.

În ziua 1 nu poate fi planificat testul 2 pentru că testele se susțin în ordinea prestabilită.

Elevul poate să susțină cele două teste conform uneia dintre cele trei programări posibile :

Posibilități de programare	Ziua 1	Ziua 2	Ziua 3	Ziua 4	Programare bună
1	Examen 1		Examen 2		Da
2	Examen 1			Examen 2	Da
3		Examen 1		Examen 2	Da
4			Examen 1	Examen 2	Nu

Concluzii:

- Pentru a realiza programarea testelor, elevul face verificări (examenele nu pot fi planificate în zile consecutive, ordinea susținerii testelor este crescătoare – nu se poate da testul 2 înaintea testului 1). Elevul face reveniri în construirea unei programări. Dacă la un moment dat testul 2 nu mai poate fi planificat în altă zi, atunci se încearcă reprogramarea primului test în altă zi și apoi se trece din nou la programarea testului 2. Sunt posibile reveniri succesive, până se găsește o zi în care poate fi programat testul 2. Aceste căutări cu avans și cu reveniri se termină când testul 1 nu mai poate fi planificat în nicio altă zi.
- Elevul a construit cele trei programări utilizând metoda **backtracking.**

Numele acestei metode este foarte sugestiv și poate fi tradus prin *căutare cu revenire,* generarea unei soluții făcându-se pe baza *urmelor precedente* (în limba engleză *back* înseamnă '*înapoi'*, iar *track* '*urmă'*).

2. DESCRIEREA GENERALĂ A METODEI BACKTRACKING

Metoda Backtracking se utilizează pentru rezolvarea unor probleme în care:

- se obțin mai multe soluții;
- soluția este formată din unul sau mai multe elemente, fiind reprezentată printr-un tablou $X=(x_1, x_2, \ldots, x_n)$ unde $x_1 \in M_1, x_2 \in M_2, \ldots, x_n \in M_n$;
- tabloul X este o structură liniară de tip *stivă*;
- mulțimile $M_1, M_2, \ldots, M_n$ sunt mulțimi finite având $c_1, c_2, \ldots, c_n$ elemente;
- elementele depuse în tablou îndeplinesc anumite condiții impuse de enunțul fiecărei probleme.

Generarea tuturor tablourilor X cu elementele produsului cartezian $M_1 \times M_2 \times \ldots \times M_n$ – numit spațiul *soluțiilor posibile* – conduce la un timp de execuție foarte mare.

Metoda Backtracking are la bază o strategie prin care se generează doar soluțiile care îndeplinesc condițiile specifice problemei, denumite *condiții interne*.

Soluțiile posibile care respectă condițiile interne se numesc *soluții rezultat*.

Dacă un element x_j primește o valoare din mulțimea M_j care este admisă în soluția rezultat, această valoare se numește *valoare validă*. Condițiile de validare sunt deduse din *condițiile interne*.

Exemplu:

Să se determine toate modalitățile de colorare a țărilor de pe o hartă, folosind un număr minim de culori precizat, spre exemplu, 4 culori. Condiția internă rezultă din așezarea țărilor pe hartă; pentru a fi vizibile, zonele geografice învecinate – țările – trebuie colorate distinct. Condițiile de validare se obțin prin compunerea următoarelor condiții simple: *dacă țara **i** este vecină cu țara **j** (în matricea de vecinătăți M[i][j] =1) atunci culoarea folosită pentru țara **i** trebuie să difere de culoarea folosită pentru țara **j** (în tabloul soluție, X[i] <> X[j])*.

Problemele care se rezolvă cu această metodă pot avea anumite particularități:

- numărul **n** de elemente care pot participa la construirea unei soluții nu este o valoare constantă (fixă);
- se poate obține și o singură soluție, denumită *soluție optimă*;
- elementele $x_1, x_2, \ldots, x_n$ ale unei soluții pot fi și ele tablouri;
- mulțimile $M_1, M_2, \ldots, M_n$ pot avea aceleași elemente.

Exemplu:

Să se genereze toate numerele naturale cu trei cifre distincte din mulțimea {1,2,4}.

Soluție:

Spațiul soluțiilor posibile S=({1,2,4}×{1,2,4}×{1,2,4}= {1,1,1},{1,1,2}, {1,1,4}, {1,2,1}, {1,2,2},…,{4,4,4})

Rezolvarea problemei prin generarea tuturor soluțiilor conduce la 27 de soluții posibile, dintre care doar 6 sunt *soluții rezultat*. În tabel s-au generat toate soluțiile posibile fiind marcate soluțiile rezultat.

Soluție posibilă	111	112	114	121	122	**124**	141	**142**	144	211	212	**214**	221	222
Soluție rezultat						√		√				√		
Soluție posibilă	224	**241**	242	244	411	**412**	414	**421**	422	424	441	442	444	
Soluție rezultat		√				√		√						

3. MECANISMUL METODEI BACKTRACKING

Soluțiile problemei sunt generate într-un tablou X de tip stivă:

- se alege din mulțimea M_1, elementul valid x_1;
- se presupune că în tabloul X s-au depus elementele $x_1 \in M_1$, $x_2 \in M_2$, …, $x_{k-1} \in M_{k-1}$ și se caută prima valoare *candidat* la construirea soluției $x_k \in M_k$;
- elementul x_k păstrează valoarea *candidat* doar dacă aceasta verifică condițiile de validare (condițiile interne);
- dacă x_k a fost depus în tabloul X, se verifică dacă s-a obținut o soluție rezultat, în caz afirmativ soluția este păstrată și se trece la construirea unei alte soluții, altfel se incrementează *k (stiva crește)* și se caută o valoare candidat pentru x_{k+1} din mulțimea M_{k+1};
- dacă în mulțimea M_k nu se mai găsesc valori valide (s-au testat toate valorile) se decrementează k (stiva scade) și se caută o altă valoare pentru x_{k-1} din mulțimea M_{k-1};
- algoritmul se oprește atunci când în mulțimea M_1 nu mai există nicio valoare candidat pentru x_1 (când stiva este vidă).

4. REPREZENTAREA ALGORITMULUI ÎN PSEUDOCOD

Algoritmul metodei Backtracking poate fi reprezentat atât iterativ, cât şi recursiv.

A) VARIANTA ITERATIVĂ

ALGORITMUL BACKTRACKING ITERATIV

```
Algoritm generare
   k← 1
   x[k] ← valoare inițială    // se inițializează primul nivel al stivei
  cât timp ( k>0 ) execută
    sem←0
    cât timp (în mulțimea M_k mai sunt valori candidat pentru x_k) și (sem=0) execută
       x[k] ← x[k]+1
      dacă (x[k] are o valoare validă) atunci          // valoarea respectă condițiile interne
                                        sem← 1
       sfârşit dacă
    sfârşit cât timp
    dacă (sem =1) atunci
                  dacă (k= numărul de elemente cerut) atunci
                                  s-a construit o soluție și se tipărește // stiva este plină
                  altfel
                                  k←k+1                    // stiva crește
                                  x[k] ← valoare inițială
                    sfârşit dacă
                 altfel
                    k ←k-1  // stiva scade
    sfârşit dacă
  sfâșit cât timp
stop algoritm
```

Descrierea algoritmului iterativ

Pe prima poziție din tabloul soluție X (k=1) se pune o valoare inițială. În cazul general, această valoare inițială este valoarea minimă admisibilă pe care o poate lua primul element X[1] din mulțimea de valori M_1 din care se scade o unitate.

Inițializarea unui element x_k se face astfel pentru ca la prima incrementare a acestuia să i se atribuie prima valoare din mulțimea M_k de valori.

Variabila k păstrează poziția curentă din tabloul X în care se va depune un nou element.

Variabila *sem* are rolul de a semnala situația în care elementul x_k respectă condițiile interne. Se caută în mulțimea M_k de valori (prin incrementarea valorii curente a elementului x_k : X[k]=X[k]+1) prima valoare care respectă condițiile interne. Această valoare, x_k, se compară cu elementele depuse în talou pentru a se verifica condițiile de validare (*valid(x_k)*). Dacă valoarea este validă, atunci *sem*← 1.

Dacă valoarea găsită pentru x_k poate participa la generarea soluției (*sem= 1*), atunci se verifică dacă s-au completat cu valori toate elementele din soluție; dacă s-a generat o soluție, aceasta este tipărită; astfel, se trece la următorul element candidat la generarea soluției (k← k+1); X[k+1] este inițializat cu o valoare corespunzătoare și sunt reluate verificările precedente.

Dacă în mulțimea M_k nu mai există valori pe care să le poată prelua x_k, *sem* rămâne 0 și se caută o nouă valoare pentru elementul x_{k-1} (k←k-1).

Generarea soluțiilor are loc cât timp pentru primul element al tabloului soluție mai există valori în mulțimea M_1 de valori. Dacă nu mai există astfel de valori, poziția curentă k, în tablou, devine zero și algoritmul de generare se oprește.

B) VARIANTA RECURSIVĂ

ALGORITMUL BACKTRACKING RECURSIV

```
Algoritm generare(k)
  pentru i ← valoare inițială, valoare maximă admisibilă din mulțimea Mk de valori  execută
     X[k] ← i
     dacă (x[k] are o valoare validă) atunci          // valoarea respectă condițiile interne
                dacă (k = numărul de elemente cerut) atunci
                          s-a construit o soluție și se tipărește // stiva este plină
                altfel
                      generare(k+1)
                sfârșit dacă
      sfârșit dacă
  sfârșit pentru
stop algoritm
```

Descrierea algoritmului recursiv

Se depune pe prima poziție din tabloul soluție (k=1, *generare(1)*) o valoare inițială (X[1]← i).

Variabila k păstrează poziția curentă din tabloul X în care se va depune un nou element.

Se caută în mulțimea M_k de valori prima valoare (prin incrementarea valorii curente a elementului x_k, X[k]=X[k]+1) care respectă condițiile interne. Această valoare x_k se compară cu elementele depuse în tablou pentru a se verifica condițiile de validare (*valid(x_k)*).

Dacă valoarea găsită pentru x_k poate participa la generarea soluției, atunci se verifică dacă s-au completat cu valori toate elementele din soluție; dacă s-a generat o soluție, aceasta este tipărită; astfel, se trece la următorul element candidat la generarea soluției prin autoapelul algoritmului (*generare(k+1)*). Prin autoapel se trece la un nou element din tabloul soluție X (se urcă în stivă).

Dacă în mulțimea M_k nu mai există valori pe care să le poată prelua x_k se caută o nouă valoare pentru elementul x_{k-1} (k←k-1). Această trecere la elementul x_{k-1} se face pe baza mecanismului recursivității la revenirea dintr-o instanță a subprogramului recursiv (prin extragerea din vârful stivei a fiecărei valori depuse, se coboară în stivă).

Generarea soluțiilor are loc cât timp, pentru primul element al tabloului soluție, mai există valori în mulțimea M_1. Dacă nu mai există astfel de valori, poziția curentă k, în tablou, devine zero și algoritmul de generare se oprește.

TEME

1. Explicați generarea soluțiilor într-un tablou unidimensional de tip stivă.

2. Evidențiați aspectele recursive ale Metodei Backtracking.

3. Stabiliți care dintre următoarele afirmații sunt adevărate sau false, prin marcarea căsuței corespunzătoare:

Nr. crt.	Afirmații	Răspuns	
		A	F
1	Metoda Backtracking se utilizează pentru rezolvarea unor probleme care admit doar o unică soluție.		
2	Soluția generată de metodă este păstrată într-o structură liniară de tip coadă.		
3	Mulțimea valorilor M_k pentru un element x_k al soluției are un număr finit de elemente.		
4	Un element x_k face parte dintr-o soluție rezultat dacă și numai dacă este diferit de elementele x_{k-1} și x_{k-2}.		
5	*Spațiul soluțiilor posibile* este format din reuniunea tuturor mulțimilor de valori posibile pentru fiecare element al tabloului soluție X.		
6	Metoda Backtracking generează doar soluțiile care îndeplinesc condițiile specifice problemei.		

4. Asociați fiecărei definiții din coloana 1 denumirea corectă din coloana 2:

Nr. crt.	Definiție
1	Condițiile specifice pe care trebuie să le respecte o soluție rezultat se numesc:
2	Soluțiile posibile care respectă condițiile interne se numesc:
3	O valoare x_k păstrată în tabloul soluție se numește:
4	Un nou element este adăugat în tabloul soluție numai după verificarea unor condiții denumite:

Denumire
Valoare validă
Condiții de continuare
Condiții interne
Soluții rezultat

5. Utilizând mecanismul Metodei Backtracking, stabiliți care dintre următoarele afirmații sunt adevărate sau false, prin marcarea căsuței corespunzătoare:

Nr. crt.	Afirmații	Răspuns	
		A	F
1	Inițializarea unui element x_k se face cu valoarea 0.		
2	Variabila *sem* are rolul de a semnala cazul în care elementul x_k respectă condițiile interne corespunzătoare.		
3	Dacă sunt îndeplinite condițiile interne corespunzătoare pentru elementul $x_{k,}$ atunci se trece la următorul element din tabloul soluție.		
4	Generarea soluțiilor are loc cât timp pentru primul element al tabloului soluție nu mai există valori în mulțimea M_1 de valori.		

6. Formulați condițiile de validare pentru următoarele probleme:

a) Să se genereze toate șirurile de lungime ***n***, formate numai din literele C, P și T , șiruri care să nu aibă două litere T alăturate. Numărul natural ***n*** (0<n<12) se citește de la tastatură. Exemplu: pentru n=3 se vor afișa (nu neapărat în această ordine) șirurile: CCP,CCT, CPT, CTC etc.

b) Să se genereze toate numerele prime de ***n*** cifre (n ∈ N, n<10) formate doar din cifrele 3, 0 și 7. Numărul natural ***n*** (0<n<10) se citește de la tastatură. Exemplu: ***n*** = 3, se vor afișa (nu neapărat în această ordine) numerele: 337, 307, 373, 733, 773.

c) Să se genereze toate posibilitățile de așezare a ***n*** dame pe o tablă șah de dimensiune ***n*** × ***n*** astfel încât ele 'să nu se atace' (două dame se atacă dacă sunt pe aceeași linie, coloană sau diagonală).

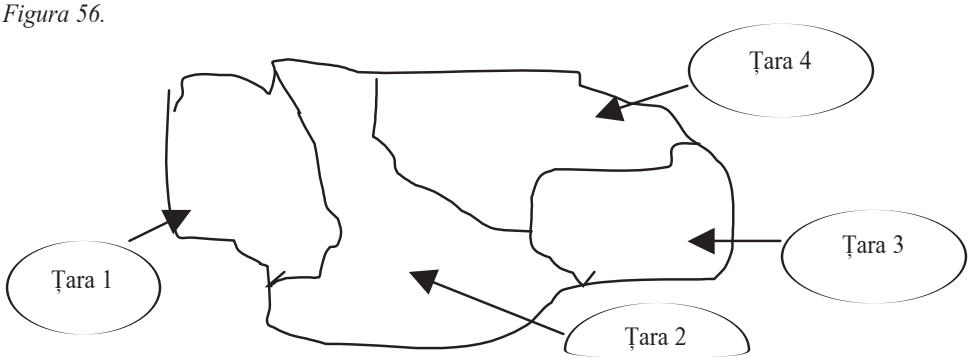
Exemplu: pentru n=4 o așezare posibilă este:

	*		
			*
*			
		*	

d) O hartă cuprinde ***n*** țări, iar pentru colorarea acestora se utilizează doar **patru** culori, astfel încât două țări cu frontieră comună vor fi colorate diferit. Culorile alese pentru realizarea acestei hărți sunt albastru, galben, portocaliu, și verde. Să se determine toate posibilitățile de colorare a hărții.

Exemplu: pentru n=4 țări având următoarea hartă:

Figura 56.



O soluție a acestei probleme este: țara1 – culoarea albastru, țara 2 - culoarea galben; țara 3 – culoarea albastru; țara 4 – culoarea portocaliu.

5. EXEMPLU DE IMPLEMENTARE A ALGORITMULUI

Să se genereze toate numerele naturale cu trei cifre distincte din mulțimea {1,2,4}.

Soluție:

Etapa I: *Identificarea elementelor specifice metodei*

Mulțimea cifrelor se notează cu C = { 1, 2, 4}, iar mulțimea valorilor este M={1,2,3}, care reprezintă indicii elementelor din tabloul C (C[1]=1, C[2]=2, C[3]=4).

Tabloul X are doar trei elemente (numărul are doar trei cifre), deci $h_{max} = 3, (X[i] \in M)_{i=1,3}$

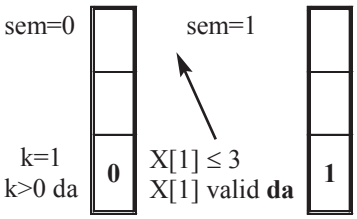
Condiții interne: cifrele numărului trebuie să fie distincte.

Condiții de validare: valoarea elementului x_k (o cifră) candidată la soluția rezultat trebuie să fie diferită de celelalte elemente $x_1, x_2, \ldots, x_{k-1}$ din tabloul X.

Etapa II: *Generarea soluțiilor*

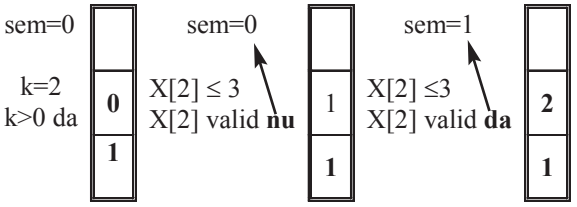
În casetele 1, 2, 3, 4 și 5 se reprezintă generarea primelor două soluții. În fiecare casetă sunt descrise operațiile de urcare și coborâre în stivă conform algoritmului prezentat.

Caseta 1 Nr. sol. **0**



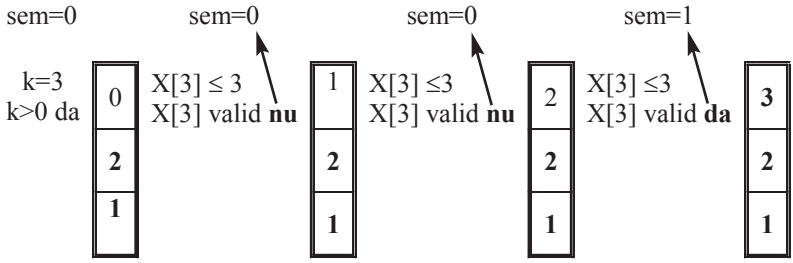
Se inițializează primul element din X cu 0 și apoi se verifică dacă este mai mic decât valoarea maximă admisibilă. X[1]=0<3.
Atunci X[1]=X[1] +1=1, care **este** o valoare validă și deci este păstrată în X, iar variabila sem=1.
Întrucât k≠3 atunci k=k+1=2 și elementul X[2] se inițializează cu 0.

Caseta 2 Nr. sol. **0**



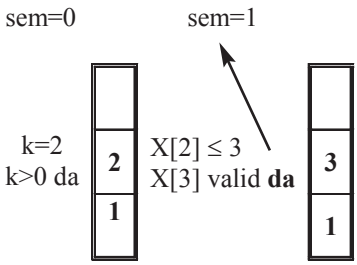
k > 0, sem=0 și X[2]=0 ≤ 3 (valoarea maximă admisibilă).
Atunci X[2]=X[2] +1=1, care **nu este** o valoare validă și deci se caută o altă valoare.
Cum X[2]=1<3, atunci X[2]=X[2] +1=2, care **este** o valoare validă, iar variabila sem=1.
Întrucât k≠3 atunci k=k+1 (*se urcă* în stivă), k=3 și elementul X[3] inițializat cu 0.

Caseta 3 Nr. sol. **1** Numărul: **124**



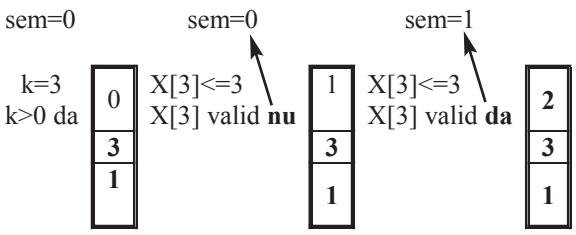
k>0, sem=0 şi X[3]=0< 3. Atunci X[3]=X[3] +1=1, care **nu este** o valoare validă şi *sem* rămânând 0 ⇒ X[3]=X[3] +1=2, care **nu este** o valoare validă (2 se regăseşte pe nivelul 2).
X[3]=2<3 şi sem=0 ⇒ X[3]=X[3] +1=3, care **este** o valoare validă şi *sem*=1.
Întrucât k=3 s-a găsit o soluţie care se păstrează.
Dar k>0, sem=0, dar X[3] nu mai poate lua altă valoare atunci k=k-1 (*se coboară* în stivă).

Caseta 4 Nr. sol. **1**



k>0, sem=0, X[2]=2<3
⇒ X[2]=X[2] +1=3, care **este** o valoare validă şi sem=1.
Întrucât k≠3 atunci k=k+1 şi elementul X[3] este iniţilizat cu 0.

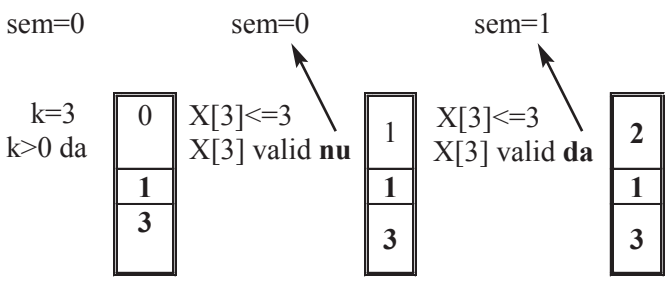
Caseta 5 Nr. sol. **2** Numărul **142**



k>0, sem=0 şi X[3]=0<3 ⇒ X[3]=X[3] +1=1, care **nu este** o valoare validă şi deci se caută o altă valoare.
sem = 0 şi X[3]=1<3 ⇒ X[3]=X[3] +1=2, care **este** o valoare validă.
Întrucât k=3 s-a găsit a doua soluție soluție care se păstrează.

În caseta 6 se reprezintă construirea soluției 5 a problemei.

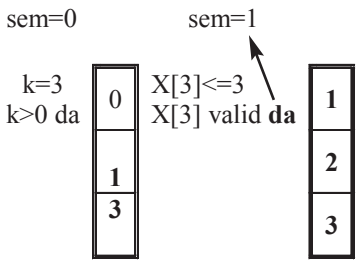
Caseta 6 Nr.sol. 5
Numărul 412



k>0, sem=0 şi X[3]=0<3, atunci X[3]=X[3] +1=1, care nu este o valoare validă.
Întrucât X[3]<3 → X[3]=X[3] +1=3 care este o valoare validă, se păstrează în tablou şi sem =1.
Întrucât k=3 şi sem=1 s-a găsit o nouă soluție care se păstrează.

În caseta 7 se reprezintă construirea ultimei soluții posibile a problemei.

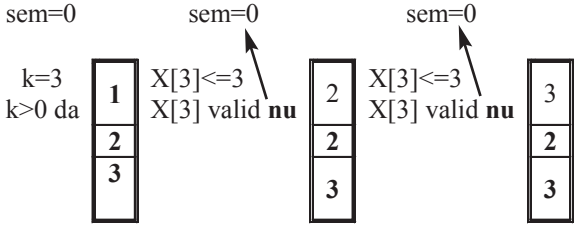
Caseta 7 Nr. sol. **6** Numărul **421**



k>0, sem=0 și X[3]=0<3 atunci X[3]=X[3]+1=1 care **este** o valoare validă și sem=1. Întrucât k=3 și s-a construit o nouă soluție care se păstrează.

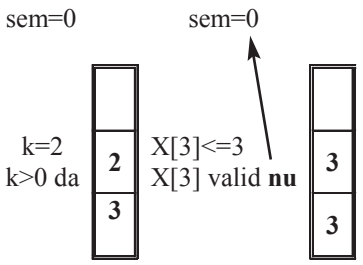
În casetele 8, 9 și 10 se descrie amănunțit, conform algoritmului Backtracking, coborârea în stivă până la golirea ei completă — algoritmul de generare s-a încheiat.

Caseta 8 Nr. sol. **6**



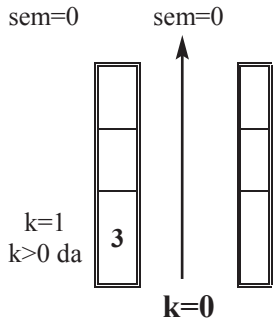
k>0, sem=0 și X[3]=1<3. Atunci X[3]=X[3]+1=2 care **nu este** o valoare validă. Întrucât X[3]=2<3 și *sem* = 0 atunci X[3]=X[3]+1=3 care **nu este** o valoare validă (este egală cu valoare depusă în X[1]). Acum X[3]=3 și nu mai poate lua altă valoare, iar sem=0. Deci k=k-1 (se coboară în stivă) și se caută o nouă valoare validă pentru X[2].

Caseta 9 Nr. sol. **6**



k>0, *sem* = 0 şi X[2]=2<3. Atunci X[2]=X[2]+1=3 care **nu este** o valoare validă. Întrucât X[2]=3 şi *sem* = 0 şi X[2] nu mai poate lua valoare atunci k=k-1 (se coboară în stivă) şi se caută o nouă valoare validă pentru X[1].

Caseta 10 Nr. sol. **6**



k>0, sem=0 şi X[1]=3.
X[1]=3 (valoarea maximă admisibilă)
Atunci k=k - 1= 0 şi algoritmul se termină.
S-au generat 6 soluţii.

Se observă cum algoritmul de generare se termină când tabloul soluție X este gol.

Etapa III: Implementarea algoritmului Backtracking.

Pentru problema dată, citirea datelor de intrare se face din fişierul ***exemplu.in*** (de pe prima linie se citeşte ***n*** iar de pe următoarea linie ***n*** cifre separate prin câte un spaţiu), iar afişarea soluţiilor pe linii distincte şi a numărului de soluţii se va face în fişierul ***exemplu.out***

VARIANTA ITERATIVĂ

LIMBAJUL PASCAL

```
var c,X :array[1..20]of integer;
    i,n,k:integer;
    f,g:text;
procedure citire;
 begin
  read(f,n);
  for i:=1 to n do
    read(f,c[i]);
  end;
function mult_val:boolean;
 begin
  mult_val:=false;
  if(x[k]<n) then
   begin
    inc(x[k]);    mult_val:=true;
  end;
end;
function valid:boolean;
 begin
  valid:=true;
  for i:=1 to k-1 do
   if(X[i]=X[k]) then valid:=false;
 end;
procedure afisare;
 begin
  for i:=1 to n do
   write(g,c[X[i]]);     writeln(g);
 end;
procedure generare;
 var sol:integer; sem, sev:boolean;
 begin
  k:=1; sol:=0;  X[k]:=0;
  while(k>0) do
   begin
     repeat
      sem:=mult_val;
      sev:=valid;
     until not(sem) or (sem and sev);
     if(sem=true)then
      if(k=n) then
              begin
                    inc(sol); afisare;
              end
             else
              begin
               inc(k); X[k]:=0;
```

LIMBAJUL C++

```
#include<fstream.h>
ifstream f("exemplu.in");
ofstream g("exemplu.out");
int c[20],X[20],i,n,k;

void citire()
{ f>>n;
  for(i=1;i<=n;i++)
    f>>c[i];   f.close();
}

int mult_val()
{if(X[k]<n)
  { X[k]++;   return 1;  }
  return 0;
 }

int valid()
{for(i=1;i<k;i++)
   if(X[i]==X[k]) return 0;
  return 1;
}
void afisare()
{ for(i=1;i<=n;i++)
     g<<c[X[i]];
  g<<endl;
}
void generare()
{  int sol=0,sem,sev;
   k=1; X[k]=0;
  while(k)
   { do{sem=mult_val();
          sev=valid();
          }while((sem & !sev);
     if(as)
      if(k==n)
             { sol++; afisare();             }
      else
       {k++; X[k]=0;  }
     else
     !k--;
   }
  g<<" s-au generat "<<sol<<" solutii ";
}
```

LIMBAJUL PASCAL	LIMBAJUL C++
else dec(k); end; writeln(g,' s-au generat ',sol,' solutii '); end; begin assign(f,'exemplu.in');reset(f); assign(g,'exemplu.out'); rewrite(g); citire; back; close(f); close(g); end.	void main() { citire(); generare(); g.close();}

Descrierea implementării iterative

Funcția *generare()/generare* reprezintă implementarea algoritmului Backtraking, prin care sunt construite soluțiile, din care sunt apelate funcțiile *mult_val()/mult_val*, *valid()/valid* și *afișare()/afisare.* Funcția *mult_val()/mult_val* are rolul de a verifica dacă mulțimea M_k de valori posibile pentru elementul x_k a fost parcursă în totalitate. Dacă mai sunt valori atunci elementul X[k] preia elementul curent din mulțimea M_k și funcția returnează valoarea 1, altfel funcția *mult_val()/mult_val* returnează valoarea 0 (nu mai sunt valori posibile pentru x_k în mutimea M_k). Funcția *valid()/valid* are rolul de a verifica dacă noua valoare depusă în tabloul X, pe poziția k, îndeplinește condițiile interne ale problemei. Funcția *afișare()/afișare* are rolul de a reprezenta fiecare soluție generată în fișier (sau pe ecran, după cum este formulată cerința fiecărei aplicații).

În funcția *generare()/generare* s-au definit variabilele semafor *sem* și *sev* care pot avea valori doar în mulțimea {0,1}. Variabilei *sem* i se atribuie valoarea returnată de funcția *mult_val()/malt_val*, iar variabilei *sev* i se atribuie valoarea returnată de funcția *valid()/valid*.

Implementarea recursivă

VARIANTA RECURSIVĂ	
LIMBAJUL PASCAL	**LIMBAJUL C++**
var c,X:array[1..20] of integer; n,k,sol:integer; f,g:text; procedure citire; var i:integer; begin read(f,n); for i:=1 to n do read(f,c[i]); end; function valid(k:integer):boolean; var i:integer; begin valid:=true; for i:=1 to k-1 do	#include<fstream.h> ifstream f("exemplu.in"); ofstream g("exemplu.out"); int c[20], X[20],n,k,sol; void citire() { f>>n; for(int i=1;i<=n;i++) f>>c[i]; f.close(); } int valid(int k) {for(int i=1;i<k;i++)

LIMBAJUL PASCAL	LIMBAJUL C++
if(X[i]=X[k])then valid:=false; end; procedure afisare(k:integer); var i:integer; begin for i:=1 to k do write(g,c[X[i]]); writeln(g); end; procedure generare(k:integer); var i:integer; begin for i:=1 to n do begin X[k]:=i; if(valid(k)=true) then if(k=n) then begin inc(sol); afisare(k); end else generare(k+1); end; end; begin assign(f,'exemplu.in');reset(f); assign(g,'exemplu.out'); assign(g,'back1.out');rewrite(g); citire; generare(1); writeln(g,' s-au generat ',sol,' solutii '); close(f);close(g); end.	if(X[i]==X[k]) return 0; return 1; } void afisare(int k) { for(int i=1;i<=k;i++) g<<c[X[i]]; g<<endl; } void generare(int k) { for(int i=1;i<=n;i++) {X[k]=i; if(valid(k)) if(k==n) { sol++; afișare(k); } else generare(k+1); } } void main() { citire(); generare(1); g<<" s-au generat "<<sol<<" solutii "; g.close(); }

Descrierea implementării recursive

Funcția *generare(int k)/generare(k:integer)* reprezintă implementarea algoritmului Backtraking recursiv, prin care sunt construite soluțiile, din care sunt apelate funcțiile *valid(int k)/valid(k:integer)* și *afișare(int k)/afișare(k:integer)*. Mulțimea de valori este generată în corpul funcției *generare(int k)/generare(k:integer)*. Funcțiile *valid(int k)/valid(k:integer)* și *afișare(int k)/afișare(k:integer)* au aceleași roluri ca și în implementarea iterativă.

TEME

1. Stabiliți valoarea de adevăr a următoarelor afirmații:

Nr. crt.	Afirmații	Răspuns	
		A	F
1	Indicele k din tabloul soluție crește cu o unitate după fiecare determinare din mulțimea de valori a unei valori valide pentru X[k].		
2	O valoare depusă pe nivelul k este validă, chiar dacă cele depuse pe nivelurile anterioare nu sunt valide		
3	După obținerea unei soluții, nivelul k din tabloul X continuă să crească.		
4	Afișarea unei soluții are loc când sunt îndeplinite condițiile interne.		
5	Elementul X[k] ia întotdeauna o singură valoare		
6	După construirea ultimei soluții, indicele curent al tabloului soluție, k, devine 0 (tabloul este gol).		

2. Asociați fiecărei acțiuni descrise în coloana 1 subprogramul corespunzător din coloana 2:

Nr. crt.	Acțiune
1	Vizualizarea unei soluții.
2	Verifică dacă noua valoare depusă pe nivelul curent k al tabloului soluție poate participa la construirea soluției.
3	Construirea soluțiilor pentru o problemă dată.
4	Alegerea din mulțimea de valori posibile pentru X[k].

Subprogram
afisare
initializare
valid
mult_val
generare

APLICAȚII REZOLVATE

1. GENERAREA PERMUTĂRILOR

Se citește de la tastatură un număr ***n*** natural nenul. Să se genereze toate permutările posibile ale elementelor mulțimii M={1,2,3,…,n}. Soluțiile generate se vor afișa pe linii distincte fișierul ***permut.out***, iar pe ultima linie a fișierului se va afișa numărul de soluții găsite.

Exemplu: n = 3

permut.out

```
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
6
```

Rezolvare:

Etapa I: Identificarea elementelor specifice metodei;

a) mulțimea valorilor este M={1, 2, 3, …,n}, și reprezintă elementele care se permută;

b) tabloul X are ***n*** elemente, deci $h_{max} = n$, $(X[i] \in M)_{i=1,n}$;

c) *condiții interne*: elementele mulțimii trebuie să fie distincte;

d) *condiții de validare*: valoarea elementului x_k candidată la soluția rezultat trebuie să fie diferită de celelalte elemente $x_1, x_2,\ldots,x_{k-1}$din tabloul X.

Etapa II: Generarea soluțiilor pentru exemplul numeric n=4 ; se completează următorul tabel:

?												
?												
2												
1												
Sol 1	Sol 2	Sol 3	Sol 4	Sol 5	Sol 6	Sol 7	Sol 8	Sol 9	Sol 10	Sol 11	Sol 12	Sol 13

Sol 14	Sol 15	Sol 16	Sol 17	Sol 18	Sol 19	Sol 20	Sol 21	Sol 22	Sol 23	Sol 24	Sol 25	Sol 26

Etapa III: Implementarea algoritmului Backtracking iterativ

VARIANTA ITERATIVĂ // PERMUTĂRI	
LIMBAJUL PASCAL	**LIMBAJUL C++**
var X :array[1..20]of integer; i,n,k:integer; g:text; function mult_val:boolean; begin mult_val:=false; if(X[k]<n) then begin inc(X[k]); mult_val:=true; end; end; function valid:boolean; begin valid:=true; for i:=1 to k-1 do if(x[i]=x[k]) then valid:=false;	#include<fstream.h> ofstream g(“perm.out”); int X[20],i,n,k; int mult_val() {if(X[k]<n) { X[k]++; return 1; } return 0; } int valid() {for(i=1;i<k;i++) if(X[i]==X[k]) return 0; return 1; } void afisare() { for(i=1;i<=n;i++) g<<X[i]<<” “;

LIMBAJUL PASCAL	LIMBAJUL C++
end; procedure afisare; begin for i:=1 to n do write(g,X[i],' '); writeln(g); end; procedure generare; var sol:integer; sem,sev:boolean; begin k:=1; sol:=0; x[k]:=0; while(k>0) do begin repeat sem:=mult_val; sev:=valid; until not(sem) or (sem and ev); if(sem=true)then if(k=n) then begin inc(sol); afisare; end else begin inc(k); X[k]:=0; end else dec(k); end; writeln(g,' s-au generat ',sol,' solutii '); end; begin assign(g,'permut.out');rewrite(g); read(n); generare; close(g); end.	g<<endl; } void generare() { int sol=0,sem, sev; k=1; X[k]=0; while(k) { do {sem=mult_val(); sev=valid(); }while((sem & !sev); if(sem) if(k==n) { sol++; afisare(); } else {k++; X[k]=0; } else !k--; } g<<" s-au generat "<<sol<<" solutii "; } void main() { cin>>n; generare(); g.close();}

2. GENERAREA ARANJAMENTELOR

Se citesc de la tastatură două numere ***n*** și ***p*** (***p*** < ***n***) naturale nenule. Să se genereze toate aranjamentele elementelor mulțimii M={1,2,3,…,n} luate câte ***p***.

Soluțiile generate se vor afișa pe linii distincte, fișierul **aranj.out**, iar pe ultima linie a fișierului se va afișa numărul de soluții găsite.

Exemplu: n = 3, p=2.

aranj.out
```
1 2
1 3
2 1
2 3
3 1
3 2
6
```

Rezolvare:

Etapa I: Identificarea elementelor specifice metodei

Mulțimea valorilor este M={1, 2, 3, …,n}, care reprezintă elementele care se permută.

Tabloul X are ***p*** elemente, pentru că se formează submulțimi doar cu p elemente, deci $h_{max} = p$, $(X[i] \in M)_{i=1,p}$

Condiții interne: elementele mulțimii trebuie să fie distincte.

Condiții de validare: valoarea elementului x_k candidată la soluția rezultat trebuie să fie diferită de celelalte elemente $x_1, x_2,\ldots,x_{k-1}$din tabloul X.

Etapa II: Generarea soluțiilor se va face ca exercițiu, pentru ***n*** = 4 și ***p*** = 3 prin completarea următorului tabel, pe baza algoritmului Backtracking:

2												
1												
Sol 1	Sol 2	Sol 3	Sol 4	Sol 5	Sol 6	Sol 7	Sol 8	Sol 9	Sol 10	Sol 11	Sol 12	Sol 13

Sol 14	Sol 15	Sol 16	Sol 17	Sol 18	Sol 19	Sol 20	Sol 21	Sol 22	Sol 23	Sol 24	Sol 25	Sol 26

Etapa III: Implementarea algoritmului Backtracking iterativ

LIMBAJUL PASCAL	LIMBAJUL C++
var X :array[1..20]of integer; i,n,k,p:integer; g:text; function mult_val:boolean; begin mult_val:=false; if(X[k]<n) then begin inc(X[k]); mult_val:=true; end; end;	#include<fstream.h> ofstream g("aranj.out"); int X[20],i,n,k; int mult_val() {if(X[k]<n) { X[k]++; return 1; } return 0; } int valid() {for(i=1;i<k;i++) if(X[i]==X[k]) return 0; return 1;

LIMBAJUL PASCAL	LIMBAJUL C++
function valid:boolean; begin valid:=true; for i:=1 to k-1 do if(x[i]=x[k]) then valid:=false; end; procedure afisare; begin for i:=1 to p do write(g,X[i],' '); writeln(g); end; procedure generare; var sol:integer; sem, sev:boolean; begin k:=1; sol:=0; x[k]:=0; while(k>0) do begin repeat sem:=mult_val; sev:=valid; until not(sem) or (sem and ev); if(sem=true)then if(k=p) then begin inc(sol); afisare; end else begin inc(k); X[k]:=0; end else dec(k); end; writeln(g,' s-au generat ',sol,' solutii '); end; begin assign(g,'aranj.out'); rewrite(g); read(n,p); generare; close(g); end.	} void afisare() { for(i=1;i<=p;i++) g<<X[i]<<" "; g<<endl; } void generare() { int sol=0,sem, sev; k=1; X[k]=0; while(k) { do {sem=mult_val(); sev=valid(); }while((sem & !sev); if(sem) if(k==p) { sol++; afisare(); } else {k++; X[k]=0; } else !k--; } g<<" s-au generat "<<sol<<" solutii "; } void main() { cin>>n>>p; generare(); g.close(); }

3. GENERAREA COMBINĂRILOR

Se citesc de la tastatură două numere ***n*** și ***p*** (**p** ≤ **n**) naturale nenule. Să se genereze toate combinările elementelor mulțimii M={1,2,3,…,n} luate câte **p**.

Soluțiile generate se vor afișa pe linii distincte, fișierul ***combin.out***, iar pe ultima linie a fișierului se va afișa numărul de soluții găsite.

Exemplu: n = 3, p=2.

combin.out
1 2
1 3
2 3
3

Rezolvare:

Etapa I: Stabilirea următoarelor elemente

a) Mulțimea valorilor este M={1, 2, 3, …,n}, care reprezintă elementele care se permută.

b) Tabloul X are ***p*** elemente, pentru că se formează submulțimi doar cu ***p*** elemente,

$h_{max} = p$, $(X[i] \in M)_{i=1,p}$

c) *Condiții interne*: elementele mulțimii trebuie să fie distincte.

d) *Condiții de validare*:

- valoarea elementului x_k candidată la soluția rezultat trebuie să fie diferită de celelate elemente $x_1, x_2,\ldots,x_{k-1}$ din tabloul X ;
- o submulțime nu conține aceleași elemente luate în altă ordine, în oricare altă submulțime ($x_1 < x_2 < \ldots x_{k-1} < x_k$).

Etapa II: Generarea soluțiilor se va face ca exercițiu, pentru ***n*** = 4 și ***p*** = 3 prin completarea următorul tabel, pe baza algoritmului Backtracking:

2												
1												
Sol 1	Sol 2	Sol 3	Sol 4	Sol 5	Sol 6	Sol 7	Sol 8	Sol 9	Sol 10	Sol 11	Sol 12	Sol 13

Sol 14	Sol 15	Sol 16	Sol 17	Sol 18	Sol 19	Sol 20	Sol 21	Sol 22	Sol 23	Sol 24	Sol 25	Sol 26

Etapa III: Implementarea algoritmului Backtracking iterativ

VARIANTA ITERATIVĂ // combinări	
LIMBAJUL PASCAL	**LIMBAJUL C++**
var X :array[1..20]of integer; i,n,k,p:integer; g:text; function mult_val:boolean; begin mult_val:=false; if(X[k]<n) then begin inc(X[k]); mult_val:=true; end; end; procedure afisare; begin for i:=1 to p do write(g,X[i],' '); writeln(g); end; procedure generare; var sol:integer; sem :boolean; begin k:=1; sol:=0; x[k]:=0; while(k>0) do begin sem:=mult_val; if(sem=true)then if(k=p) then begin inc(sol); afisare; end else begin inc(k); X[k]:=X[k-1]; end else dec(k); end; writeln(g,' s-au generat ',sol,' solutii '); end; begin assign(g,'combin.out');rewrite(g); read(n,p); generare; close(g); end.	#include<fstream.h> ofstream g("combin.out"); int X[20],i,n,k; int mult_val() {if(X[k]<n) { X[k]++; return 1; } return 0; } void afisare() { for(i=1;i<=p;i++) g<<X[i]<<" "; g<<endl; } void generare() { int sol=0,sem; k=1; X[k]=0; while(k) { sem=mult_val(); if(sem) if(k==p) { sol++; afisare(); } else {k++; X[k]=X[k-1]; } else !k--; } g<<" s-au generat "<<sol<<" solutii "; } void main() { cin>>n>>p; generare(); g.close(); }

TEME

1. Stabiliți valoarea de adevăr a următoarelor afirmații:

Nr. crt.	Afirmații	Răspuns A	Răspuns F
1	Nivelul curent X[k] al tabloului soluție este inițializat cu valoarea minimă din mulțimea M_k de valori posibile.		
2	O soluție pentru generarea permutărilor elementelor mulțimii {1, 2,…,n} are exact **n** elemente.		
3	Fie mulțimea { 1, 2, …, ***m***}. Prin generarea tuturor aranjamentelor, respectiv a tuturor combinărilor elementelor acestei mulțimi în submulțimi de câte ***t*** elemente (***t*** ≤ ***m***), se obțin ***n1*** soluții, respectiv ***n2*** soluții. Atunci **n1>n2**.		
4	Prin generarea combinărilor mulțimii {1,2,…,***m***} în submulțimi de câte ***t*** elemente (***t*** ≤ ***m***) se obțin soluții care au ***m*** elemente.		
5	Numărul de soluții obținut la generarea tuturor permutărilor unei mulțimi cu ***p*** elemente este ***p!***		
6	Numărul de soluții obținut la generarea tuturor aranjamentelor de câte **k** elemente ale unei mulțimi cu **p** (***k*** ≤ ***p***) elemente este: $C_p^k = \frac{p!}{k! \cdot (p-k)!}$		
7	Numărul de soluții obținut la generarea tuturor combinărilor de câte **k** elemente ale unei mulțimi cu **p** (***k*** ≤ ***p***) elemente este: $A_p^k = \frac{p!}{(p-k)!}$		
8	Submulțimile obținute prin generarea tuturor combinărilor de câte ***k*** elemente ale unei mulțimi cu ***p*** (***k*** ≤ ***p***) sunt distincte.		

2. Se consideră următorul program:

LIMBAJUL PASCAL	LIMBAJUL C++
var X :array[1..20]of integer; i,n,k,p:integer; g:text; procedure citire; begin read(n,p); end;	#include<fstream.h> ofstream g(“date.out”); int X[20],n,k,p; void citire() { cin>>n>>p; }

LIMBAJUL PASCAL	LIMBAJUL C++
function mult_val:boolean; begin mult_val:=false; if(X[k]<n) then begin dec(X[k]); mult_val:=true; end; end; procedure afisare; begin for i:=1 to p do write(g,X[i],' '); writeln(g); end; procedure generare; var sol:integer; sem:boolean; begin k:=1; sol:=0; X[k]:=n+1; while(k>0) do begin sem:=mult_val; if(sem=true)then if(k=p) then begin inc(sol); afisare; end else begin inc(k); X[k]:=X[k-1]; end else dec(k); end; writeln(g,' s-au generat ',sol,' solutii '); end; begin citire; generare; close(g); end.	int mult_val() {if(X[k]>1) { X[k]—; return 1; } else return 0; } void afisare() { int i; for(i=1;i<=p;i++) g<<X[i]<<' '; g<<endl; } void generare() { k=1; int sol=0,sem; X[k]=n+1; while(k) { sem=mult_val(); if(sem) if(k==p) { sol++; afisare(); } else {k++; X[k]=X[k-1]; } else k--; } g<<" s-au generat "<<sol<<" solutii "; } void main() { citire(); generare(); g.close(); }

a) Dacă valorile citite de la tastatură sunt ***n*** = 4 și ***p*** = 3, stabiliți ce se afișează după execuția programului.

b) Formulați un enunț pentru o problemă al cărei algoritm de rezolvare se implementează cu ajutorul programului dat.

c) Construiți varianta recursivă a programului.

3. Se citește un număr natural nenul ***t*** (0<***t***<5). Se generează numere naturale cu ***t*** cifre distincte.

Alegeți varianta corectă de răspuns pentru următoarele întrebări:

a) Care este valoarea minimă care poate fi depusă pe primul nivel al tabloului soluție? Justificați.

1) -1 sau 0; 2) 9; 3) 0; 4) 0 sau 1.

b) Care este valoarea maximă care poate fi depusă pe nivelul curent al tabloului soluție? Justificați.

1) 4; 2) 9; 3) 10; 4) 5.

c) Dacă ***t*** = 4, precizați care dintre următoarele numere poate fi soluție a problemei:

1) 5678; 2) 1019; 3) 1239; 4) 1234.

d) Dacă ***t*** = 4 și s-a generat soluția **2798** (număr natural cu 4 cifre distincte), care este următorul număr generat?

e) Se generează toate numerele naturale cu 4 cifre distincte **pare.** Precizați care dintre următoarele numere nu poate fi soluție a problemei:

1) 5678; 2) 2046; 3) 2034; 4) 4240.

f) La baza generării numerelor naturale cu ***t*** cifre distincte stă algoritmul de generare a:

1) permutărilor primelor ***t*** numere naturale;
2) aranjamentelor mulțimii {1, 2, …,10} în submulțimi de câte ***t*** elemente;
3) combinărilor mulțimii {0, 1, 2, …,9} în submulțimi de câte ***t*** elemente;
4) aranjamentelor mulțimii {0, 1, 2, …,9} în submulțimi de câte ***t*** elemente.

APLICAȚII DE LABORATOR

1. Să se scrie programele pentru aplicațiile rezolvate și, după execuția lor, să se completeze numărul de soluții găsite :

a) permutări: pentru ***n*** = 4 numărul de soluții găsite =

b) aranjamente: pentru ***n*** = 4 și ***p*** = 3 numărul de soluții găsite =

c) combinări: pentru ***n*** = 4 și ***p*** = 3 numărul de soluții găsite =

Modificați programul de generare iterativă a permutărilor mulțimii primelor ***n*** numere naturale, astfel încât inițializarea nivelului curent al tabloului soluție să pornească de la valoarea maximă admisibilă din mulțimea de valori (folosiți ca model programul de la tema 2).

2. Scrieți varianta recursivă a programului de generare a tuturor aranjamentelor cu ***p*** elemente din elemente mulțimii M={1,2,3,…,n}. Se citesc de la tastatură două numere naturale nenule ***n*** și ***p*** (**p ≤ n**). Soluțiile generate se vor afișa pe linii distincte în fișierul ***aranj.out***, iar pe ultima linie a fișierului se va afișa numărul de soluții găsite.

3. Scrieți varianta recursivă a programului de generare a tuturor combinărilor cu ***p*** elemente din elemente mulțimii M={1,2,3,…,n}. Se citesc de la tastatură două numere naturale nenule ***n*** și ***p*** (**p ≤ n**). Soluțiile generate se vor afișa pe linii distincte în fișierul ***combin.out***, iar pe ultima linie a fișierului se va afișa numărul de soluții găsite.

4. Se citește de la tastatură un număr natural ***t*** ($0 < t < 6$).

a) Să se scrie un program (iterativ sau recursiv) pentru generarea tuturor numerelor naturale cu ***t*** cifre distincte. Soluțiile se vor afișa pe linii distincte în fișierul ***numere1.out***, iar pe ultima linie se va afișa numărul de soluții.

Exemplu: Pentru ***t*** =3 se obțin astfel de numere: 123, 124, 125,…

b) Modificați programul scris la punctul a) astfel încât să se genereze doar numerele naturale cu ***t*** cifre distincte pare. Soluțiile se vor afișa pe linii distincte în fișierul ***numere2.out***, iar pe ultima linie se va afișa numărul de soluții.

Exemplu: Pentru ***t*** =3 se obțin astfel de numere: 204, 206, 208,…

c) Modificați programul scris la punctul b) astfel încât să se genereze doar numerele naturale cu ***t*** cifre distincte pare, dispuse în ordine crescătoare. Soluțiile se vor afișa pe linii distincte în fișierul ***numere3.out***, iar pe ultima linie se va afișa numărul de soluții.

Exemplu: Pentru ***t*** = 4 se obțin astfel de numere: 246, 248, 468,…

d) Modificați programul scris la punctul a) astfel încât să se genereze doar numerele naturale cu ***t*** cifre distincte care au cifra cea mai semnificativă 8. Soluțiile se vor afișa pe linii distincte în fișierul ***numere4.out***, iar pe ultima linie se va afișa numărul de soluții.

Exemplu: Pentru ***t*** = 3 se obțin astfel de numere: 801, 802, 803,…

PROBLEME PROPUSE

1. Se citește de la tastatură un cuvânt cu cel mult 20 de caractere. Să se genereze toate cuvintele obținute prin permutarea literelor cuvântului dat. Cuvintele generate vor fi afișate în fișierul ***cuvinte.out***, pe linii distincte, iar pe ultima linie se va afișa numărul de cuvinte generate.

Exemplu: Se citește cuvântul *cor*.

În fișierul ***cuvinte.out*** vor fi afișate următoarele cuvinte:

cuvinte.out
cor
cro
ocr
orc
rco
roc
S-au generat 6 cuvinte

2. Se citește de la tastatură un număr natural nenul ***k*** ($2<k<9$). Să se genereze toate numerele naturale formate din ***k*** cifre distincte astfel încât diferența în valoare absolută între oricare două cifre consecutive să fie 2. Numerele generate vor fi afișate pe linii distincte în fișierul ***dif_doi.out***, iar pe ultima linie se va afișa numărul de soluții.

Exemplu: ***k*** = 3

dif_doi.out
135 246 357
420 468 531
579 642 753
864 975
S-au generat 11 numere

3. Fie șirul ***a*** cu ***n*** ($n>=3$) numere reale. Să se genereze cu elementele șirului ***a*** toate '*permutările vale'*.

Datele de intrare se citesc din fișierul ***vale.in*** astfel: de pe prima linie se citește dimensiunea șirului ***n***, iar de pe următoarea linie ***n*** numere reale separate prin câte un spațiu.

'Permutările vale' generate se vor afișa pe linii distincte în fișierul ***vale.out***, pe linii distincte, iar pe ultima linie a fișierului se va afișa numărul de soluții determinat.

O *permutare vale* are următoarea definiție:
a[i] > a[i+1] , pentru i=1, 2,…, k-1
a[i] < a[i+1] , pentru i= k, k+1, …,n

$$\text{unde } k = \begin{cases} \left[\frac{n}{2}\right], \text{ dacă } \boldsymbol{n} \text{ este par} \\ \left[\frac{n}{2}\right]+1, \text{ dacă } \boldsymbol{n} \text{ este impar} \end{cases}$$

Exemplu:

vale.in	***vale.out***
6 4 12 13 7 5 2	13 12 2 4 5 7 13 7 2 4 5 12 13 5 2 4 7 12 …….

4. Fie șirul ***a*** cu ***n*** (n>=3) numere reale. Să se genereze cu elementele șirului ***a*** toate '*permutările munte*' .

O *permutare munte* se definește astfel:
a[i] < a[i+1] , pentru i=1, 2,…, k-1
a[i] > a[i+1] , pentru i= k, k+1, …,n

$$\text{unde } k = \begin{cases} \left[\frac{n}{2}\right], \text{ dacă } \boldsymbol{n} \text{ este par} \\ \left[\frac{n}{2}\right]+1, \text{ dacă } \boldsymbol{n} \text{ este impar} \end{cases}$$

Datele de intrare se citesc din fișierul ***munte.in*** astfel: de pe prima linie se citește dimensiunea șirului ***n***, iar de pe următoarea linie ***n*** numere reale separate prin câte un spațiu.

Permutările munte generate se vor afișa în fișierul ***munte.out***, pe linii distincte, iar pe ultima linie a fișierului se va afișa numărul de soluții determinat.

Exemplu:

munte.in	***munte.out***
6 4 12 13 7 5 2	2 4 13 12 7 5 13 7 2 4 5 12 13 5 2 4 7 12 …….

5. Să se genereze toate tablourile pătratice de ordin ***n***, care au proprietatea că pe fiecare linie cât și pe fiecare coloană se află un caracter citit ***c***. Ordinul ***n*** (*n* număr natural nenul) al tabloului și caracterul ***c*** se citesc de la tastatură, iar toate tablourile generate se vor afișa în fișierul ***matrice.out***, separate printr-o linie goală. Pe ultima linie din fișier se va afișa numărul de tablouri generate.

Exemplu : n = 3 c='m' Matrice.out

S1: m 0 0 0 m 0 0 0 m	S3: 0 m 0 m 0 0 0 0 m	S5: 0 0 m m 0 0 0 m 0
S2: m 0 0 0 0 m 0 m 0	S4: 0 m 0 0 0 m m 0 0	S6: 0 0 m 0 m 0 m 0 0

S-au generat 6 soluții

Indicație: În tabloul soluție X se vor genera permutările mulțimii {1, 2, …, n}, nivelul X[k] va păstra indicele coloanei pe care se va afla caracterul ***c***, iar nivelul ***k*** reprezintă linia pe care se va pune caracterul (caracterul ***c*** se află pe poziția (i, j) = (k, X[k]) în tabloul generat).

6. Să se genereze toate numerele cu ***n*** cifre distincte ($n \in N^*$, $n<=10$) cu proprietatea că au suma cifrelor egală cu ***3n+1***. Numărul ***n*** se citește de la tastatură, iar numerele generate sunt afișate pe linii distincte în fișierul ***numere.out***. Pe ultima linie a fișierului se va afișa numărul de soluții.

Exemplu: n=2 , în fișierul ***numere.out*** se generează 7 numere, și anume:
16, 25, 34, 43, 52, 61, 70.

7. Să se genereze toate numerele cu ***n*** cifre distincte (***n***$\in N^*$, ***n***$<=10$) cu proprietatea că au suma cifrelor egală cu ***S***. Numerele ***n*** și ***S*** se citesc din fișierul ***date.in***, iar numerele generate sunt afișate pe linii distincte în fișierul ***date.out***. Pe ultima linie a fișierului se va afișa numărul de soluții.

Exemplu: n=4 , S=30 în fișierul ***date.out*** se generează 24 numere, și anume:
6789, 6798, 6879, …, 9876.

Care este valoarea maximă a sumei ***S*** pentru generarea numerelor cu 6 cifre distincte?

Care este formula pentru determinarea valorii maxime admisibile pentru ***S,*** atunci când se generează numere cu ***n*** cifre distincte (cazul general)?

8. Fie un șir ***z*** cu ***p*** ($p \in N^*$) numere întregi. Să se genereze toate posibilitățile de rearanjare a elementelor șirului dat astfel încât:

a) să nu existe două elemente consecutive negative;

b) să nu existe două elemente consecutive pozitive.

Datele de intrare se citesc din fișierul ***sir.in*** astfel: de pe prima linie se citește dimensiunea ***p*** a șirului, iar de pe următoarea linie se citesc ***p*** numere întregi separate prin câte un spațiu.

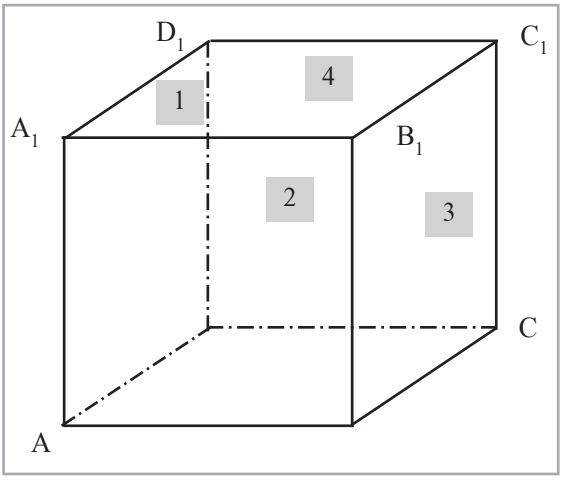
Șirurile generate la punctul ***a*** se vor afișa pe linii distincte în fișierul ***sir.out***, iar pe ultima linie se va afișa numărul de șiruri determinate. Șirurile generate la punctul ***b*** se vor afișa pe linii distincte la sfârșitul fișierului ***sir.out***, iar pe ultima linie se va afișa numărul de șiruri determinate.

9. S-a construit un nou centru de locuințe format din ***t*** (***t***$\in N^*$, ***t***>20) clădiri sub forma unor piramide patrulatere regulate. Arhitectul și designer-ul aleg ***s*** ($s \in N^*$, $s>= 4$) culori de vopsea specială pentru exteriorul clădirilor. Apoi își propun să vopsească fiecare față laterală a unui bloc utilizând culori diferite din cele ***s*** disponibile, dar fațada principală (notată cu **2 – fața ABB_1A_1**) trebuie să aibă aceeași culoare ***c*** (inclusă în cele ***s*** culori alese). Să se genereze toate posibilitățile de colorare a unei clădiri cu cele ***s*** culori.

Datele de intrare se citesc din fișierul ***culori.in*** astfel: de pe prima linie se citesc două numere naturale nenule ***t*** și ***s*** și un șir de caractere ***c*** ***s***eparate prin câte un spațiu, iar de pe următoarea linie se citesc ***s*** șiruri de caractere de lungime maximă 12 care reprezintă culorile alese de arhitect.

În fișierul ***culori.out*** se vor afișa toate soluțiile generate pe linii distincte, iar pe ultima linie numărul de soluții generate.

Figura 57



culori.in	***culori.out***			
5 portocaliu galben portocaliu albastru alb verde	Fața 1	Fața2	Fața 3	Fața 4
	Galben	Portocaliu	Albastru	Alb
	Galben	Portocaliu	Albastru	Verde
	Galben	Portocaliu	Alb	Albastru
	.....			

Exemplu:

Figura 58

inject.in	***inject.out***
2 3	f(1) = 1 f(2) = 2 f(1) = 1 f(2) = 3 f(1) = 2 f(2) = 1 f(1) = 2 f(2) = 3 f(1) = 3 f(2) = 1 f(1) = 3 f(2) = 2 s-au generat 6 soluții

Figura 59

biject.in	***biject.out***
3	f(1) = 1 f(2) = 2 f(3) = 3 f(1) = 1 f(2) = 3 f(3) = 2 f(1) = 2 f(2) = 1 f(3) = 3 f(1) = 2 f(2) = 3 f(3) = 1 f(1) = 3 f(2) = 1 f(3) = 2 f(1) = 3 f(2) = 2 f(3) = 1 s-au generat 6 soluții

10. Să se genereze toate funcțiile injective f: A → B, unde A={1, 2, …, m} și B = {1, 2, …, n} sunt două mulțimi de numere (m ≤ n).

Datele de intrare se citesc astfel: din fișierul ***inject.in*** se citesc două numere naturale nenule ***m*** și ***n*** separate prin câte un spațiu (care reprezintă cardinalele celor două mulțimi A și B).

În fișierul ***inject.out*** se vor scrie toate funcțiile injective, iar pe ultima linie numărul de funcții injective determinat. Afișarea se va face ca în exemplul din figura 58.

11. Să se genereze toate funcțiile bijective f: A → B, unde A={1, 2, …, m} și B = {1, 2, …, ***m***} sunt două mulțimi de numere.

Din fișierul ***biject.in*** se citește numărul natural nenul ***m*** (care reprezintă numărul de elemente al celor două mulțimi A și B).

În fișierul ***biject.out*** se vor scrie toate funcțiile bijective, iar pe ultima linie numărul de funcții bijective determinat. Afișarea se va face ca în exemplul din figura 59.

12. Identificați varianta corectă de răspuns la următoarele întrebări:

i) Care este relația matematică care determină numărul de funcții injective f: A → B, unde A={1, 2, …, m} și B = {1, 2, …, n}:

a) n^m; b) m^n; c) nu există; d) m×n; e) m! + n!.

ii) Care este relația matematică care determină numărul de funcții bijective f: A → B, unde A={1, 2, …, m} și B = {1, 2, …, m}

a) nu există; b) m^m; c) m^2; d) m! ; e) 2m.

4. GENERAREA PRODUSULUI CARTEZIAN

Se consideră ***n*** mulțimi M_1, M_2, …, M_n. Mulțimile au un număr finit de elemente: $c_1 = |M_1|$, $c_2 = |M_2|$,…, $c_n = |M_n|$. Să se genereze mulțimile produsului cartezian $M_1 \times M_2 \times \ldots \times M_n$.

Datele de intrare se vor citi din fișierul ***cartez.in*** astfel: de pe prima linie numărul natural ***n***, iar de pe următoarea linie ***n*** valori naturale separate prin câte un spațiu, care reprezintă cardinalele celor ***n*** mulțimi.

Soluțiile generate se vor afișa pe linii distincte în fișierul **cartez.out**, iar pe ultima linie a fișierului se va afișa numărul de soluții găsite.

Exemplu:

cartez.out
2
2 3 2

cartez.out
{1, 1, 1}
{1, 1, 2}
{1, 2, 1}
{1, 2, 2}
{1, 3, 1}
{1, 3, 2}
{2, 1, 1}
{2, 1, 2}
……..
{2, 3, 2}
{3, 1, 1}
{3, 1, 2}
{3, 2, 1}
{3, 2, 2}
{3, 3, 1}
{3, 3, 2}
18

Rezolvare:

Etapa I: Identificarea elementelor specifice metodei

Tablou $X=(x_1, x_2, \ldots, x_n)$ unde $x_1 \in M_1$, $x_2 \in M_2$, …, $x_n \in M_n$.

Mulțimile M_1, M_2, …, M_n sunt mulțimi finite având c_1, c_2, …, c_n elemente.

Mulțimile au următoarele elemente: $M_1 = \{1, 2, \ldots, c_1\}$, $M_2 = \{1, 2, \ldots, c_2\}$, …, $M_n = \{1, 2, \ldots, c_n\}$. În această aplicație se observă că mulțimile de valori nu mai coincid.

Tabloul X are ***n*** elemente, pentru că se formează submulțimi doar cu **n** elemente,

$$h_{max} = n, \ (X[i] \in M_i)_{i=1,n}$$

Etapa II: Generarea soluțiilor se va face ca exercițiu, pentru n=4 și $M_1 = \{1, 2\}$, $M_2 = \{1, 2, 3\}$, $M_3 = \{1, 2\}$, $M_4 = \{1, 2\}$prin completarea următorului tabel, pe baza algoritmului Backtracking:

2												
1												
Sol 1	Sol 2	Sol 3	Sol 4	Sol 5	Sol 6	Sol 7	Sol 8	Sol 9	Sol 10	Sol 11	Sol 12	Sol 13

Sol 14	Sol 15	Sol 16	Sol 17	Sol 18	Sol 19	Sol 20	Sol 21	Sol 22	Sol 23	Sol 24	Sol 25	Sol 26

Etapa III: Implemetarea algoritmului Backtracking iterativ:

LIMBAJUL PASCAL	LIMBAJUL C++
var X,card:array[1..20] of integer; i,n,m,k:integer; f,g:text; function mult_val:boolean; begin mult_val:=false; if(X[k]<card[k]) then begin inc(X[k]) ; mult_val:=true; end end; procedure afisare; begin for i:=1 to n do write(g,X[i],' '); writeln end;	#include<fstream.h> ifstream f("cartez.in"); ofstream g("cartez.out"); int X[20], card[20],i,n,k; int mult_val() {if (X[k]<card[k]) { X[k]++ ; return 1 ; } return 0; } void afisare() { for(i=1;i<=p;i++) g<<X[i]<<" "; g<<endl; }

```
procedure generare;
var sol:integer;  sem:boolean;
 begin
  k:=1; sol:=0; X[k]:=0;
  while(k>0) do
   begin
    sem:=mult_val;
    if(sem=true) then
      if(k=n) then  begin
                      inc(sol); afisare;
                    end
      else begin
           inc(k); X[k]:=0; end
     else  dec(k);
  end;
  writeln(g,' s-au generat ',sol,' solutii ');
end;
begin
  assign(f,'cartez.in');reset(f);
      assign(g,'cartez.out');rewrite(g);
        read(f,n);
      for i:=1 to n do  read(f,card[i]);
  generare; close(g);  close(f);
end.
```

```
void generare()
{  int sol=0,sem;
   k=1; X[k]=0;
  while(k)
   { sem=mult_val();
     if(sem)
      if(k==n)
            { sol++; afisare();          }
      else
       {k++; X[k]=0;  }
     else
      k- -;
   }
  g<<" s-au generat "<<sol<<" solutii ";
}
void main()
{ f>>n;
   for(i=1;i<=n;i++)
f>>card[i];
   generare();
   f.close();
g.close();
}
```

TEME

1. Stabiliți valoarea de adevăr a următoarelor afirmații legate de generarea produsului cartezian a ***n*** mulțimi:

Nr. crt.	Afirmații	Răspuns	
		A	F
1	Produsul cartezian este un tablou în care fiecare element se obține ca produs al elementelor din cele ***n*** mulțimi.		
2	Produsul cartezian este o mulțime de mulțimi ce conțin elemente din cele ***n*** mulțimi date.		
3	Produsul cartezian este o mulțime de mulțimi ce conțin câte un element din fiecare dintre cele ***n*** mulțimi date.		

4	Fiecare element dintr-o soluție care se generează depinde de elementele anterioare.		
5	Fiecare element dintr-o soluție care se generează depinde de elementele următoare.		
6	Un element al soluției care se generează nu depinde de restul elementelor.		
7	O soluție generată are numărul de elemente egal cu maximul cardinalelor celor ***n*** mulțimi date.		
8	O soluție generată are numărul de elemente egal cu numărul ***n*** de mulțimi date.		
9	O soluție generată are numărul de elemente egal cu produsul cardinalelor celor ***n*** mulțimi date.		

2. Se consideră trei mulțimi A, B și C având cardinalele a, b respectiv c.

i) Numărul de mulțimi generate prin produsul cartezian al celor trei mulțimi este:

a) a+b+c; b) a*b*c; c) nedeterminat; d) 3 * (a+b+c).

ii) Elementul X[2] al tabloului soluție:

a) poate lua doar valoarea b;

b) poate să ia valori care sunt cuprinse în intervalul [1,b];

c) poate să ia valori care sunt cuprinse în intervalul (1,b);

d) poate să ia valori care sunt cuprinse în intervalul [1,c].

APLICAȚII DE LABORATOR

1. Să se genereze toate funcțiile f: A → B, unde A={1, 2, …, n} și B = {1, 2, …, m} sunt două mulțimi de numere. Programul se va salva cu numele ***nr_fct.pas*** sau ***nr_fct.c*** sau ***nr_fct.cpp***.

nr_fct.in	***nr_fct1.out***
3 2	f(1) = 1 f(2) = 1 f(3) = 1 f(1) = 1 f(2) = 1 f(3) = 2 f(1) = 1 f(2) = 2 f(3) = 3 f(1) = 1 f(2) = 2 f(3) = 1 .....

Datele de intrare se citesc astfel: din fișierul ***nr_fct.in*** se citesc două numere naturale nenule ***n*** și ***m*** separate prin câte un spațiu (care reprezintă cardinalele celor două mulțimi A și B).

În fișierul ***nr_fct.out*** se vor scrie toate funcțiile, iar pe ultima linie numărul de funcții determinate. Afișarea se va face ca în exemplul din figura alăturată .

Care este formula matematică care determină numărul de funcții :

a) n^m; b) n*m; c) m^n; d) nu există.

2. Să se genereze toate funcțiile surjective f: A → B, unde A={1, 2, …, n} și B = {1, 2, …, m} sunt două mulțimi de numere. Programul se va salva cu numele ***surject1.pas*** sau ***surject1.c*** sau ***surject1.cpp.***

surject1.in	***surject1.out***
3 2	f(1) = 1 f(2) = 1 f(3) = 2 f(1) = 1 f(2) = 2 f(3) = 1 f(1) = 1 f(2) = 2 f(3) = 2 f(1) = 2 f(2) = 1 f(3) = 1 f(1) = 2 f(2) = 1 f(3) = 2 f(1) = 2 f(2) = 2 f(3) = 1 s-au generat 6 soluții

Datele de intrare se citesc astfel: din fișierul ***surject1.in*** se citesc două numere naturale nenule ***n*** și ***m*** separate prin câte un spațiu (care reprezintă cardinalele celor două mulțimi A și B).

În fișierul ***surject1.out*** se vor scrie toate funcțiile surjective, iar pe ultima linie numărul de funcții surjective determinate. Afișarea se va face ca în exemplul din figura de mai sus.

Indicație: Se poate modifica programul construit la prima problemă după salvarea lui cu numele de ***surject1.pas*** sau ***surject1.c*** sau ***surject1.cpp***.

3. Să se genereze toate funcțiile surjective f: A → B, unde A={a_1, a_2, …, a_n} și B = {b_1, b_2, …, b_m} sunt două mulțimi de numere reale. O funcție generată trebuie să îndeplinească condiția: $\sum_{i=1}^{v} f_{(ai)} \leq V$,

unde v este o valoare reală cunoscută.

surject2.in	***surject2.out***
3 2 -2 3 7 -3 1 -3	f(-2) = -3 f(3) = -3 f(7) = 1 f(-2) = -3 f(3) = 1 f(7) = -3 f(-2) = 1 f(3) = -3 f(7) = -3 s-au generat 3 soluții

Datele de intrare se citesc din fișierul ***surject2.in*** astfel: de pe prima linie se citesc două numere naturale nenule ***n*** și ***m*** separate prin câte un spațiu (care reprezintă cardinalele celor două mulțimi), de pe a doua linie se citesc **n** numere reale separate prin câte un spațiu (care reprezintă elementele mulțimii A), de pe linia a treia se citesc ***m*** numere reale separate prin câte un spațiu (care reprezintă elementele mulțimii B), iar de pe ultima linie numărul real ***v***.

În fișierul ***surject2.out*** se vor scrie toate funcțiile surjective, iar pe ultima linie numărul de funcții surjective determinate.

Afișarea se va face ca în exemplul din figura de mai sus.

Indicație: Se poate modifica programul construit la problema 2 după salvarea lui cu numele de ***surject2.pas*** sau ***surject2.c*** sau ***surject2.cpp***.

PROBLEME PROPUSE

1. Să se genereze toate cuvintele de lungime ***n*** (***n***<=10) ale alfabetului Morse (formate doar din caracterele '-' și '.') care nu încep și nu se termină cu caracterul '.'

2. Să se genereze toate subșirurile de ***c*** (***c*** ≤ 6) cuvinte dintr-un șir ***x*** format din ***p*** cuvinte de lungime maximă 20 (c∈N*, p∈N* , 0<c<p<10).

Datele de intrare se citesc din fișierul ***cuvinte.in*** astfel: de pe prima linie două numere naturale ***p*** și ***c*** separate prin câte un spațiu, iar de pe următoarea linie se citesc ***p*** cuvinte de lungime maximă 20, separate prin câte un spațiu. Subșirurile generate vor fi afișate pe linii distincte în fișierul ***cuvinte.out***.

3. Să se determine toate soluțiile ecuației 4x+ 2y = 41 , unde (***x***, ***y***) ∈N^2

4. Să se determine toate soluțiile ecuației 3 x+ 2 y +5 z =100, unde (***x***, ***y***, ***z***) ∈N^3,

5. GENERAREA PARTIȚIILOR UNEI MULȚIMI
(GENERAREA SUBMULȚIMILOR UNEI MULȚIMI)

Se citește de la tastatură un număr ***n*** natural nenul. Să se genereze toate partițiile (submulțimile) mulțimii {1, 2, 3, 4, …, n}. Partițiile (submulțimile) vor fi afișate pe linii distincte în fișierul **submult.out**, iar pe ultima linie se va afișa numărul de partiții (submulțimi) generate.

Exemplu: Pentru **n= 3** submulțimile generate sunt reprezentate în caseta alăturată.

submult.out

```
partiția 1
{ 1 2 3 }
partiția 2
{ 1 2 }  { 3 }
partiția 3
{ 1 3 }  { 2 }
partiția 4
{ 1 }  { 2 3 }
partiția 5
{ 1 }  { 2 }  { 3 }
S-au generat 5 partiții
```

Rezolvare:

Definiție: Se numește partiție a mulțimii **P** = {1, 2, 3,…,n} o familie $P_1, P_2, ..., P_m$ de submulțimi ale sale care îndeplinesc următoarele condiții:

- $P_i \neq \emptyset$ pentru $\forall$ i∈ {1, 2, …, m};
- $P_i \cap P_j = \emptyset$ pentru $\forall$ i, j ∈ {1, 2, …, m}, i ≠ j (submulțimile sunt disjuncte între ele sau nu au elemente comune);
- $P_1 \cup P_2 \cup ... \cup P_m = P$.

Reprezentarea partițiilor mulțimii P se poate face în tabloul X=(x_1, x_2, …, x_n), în care elementul x_k (X[k]) reprezintă indicele clasei din care face parte elementul k. Clasa elementului 1 este submulțimea P_1. Clasa elementului 2 este submulțimea P_1 dacă elementul 2 face parte tot din submulțimea P_1, altfel clasa elementului 2 este submulțimea P_2 . Clasa elementului k este P_1, sau P_2,…, sau P_k.

Pentru exemplul dat mai sus soluția 3 afișată : { 1, 3 } { 2 } corespunde următoarelor valori din tabloul soluție X : X[1] = 1, X[2]=2, X[3]=2. Elementele 1 și 3 din mulțimea P={1, 2, 3} fac parte din submulțimea 1, iar elementul 2 face parte din submulțimea 2 a partiției curente.

Etapa I: Identificarea elementelor specifice metodei

Tabloul X=(x_1, x_2, …, x_n) unde $x_1 \in P_1$, $x_2 \in P_2$, …, $x_n \in P_m$.

Mulțimile P_1, P_2, …, P_m (denumite și clase) sunt mulțimi finite având cel mult ***m*** elemente.

Mulțimile de valori nu coincid.

Tabloul X are ***n*** elemente, pentru că se construiesc partițiile cu cele ***n*** elemente ale mulțimii P, deci $h_{max} = n$, $(X[i] \in P_j)_{i=1,n\ ;\ j=1,m}$

Condiții interne: Dacă toate elementele mulțimii P={1, 2,…, n} fac parte din aceeași clasă (aceeași submulțime a partiției curente) atunci toate elementele tabloului X au aceeași valoare. Clasele (submulțimile) partiției curente se numerotează cu valori consecutive (crescătoare), fără 'salturi'. Deci pentru oricare i ∈ {1, 2,…, n} există un element ***j*** al acestei mulțimi astfel încât $|X[i] - X[j]| \leq 1$. De exemplu: dacă n=10 și P1={1, 5, 9} P2={2, 3, 7, 8, 10}, P3 = {4, 6}atunci tabloul soluție X are următoarele valori : X= {1, 2, 2, 3, 1, 3, 2, 2, 1, 2}.

Fie ***max*** valoarea maximă a elementelor (x_1, x_2,…, x_{k-1}). Elementul x_k poate lua valori din mulțimea de valori {1, 2, …, ***max***} (elementul k poate să aparțină uneia dintre submulțimile (clasele) elementelor 1, 2, …, k-1) sau ***max*** *+1* (elementul k generează o nouă clasă (submulțime) a partiției curente). Indicele fiecărei clase x_k nu trebuie să depășească valoarea indicelui k al elementului.

Afișarea unei partiții se face prin identificarea, pe rând, a tuturor elementelor unei submulțimi, acestea fiind caracterizate prin aceeași valoare în componentele corespunzătoare din tabloul soluție X.

Etapa II: Generarea soluțiilor se va face ca exercițiu, pentru ***n***=4 și P = {1, 2, 3, 4} prin completarea următorului tabel, pe baza algoritmului Backtracking:

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

Partiția 1	{ 1, 2, 3, 4}	Partiția 10	
Partiția 2	{ 1, 2, 3} {4}	Partiția 11	
Partiția 3		Partiția 12	
Partiția 4		Partiția 13	
Partiția 5		Partiția 14	

Partiția 6		Partiția 15	
Partiția 7		Partiția 16	
Partiția 8		Partiția 17	
Partiția 9		Partiția 18	
S-au generat …… partiții			

Etapa III: Implemetarea algoritmului Backtracking iterativ

LIMBAJUL PASCAL	LIMBAJUL C++
var X :array[1..20]of integer; i,n,k:integer; g:text; function mult_val:boolean; var max:integer; begin if(k=1) then max:=1 else begin max:=X[1]; for i:=1 to k-1 do if max<X[i] then max:=X[i] end; if(X[k]<max+1)and (X[k]<k) then begin inc(X[k]); mult_val:=true; end else mult_val:=false; end; procedure afisare; var max,j:integer; begin max:=X[1]; for i:=2 to n do if max<X[i] then max:=X[i]; for i:=1 to max do begin write(g,' { '); for j:=1 to n do if X[j]=i then write(g,j,' '); write(g,'}'); end;	#include<fstream.h> ofstream g("submult.out"); int X[20],n,k; int mult_val() {int max,i; if(k==1) max=1; else { max=X[1]; for(i=2;i<k;i++) if(max<X[i]) max=X[i]; } if((X[k]<max+1) &&(X[k]<k)) { X[k]++; return 1; } else return 0; } void afisare() { int max,i,j; max=X[1]; for(i=2;i<=n;i++) if(max<X[i]) max=X[i]; for(i=1;i<=max;i++) { g<<" { "; for(int j=1;j<=n;j++) if(X[j]==i) g<<j<<' '; g<<"} "; } g<<endl;

LIMBAJUL PASCAL	LIMBAJUL C++
writeln(g) end; procedure generare; var sol:integer; sem,ev:boolean; begin k:=1; sol:=0; x[k]:=0; while(k>0) do begin sem:=mult_val; if(sem=true)then if(k=n) then begin inc(sol);writeln(g,' partitia ',sol); afisare; end else begin inc(k); X[k]:=0; end else dec(k); end; writeln(g,' s-au generat ',sol,' solutii '); end; begin assign(g,'submult.out');rewrite(g); read(n); generare; close(g); end.	} void generare() { k=1; int sol=0,sem; X[k]=0; while(k) { sem=mult_val(); if(sem) if(k==n) { sol++; g<<" partitia "<<sol<<endl; afisare(); } else {k++; X[k]=0; } else k - -; } g<<" s-au generat "<<sol<<" partitii "; } void main() { cin>>n; generare(); g.close(); }

TEME

1. Stabiliți valoarea de adevăr a următoarelor afirmații, legate de generarea partițiilor unei mulțimi **P** = {1, 2, 3, ..., n}.

Nr. crt.	Afirmații	Răspuns	
		A	F
1	Se numește partiție a mulțimii **P** = {1, 2, 3, ..., n} o familie P_1, P_2, P_m, de submulțimi ale sale care îndeplinesc următoarele condiții: • Pi = ∅ pentru ∀ i ∈{1, 2, ..., m}; • Pi ∩ Pj pentru ∀ i, j ∈{1, 2, ..., m} i ≠ j; • (P1 ∪ P2 ∪ ... ∪ Pm = P		
2	Dacă toate elementele mulțimii P = {1, 2, ..., n} fac parte din aceeași clasă (aceeași submulțime a partiției curente) atunci toate elementele tabloului X au aceeași valoare.		
3	Elementul x_k poate lua valori din mulțimea de valori {1, 2, ..., *max*} (elementul k poate să aparțină uneia dintre submulțimile (clasele) elementelor 1, 2, ..., k-1) sau *max* + 1 (elementul k generează o nouă clasă (submulțime) a partiției curente).		
4	Indicele fiecărei clase x_k poate să depășească valoarea indicelui k al elementului.		
5	Clasele (submulțimile) partiției curente se numerotează cu valori consecutive (crescătoare), fără 'salturi'. Pentru oricare i ∈{1, 2, ..., n} există un element j al acestei mulțimi astfel încât $\lvert X[i] - X[j] \rvert \geq 1$.		
6	Reprezentarea partițiilor mulțimii P se poate face în tabloul X = (x_1, x_2, ..., x_n), în care elementul x_k(X[k]) reprezintă indicele clasei din care face parte elementul k.		
7	Mulțimile de valori coincid.		
8	Mulțimile P_1, P_2, P_m, (denumite și clase) sunt mulțimi finite având cel mult ***n*** elemente.		
9	Tabloul soluție X are ***n*** elemente (h_{max} = *n,* $(X[i] \in P_j)_{i=1,n;\ j=1,m}$).		

2. Se citeşte un număr ***n*** natural de la tastatură (intrarea standard).

a) Să se determine descompunerile numărului ***n*** ca sumă de numere naturale. Toate descompunerile şi numărul acestora se vor afişa pe linii distincte în fişierul ***descomp.out.***

Exemplu: n = 4

descomp.out

```
4 = 1 + 1 + 1 + 1
4 = 1 + 1 + 2
4 = 1 + 2 + 1
4 = 1 + 3
4 = 2 + 1 + 1
4 = 2 + 2
4 = 3 + 1
4 = 4
S-au generat 8 descompuneri
```

b) Să se determine descompunerile *distincte* ale numărului ***n*** ca sumă de numere naturale. Toate descompunerile şi numărul acestora se vor afişa pe linii distincte în fişierul ***descomp1.out.***

Exemplu: n = 4

Descomp1.out

```
4 = 1 + 1 + 1 + 1
4 = 1 + 1 + 2
4 = 1 + 3
4 = 2 + 2
4 = 4
S-au generat 5 descompuneri
```

c) Să se determine descompunerile numărului ***n*** ca sumă de numere naturale distincte. Toate descompunerile şi numărul acestora se vor afişa pe linii distincte în fişierul ***descomp2.out.***

Exemplu: n = 4

Descomp2.out

```
4 = 1 + 3
4 = 3 + 1
4 = 4
S-au generat 4 descompuneri
```

d) Să se determine descompunerile numărului ***n*** ca sumă de ***k*** numere naturale. Numărul ***k*** este citit tot de la tastatură. Toate descompunerile şi numărul acestora se vor afişa pe linii distincte în fişierul ***descomp3.out.***

Exemplu: n = 9 și k=3

Descomp3.out	
9 = 1 + 2 + 6	9= 2 + 4 + 3
9 = 1 + 3 + 5	……
9 = 2 + 1 + 6	
9 = 2 + 3 + 4	
S-au generat …… soluții	

e) Să se determine descompunerile distincte ale numărului ***n*** ca sumă de ***k*** numere naturale distincte. Numărul ***k*** este citit tot de la tastatură. Toate descompunerile și numărul acestora se vor afișa pe linii distincte în fișierul ***descomp4.out.***

f) În tabelul de mai jos să se completeze numărul de soluții obținut după execuția programelor scrise la punctele a), b), c), d) și e) pentru datele de intrare indicate.

Programul a)	Programul b)	Programul c)	Programul d)
N= 8	N=12	N= 16 și k = 4	N=24 si k = 3
Nr. Soluții =	Nr. Soluții =	Nr. Soluții =	Nr. Soluții =

PROBLEME PROPUSE

1. Se citește de la tastatură un număr natural nenul ***n***. Să se genereze toate descompunerile numărului **n** ca sumă de numere prime distincte. În fișierul ***s_prim.out*** se vor afișa pe linii distincte toate descompunerile generate.
Exemplu: pentru n=30 o parte dintre soluțiile generate sunt: (17,13), (7,23), (2, 5, 11, 13) etc.

2. Fie ***n*** un număr natural nenul. Să se genereze toate descompunerile numărului ***n*** ca sumă doar de două numere naturale nenule ***a*** și ***b*** (a,b<n). Datele de intrare: ***n***, ***a*** și ***b*** se citesc din fișierul ***date.in.*** de În fișierul ***date.out*** se vor afișa pe linii distincte toate descompunerile generate.
Exemplu: n = 38, a= 3, b=7 atunci soluțiile generate sunt: (3, 3, 3, 3, 3, 3, 3, 3, 7, 7), (3, 7, 7, 7, 7, 7).

3. De la tastatură se citește un număr natural nenul ***n***. Să se determine toate posibilitățile de reprezentare a numărului ca sumă de numere naturale, cu proprietatea că suma numerelor (din descompunerea curentă) inversate este subunitară. Toate descompunerile determinate se vor afișa pe monitor, pe linii distincte.
Exemplu: n=10 soluțiile generate sunt: (2, 8), (3, 3, 4), (3, 7),…, (4, 6), (5, 5), (10).

4. Se citește de la tastatură un număr natural nenul ***m***. Să se genereze toate descompunerile numărului ***m*** ca sumă de numere naturale distincte al căror cel mai mare divizor comun este maxim. Descompunerile generate vor fi afișate pe linii distincte în fișierul ***date.out***.
Exemplu: ***m***=68 soluția : (17,51).

5. Să se genereze toate șirurile strict crescătoare formate din numerele naturale cu proprietatea că primul element din șir este ***p***, iar ultimul element al șirului este ***p+k***. Numerele naturale nenule ***p*** și ***k*** (0<p<20, 0<***k***<15) se citesc de la tastatură. Șirurile generate se vor afișa pe linii distincte în fișierul ***sir.txt***, iar pe ultima linie a fișierului se va afișa numărul de șiruri determinate. **Exemplu:** pentru ***n*** = 7 și ***k*** = 3 se vor afișa șirurile: (7,8,10), (7,9,10) (7,10), (7,8,9,10) (ordinea generării poate să nu coincidă cu cea afișată).

MINIPROIECT ÎN ECHIPĂ. COMPANIA EFICIENT

Managerul companiei Eficient dorește să afle rapid vînzările realizate de oricare dintre angajați și să organizeze distribuitorii în grupe de câte trei membri, astfel încât vârsta medie a unui grup să nu depășească 21 de ani.

Cerințe:

1. Analizați noile solicitări ale managerului și stabiliți prelucrările necesare pentru rezolvarea acestora.

2. Determinați efectele adăugării acestor prelucrări la programele existente.

3. Implementați unitățile de program corespunzătoare noilor prelucrări.

4. Completați documentația proiectului cu concluziile rezultate din analiza și rezolvarea noilor cerințe.

Capitolul 4

ELEMENTE DE TEORIA GRAFURILOR

1. SCURT ISTORIC

Primele elemente de teoria grafurilor au apărut încă din anul 1736, când matematicianul *Leonhard Euler* a publicat, la *Academia de Științe din Petersburg,* un articol în care propunea, sub formă de joc matematic, rezolvarea problemei podurilor din Königsberg (astăzi orașul Kaliningrad). Aici existau șapte poduri peste delta râului Pregel, desenate ca în figura 60:

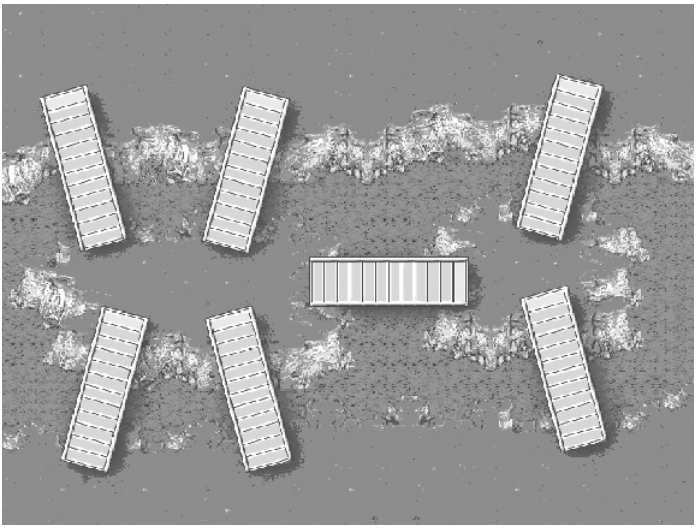


Figura 60.

Problema lui Euler:

Se poate face o plimbare, care să includă în traseul său, toate cele șapte poduri, astfel încât fiecare pod să fie traversat o singură dată, iar punctul de oprire să coincidă cu punctul de plecare?

Euler a demonstrat că acest lucru este imposibil: indiferent de punctul de la care se începe plimbarea, nu se poate parcurge tot traseul fără a trece de două ori pe un pod.

În anul 1857, matematicianul *William Hamilton* a inventat un joc ce are la bază noțiuni de teoria grafurilor. Acest joc avea o piesă de lemn în forma unui dodecaedru: poliedru cu 12 fețe care sunt toate pentagoane regulate, iar în fiecare din cele 20 de vârfuri se întâlnesc câte 3 muchii.

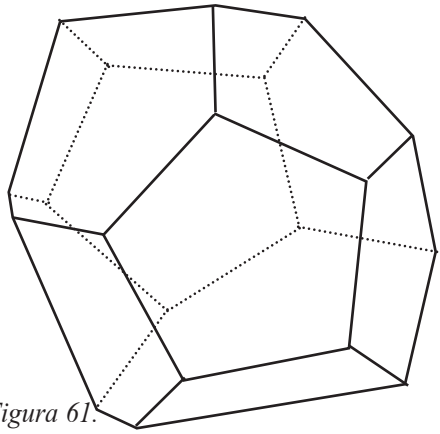
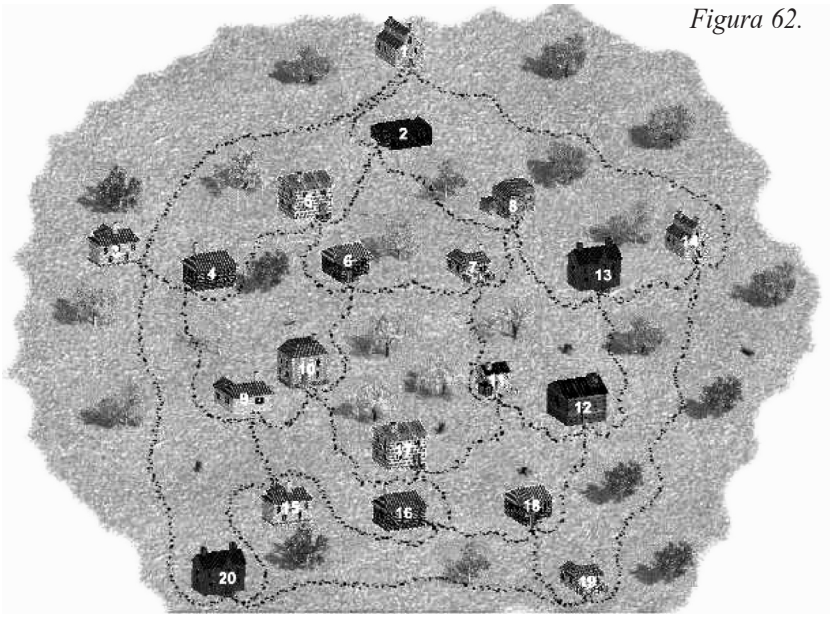


Figura 61.

Fiecare vârf al dodecaedrului lui Hamilton era marcat cu numele unui oraș. Jocul consta în determinarea unui drum de-a lungul muchiilor dodecaedrului, care să treacă prin fiecare din cele 20 de orașe exact o dată și să se întoarcă în orașul din care a plecat. Pentru a ușura memorarea trecerilor efectuate, în fiecare vârf al dodecaedrului era câte un cui cu o floare, astfel încât în jurul acestor cuie putea să se întindă un fir care să indice drumul parcurs în această călătorie imaginară.

O problemă mai generală este aceea a voiajorului comercial: *Un voiajor comercial trebuie să prezinte în* **n** *orașe produsele firmei pe care o reprezintă, după care se întorce în orașul din care a plecat. Cunoscându-se costul deplasării între oricare două dintre cele* **n** *orașe, se cere să se determine un traseu care să viziteze o singură dată cele* ***n*** *orașe și care să aibă un cost total minim.*

Figura 62.



Elemente din lumea reală ce pot fi modelate prin teoria grafurilor:
— puncte distribuite pe o suprafață și legăturile dintre ele: insule-poduri, orașe-drumuri;
— relațiile interpersonale;
— ordonarea unor activități;
— circuite electrice: la mijlocul secolului trecut, fizicianul Kirchhoff a studiat rețelele electrice bazându-se pe elemente de teoria grafurilor și a contribuit astfel la dezvoltarea acesteia;
— harta căilor ferate este o reprezentare sub formă de graf a legăturilor feroviare dintre localități:

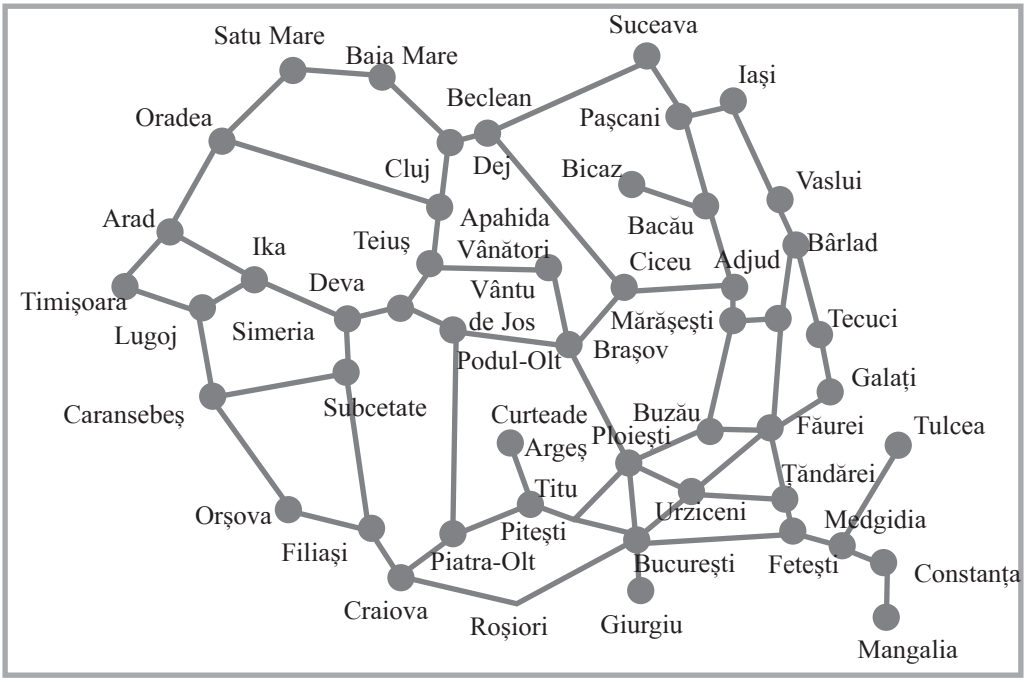


Figura 63.

• formulele de structură ale substanțelor chimice sunt grafuri pentru care legăturile dintre vârfuri corespund legăturilor dintre grupările sau atomii care compun molecula.

În figura 64 este prezentată molecula de glucoză.

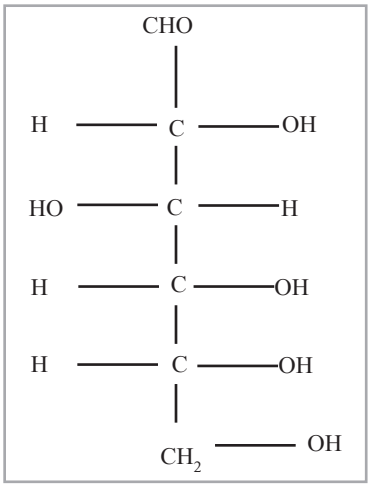


Figura 64.

2. GRAFURI NEORIENTATE

2.1. Noțiuni de bază

Definiție

Se numește *graf neorientat* o pereche ordonată de mulțimi (X,U) cu semnificația: X este o mulțime finită și nevidă de elemente numite *noduri*, U este o mulțime de perechi neordonate (submulțimi cu două elemente din X), numite *muchii.*

Un graf neorientat se notează cu G=(X,U), unde X se numește *mulțimea nodurilor* grafului G, iar U se numește mulțimea muchiilor.

O submulțime {x,y} de vârfuri din X se notează cu u=[x,y] (u este muchie iar x și y sunt extremitățile), u ∈ U.

În cazul general, într-un graf neorientat G = (X,U), se folosesc notațiile:

- card (X) = *n* numărul de noduri din graf;
- card (U) = *m* numărul de muchii din graf.

Multe dintre problemele de interes practic pot fi reprezentate prin grafuri. Structura unui website poate fi reprezentată folosind un graf: nodurile sunt paginile website-ului; există o muchie care leagă pagina oarecare A de pagina oarecare B, dacă și numai dacă pagina A conține un link către pagina B.

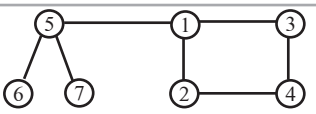
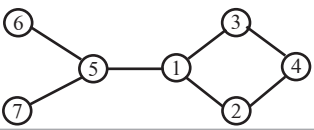


Figura 65

Figura 66



Pentru vizualizare și înțelegere intuitivă, un graf poate fi reprezentat cu ajutorul unei figuri plane, în care fiecărui nod x∈X i se asociază un punct în plan, iar fiecărei muchii u=[x,y] i se asociază o linie ce unește punctele corespunzătoare nodurilor x și y.

Grafurile din figura 65 și figura 66 sunt echivalente.

Exemplul 1:

Fie G=(X,U) un graf neorientat cu reprezentarea din figura 67.

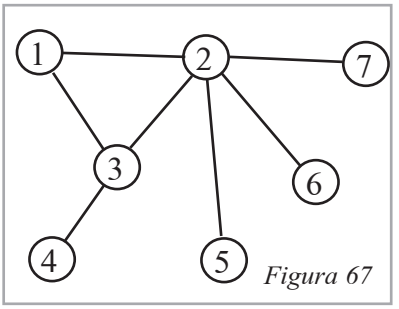


Figura 67

Definiție

Fie u∈U , u=[x,y] . Nodurile x și y din X sunt *adiacente* în *G* iar *u* și *x* sunt *incidente* (la fel u și y).

– Nodul 1 este adiacent cu nodul 2 deoarece ele sunt extremități ale muchiei [1,2].

– Nodul 2 este adiacent cu nodul 6 deoarece ele sunt extremități ale muchiei [2,6].
– Nodul 3 este incident cu muchia [2,3] deoarece el este extremitate a muchiei.

Observație:

Nu există diferență între muchia [1,2] și muchia [2,1] (nu există orientare a muchiei).

Exemplul 2:

X = {a, b, c, d, e, f, g, h, i, j}

U = { [a,b]; [a,c]; [a,d]; [f,g]; [f,h]; [g,h]; [g,j]; [h,i]; [j,i] }

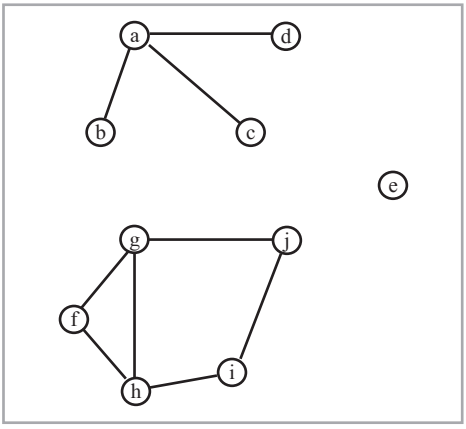


Figura 68.

Exemplul 3 (transpus din fizică):

X = {1, 2, 3, 4, 5, 6, 7,8}

U = { [1,2]; [1,3]; [1,5]; [2,4]; [2,8]; [3,4]; [5,6]; [6,7];[6,8]; [7,8] }

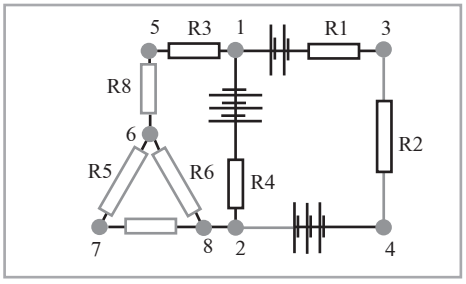


Figura 69.

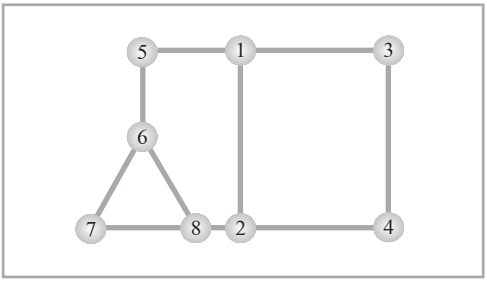


Figura 70.

2.2. Gradul unui nod

Definiție:

prin gradul unui nod x, notat $d(x)$, se înțelege numărul de muchii incidente cu nodul x.

Exemplul 1:

Fie graful neorientat G=(X,U) (figura 71),

X = {1, 2, 3, 4, 5, 6} și U = { [1,2] ; [1,3] ; [1,4] ; [1,5] ; [3,4] ; [3,5] ; [4,6]};

n = 6 (numărul de noduri),

m = 7 (numărul de muchii).

Se determină gradul fiecărui nod:

d(1)=4, deoarece muchiile incidente cu nodul 1 sunt în număr de 4
([1,2], [1,3], [1,4] și [1,5]) ;
d(2)=1, deoarece există o singură muchie incidentă cu nodul 2 ([1,2]) ;
d(3)=3, deoarece muchiile incidente cu nodul 3 sunt în număr de 3
([1,3], [3,4] și [3,5]) ;
d(4)=3, deoarece muchiile incidente cu nodul 4 sunt în număr de 3
([1,4], [3,4] și [4,6]) ;
d(5)=2, deoarece muchiile incidente cu nodul 5 sunt în număr de 2
([1,5] și [3,5]) ;
d(6)=1, deoarece există o singura muchie incidentă cu nodul 6([4,6]).

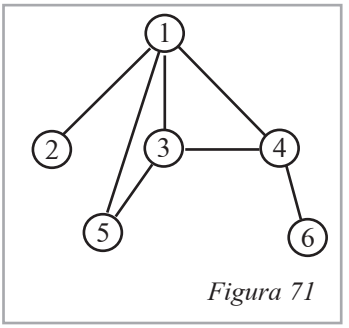


Figura 71

Definiție:
Se numește *nod izolat* un nod cu gradul 0.
Se numește *nod terminal* un nod cu gradul 1.

Exemplul 2 (figura 72):
d(a)=*d*(g)=*d*(h)=3 ;
d(b)=*d*(c)=*d*(d)=1; b,c,d sunt noduri terminale;
d(f) = *d*(j)=*d*(i)=2;
d(e) = 0, deci nodul *e* este nod izolat (nu există nicio muchie incidentă cu nodul (*e*).

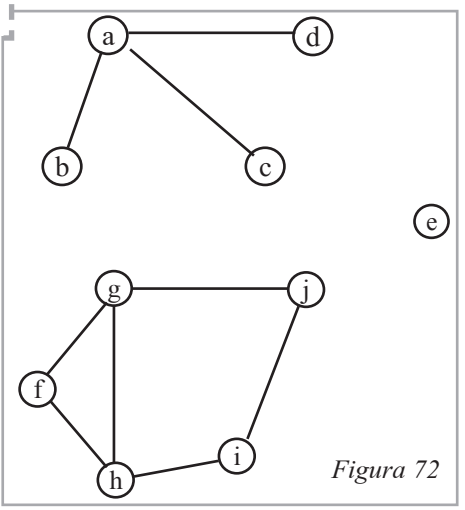


Figura 72

TEME

1. Fie G = (X,U), un graf neorientat;
X = { 1, 2, 3, 4, 5} și
U = { [1,3]; [1,6]; [2,4]; [2,5]; [3,6]; [4,5]}
Cerințe:
a) Realizați reprezentarea grafică a acestui graf.
b) Determinați nodurile de grad par.
c) Determinați muchiile incidente cu nodul 2 și nodul de grad maxim.

2. Pentru graful din figura 73, determinați nodul cu cel mai mare grad.

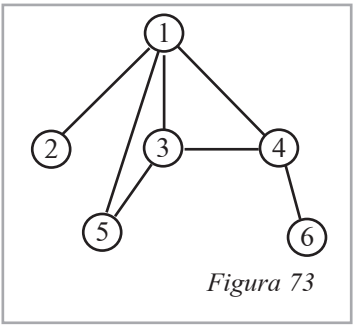


Figura 73

3. Pentru graful neorientat din figura 74, determinați:

a) Mulțimea muchiilor și mulțimea nodurilor.

b) Nodurile izolate.

c) Nodurile cu grad impar.

d) Nodurile adiacente cu nodul 3.

e) Nodurile impare de grad par.

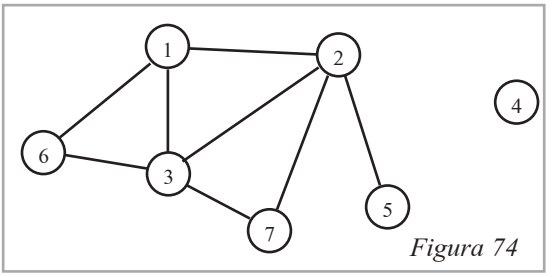


Figura 74

4. Se cunosc contractele încheiate între firmele A, B, C, D, E, F:

Firma A a încheiat contracte cu firmele C și D.

Firma B a încheiat contracte cu firmele C, E, F.

Firma E a încheiat un contract cu firma F.

Cerințe:

a) asociați datelor din problemă reprezentarea grafică corespunzătoare;

b) determinați mulțimea de noduri și mulțimea de muchii pentru graful obținut;

c) determinați firma ce a încheiat cele mai multe contracte și firma ce a încheiat cele mai puține contracte;

d) determinați numărul de contracte încheiate între firmele A, B, C, D, E, F;

e) precizați care este semnificația numărului de contracte încheiate de o firmă.

5. Să se determine mulțimea U a grafului neorientat G = (X,U), unde,

$X = \{1, 2, 3\}$ și $d(1) = 1$, $d(2) = 2$, $d(3) = 1$.

6. Fie funcția f: $\{1, 2, 3, 4, 5\} \rightarrow \{2, 3\}$ definită

$$f(x) = \begin{cases} 3, \textit{dacă } x \textit{ este numar par} \\ 5, \textit{dacă } x \textit{ este numar impar} \end{cases}$$

Să se determine mulțimea U a grafului G= (X,U) obținut prin asocierea cu funcția f.

Realizați reprezentarea grafică a lui G.

7. Fie G = (X,U) un graf neorientat cu 10 noduri. Toate nodurile sunt izolate. Precizați numărul de muchii ale grafului.

2.3. Reprezentarea în memorie a grafurilor neorientate

Fie G = (X,U) un graf neorientat, unde X = {1, 2, 3, …, n}, n∈N*, este mulțimea nodurilor, iar U este mulțimea muchiilor. Reprezentarea în memoria calculatorului a unui graf neorientat se poate face prin mai multe modalități; alegerea formei de reprezentare a unui graf depinde de problema pentru care se solicită algoritmul de rezolvare.

2.4. Memorarea grafurilor folosind matricea de adiacență

Matricea de adiacență atașată unui graf este o matrice pătrată cu *n* linii și *n* coloane (*n* este numărul de noduri din mulțimea *X*), cu elementele:

$$A[i, j] = \begin{cases} 1, [i, j] \in U; \\ 0, [i, j] \notin U. \end{cases}$$

Observații:

1. Matricea de adiacență atașată unui graf neorientat este o matrice simetrică față de diagonala principală, deoarece pentru orice muchie [i,j] ∈ U este adevarată relația

A[i,j] = A[j,i].

2. Muchia [i,j] există în graful neorientat G = (X,U) reprezentat prin matricea de adiacență, dacă A[i,j] = 1.

3. Suma valorilor din matricea de adiacență atașată unui graf neorientat este egală cu 2**m*, unde *m* este numărul muchiilor din graf.

4. Fie x∈X. Gradul nodului *x* este egal cu suma valorilor elementelor matricei de adiacență atașată grafului neorientat, de pe linia *x* sau coloana *x*.

Figura 75

Exemplul 1:

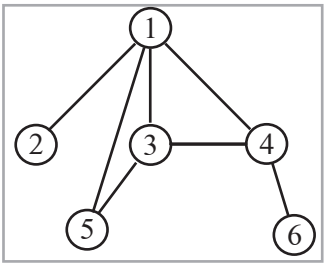
Fie G = (X,U) , din figura 75.

X = {1, 2, 3, 4, 5, 6} și U = { [1,2], [1,3], [1,4], [1,5], [3,4], [3,5], [4,6]};

n = 6 (numărul de noduri),

m =7 (numărul de muchii).

Matricea de adiacență asociată acestui graf este:



$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Elementele din matricea de adiacență situate pe diagonala principală au valoarea 0. Acestea sunt de forma A[i,i] și, cum nu există muchie de la un nod la el însuși pentru grafurile studiate, rezultă că A[i,i] = 0.

Se observă că d(1) = 4 (numărul elementelor egale cu 1 de pe prima linie), d(2) = 1 (numărul elementelor egale cu 1 de pe linia 2).

Pentru graful dat, suma elementelor din matricea atașată este 14, adică 2*m.

Matricea de adiacență atașată unui graf neorientat poate fi citită direct de la tastatură sau din fișierul de intrare sau poate fi completată inițial cu 0, urmând a fi citite din fișierul de intrare *m* perechi de vârfuri ce reprezintă muchiile de forma [x,y]; se completează A[x,y] = 1 și A[y,x] = 1.

Exemplu de citire a matricei de adiacență atașată unui graf neorientat dintr-un fișier dat:

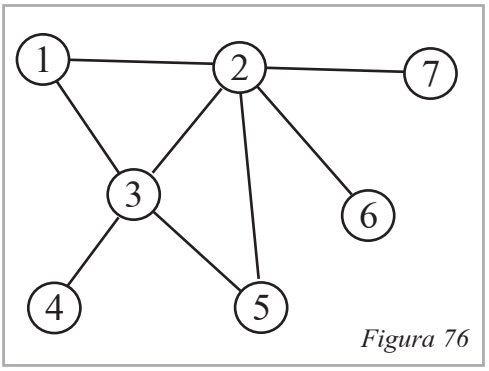


Figura 76

Matrice.txt

```
7
0 1 1 0 0 0 0
1 0 1 0 1 1 1
1 1 0 1 1 0 0
0 0 1 0 0 0 0
0 1 1 0 0 0 0
0 1 0 0 0 0 0
0 1 0 0 0 0 0
```

Varianta Pascal	Varianta C++
procedure citirematrice(var a :matrice ; var n :integer); var i,j:integer; begin assign(f,'matrice.txt'); reset(f); readln(f,n); for i:=1 to n do for j:=1 to n do read(f,a[i,j]); close(f); end;	void citirematrice(int a[11][11], int &n) {int i,j; fstream f("c:\\matrice.txt",ios::in); f>>n; for (i=1;i<=n;i++) for (j=1;j<=n;j++) f>>a[i][j]; f.close(); }

Exemplul 2:

Fie G = (X,U) un graf neorientat,

n este numărul de noduri din *G*, *m* este numărul de muchii din *G*.

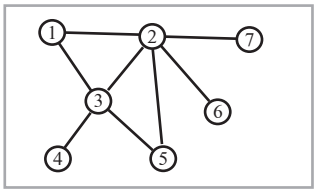
Următoarea secvență de program calculează gradul pentru fiecare nod al grafului G reprezentat prin matricea de adiacență:

Varianta Pascal	Varianta C ++
for i :=1 to n do begin gr:=0; for j:=1 to n do if a[i,j]=1 then gr:=gr+1; writeln('Gradul lui ',I,' este ',gr) end ;	for (i=1;i<=n;i++) { gr=0 ; for (j=1;j<=n;j++) if (a[i][j]) gr++; cout<<"Gradul lui "<<i; cout<<" este "<<gr<<endl; }

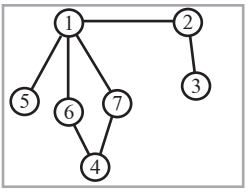
TEME

1. Construiți matricea de adiacență pentru fiecare dintre grafurile de mai jos:

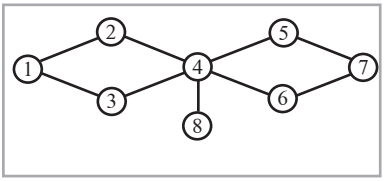
a) G1



b) G2



c) G3



2. Determinați valoarea de adevăr a următoarelor afirmații:

a) Matricea de adiacență a unui graf neorientat are un număr impar de valori de 1.

b) Orice matrice de adiacență a unui graf neorientat are pe diagonala secundară numai valori nule.

c) Din matricea de adiacență a unui graf neorientat se poate afla gradul oricărui nod.

d) Suma componentelor matricei de adiacență atașată unui graf neorientat este egală cu 2*n, unde n = numărul de noduri ale grafului.

e) Din matricea de adiacență atașată unui graf neorientat se poate determina numărul de muchii din graf.

f) Matricea de adiacență a unui graf neorientat este simetrică față de diagonala principală.

3. Fie un graf G = (X, U) reprezentat prin matricea de adiacență:

Precizați care dintre următoarele muchii aparțin grafului:

a) [3,4] b) [2,3] c) [1,2] d) [4,5]

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

4. Fie un graf G = (X,U) unde X = {1, 2, 3, 4, 5, 6, 7, 8}.

În graf există muchii între perechile de noduri *x* și *y* cu proprietatea ca *x* și *y* sunt prime între ele.

a) Determinați mulțimea muchiilor U.

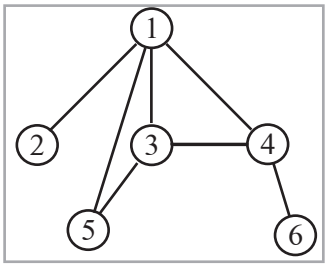
b) Construiți matricea de adiacență atașată grafului dat.

c) Determinați suma elementelor matricei de adiacență.

2.5. Memorarea grafurilor folosind listele de adiacență

Reprezentarea unui graf neorientat se poate realiza utilizând listele de adiacență a nodurilor: pentru fiecare nod se alcătuiește lista nodurilor adiacente cu el.

Figura 77.



Exemplu:
Pentru graful G = (X,U)
se formează următoarele liste de adiacență:

Lista vecinilor lui 1: 2, 3, 4, 5
Lista vecinilor lui 2: 1
Lista vecinilor lui 3: 1, 4, 5
Lista vecinilor lui 4: 1, 3, 6
Lista vecinilor lui 5: 1, 3
Lista vecinilor lui 6: 4.

Observații:

1. Numărul de elemente din lista vecinilor unui nod reprezintă gradul său. Dacă nodul este izolat, atunci lista lui de adiacență este vidă.

2. Memorarea grafului prin intermediul listelor de adiacență prezintă avantajul că, în general, ocupă mai puțin spațiu. Dezavantajul utilizării listelor de adiacență îl constituie accesul la informații.

3. Suma lungimilor listelor este egală cu 2**m*, unde *m* reprezintă numărul de muchii.

Următoarea secvență de program construiește listele de adiacență pentru un graf neorientat *G,* pornind de la matricea de adiacență atașată acestuia. Se obține o matrice ce va conține pe linia *i* (1<=i<=n) lista de adiacență a nodului *i.* Se obține o matrice de liste.

Varianta Pascal	Varianta C ++
type matrice=array[1..20,1..20] of integer ; var a,liste:matrice;	int a,liste[21][21]

Se inițializează cu 0 toate elementele matricei de liste.

Varianta Pascal	Varianta C ++
procedure liste_de_adiac(a :matrice ; n :integer ;var liste :matrice) ; var i,j,k:integer; begin for i:=1 to n do for j:=1 to n do liste[i,j]:=0; for i :=1 to n do begin k:=1;	void liste_de_adiac(int a[21][21], liste[21][21]) int i,j,k ; { for (i=1;i<=n;i++) for (j=1;j<=n;j++) liste[i][j]=0; for (i=1;i<=n;i++) {k=1; for (j=1;j<=n;j++)

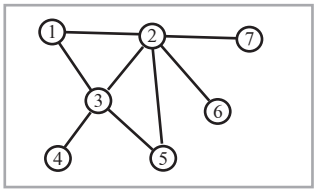
```
        for j:=1 to n do
            if a[i,j]=1 then
              begin
                liste[i,k]:=j;
                k:=k+1;
              end;
    end;
end;
```

```
                if (a[i][j])
                  {liste[i][k]=j;
                   k++;
                   }
            }
}
```

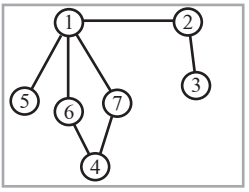
TEME

1. Construiți listele de adiacență pentru fiecare dintre grafurile de mai jos:

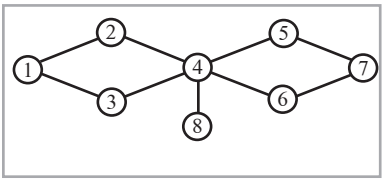
a) G1



b) G2



c) G3



2. Fie un graf G = (X, U) reprezentat prin matricea de adiacență:

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

a) Specificați dacă graful G are noduri izolate.
b) Câte muchii are graful G?
c) Determinați gradul fiecărui nod.
d) Determinați listele de adiacență pentru graful G.

3. Fie graful lui Petersen (figura 78):
a) Construiți listele de adiacență.
b) Analizați listele construite; ce observați ?

4. Fie un graf G = (X,U), unde X = {1, 2, 3, 4, 5, 6, 7, 8}.
În graf există muchii între perechile de noduri *x* și *y* cu proprietatea ca *x* și *y* sunt prime între ele. Determinați listele de adiacență pentru graful *G*.

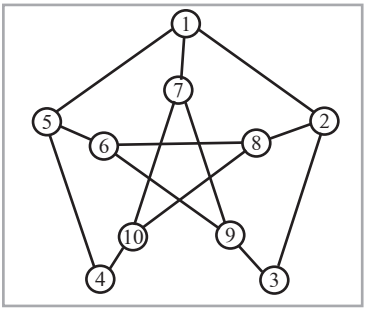


Figura 78

2.6. Memorarea grafurilor neorientate folosind lista muchiilor

Fie G = (X,U) un graf neorientat cu *n* noduri și *m* muchii.

Lista extremităților muchiilor este formată dintr-o matrice de două coloane și *m* linii, în care pe fiecare linie există două valori ce reprezintă nodurile componente ale unei muchii. Pentru memorarea acestei matrice sunt necesare 2**m* locații de memorie.

Exemplu:

Fie graful G = (X,U) (figura 79).

Pentru acest graf, lista muchiilor este reținută în matricea A:

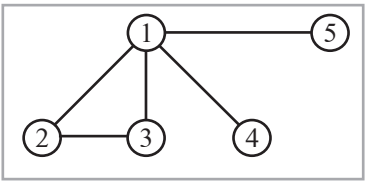
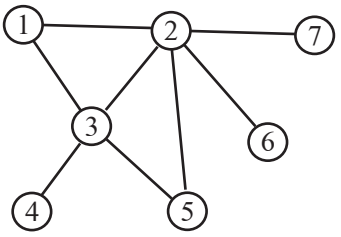


Figura 79

$$A = \begin{pmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \\ 2 & 3 \end{pmatrix}$$

Secvența următoare de program construiește lista muchiilor pentru un graf neorientat. Citirea datelor se face dintr-un fișier organizat astfel: pe prima linie se precizează numărul de noduri și numărul de muchii, iar pe fiecare dintre următoarele linii cele două extremități ale unei muchii:



Muchii.txt

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

Varianta Pascal

```
type matrice=array[1..20,1..2] of integer ;
procedure citire_muchii(var a :matrice ;
var n,m :integer) ;
var i:integer;
begin
  assign(f,'muchii.txt');
  reset(f);
  readln(f,n,m);
  for i:=1 to m do
      begin
         read(f,a[i,1]);
         read(f,a[i,2]);
       end;
  close(f);
end;
```

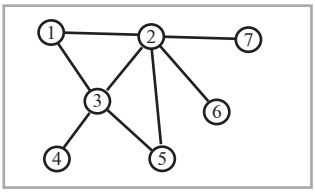
Varianta C ++

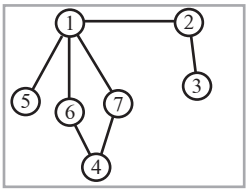
```
int a[21][3]
Void citire_muchii(int a[21][3],
int &n, int &m)
int i;
{
  fstream f("c:\\muchii.txt",ios::in);
  f>>m;
  f>>n;
  for (i=1;i<=n;i++)
   {
     f>>a[i][1];
     f>>a[i][2];
    }
f.close();
}
```

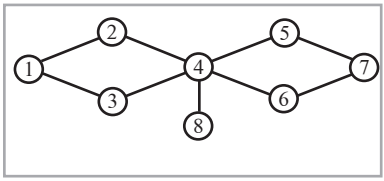
TEME

1. Construiți lista muchiilor pentru fiecare dintre grafurile de mai jos:

a) G1 b) G2 c) G3







2. Fie un graf G = (X,U), unde X = {1, 2, 3, 4, 5, 6, 7, 8}.

În graf există muchii între perechile de noduri *x* și *y* cu proprietatea că *x* și *y* sunt prime între ele. Determinați lista muchiilor pentru graful G.

3. Fiind dat un graf neorientat memorat prin matricea sa de adiacență, să se scrie câte un subprogram pentru rezolvarea următoarelor cerințe:
- afișarea nodurilor de grad maxim;
- afișarea nodurilor de grad minim;
- afișarea nodurilor izolate;
- afișarea nodurilor terminale.

4. Se cunosc listele de adiacență corespunzătoare unui graf neorientat. Să se scrie un program care construiește matricea sa de adiacență.

5. Harta unei țări este formată din mai multe regiuni. O parte dintre regiuni au frontiera comună. Scrieți un program care să afișeze regiunea care se învecinează cu cele mai multe dintre regiuni.

6. Implementați memorarea grafurilor neorientate prin lista muchiilor, folosind un vector de articole. (Indicație: fiecare articol conține două câmpuri corespunzătoare extremităților unei muchii.)

3. CLASE SPECIALE DE GRAFURI

3.1 Grafuri complete

Definiție:

Un graf G=(X,U) se numește graf complet dacă între oricare două noduri ale sale există muchii. Un graf complet se notează Kn.

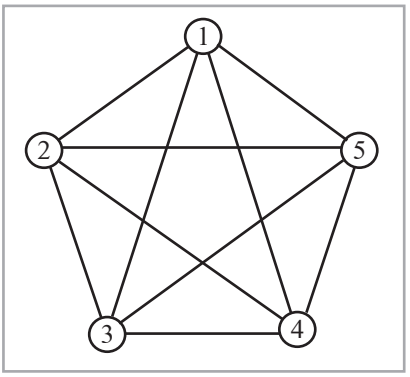


Figura 80.

Exemplu:

Graf complet cu 5 noduri :

Proprietăți:

1. d(1)=d(2)=...=d(n)=n-1;
2. Graful Kn are $\frac{n*(n-1)}{2}$ muchii;
3. Dintre toate grafurile cu *n* noduri, Kn are numărul maxim de muchii.

Aplicație: Să se verifice dacă există un graf cu 100 de noduri și 4951 de muchii.

Rezolvare:

Un graf complet cu 100 de noduri are 4950 de muchii; un graf complet are numărul maxim de muchii dintre toate grafurile cu 100 de noduri; rezultă că nu există niciun graf cu 100 de noduri și 4951 de muchii.

3.2. Grafuri parțiale

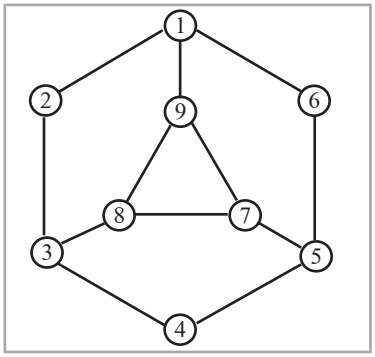
Definiție:

Fie graful G=(X,U). Un graf parțial al lui G este un graf G1=(X,V), unde V⊆U.

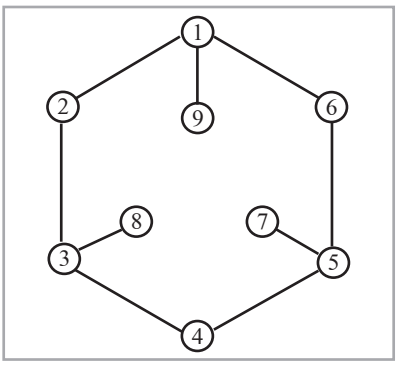
Se observă că G1 se obține din G păstrând toate nodurile și eliminând un număr de muchii.

Exemplu :

G=(X,U)



G1=(X,V)-graf parțial al lui G



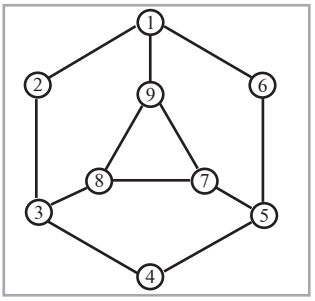
Din graful neorientat G au fost eliminate muchiile [7 ,8], [7,9], [8,9] și a rezultat graful parțial G1.

3.3. Subgrafuri

Definiție:

Fie graful G=(X,U). Un subgraf al lui G este un graf G1=(X1,U1), unde X1⊂X și U1⊂U, iar U1 va conține numai muchiile care au ambele extremități în X1.

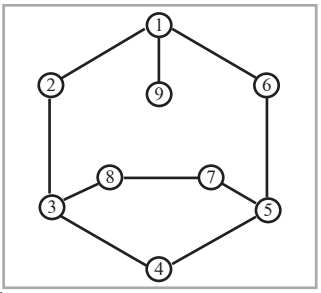
Se observă că G1 se obține din G eliminând unele noduri și păstrând doar acele muchii care au ambele extremități în mulțimea nodurilor rămase.



G=(X,U)

X={1,2,3,4,5,6,7,8,9}

G1=(X1,U1)



X1={1,2,3,4,5,6,7,8}

Se observă că din mulțimea X a fost eliminat nodul 9. Mulțimea U1 va conține aceleași elemente ca și U mai puțin muchiile [9,1], [9,8] și [9,7].

TEME

1. Fie G și G1 două grafuri neorientate ce au matricele de adiacență A, respectiv A1.

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} \qquad A1 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Stabiliți valoarea de adevăr pentru următoarele afirmații :
a) G este un graf complet cu 5 noduri.
b) G1 este subgraf al grafului G.
c) G1 este graf parțial pentru graful G.
d) Dacă la graful G1 se adaugă încă 4 muchii, acesta va deveni graf complet.

2. Fie G un graf ce conține patru noduri și zero muchii. Câte muchii trebuie adăugate grafului G, asfel încât acesta să devină graf complet?

3. Stabiliți valoarea de adevăr a următoarelor afirmații :
Un graf este complet dacă:
a) Fiecare nod are cel puțin o muchie adiacentă.
b) Are n*(n-1)/2 muchii (n reprezintă numărul de noduri).
c) Gradele tuturor nodurilor sale sunt pare.
d) Are n*(n+1)/2 muchii (n reprezintă numărul de noduri).

4. Fie G=(X,U) un graf neorientat.
Stabiliți valoarea de adevăr a următoarelor afirmații :
a) Un graf parțial se obține prin suprimarea unor noduri.
b) Un subgraf se obține prin suprimarea unor muchii.
c) Orice subgraf al lui G este graf parțial al grafului G.

5. Câte noduri are graful complet cu 15 muchii?
a) 10; b) 5; c) 6; d) 12.

6. Fie G=(X,U) un graf neorientat reprezentat prin listele de adiacență:
Lista vecinilor lui 1: 3, 4.
Lista vecinilor lui 2: 3, 4.
Lista vecinilor lui 3: 1, 2.
Lista vecinilor lui 4: 1, 2.
Câte muchii trebuie adăugate la graful G astfel încât acesta să devină graf complet?

7. Fie G un graf neorientat cu *n* noduri și *m* muchii și G1 un graf neorientat cu *n1* noduri și *m1* muchii. Se citesc de la tastatura *m* perechi de numere întregi reprezentând

extremitățile muchiilor grafului G și *m1* perechi de numere întregi reprezentând extremitățile muchiilor grafului G1. Să se verifice dacă G1 este graf parțial al lui G.

8. Din cauza unor probleme provocate de vremea nefavorabilă, mai multe relee de amplificare și retransmitere a semnalului pentru o telefonie mobilă au fost avariate. Cunoscându-se legăturile inițiale de transmitere a semnalului și releele avariate, să se determine care sunt legăturile de transmitere a semnalului care încă mai funcționează.

4. PARCURGEREA GRAFURILOR NEORIENTATE

Prin parcurgerea sau traversarea unui graf neorientat se urmărește examinarea nodurilor sale, plecând dintr-un nod dat *x*, astfel încât fiecare nod, la care se poate ajunge din x pe muchii adiacente două câte două, să fie marcat o singură dată.

4.1. Metoda de parcurgere Breadth First (BF) – parcurgerea în lățime

Fie G = (X,U) un graf neorientat. Algoritmul BF realizează o traversare „în lățime" a grafului: se marchează nodul inițial *x*, apoi vecinii acestuia, apoi vecinii nevizitați ai acestora și așa mai departe.

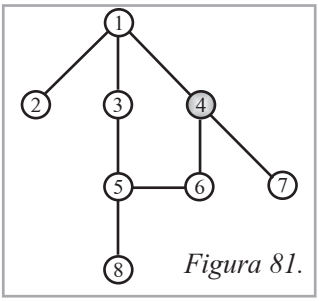


Figura 81.

Exemplul 1:

Pentru graful din figura 81, dacă nodul de start este 1, ordinea de parcurgere a nodurilor va fi: 1, 2, 3, 4, 5, 6, 7, 8.

Dacă nodul de start este 4, atunci prin metoda BF se obține următoarea ordine de parcurgere a grafului: 4, 1, 6, 7, 2, 3, 5, 8.

Algoritmul folosește o structură de tip *coadă* (fir de așteptare), *c,* în care se introduc pe rând nodurile vizitate și un vector *vizitat* definit astfel:

$$\text{vizitat}[i] = \begin{cases} 1, \text{dacă nodul } i \text{ a fost vizitat} \\ 0, \text{altfel} \end{cases}$$

Raționamentul metodei BF

Inițial, vectorul vizitat are toate valorile nule (nu a fost vizitat niciun nod).

Nodul de start este 4 și se memorează în coadă pe poziția 1.

	1	2	3	4	5	6	7	8
vizitat=	0	0	0	0	0	0	0	0

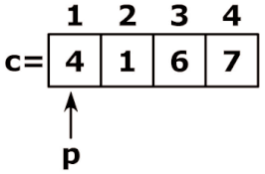
Se marchează nodul 4 ca fiind vizitat.

	1
c=	4
	p

	1	2	3	4	5	6	7	8
vizitat=	0	0	0	1	0	0	0	0

p = marcator ce indică nodul ai cărui vecini nevizitați urmează să fie memorați în coadă.

Se memorează în coadă, pe pozițiile următoare, vecinii nodului 4, adică 1, 6, 7.



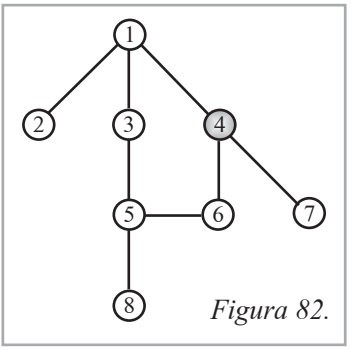
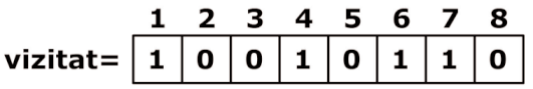


Figura 82.

Se marchează nodurile 1, 6 și 7 ca fiind vizitate.



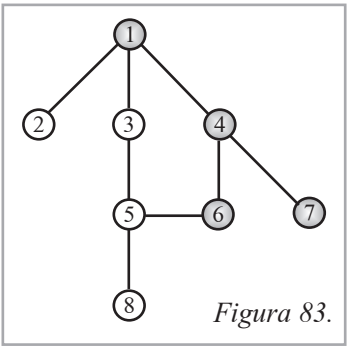
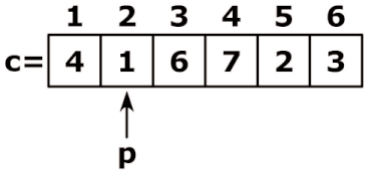


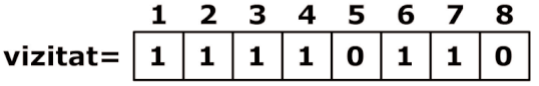
Figura 83.

Marcatorul *p* se deplasează cu o poziție în coadă.

Se memorează în coadă, pe pozițiile următoare, vecinii nevizitați ai nodului 1, adică nodurile 2 și 3:



Se marchează nodurile 2 și 3 ca fiind vizitate:



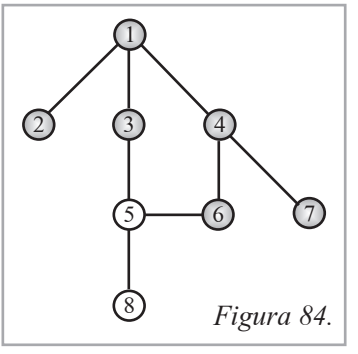
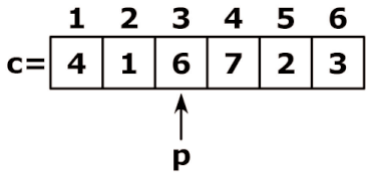
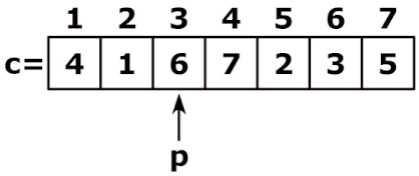


Figura 84.

Marcatorul p se deplasează cu o poziție în coadă:



Se memorează în coadă, pe pozițiile următoare, vecinul nevizitat al nodului 6, adică 5:



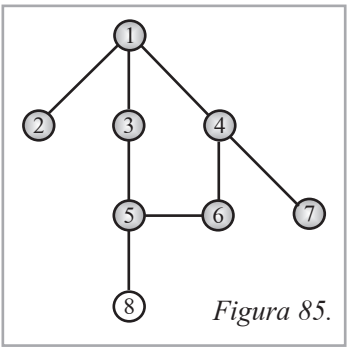
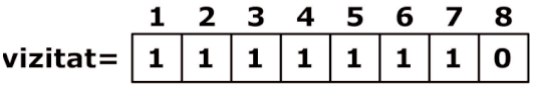
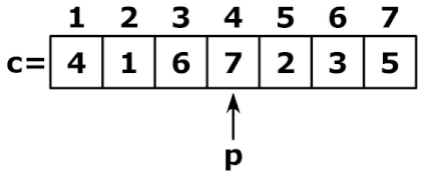


Figura 85.

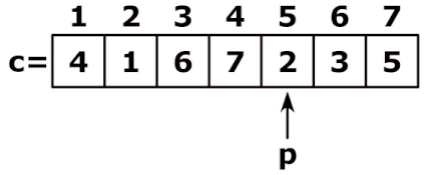
Se marchează nodul 5 ca fiind vizitat.



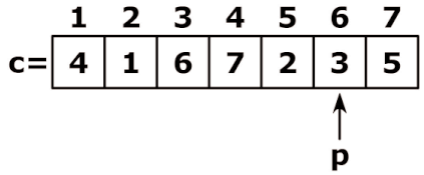
Marcatorul *p* se deplasează cu o poziție în coadă:



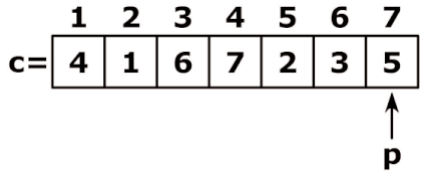
Nodul 7 are toți vecinii vizitați.
Marcatorul *p* se deplasează cu o poziție în coadă:



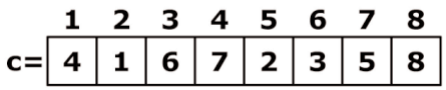
Nodul 2 are toți vecinii vizitați.
Marcatorul *p* se deplasează cu o poziție în coadă:



Nodul 3 are toți vecinii vizitați. Marcatorul *p* se deplasează cu o poziție în coadă:



Se memorează în coadă, pe poziția următoare, vecinul nevizitat al nodului 5, adică 8:



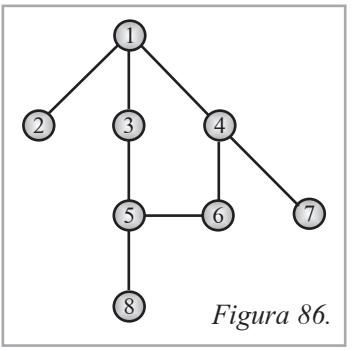
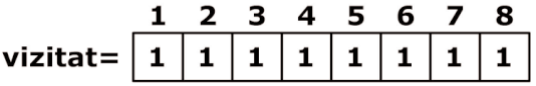
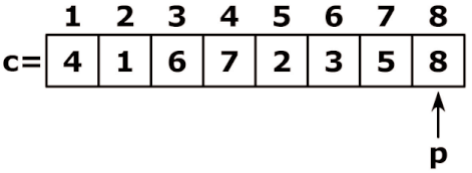


Figura 86.

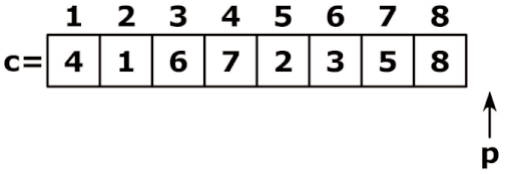
Se marchează nodul 8 ca fiind vizitat.



Marcatorul *p* se deplasează cu o poziție în coadă:



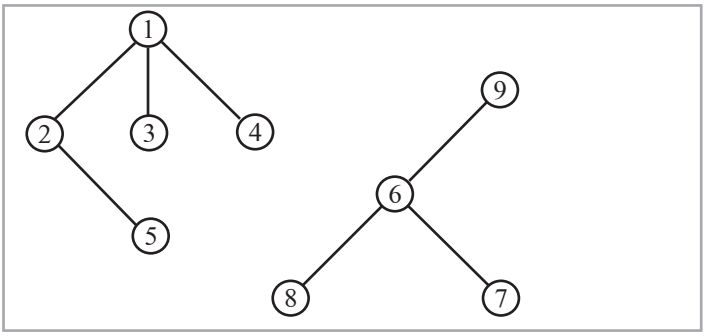
Nodul 8 are toți vecinii vizitați.
Marcatorul *p* se deplasează cu o poziție în coadă:



Au fost vizitate (marcate) toate nodurile.

Figura 87.

Exemplul 2:
Fie graful G = (X,U):



Dacă nodul de start este 2, atunci ordinea de parcurgere a nodurilor grafului în parcurgerea BF este 1, 5, 3, 4.

Observație:

Nodurile 6, 7, 8, 9 nu vor fi atinse.

Reprezentarea algoritmului BF în pseudocod

```
pentru j = 1 la n execută
                vizitat[j] ← 0
sfârșit pentru;
citește i;
c[1] ← i;
/*se memorează în coadă nodul i pe prima poziție*/
p ← 1 /*p reprezintă poziția elementului cheie*/
u ← 1 /*u reprezintă poziția ultimului element din coadă*/
vizitat[i] ← 1;
cât timp  p ≤ u execută
                /*cât timp coada nu e vidă*/
                x ← c[p];
        /*x este elementul cheie, adică nodul ai cărui succesori nevizitați vor fi memorați
        în coadă*/
                pentru toți j ∈ Listei vecinilor lui x nevizitați încă execută
                        /*pentru toți vecinii nodului x nevizitați încă*/
                        u ← u+1;
                        c[u] ← j;
        /*se memorează în coada vecinul j nevizitat încă*/



                        prelucrare (j);
                        /*de regulă se face afișarea nodului j*/
                        vizitat[j] ← 1;
                sfârșit pentru;
                p ← p+1;
                /*se alege următorul nod ca fiind nod cheie*/
sfârșit cât timp;
```

Ordinul de complexitate al algoritmului de parcurgere "în lățime" a grafurilor neorientate este $0(n^2)$ deoarece instrucțiunile din ciclul *atâta timp cât* se execută de cel mult n-1 ori, iar cele interioare, structura *pentru* se execută de cel mult n ori.

TEME

1. Alcătuiți lista nodurilor în ordinea dată de vizitarea lor prin metoda parcurgerii BF(Breadth First), pentru grafurile din figura 88 și figura 89:

Figura 88.

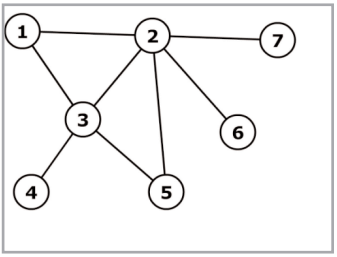
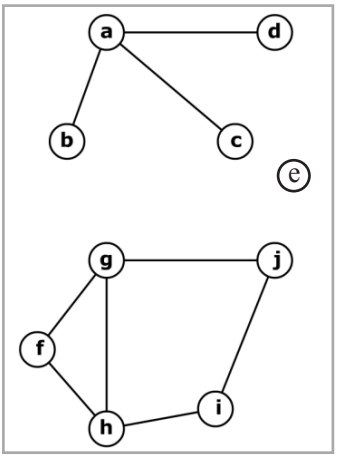
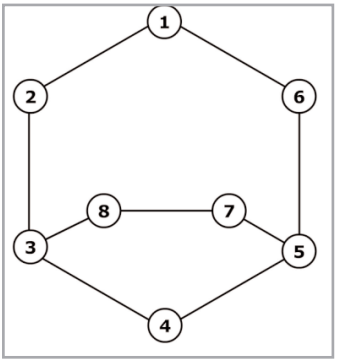


Figura 89.



2. Parcurgeți graful următor în lățime (BF), completați vectorul *vizitat* și *coada* (firul de așteptare) cu nodurile vizitate, dar ai căror vecini nu au fost vizitați încă.

Figura 90.



3. Implementați în limbajul de programare studiat algoritmul BF.

4. Justificați necesitatea tabloului de tip *coadă* pentru implementarea metodei BF.

5. Dezvoltați un algoritm de parcurgere a unui graf în lățime (BF) care, testat și pe graful de la problema 3, să viziteze toate nodurile.

6. Dezvoltați un algoritm recursiv de parcurgere în lățime a unui graf.

4.2. Metoda de parcurgere Depth First (DF) – parcurgerea în adâncime

Fie G = (X,U) un graf neorientat unde *n* reprezintă numărul de noduri ale sale.

Algoritmul DF realizează o parcurgere "*în adâncime*" a grafului: se pornește de la un nod **x** și se atinge primul dintre vecinii nevizitați ai acestuia; algoritmul continuă până se ajunge la un nod ce are toți vecinii vizitați. În acest moment, se revine la nodul anterior și se caută un nod nevizitat.

Exemplul 1:

Fie G = (X,U), un graf neorientat. (Figura 91)

Pentru graful dat, ordinea de parcurgere a nodurilor prin metoda DF, pornind din nodul unu, este: 1, 2, 3, 5, 6, 4, 7, 8.

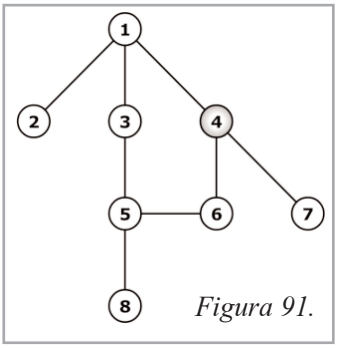


Figura 91.

Dacă nodul de start este 4, atunci prin metoda DF se obține următoarea ordine de parcurgere a grafului:

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

Algoritmul folosește o structură de tip stivă în care va introduce (sau elimina) nodurile vizitate din graf și un vector cu **n** componente, pentru marcarea nodurilor vizitate definit astfel:

$$\text{vizitat}[i] = \begin{cases} 1, & \text{dacă nodul } \mathbf{i} \text{ a fost vizitat;} \\ 0, & \text{altfel .} \end{cases}$$

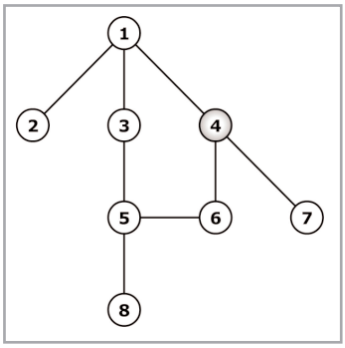
Raționamentul metodei DF

Inițial vectorul vizitat are toate valorile nule:

	1	2	3	4	5	6	7	8
vizitat=	0	0	0	0	0	0	0	0

Se introduce în stivă nodul de start. Fie acesta nodul 4.
Se afișează nodul 4 și se marchează ca fiind vizitat.

Figura 92.



S:

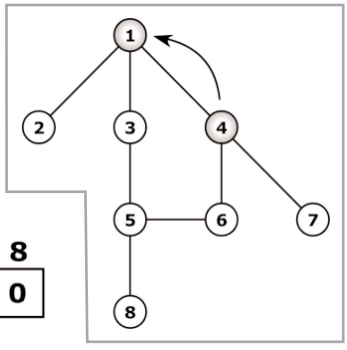
ps → 4

	1	2	3	4	5	6	7	8
vizitat=	0	0	0	1	0	0	0	0

Se determină primul dintre vecinii nodului din vârful stivei, se mărește cu o unitate vârful stivei și pe această poziție se înregistrează nodul 1.

Se afișează nodul 1 și se marchează ca fiind vizitat.

Figura 93.



S:

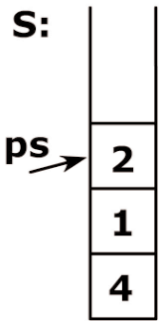
ps → 1

4

	1	2	3	4	5	6	7	8
vizitat=	1	0	0	1	0	0	0	0

Se determină primul dintre vecinii nevizitați ai nodului 1; acesta este nodul 2, se mărește cu o unitate vârful stivei (stiva crește); pe noua poziție din stivă se înregistrează nodul 2.

Se afișează nodul 2 și se marchează nodul 2 ca fiind vizitat.



Se încearcă determinarea unui vecin nevizitat al nodului 2. Singurul vecin al nodului 2 este nodul 1 care a fost deja vizitat. Se extrage din stivă nodul 2 și se micșorează cu o unitate vârful stivei.

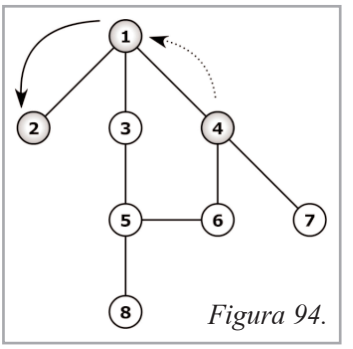


Figura 94.

	1	2	3	4	5	6	7	8
vizitat=	1	1	0	1	0	0	0	0

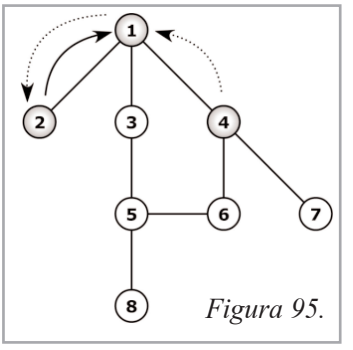
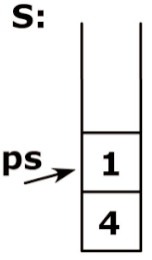
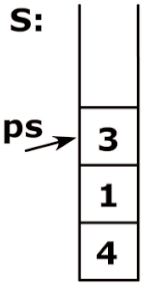


Figura 95.



Se determină primul dintre vecinii nevizitați ai nodului 1; acesta este nodul 3 (nodul 2 a fost deja vizitat); se mărește cu o unitate vârful stivei și pe noua poziție se înregistrează nodul 3.

Se afișează nodul 3 și se marchează nodul 3 ca fiind vizitat.



Se determină primul dintre vecinii nevizitați ai nodului 3; acesta este nodul 5, se mărește cu o unitate vârful stivei și pe noua poziție din stivă se înregistrează nodul 5.

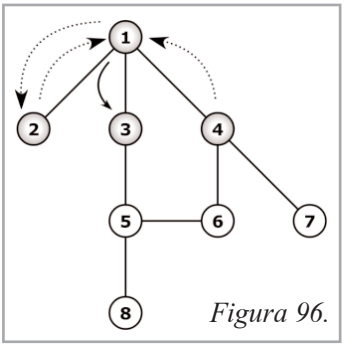


Figura 96.

	1	2	3	4	5	6	7	8
vizitat=	1	1	1	1	0	0	0	0

Se afișează nodul 5 și se marchează ca fiind vizitat.

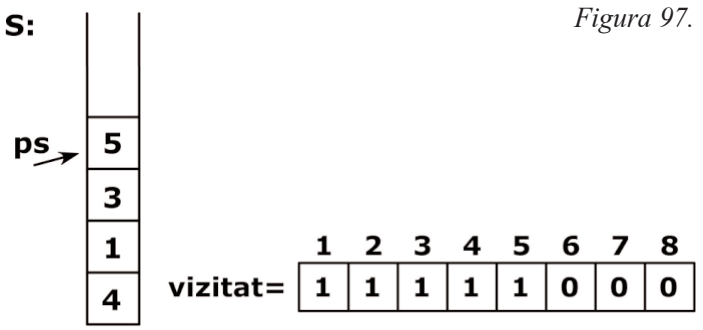


Figura 97.

Se determină primul dintre vecinii nevizitați ai nodului 5; acesta este nodul 6, se mărește cu o unitate vârful stivei și pe noua poziție din stivă se înregistrează nodul 6.

Se afișează nodul 8 și se marchează ca fiind vizitat. Algoritmul continuă până când stiva devine vidă.

Reprezentarea în pseudocod a algoritmului DF de parcurgere a grafurilor

```
pentru i = 1 la n execută
        vizitat [i] ← 0;
sfârșit pentru;
ps ← 1;
citește x;
s[ps] ← x;
vizitat[x] ← 1;
scrie x;
/*se memorează în stivă pe poziția 1 nodul de plecare, se marchează ca fiind vizitat și se prelucrează informația atașată – afișare în acest caz*/
cât timp ps > 0 execută
        /*cât timp stiva nu este vidă se încearcă determinarea primului nod vecin cu nodul indicat de ps din stivă și care nu e vizitat încă*/
        găsit ← false;
        i ← 2;
    cât timp (i <= n) AND (găsit = false) execută
            dacă (A[s[ps],i] = 1) și (vizitat[i] = 0) atunci
                    găsit ← true
                    altfel i ← i+1
            sfârșit dacă
    sfârșit cât timp;
    dacă găsit = true atunci
                scrie(i);
                vizitat[i] ← 1;
                ps ← ps+1;
                s[ps] ← i;
                altfel ps ← ps-1;
    sfârșit dacă
sfârșit cât timp
```

TEME

1. Alcătuiți lista nodurilor în ordinea dată de vizitarea lor prin metoda parcurgerii DF, pentru grafurile din figura 98 și din figura 99.

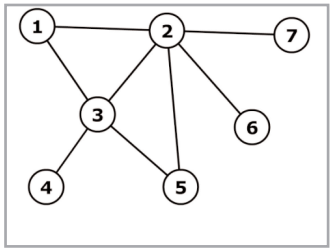


Figura 98.

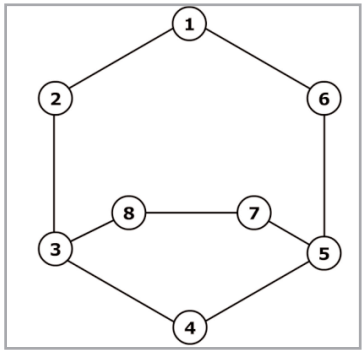


Figura 99.

2. Parcurgeți graful din figura 100 în adâncime (DF) și completați vectorul *vizitat* și stiva nodurilor vizitate, dar ale căror vecini nu au fost vizitați încă.

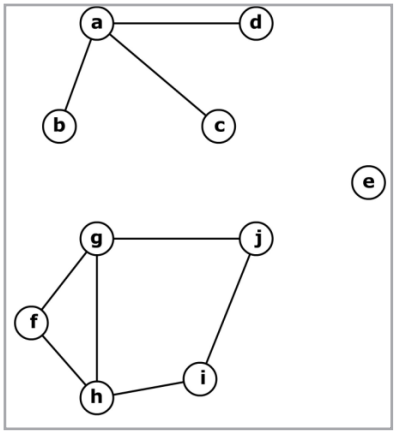


Figura 100.

3. Implementați în limbajul de programare studiat algoritmul DF.

4. Justificați necesitatea tabloului de tip stivă pentru implementarea metodei DF.

5. Descrieți metoda de programare care stă la baza parcurgerii DF.

6. Dezvoltați un algoritm de parcurgere a unui graf în adâncime (DF) care, testat și pe graful de la problema 2, să viziteze toate nodurile.

7. Dezvoltați un algoritm recursiv de parcurgere în adâncime a unui graf.

5. NOȚIUNEA DE CONEXITATE ÎN GRAFURI NEORIENTATE

Studiu de caz: Traseu turistic.

Pe harta unei regiuni apar puncte turistice legate prin drumuri ca în figura 101. Dacă un grup de turiști se află în punctul turistic A, se cere să se determine toate posibilitățile de constituire de trasee turistice care să îi conducă în punctul turistic B, fără să viziteze de două ori același punct turistic.

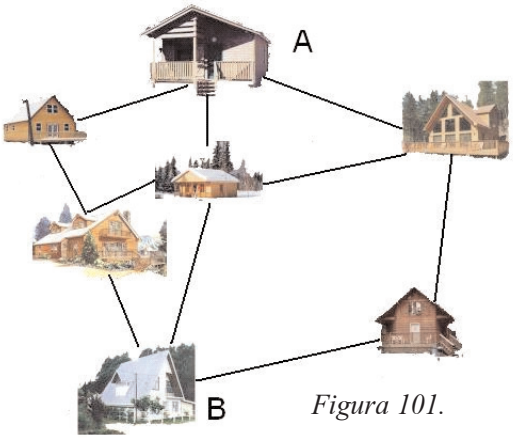


Figura 101.

Graful asociat acestei probleme:

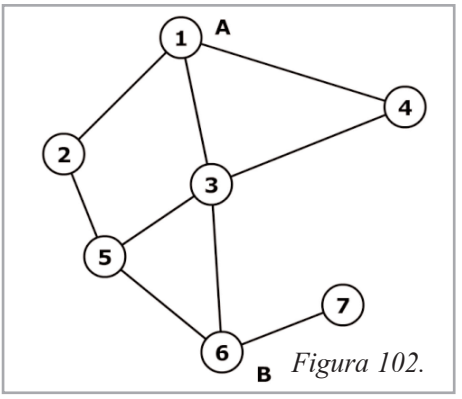
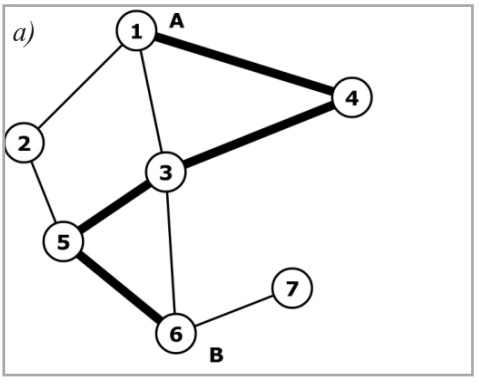
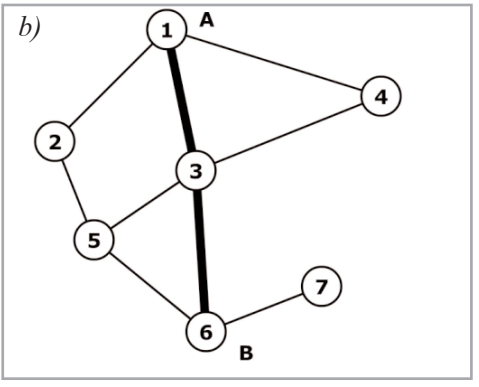


Figura 102.

Figura 103.

Exemple de trasee:





Definiție :

Fie G=(X,U) un graf neorientat. Se numește *lanț* în graful G o succesiune de noduri $L=(x_1, x_2, x_3 \ldots, x_k)$, unde $x_1, x_2, x_3 \ldots, x_k \in X$ cu proprietatea că oricare două noduri consecutive sunt adiacente ($[x_i, x_{i+1}] \in U$ unde i=1, k-1) și muchiile sunt distincte două câte două.

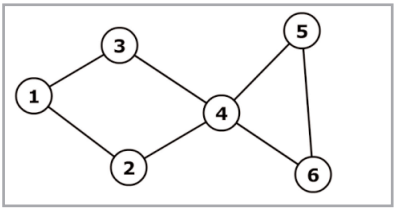
Lungimea lanțului este egală cu numărul de muchii componente.

Exemple:

1. Pentru graful G=(X,U), unde
X={A,B,C,D,E,F}
U={[A,C];[A,E];[B,C];[B,D];[B,F];[C,D];[E,F]}
există următoarele exemple de lanțuri :
L=(A,C,D,B) de lungime 3;
L=(C,D,B,C,A) de lungime 4;

2. Fie G=(X ,U) (figura 104):
L=(3, 4, 6, 5, 4, 2) este un lanț de lungimea 5.

Figura 104.



Definiție:

Se numește *lanț elementar* un lanț în care nodurile componente sunt distincte două câte două.

Exemplu:

Fie graful din figura 105.
L=(b,d,e,f) este lanț elementar.
L=(a,b,f,d,b,e) este un lanț neelementar.

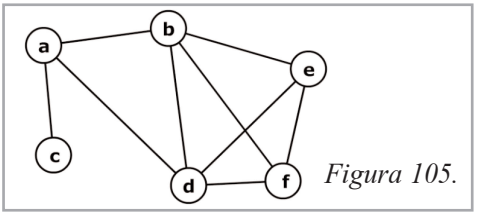


Figura 105.

Definiție:

Se numește ciclu într-un graf un lanț $L=(x_1, x_2, x_3, \ldots, x_k)$, unde $x_1, x_2, x_3, \ldots, x_k \in X$ cu proprietatea că $x_1 = x_k$ și muchiile componente sunt distincte două câte două.

Definiție:

Un *ciclu elementar* este un *ciclu* în care toate nodurile, cu excepția primului și a ultimului, sunt distincte două câte două.

Exemplu :

Fie G=(X ,U) un graf neorientat ce are reprezentarea din figura 106.
L = (3,4,5,6,3) este ciclu elementar
L = (1,2,3,4,5,6,3,1) este ciclu neelementar

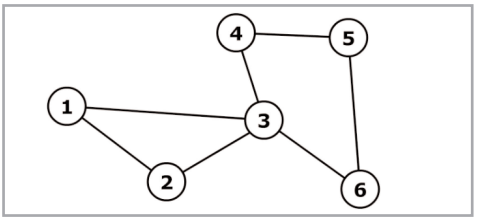


Figura 106.

Definiție:

Fie G=(U,V) un graf neorientat. G este un graf *conex* dacă, oricare ar fi două noduri ale sale x și y, există cel puțin un lanț prin care se poate ajunge de la nodul x la nodul y.

Exemple:

1. Fie G=(X,U) un graf neorientat (figura 107):

Se observă că, pentru oricare două noduri, există un lanț care le unește.

Prin parcurgerea acestui graf, se observă că vor fi marcate toate nodurile: există lanțuri ce unesc oricare două noduri din acest graf.

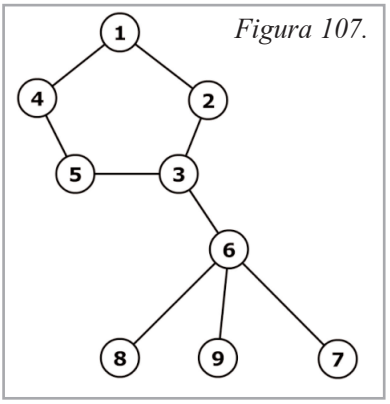


Figura 107.

Observații:

• La parcurgerea BF a grafului, ordinea de parcurgere a nodurilor este 1,2,4,3,6,5,7,8,9 și vectorul vizitat va avea toate componentele completate cu valoarea 1.

• La parcurgerea DF a grafului ordinea de parcugere a nodurilor este: 1, 2, 3, 5, 4, 6, 7, 8, 9, iar vectorul vizitat va avea toate nodurile completate cu valoarea 1.

2. Fie graful G=(X,U) din figura 108:

Prin parcurgerea DF pornind din nodul 3, se obține următoarea ordine a nodurilor: 3,1,4,5,8. Vectorul vizitat este:

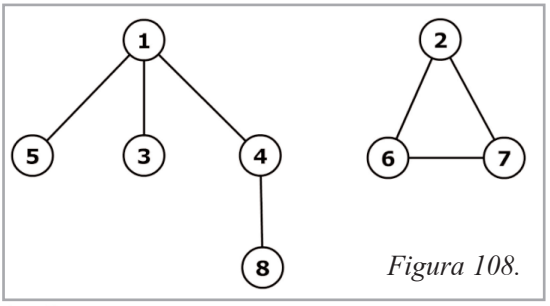


Figura 108.

	1	2	3	4	5	6	7	8
vizitat=	1	0	1	1	1	0	0	1

Concluzie

Se observă că, nodurile 2, 6, 7 nu au fost vizitate, graful nu este graf conex.

Dacă se aplică acestui graf un algoritm de parcurgere BF pornind din nodul 3, atunci ordinea de parcurgere a nodurilor sale este: 3, 1, 4, 8, 5. Vectorul vizitat este:

	1	2	3	4	5	6	7	8
vizitat=	1	0	1	1	1	0	0	1

Nodurile 2, 6 și 7 nu au fost vizitate și nu se poate ajunge la ele pornind din nodul 3 și mergând pe muchii adiacente.

Concluzie

Graful G nu este graf conex.

Dacă se reia parcurgerea grafului pornind dintr-un nod care nu a fost *vizitat* se obține, pentru o parcurgere BF, ordinea 2, 6, 7; Vectorul vizitat este:

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

Pentru a putea fi marcate toate nodurile, au fost necesare două parcurgeri ale grafului G.

Definiție:

Se numește *componenta conexă* a grafului neorientat G=(X,U) un *subgraf conex* $G_1 = (X_1,U_1)$ a lui G, cu proprietatea că nu există niciun lanț care să lege un nod din mulțimea X cu un nod din mulțimea complementară $X-X_1$.

Pentru graful G=(X,U) din figura 109:
Se obțin subgrafurile:
$G_1 =(X_1,U_1)$,
cu $X_1 =\{1,3,4,5,8\}$
$U_1 =\{[1,3];[1,4];[1,5];[4,8]\}$
și
$G_2=(X_2,U_2)$,
cu $X_2=\{2,6,7\}$
$U_2=\{[2,6];[2,7];[6,7]\}$

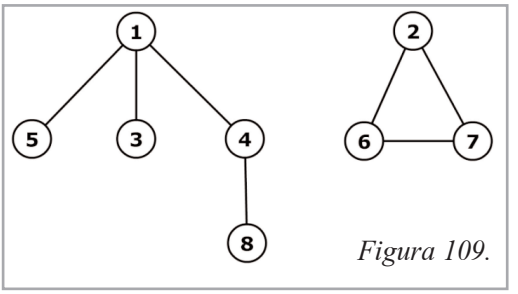


Figura 109.

Observații:

• G_1, G_2 sunt grafuri conexe deoarece există cel puțin un lanț între oricare două noduri din G_1 respectiv G_2.

• Pentru $X_1 =\{1,3,4,5,8\}$, obținem mulțimea complementară

$X- X_1=\{2,6,7\}$; se observă că nu există niciun lanț care să lege un nod din X_1 cu un nod din $X-X_1$.

• Pentru $X_2=\{2,6,7\}$ obținem mulțimea complementară $X-X_2=\{1,3,4,5,8\}$ și se observă că nu există niciun lanț care să lege un nod din X_2 cu un nod din $X-X_2$.

Concluzie

G_1 și G_2 sunt componente conexe pentru graful G.

Observație:

Un graf conex are o singură componentă conexă.

Reprezentarea în pseudocod a algoritmului de verificare a conexității unui graf neorientat.

```
pentru  j  = 2 la n execută
    vizitat [j]  ← 0;
sfârșit pentru
c[1] ← 1;
vizitat [1] ← 1;
p ← 1;
u ← 1;
cât timp  p ≤ u execută
    pentru toti j ∈ L_x nevizitati execută
                u ← u+1;
                c[u] ← j;
                vizitat[j] ← 1;
    sfârșit pentru;
    p ← p+1;
sfârșit cât timp;
conex ← true;
i ← 1;
cât timp  (conex=true) AND (i<=n) execută
    dacă vizitat[i] = 0 atunci
                conex ← false
                altfel i ← i+1;
    sfârșit dacă
sfârșit cât timp
dacă conex=true atunci
    scrie ('Graful este conex.');
    altfel scrie ('Graful nu este conex.')
sfârșit dacă;
```

Arhipelag – aplicație conexitate

*Harta unui **arhipelag** este codificată printr-o matrice binară pătrată unde suprafețele terestre sunt marcate cu 1 și cele marine cu 0. Să se determine câte insule fac parte din arhipelag. O insulă este compusă din unul sau mai multe elemente ale matricei marcate cu 1 care se învecinează pe direcțiile N, S, E, V.*

Date de intrare: se citesc din fișierul "Date.in":
Pe prima linie: **n** = dimensiunea matricei pătratice.
Pe următoarele linii, elementele matricei.
Date de ieșire: **nr** (numărul de insule).

Exemplu:

Date.in

```
5
1 1 0 1 1
0 0 0 0 1
0 0 0 1 0
1 0 0 0 0
1 0 1 1 1
```

Date de ieşire:

```
5
```

Varianta Pascal

```
program insule;
type matrice=array[0..100,0..100] of byte;
var A:matrice;
      n,i,j:byte
procedure citire;
var f:text;
begin
assign(f,'date.in');
reset(f);
readln(f,n);
for i:=1 to n do
for j:=1 to n do
read(f,A[i,j]);
close(f);
end;
procedure df(x,y:integer);
begin
A[x,y]:=2;
if A[x-1,y]=1 then df(x-1,y);
if A[x+1,y]=1 then df(x+1,y);
if A[x,y-1]=1 then df(x,y-1);
if A[x,y+1]=1 then df(x,y+1);
end;
begin
citire;
for i:=0 to n+1 do
   begin A[0, i]:=0; A[n+1, i]:=0 end;
for i:=1 to n do
   begin A[i, 0]:=0; A[i, n+1]:=0 end;
nr:=0;
for i:=1 to n do
for j:=1 to n do
if A[i,j]=1 then begin
nr:=nr+1;
df(i,j);
end;
write('numarul de insule este: ',nr);
readln;
end.
```

Varianta C++

```
#include <fstream.h>
int n,a[100][100];
void citire() {
      int i,j;
      fstream f("date.in",ios::in);
      f>>n
      from (i=1; i<=n; i++)
            for(j=1;j<=n;j++)
                  f>>a[i,j];
      f.close();
}
void df(int x, int y){
      int i;
      a[x][y]=2;
      if (a[x-1][y])
            df(x-1,y);
      if (a[x+1][y])
            df(x+1,y);
      if (a[x][y-1])
            df(x,y-1);
      if (a[x][y+1])
            df(x,y+1);
}
void main() {
      int i,j,nr=0;
      citire();
      for (i=0; i<n=1; i++)
      {A[0][i]=0; A[n+1][i]=0};
       for (i=0; i≤n+1; i++)
      {A[i][0]=0; A[i][n+1]=0};
      for (i=1;i<=n;i++)
            for (j=1;j<=n;j++)
                  if (a[i][j]==1){
                        nr++;
                        df(i,j);
                  }
      cont<<"nr. de insule este"<<nr<<endl;
}
```

TEME

1. Fiind dat un graf G=(X,U), să se determine un lanț elementar al acestuia de lungime maximă.

2. Să se determine dacă într-un graf cu **n** noduri și **m** muchii există un ciclu care să conțină un nod oarecare **k** introdus de la tastatură. Elementele geafului se citesc din fișierul *graf.in* cu următoarea structură: pe prima linie, valorile pentru **n** și **m**; pe următoarele **m** linii, câte două valori reprezentând extremitățile unei muchii.

3. Să se verifice dacă un graf este *hamiltonian*- conține un *circuit hamiltonian*: circuit elementar care trece prin toate nodurile grafului.

4. Implementați în limbajul studiat algoritmul de verificare a conexității unui graf.

5. Explicați metoda de verificare a conexității folosită în algoritmul prezentat.

6. Formulați exemple de probleme care să conducă la determinarea conexității.

7. Într-un grup de persoane se cunosc relațiile de prietenie. Să se determine cercurile de persoane cu proprietatea că între oricare două persoane din același cerc sunt relații de prietenie (aplicând „principiul «prietenul prietenului meu este prietenul meu»”) și numărul minim de relații de prietenie (precum și prieteniile respective) care ar trebui realizate pentru a exista un singur cerc de prieteni.

8. În fișierul „Date.in”, pe prima linie, se află numărul de noduri ***n*** ale unui graf neorientat G=(X,U) , pe următoarele *n* linii, matricea de adiacență atașată grafului, iar pe ultima linie două noduri (*v1* și *v2*). Să se determine lanțurile elementare de lungime minimă existente între nodurile v1 și v2.

6. GRAFURI ORIENTATE

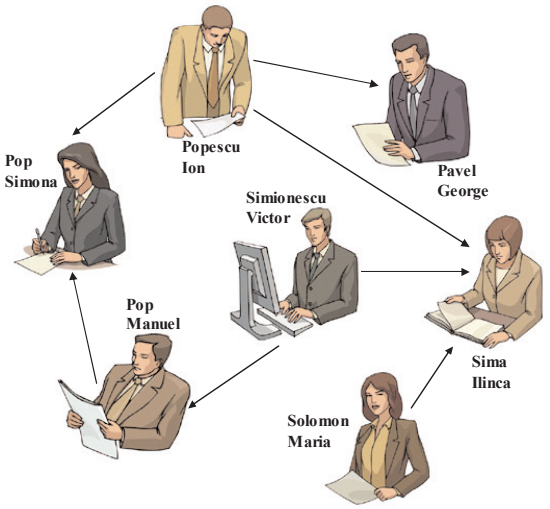
Structura unui graf poate fi extinsă prin asocierea unei funcții pentru fiecare muchie a grafului. Grafurile cu funcții pot fi folosite pentru reprezentarea a diferite situații; de exemplu: dacă un graf reprezintă o rețea de drumuri, funcția poate reprezenta lungimea fiecărui drum.

Alt mod de a extinde un graf simplu este asocierea unei direcții muchiilor grafului (A e legat de B, dar B nu e neapărat legat de A, ca într-o pagină web), un astfel de graf se numește *graf orientat* sau *digraf*.

Simpozion. Studiu de caz

La un simpozion pe teme de IT se întâlnesc profesori din diverse județe ale țării. În grupul respectiv există profesori ce sunt cunoscuți de alți participanți la simpozion. Să se găsească profesorul (profesorii) cel mai cunoscut din grup și profesorul (profesorii) ce nu este cunoscut de niciun alt membru al grupului.

Exemplu:



Interpretarea reprezentării grafice:

- Domnul Popescu Ion îl cunoaşte pe domnul Pavel George, pe doamna Pop Simona şi pe doamna Sima Ilinca.
- Domnul Simionescu Victor îl cunoaşte pe domnul Pop Manuel şi o cunoaşte pe doamna Sima Ilinca.
- Doamna Solomon Maria o cunoaşte pe doamna Sima Ilinca.
- Domnul Pop Manuel o cunoaşte pe doamna Pop Simona.

Observaţii:

Doamna Sima Ilinca este cea mai cunoscută persoană din grup.

Domnii Popescu Ion şi Simionescu Victor şi doamna Solomon Maria nu sunt cunoscuţi de niciun alt membru al grupului.

Concluzie:

Persoanele şi relaţiile dintre ele formează un graf orientat.

Definiţie

Se numeşte **graf orientat** o pereche ordonată de mulţimi G=(X,U), unde:

X este o mulţime finită şi nevidă de elemente numite vârfuri.

U este o mulţime formată din perechi ordonate de elemente ale lui X, numită mulţimea arcelor.

Observaţii:

- Arcul [x,y] este diferit de arcul [y,x].
- Pentru fiecare arc u=[x,y], x reprezintă extremitatea iniţială, iar y reprezintă extremitatea finală a arcului.
- Se spune că „arcul iese din vârful x şi intră în vârful y”.
- Vârfurile x şi y sunt adiacente.

- Arcul u este incident cu vârful x (respectiv y).
- Vârful y se numeşte succesor al vârfului x.
- Vârful x se numeşte predecesor al vârfului y.
- Dacă G=(X,U) este un graf orientat şi U este o mulţime nevidă, atunci spunem că G este graf nul şi reprezentarea lui în plan se reduce la puncte izolate.

Exemplul 1:
Graful asociat problemei din studiul de caz este:
G=(X,U)
X={1,2,3,4,5,6,7}
U={[1,2];[1,3];[1,7];[4,3];[4,6];[5,3];[6,7]}

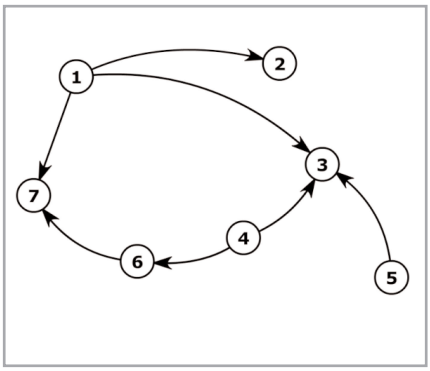


Figura 110

TEME

1. Formulaţi un exemplu de situaţie care să poată fi modelată prin graful din figura 111 (explicaţi semnificaţia vârfurilor şi a arcelor).

Fie $G_1=(X_1,U_1)$ unde

$X_1=\{1,2,3,4,5,6,7,8\}$

$U_1=\{[1,6];[2,3];[2,7];[3,2];[3,4];[4,7];[4,5];[7,2];[7,6];[8,5];[8,6]\}$

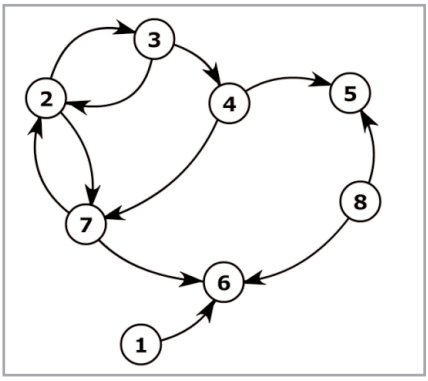


Figura 111.

2. Se cunosc **n** localităţi şi **m** drumuri între aceste localităţi.
a) Formulaţi un exemplu de graf neorientat care să aibă la bază aceste date.
b) Formulaţi un exemplu de graf orientat care să aibă la bază aceste date.

Gradul unui vârf

Definiție

Fie G=(X,U) un graf orientat și x ∈ X. Se numește *grad exterior* vârfului x numărul arcelor care ies din vârful x, adică numărul arcelor de tipul [x,y] ∈ U, y ∈ X.

Gradul exterior se notează cu $d^+(x)$,

Se numește *grad interior* al vârfului x numărul arcelor care intră în vârful x, adică numărul arcelor de tipul [y,x] ∈ U, y ∈ X. Gradul interior se notează $d^-(x)$,

Exemple

Fie graful din figura 112.

Se determină gradele vârfurilor:

$d^+(1)=3$; $d^-(1)=0$
$d^+(2)=0$; $d^-(2)=1$
$d^+(3)=0$; $d^-(3)=3$
$d^+(4)=2$; $d^-(4)=0$
$d^+(5)=1$; $d^-(5)=0$
$d^+(6)=1$; $d^-(6)=1$
$d^+(7)=0$; $d^-(7)=2$

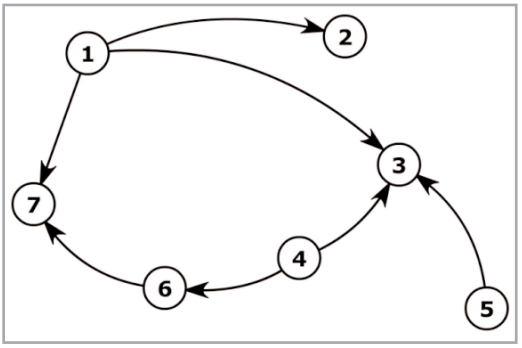


Figura 112.

Revenind la studiul de caz, putem spune că:

- profesorul cel mai cunoscut este definit de vârful cu gradul interior cel mai mare.
- profesorul care nu este cunoscut de niciun alt membru al grupului va fi definit de vârful cu gradul exterior egal cu 0.

Pentru fiecare profesor putem alcătui lista cu persoanele pe care le cunoaște precum și lista cu persoanele care îl cunosc. În acest sens se obțin mulțimile:

$\Gamma^+(x)=\{y \in X \mid [x,y] \in U\}$ mulțimea succesorilor vârfului x.
$\Gamma^-(x)=\{y \in X \mid [y,x] \in U\}$ mulțimea precedenților vârfului x.

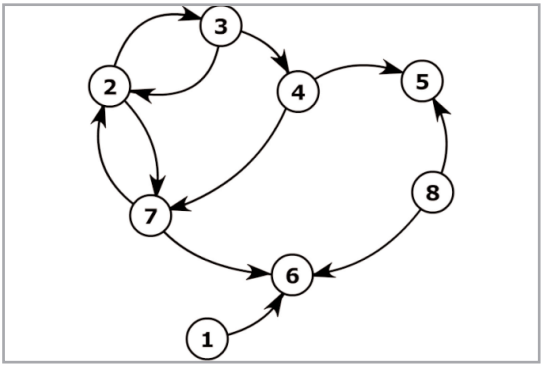
Figura 113.

Exemple

Pentru graful din figura 113.

Se obține:

$d^+(4)=2$; $\Gamma^+(4)=\{5,7\}$;
$d^-(4)=1$; $\Gamma^-(4)=\{3\}$
$d^+(6)=0$; $\Gamma^+(6)=\{\}$;
$d^-(6)=3$; $\Gamma^-(6)=\{1,7,8\}$



Observație:

card($\Gamma^+(x)$)= $d^+(x)$ card($\Gamma^-(x)$)= $d^-(x)$ pentru $x \in X$.

Reprezentarea în memorie a grafurilor orientate

Fie G=(X,U), unde X este o mulțime cu *n* vârfuri, iar *U* este o mulțime cu *m* arce.

Pentru reprezentarea în memorie a grafurilor orientate se pot folosi mai multe modalități: *matricea de adiacență*, *listele de adiacență*, *matricea vârfuri-arce* și *vectorul de arce*.

6.1. Memorarea grafurilor orientate prin matricea de adiacență

Matricea de adiacență asociată unui graf orientat este definită prin:

$$A[i,j] = \begin{cases} 1, \text{ dacă } [i,j] \in U \\ 0, \text{ altfel.} \end{cases}$$

Observație:

Deoarece [x,y]<>[y,x] pentru $x,y \in X$, atunci se constată că matricea de adiacență atașată unui graf neorientat nu este simetrică.

Exemple:

Pentru graful din figura 114, matricea de adiacență atașată este:

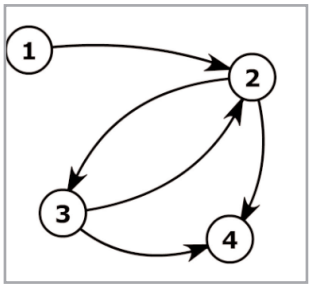


Figura 114.

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Observații:

Matricea de adiacență este o matrice pătrată cu *n* linii și *n* coloane.

Multe elemente ale matricei de adiacență sunt nule; folosind o astfel de reprezentare se consumă spațiu de memorie (matricea este *rară*).

Numărul de valori egale cu 1 de pe linia *i* din matrice ($1 \le i \le n$), reprezintă *gradul exterior* al vârfului *i*.

Numărul de valori egale cu 1, de pe coloana j din matrice ($1 \le j \le n$), reprezintă *gradul interior* al vârfului *j*.

Numărul de valori egale cu 1 din matrice reprezintă numărul de arce existente în graful orientat.

TEME

1. Precizați care dintre următoarele variante reprezintă matricea de adiacență atașată grafului orientat din figura 115:

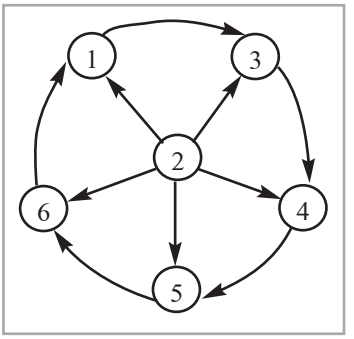
a) 100100
101111
000010
001000
000001
100000

b) 100100
111111
001010
001100
000011
100001

c) 010100
101111
101000
010001
100000
010100

d) 001000
101111
000100
000010
000001
100000

Figura 115.



2. Pentru graful din figura 115, determinați mulțimea vârfurilor cu proprietatea că diferența dintre gradul exterior și gradul interior este -1:

a) {} b) {2} c) {1,3,4,5,6} d) {1,2,3,4,5,6}

3. Pentru graful din figura 115, determinați numărul de vârfuri cu gradul interior egal cu 2.

a) 1; b) 5; c) 0; d) 4.

4. Se dă matricea de adiacență atașată unui graf orientat:

$$A = \begin{matrix} 0101 \\ 0000 \\ 0100 \\ 0110 \end{matrix}$$

Care este numărul de vârfuri cu proprietatea că gradul exterior este mai mare sau egal cu gradul interior?

a) 0; b) 1; c) 2; d) 3.

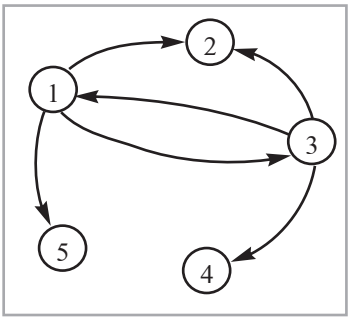
5. Fie G un graf orientat, ca în figura 116:

Se dau doi vectori *dext și dint* unde dext[i] memorează gradul exterior al vârfului i, iar *dint[i]* gradul interior, 1<=i<=5.

Care dintre perechile de vectori de mai jos se referă la graful din figura 116:

dext=(3,0,2,1,0); dint=(1,2,1,0,1)
dext=(2,0,2,2,0); dint=(1,0,0,0,2)
dext=(3,0,2,0,0); dint=(1,2,1,1,1)
dext=(2,0,2,0,0); dint=(0,2,0,0,0)

Figura 116.



6. Determinați valoarea de adevăr pentru următoarele afirmații:

a) Matricea de adiacență atașată unui graf orientat este pătrată și simetrică.

b) Numărul de valori egale cu 1 din matricea de adiacență atașată unui graf orientat este egal cu numărul de arce corespunzător acelui graf.

c) Numărul de valori egale cu 1 de pe coloana j din matrice (1<=j<=n) reprezintă gradul interior al vârfului j.

d) Matricea de adiacență atașată unui graf orientat conține pe diagonala principală numai valori egale cu 1.

7. Realizați un program, în limbajul studiat, pentru rezolvarea problemei propusă în studiul de caz (simpozion).

6.2. Memorarea unui graf orientat folosind listele de adiacență

Prin această metodă, se reține, pentru fiecare vârf, mulțimea succesorilor săi.

Exemplu

Pentru graful din figura 117, listele de adiacență vor fi:

Lista vârful 1: 2, 3, 7
Lista vârfului 2:———-
Lista vârfului 3:———-
Lista vârfului 4: 3, 6
Lista vârfului 5: 3
Lista vârfului 6: 7
Lista vârfului 7:———-

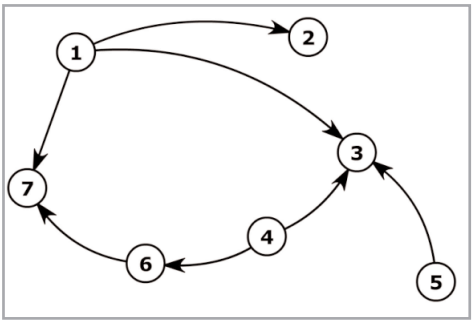


Figura 117.

TEME

1. Fie G un graf orientat (figura 118).

Pentru fiecare vârf determinați listele de adiacență.

2. Se dă următoarea matrice de adiacență a unui graf orientat:

$$A = \begin{matrix} 0101 \\ 0000 \\ 0100 \\ 0110 \end{matrix}$$

Pentru fiecare vârf determinați listele de adiacență.

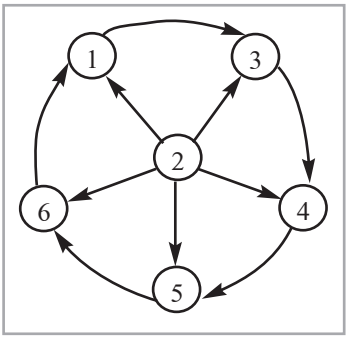


Figura 118.

3. Se dă următoarea matrice de adiacență a unui graf orientat:

$$A = \begin{matrix} 0111 \\ 0000 \\ 0101 \\ 0000 \end{matrix}$$

Determinați vârful a cărui listă de adiacență are un număr maxim de componente:
a)1; b)2; c)3; d)4.

6.3. Reprezentarea unui graf orientat folosind matricea vârfuri-arce

Matricea vârfuri-arce este o matrice cu *n* linii și *m* coloane definită astfel:

$$a[i,j]=\begin{cases} 1, \text{ dacă vârful } i \text{ este extremitate inițială a arcului } u_j \\ -1, \text{ dacă vârful } i \text{ este extremitate finală a arcului } u_j \\ 0, \text{ dacă vârful } i \text{ nu este extremitate a arcului } u_j \end{cases}$$

Exemplu:
Fie G=(X,U) (figura 119).
Matricea vârfuri arce este:

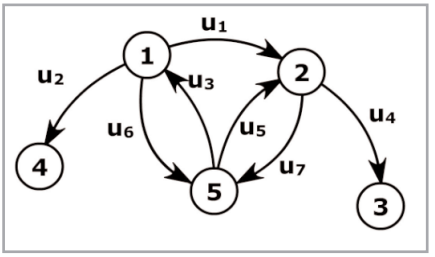


Figura 119.

$$A=\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 1 & 1 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & 1 & 1 \end{pmatrix}$$

Interpretarea matricei vârfuri-arce.

Cum a fost determinată ultima linie din matrice? Această linie corespunde vârfului 5.
a[5,1]= 0 deoarece vârful 5 nu este incident cu arcul u_1.
a[5,2]= 0 deoarece vârful 5 nu este incident cu arcul u_2.
a[5,3]= -1 deoarece vârful 5 este extremitate finală a arcului u_3.
a[5,4]= 0 deoarece vârful 5 nu este incident cu arcul u_4.
a[5,5]= -1 deoarece vârful 5 este extremitate finală a arcului u_5.
a[5,6]= 1 deoarece vârful 5 este extremitate inițială a arcului u_6.
a[5,7]= 1 deoarece vârful 5 este extremitate inițială a arcului u_7.

Observații:

Numărul de valori egale cu 1 din linia *i* a matricei vârfuri-arce reprezintă gradul exterior al vârfului *i*.

Numărul de valori egale cu -1 din linia *i* a matricei vârfuri-arce reprezintă gradul interior al vârfului *i*.

Pe fiecare coloană *i* a matricei vârfuri-arce există o singură valoare egală cu 1 (pe linia *x* din matrice deoarece *x* este extremitate inițială a arcului u_i) și o singură valoare egală cu -1 (pe linia *y* din matrice deoarece *y* este extremitate finală a arcului u_i), pentru $u_i=[x,y]$; $1 \le x, y \le n$, $u_i \in U$, $1 \le i \le m$.

TEME

1. Determinați matricea vârfuri-arce corespunzătoare grafului orientat din figura 120.

a)
```
 1 -1  0  0
 1  0  0 -1
 0 -1  1  0
 1  0 -1  0
 0  0  1 -1
```

b)
```
 1  1  0  1  0
-1  0 -1  0  0
 0  0  1 -1 -1
 0  0  1 -1 -1
 0 -1  0  0  1
```

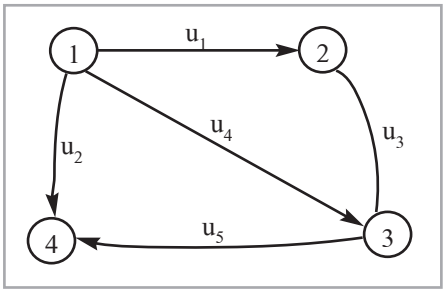


Figura 120.

c)
```
 1  1  0  1  0
-1  0 -1  0  0
 0  0  1 -1  1
 0 -1  0  0 -1
```

d)
```
 1  1  0  1
-1  0 -1  0
 0  0  1 -1
 0 -1  0  0
```

2. Determinați valoarea de adevăr pentru următoarele afirmații:

a) Matricea vârfuri-arce atașată unui graf orientat este o matrice pătrată.

b) Numărul de valori egale cu 1 din linia *i* a matricei vârfuri-arce reprezintă gradul exterior al vârfului *i*.

c) Numărul de valori egale cu -1 din coloana *i* a matricei vârfuri-arce reprezintă gradul exterior al vârfului *i*.

d) Pe fiecare coloană a matricei vârfuri-arce există o singură valoare egală cu 1 și o singură valoare egală cu -1.

6.4. Reprezentarea unui graf orientat ca un vector de arce

Un arc poate fi privit ca o celulă cu două câmpuri componente reprezentând extremitățile sale: vârful inițial și vârful final.

Fie $u_i \in U$; $1 \le i \le m$; $u_i=[x,y]$; $1 \le x,y \le n$.

Se obține un vector de înregistrări numit vector de arce.

Figura 121.

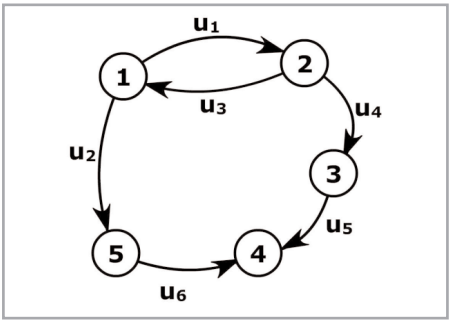
Exemplu:

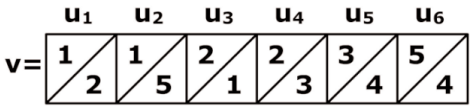
G=(X,U) (figura 121)

X={1,2,3,4,5}

U={[1,2];[1,5];[2,1];[2,3];[3,4];[5,4]}

va avea următoarea reprezentare folosind un vector de arce.



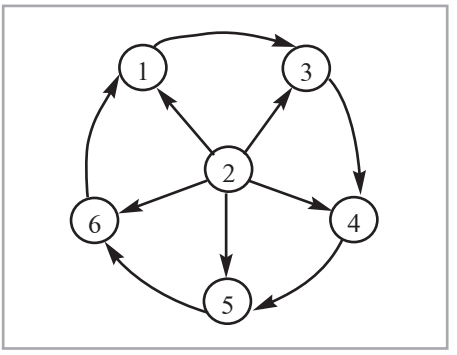


TEME

1. Fie G un graf orientat (figura 122).

Determinați vectorul de arce corespunzător.

Figura 122.



2. Se dă următoarea matrice de adiacență a unui graf orientat:

$$A=\begin{matrix}0111\\0000\\0101\\0000\end{matrix}$$

Determinați vectorul de arce corespunzător.

Parcurgerea grafurilor orientate

Pentru prelucrarea informațiilor atașate vârfurilor unui graf orientat se folosesc cele două parcurgeri studiate la grafurile neorientate: parcurgerea în lățime BF și parcurgerea în adâncime DF.

Diferența față de parcurgerea grafului neorientat constă în urmărirea direcțiilor impuse de arce.

Parcurgerea în lățime a grafurilor orientate (BF)

Fie G=(X,U) un graf orientat (figura 123).

Pornind din vârful 1 parcurgerea va fi: 1,2,3,5,4.

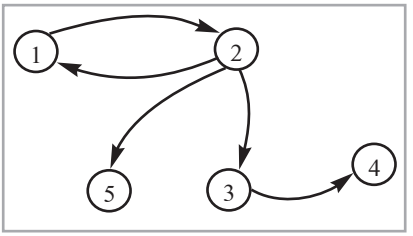


Figura 123.

Parcurgerea în adâncime a grafurilor orientate (DF)

Pentru graful de mai sus ordinea de parcurgere a vârfurilor, pornind din vârful 1, este: 1, 2, 3, 4, 5.

TEME

1. Alcătuiți lista vârfurilor în ordinea dată de vizitarea lor prin metoda parcurgerii BF (respectiv DF) pentru grafurile din figura 124:

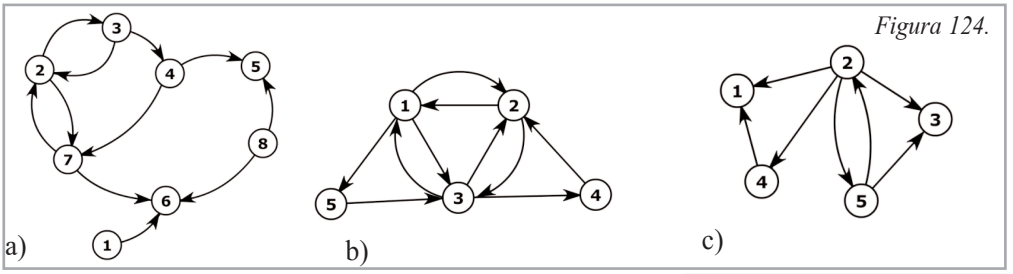


Figura 124.

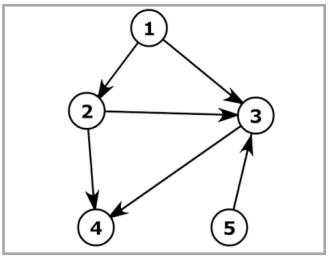


Figura 125.

2. Pentru graful din figura 125, rezolvați următoarele cerințe:

a) parcurgeți graful pe lățime-BF și completați vectorul *vizitat* și firul de așteptare folosit în algoritm;

b) ce particularități au vârful 4 și vârful 5?

c) ce semnificație are noțiunea de conexitate pentru un graf orientat?

3. Parcurgeți graful din figura 125 în adâncime-DF și completați vectorul *vizitat* al vârfurilor vizitate și stiva folosită în algoritm.

4. Implementați în limbajul de programare studiat algoritmul BF (respectiv DF), folosind listele de adiacență pentru memorarea grafului orientat.

5. Dezvoltați un algoritm recursiv de parcurgere în lățime a unui graf orientat.

6. Dezvoltați un algoritm recursiv de parcurgere în adâncime a unui graf orientat.

Drumuri și circuite în grafuri orientate

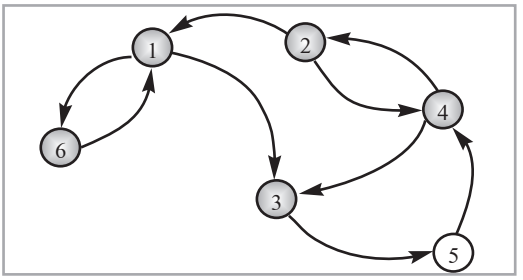
Definiție

Fie G=(X,U) un graf orientat. Se numește *drum în* graful G, D=(x_1,x_2,...x_k) o succesiune de vârfuri cu proprietatea ca [x_i,x_{i+1}]∈ U, unde 1<=i<=k-1.

Observații:

- Vârfurile x_1 și x_k sunt numite extremitățile drumului.
- Lungimea drumului este dată de numărul de arce componente.

Figura 126.



Exemple:

D_1=(1,6,1,3,5,4) este un drum de lungime 5.

D_2=(4,2,1,6) este un drum de lungime 3.

D_3=(3,5,4,2,1,6,1,3) este un drum de lungime 7.

Definiție

Se numește *drum elementar* într-un graf orientat, un drum pentru care vârfurile componente sunt distincte două câte două.

Exemple:

D_2=(4,2,1,6) este un drum elementar.

D_1=(1,6,1,3,5,4) este drum neelementar.

Definiție

Se numește *circuit* într-un graf orientat un drum pentru care cele două extremități coincid.

Exemplu: D_4=(5,4,2,1,3,5).

Definiție

Se numește *circuit elementar* un circuit în care toate elementele sunt distincte două câte două, cu excepția primului și a ultimului element.

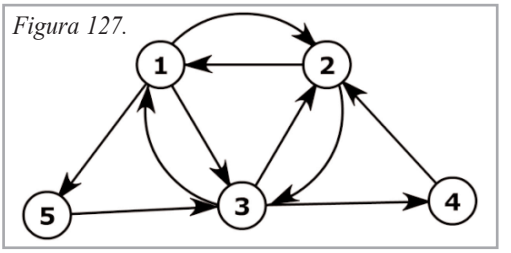
Exemple: D_4 =(5,4,2,1,3,5) este circuit elementar.

D_3=(3,5,4,2,1,6,1,3) este circuit neelementar.

TEME

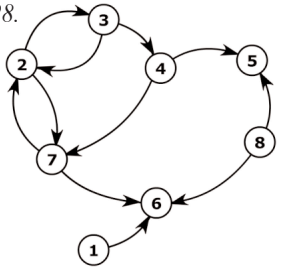
1. Pentru graful din figura 127, determinați un drum ce are extremitate inițială vârful 1 și extremitate finală vârful 2.

Figura 127.



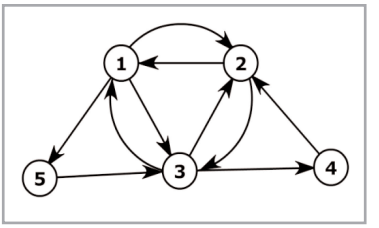
2. Pentru graful din figura 128, determinați toate drumurile elementare de lungime 3.

Figura 128.



3. Pentru graful din figura 129, determinați toate circuitele.

Figura 129.



Matricea drumurilor atașată unui graf orientat

Fie G=(X,U) un graf orientat cu *n* vârfuri. Asociem acestui graf o matrice pătrată cu *n* linii și *n* coloane denumită *matricea drumurilor* și definită astfel:

$$M[i,j] = \begin{cases} 1, \textit{ dacă există drum de la vârful i la vârful j} \\ 0, \textit{ altfel.} \end{cases}$$

Exemplu:

Pentru graful din figura 130 matricea drumurilor este:

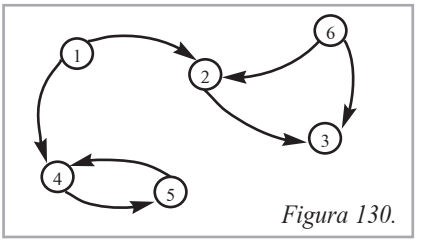


Figura 130.

$$M = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Se observă că M[1,3]=1 deoarece există drumul (1,2,3) între vârful 1 și vârful 3.

Observație:

În urma parcurgerii DF a grafului din figura 130, se vor atinge următoarele vârfuri, în ordinea: 1, 2, 3, 4, 5.

Deci 2, 3, 4, 5 sunt vârfurile la care putem ajunge, pornind din vârful 1 și urmărind sensul arcelor. Astfel, pe linia 1 din matricea drumurilor vom obține valoarea 1 pe coloanele 2, 3, 4 și 5.

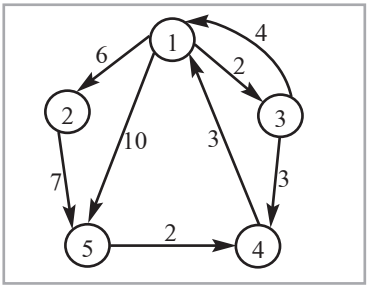
Pentru completarea matricei drumurilor se aplică parcurgerea DF pentru celelalte vârfuri (2, 3, 4, 5 și 6).

Matricea costurilor atașată unui graf orientat

În multe situații reale legăturile (arcele) dintre două puncte (vârfuri) au asociat un cost cu diferite semnificații: lungime, timp de parcurgere.

Pentru astfel de grafuri se construiește matricea costurilor:

Exemplu: Fie G = (X,U) un graf orientat:



$$C[i,j]=\begin{cases}\cos t(i,j),[i,j]\in U;\\ 0,[i,j]\notin U.\end{cases}$$

$$C=\begin{pmatrix}0&6&2&0&10\\0&0&0&0&7\\4&0&0&3&0\\3&0&0&0&0\\0&0&0&2&0\end{pmatrix}$$

TEME

1. Pentru grafurile din figura 131 (a, b, c), determinați matricea drumurilor.

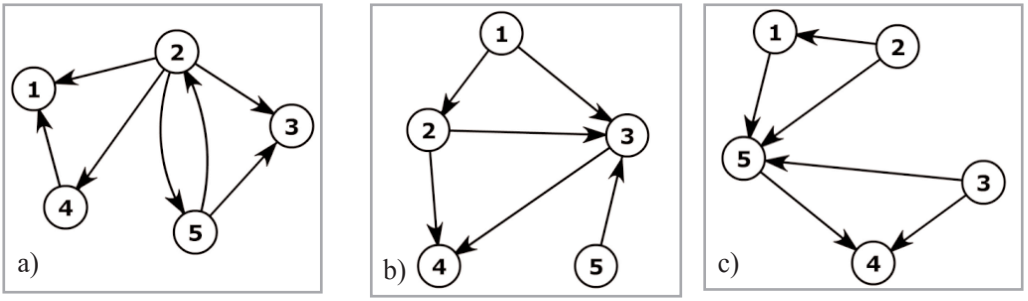


Figura 131.

2. Determinați graful orientat ce are matricea costurilor:

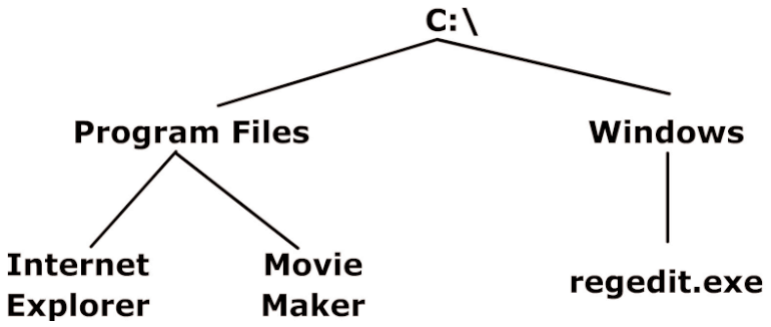
$$C=\begin{pmatrix}0&10&0&0&0\\4&0&2&0&0\\0&0&0&0&4\\2&0&5&0&0\\0&0&9&0&0\end{pmatrix}$$

7. ARBORI

Definiție

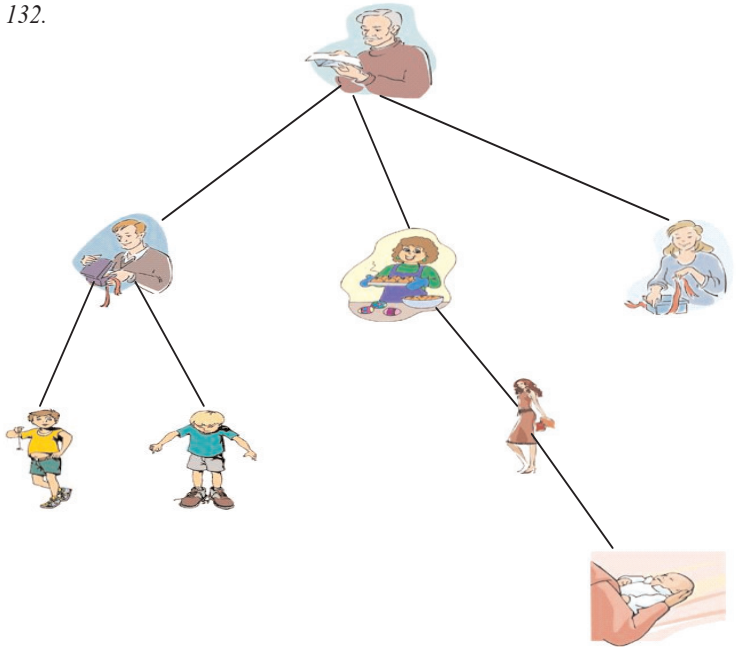
Un arbore este un graf conex aciclic (fără cicluri).

O structură de date de tip arbore devine intuitivă dacă ne imaginăm modalitatea de organizare a directoarelor și fișierelor într-un sistem de operare.



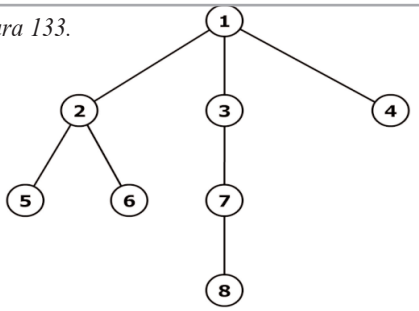
Asemănător acestui exemplu este și modelul arborelui genealogic al unei familii:

Figura 132.



După modelul arborelui genealogic de mai sus, putem obține graful din figura 133.

Figura 133.



Observații:

Graful este conex.

Graful nu conține cicluri.

Concluzii:

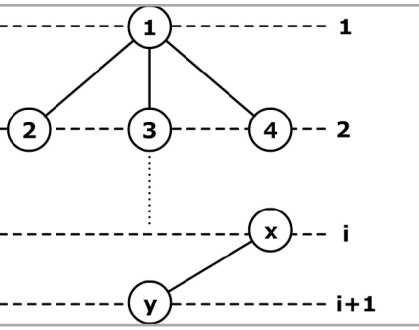
G este un arbore.

Un arbore poate fi așezat pe niveluri.

Se alege un vârf care se așază pe nivelul 1 și acesta se va numi *rădăcina*.

Pe nivelul 2 vor fi așezate acele vârfuri care sunt legate printr-o muchie direct de rădăcină și procedeul continuă până vor fi așezate toate nodurile pe nivelele corespunzătoare (figura 134).

Figura 134.



Notații:

X= vârf *părinte* sau *predecesor*, tată;

Y= vârf *fiu* sau copil, *succesor*.

Vârfurile care se află pe ultimul nivel se numesc vârfuri *terminale* sau *frunze*.

Vârfurile pot conține o informație utilă (cheie) care poate fi de orice tip.

Exemplu:

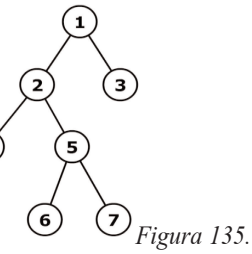


Figura 135.

Fie arborele din figura 135. Se observă că:

- Vârful 1 este rădăcina arborelui.
- Vârful 1 este părinte pentru vârful 2 și vârful 3.
- Vârful 2 este descendent (fiu, copil) pentru vârful 1.
- Vârful 3 este descendent (fiu, copil) pentru vârful 1.
- Vârfurile 3, 4, 6 și 7 sunt vârfuri terminale (frunze).

Teoremă

Un arbore cu n vârfuri are n-1 muchii.

Demonstrație prin inducție matematică:

1. P(n): este o propoziție adevărată. Un arbore care conține doar un singur vârf are un număr de muchii egale cu 0.

2. Să demonstrăm că *P*(m) este adevărată, cu *m>1*, presupunând că afirmațiile *P(k)* sunt adevărate pentru *k<m*.

Dacă se elimină o muchie dintr-un arbore, atunci se ajunge la un graf care nu va mai fi conex și se obțin doi arbori. Fie *p,* respectiv *q,* numărul de vârfuri din cei doi arbori, unde:

$$m=p+q;\ p<m,\ q<m$$

Primul arbore va avea p-1 muchii iar cel de-al doilea va avea q-1 muchii. În total, dacă se adaugă și muchia inițial eliminată, se obține:

p-1+q-1+1=m-1 muchii, fapt ce probează afirmația $P(m)$.

Pentru reprezentarea în memorie a arborilor, pot fi utilizate aceleași reprezentări pe care le folosim pentru grafuri.

Arbori binari

Studiu de caz

Andrei are o hartă montană (figura 136). El se află în cabana „Vârful cu brazi" și dorește să ajungă împreună cu prietenii săi la „Cascada binară". Traseul pe care urmează să îl parcurgă se poate ramifica, în diferite puncte, în două cărări. Ei observă că drumul spre cascadă corespunde alegerii repetate a potecii din stânga.

Ajutați-i să descopere numărul de intersecții ce trebuie traversate precum și lungimea traseului.

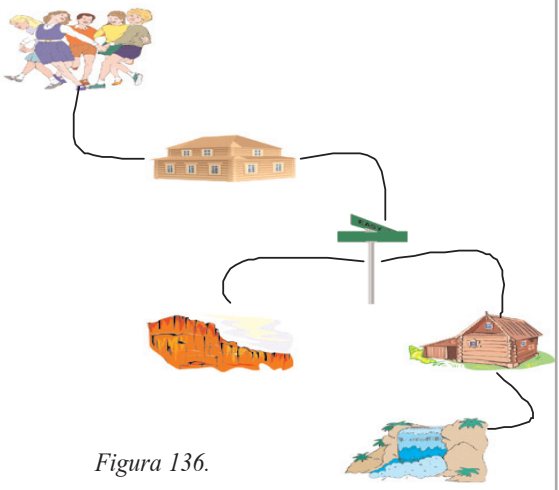


Figura 136.

Rezolvarea problemei conduce la noțiunea de arbore binar.

Definiție:

Un *arbore binar* este un arbore cu proprietatea că oricare vârf, în afară de frunze, are cel mult doi descendenți (succesori).

Exemplu:

Arborele binar din figura 137 are drept rădăcină nodul 1.

Nodul 1 are doi descendenți direcți: nodul 2 (descendent stâng) și nodul 3 (descendent drept).

Nodul 2 are un singur descendent direct: nodul 4 (descendent stâng).

Nodul 3 are doi descendenți direcți: nodul 5 (descendent stâng) și nodul 6 (descendent drept).

Nodurile 4, 5 și 6 sunt noduri terminale (frunze) pentru arborele binar dat.

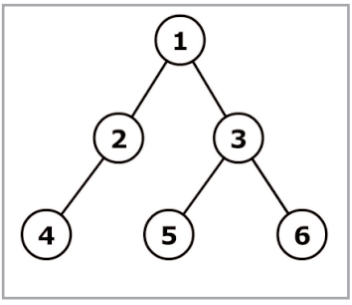


Figura 137.

Un arbore binar poate fi reprezentat în memorie prin două tablouri:

S și D cu n elemente (n= numărul de noduri din graf), unde:

S[i]= succesorul stâng nodului i.

D[i]= succesorul drept al nodului i.

Exemplu:

Pentru arborele din figura 138 vectorii S și D vor avea următoarele valori:

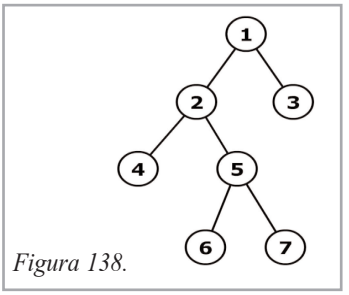


Figura 138.

i	1	2	3	4	5	6	7
S[i]	2	4	0	0	6	0	0

i	1	2	3	4	5	6	7
D[i]	3	5	0	0	7	0	0

Observații:

În tablourile S și D, nodurile terminale au valori nule.

Rădăcina este singurul nod care nu apare ca valoare în tabourile S și D.

TEME

1. Verificați dacă graful G=(X,U), X = {1, 2, 3, 4, 5, 6, 7}
U = { [1,2]; [1,3]; [2,5]; [2,6]; [2,7]} este arbore.

2. Așezați pe niveluri arborele din figura 139. Fixați drept rădăcină nodul 4.

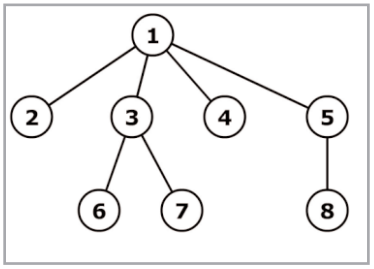


Figura 139.

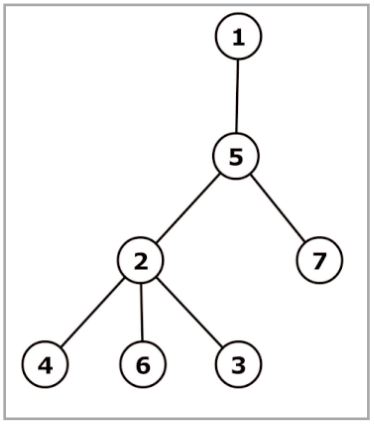


Figura 140.

3. Determinați numărul de niveluri ce se obțin prin rearanjarea arborelui din figura 140 pentru a avea drept rădăcină vârful 3.

4. Verificați care dintre următoarele matrice de adiacență corespunde unui arbore binar.

$$A1 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \quad A2 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \quad A3 = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

5. Să se determine vectorii S și D pentru arborele binar din figura 141.

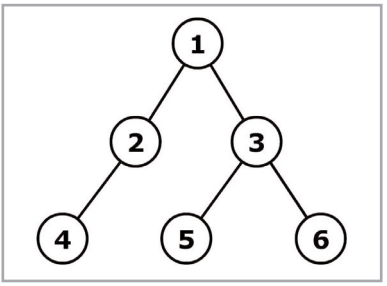


Figura 141.

6. Determinați rădăcina unui arbore binar reprezentat prin următorii vectori:
S=(0,4,0,0,3); D=(0,1,0,0,2).

Arborele parțial de cost minim

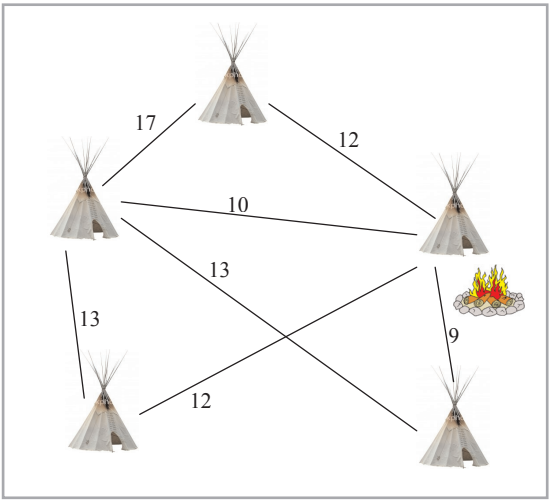
Studiu de caz APECEM

Șeful tribului Apecem dorește să construiască poteci care să lege corturile supușilor săi astfel încât resursele folosite să fie minime și în final să existe un traseu între oricare două corturi. Dacă îl veți ajuta, vă va trimite o invitație la petrecea tribală anuală.

Se cunosc:

- n = numărul de corturi
- m = numărul de poteci ce pot fi construite precum și lungimea acestora.

Se cere: determinarea unui traseu de cost minim care să cuprindă toate corturile.



Rezolvare

Se atașează grafului neorientat obținut G o matrice de costuri.

Vom determina un graf parțial al grafului G (*toate corturile*) care să fie conex, de cost minim (suma costurilor muchiilor sale să fie minimă) și să nu existe circuite (să nu se treacă de două ori prin același punct).

Acesta va fi un arbore parțial de cost minim (APM).

Algoritmul lui Kruskal de determinare a APM

Pentru început, se consideră cele **n** corturi izolate (**n** arbori disjuncți care conțin fiecare câte un nod al grafului inițial).

La fiecare pas al algoritmului, se introduce câte o potecă de cost minim (unificarea a doi dintre arborii existenți); se selectează din graf acea muchie, *u*, de cost minim, dintre cele nealese încă, ce are o extremitate într-un arbore și cealaltă extremitate în alt arbore. În urma unificării celor doi arbori, se constituie un alt arbore ce va conține toate nodurile și muchiile celor doi arbori inițiali precum și muchia u.

Exemplu: Fie graful atașat problemei din studiul de caz (figura 142).

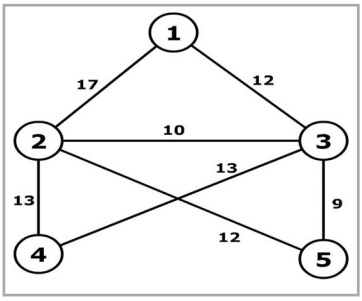


Figura 142.

Se folosește o matrice *A* cu *m* coloane (corespunzătoare celor *m* muchii) și 3 linii (care corespund celor două extremități *x,y și* costului muchiei *c*).

$$A = \begin{matrix} x \\ y \\ c \end{matrix}\begin{pmatrix} 1 & 1 & 2 & 2 & 2 & 3 & 3 \\ 2 & 3 & 3 & 5 & 4 & 4 & 5 \\ 17 & 12 & 10 & 12 & 13 & 13 & 9 \end{pmatrix}$$

Costul APM este inițializat cu 0 (*cost* = 0).

Se organizează matricea realizându-se o ordonare crescătoare a muchiilor după *cost*.

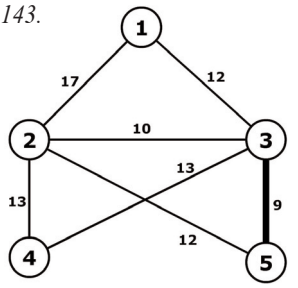
$$A = \begin{matrix} x \\ y \\ c \end{matrix}\begin{pmatrix} 3 & 2 & 1 & 2 & 2 & 3 & 1 \\ 5 & 3 & 3 & 5 & 4 & 4 & 2 \\ 9 & 10 & 12 & 12 & 13 & 13 & 17 \end{pmatrix}$$

Se folosește un tablou unidimensional S unde, S[i]= indicele arborelui din care face parte nodul *i* la un moment dat.

Inițial fiecare nod este în propriul său arbore:

S = (1, 2, 3, 4, 5).

Figura 143.



Se alege muchia de cost minim ce are o extremitate într-un arbore și altă extremitate în alt arbore dintre cei existenți ([3,5]).

Se realizează unificarea celor doi arbori și vectorul S devine:

S = (1, 2, 3, 4, 3).

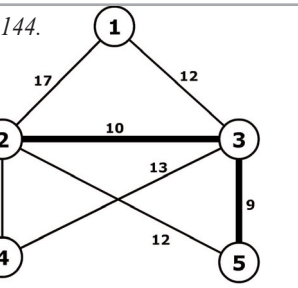
Costul APM va fi: cost = 0 + 9 + 9.

Muchia selectată la pasul următor va fi [2,3] (este de cost minim dintre cele neselectate încă și are extremitatea 2 în arborele cu indicele 2 și extremitatea 3 în arborele indice 3).

Se realizează unificarea celor doi arbori și Vectorul S devine:

S = (1, 2, 2, 4, 2).

Figura 144.



Costul APM va fi:

Cost= 9+10=19.

Muchia selectată la pasul următor va fi [1,3] (este de cost minim dintre cele rămase neselectate încă și are extremitatea 1 în arborele etichetat cu 1 și extremitatea 3 în arborele etichetat cu 2).

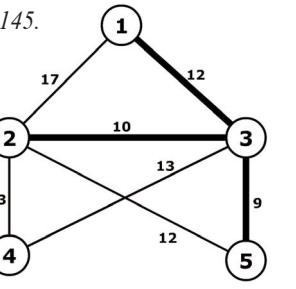
Se realizează unificarea celor doi arbori și vectorul S devine:

S = (1, 1, 2, 4, 1).

Costul APM va fi:

Cost =19+12 =31.

Figura 145.



Muchia selectată la pasul următor va fi [2,4] (este de cost minim dintre cele rămase neselectate încă și are extremitatea 2 în arborele etichetat cu 1 și extremitatea 4 în arborele etichetat cu 4).

Se realizează unificarea celor doi arbori și vectorul S devine:

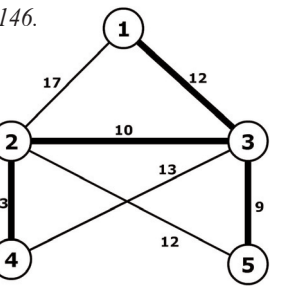
S = (1, 1, 1, 1, 1).

Costul APM va fi:

Cost= 31+13= 44.

Algoritmul se oprește deoarece au fost selectate n-1 muchii (un arbore cu n noduri conține n-1 muchii).

Figura 146.



Observație:
APM pentru un graf dat nu este unic.

Reprezentarea algoritmului în pseudocod

```
citește (n);
citește (m);
pentru i=1, m execută
        citeste (x,y,c);
        A[1,i]←x;
        A[2,i]←y;
        A[3,i]←c;
sfârșit pentru;
pentru i=1, n execută
        S[i]←i;
/*Se pozitioneaza fiecare nod în propriul sau arbore*/
sfârșit pentru;
cost←0; /*cost APM/*
sortează_matrice;
/*Se sorteaza matricea dupa ultima linie corespunzatoare costului muchiilor*/
k←0;
/*K reprezinta numarul de muchii ce vor fi adaugate în APM*/
i←0;
cât timp k< n-1 execută
    dacă S[A[1,i]]<>S[A[2,i]] atunci
                        k←k+1;
                        cost←cost + A[3,i];
                        scrie(A[1,i]);
                        scrie(A[2,i]);
                        x←A[1,i];
                        y←A[2,i];
                        pentru j=1,n execută
                                dacă S[j]=x atunci S[j]‹y
                                sfârșit dacă
                        sfârșit pentru
    sfârșit dacă
i←i+1;
sfârșit cât timp
scrie  (cost);
```

TEME

1. Să se determine dacă un graf G este arbore.

2. Determinați, folosind algoritmul lui Kruskal, arborele parțial de cost minim pentru grafurile din figura 147.

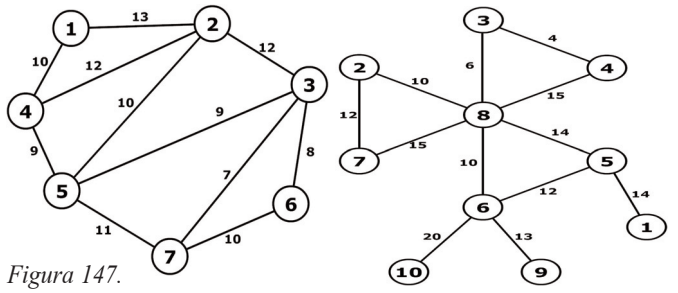


Figura 147.

3. Formulați un exemplu de graf neorientat G pentru care arborele parțial de cost minim nu este unic.

4. Codificați în limbajul de programare studiat algoritmul lui Kruskal. Citirea datelor se face din fișierul *date.in* ce va avea pe prima linie numărul de noduri, pe a doua linie numărul de muchii, iar pe următoarele *m* linii trei valori (extremitățile muchiei și costul său).

5. Justificați ordonarea crescătoare a muchiilor, după cost.

6. Formulați exemple de probleme care să conducă la determinarea arborelui parțial de cost minim.

PROBLEME PROPUSE

Asociați fiecărei probleme modelul de graf corespunzător. Identificați elementele specifice și algoritmul care conduce la rezolvarea problemei.

1. Un grup de *n* copii și-au interconectat calculatoarele personale astfel încât ei își pot transmite „mesaje” și „date” „din calculator în calculator”. Se cunosc calculatoarele între care există legături directe. Să se precizeze dacă:
   a) Oricare dintre copii poate transmite mesaje către ceilalți n-1 copii.
   b) Există posibilitatea ca cel puțin unul dintre copii să primească înapoi mesajul transmis în rețea.

2. Într-un grup de *n* persoane se formează un „lanț al slăbiciunilor” sau „lanțul influenței”. Cunoscând persoanele care „se influențează” direct, să se determine cel mai lung „lanț al influenței” din grupul respectiv.

3. Se cunoaște amplasarea a celor *n* localități ale unui județ și drumurile care leagă între ele câte două localități. Să se determine:
   a) localitățile izolate;
   b) toate posibilitățile de deplasare între două localități oarecare.

4. La un simpozion participă *n* persoane care nu se cunosc direct dar pot face cunoștință unele cu altele prin intermediul unei cunoștințe comune. Știindu-se persoanele care se cunosc, să se determine dacă:

a) O persoană oarecare poate să își facă cel puțin o cunoștință nouă la acest simpozion.
b) Există persoane care cunosc „dinainte” toate celelalte persoane.
c) Există persoane care se cunosc „dinainte” între ele, formând un grup „închis”, în care nu poate să pătrundă nicio „cunoștință” nouă.

5. Se cunoaște amplasarea conductelor de gaz prin care o centrală alimentează blocurile unui nou cartier și costurile corespunzătoare fiecărui segment de conductă (un segment de conductă face legătura între două blocuri). Să se determine cel mai ieftin traseu pentru alimentarea cu combustibil gazos a noului cartier.

6. Fiind date cele *n* puncte din teritoriul de pază al unei patrule și căile de acces între acestea, să se determine traseul pe care se poate deplasa patrula astfel încât să verifice într-un singur „rond” toate punctele o singură dată. Dacă problema nu are soluție, să se determine, dacă există, zonele în care se poate instala o astfel de patrulă (numărul zonelor și numărul punctelor din care este formată fiecare zonă).

7. O agenție de turism are spații de cazare și agrement în *n* orașe. Pentru atragerea clienților și îmbunătățirea aprovizionării, agenția are de rezolvat următoarele probleme:

- Să ofere clienților, pentru sfârșitul săptămânii (vineri, sâmbătă și duminică), cât mai multe variante de excursii „în circuit” cu oprire câte o zi în fiecare oraș.
- Să ofere clienților, pentru vacanță, excursii „în circuit” cu posibilitatea de a vizita toate cele *n* orașe trecând o singură dată prin fiecare.
- Să-și construiască un sistem propriu de comunicare între cele *n* orașe, astfel încât oricare două orașe să comunice direct sau prin intermediul altora, printr-un număr minim de căi de acces.

Cerințe:

Asociați cerințelor din problemă modelul de graf corespunzător și rezolvați problemele acestei agenții, folosind metodele de lucru specifice teoriei grafurilor.

PROIECT ÎN ECHIPĂ

Elemente de teoria grafurilor – abordare interdisciplinară

Scopul proiectului: aplicarea cunoștințelor de teoria grafurilor în rezolvarea unor probleme concrete de fizică, biologie, geografie, sociologie și altele.

Sugestie de lucru: se formează echipe de 6–8 elevi; fiecare elev tratează o problemă/domeniu; rezolvarea conduce la realizarea de programe și o documentație scrisă. Proiectul poate fi susținut și prin prezentare PowerPoint.

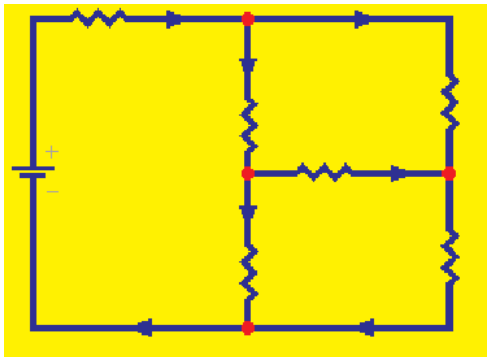
Aplicații propuse pentru rezolvarea proiectului

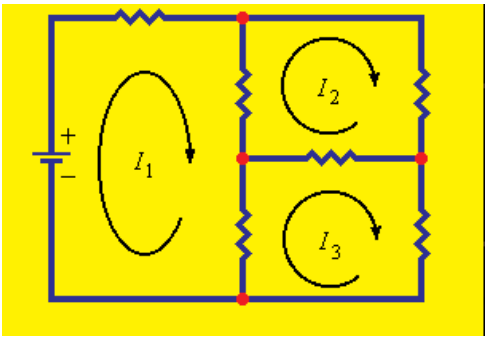
A. Circuite electrice

1) Identificarea elementelor specifice grafurilor pentru un circuit electric

Cerințe:

- Asociați circuitului electric tipul de graf corespunzător.
- Identificați elementele specifice grafurilor și descrieți semnificația acestora în circuitul electric.





Sugestie de rezolvare

Nodurile unui circuit electric corespund vârfurilor grafului (1,2,3,4).

Ramura ce leagă două noduri ale circuitului electric reprezintă arcul grafului ([1,2], [1,3], [2,3], [2,4], [3,4], [4,1]).

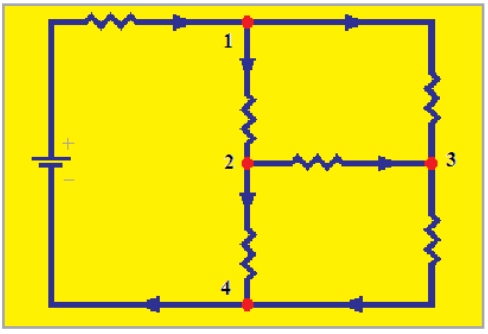
Ochiul unui circuit electric reprezintă circuitul (ciclul) unui graf.

Fie graful G=(X,U) , X – mulțimea vârfurilor grafului și U – mulțimea arcelor grafului.

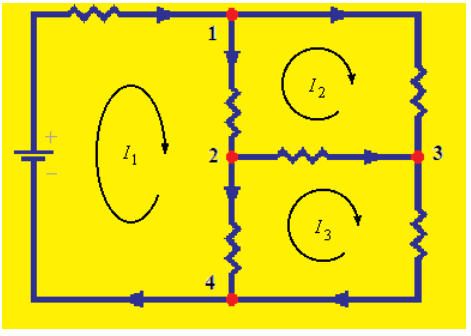
Gradul exterior *(al vârfului x)**:*** numărul arcelor de forma [x,y] (există legătura orientată de la x la y).

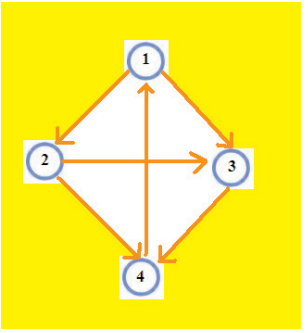
Gradul interior *(al vârfului x):* numărul arcelor de forma [y,x] (există legătura orientată de la y la x).

Exemplu (vârf 2):
Grad exterior = 2
Grad interior = 1



2) Evidențierea Legilor lui Kirchhoff folosind graful asociat





Cu ajutorul grafului corespunzător, prin stabilirea ***gradului interior*** și a ***gradului exterior*** al unui vârf, stabilim semnul curenților electrici și automat sensul acestora (intră sau ies din nodul respectiv).

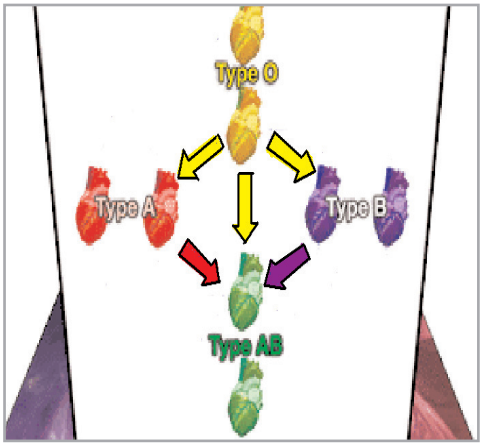
B. ***Grupele de sânge***

1. Realizați o scurtă prezentare a grupelor de sânge, urmărind planul de idei și graful asociat.

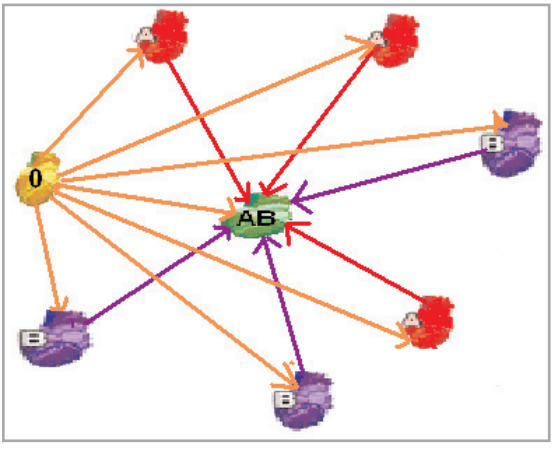
- De ce trebuie să îți cunoști grupa de sânge?
- Pierderile mari de sânge pot fi recuperate prin transfuzii?
- Orice alte combinații decât cele indicate în schemă provoacă accidente grave, mortale?

2. Care este semnificația următoarelor elemente:

- gradul unui vârf;
- gradul exterior al unui vârf;
- gradul interior al unui vârf.

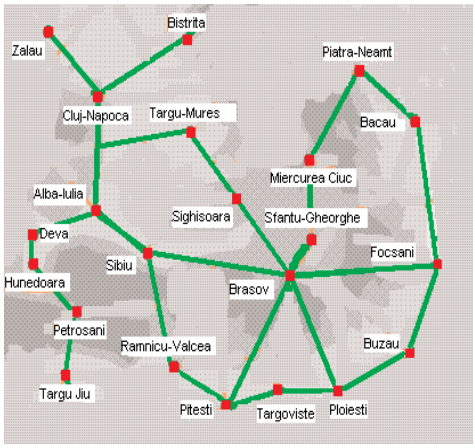


Grad exterior al vârfului 0=7.
Grad interior al vârfului 0=0
=> nu poate primi sânge de la nicio altă grupă.
Grad exterior vârf AB=0 => nu poate dona niciunei persoane.



C. *Hărți*

1. Harta rutieră



Noduri = Orașe.

Muchii = Drumuri.

Graf conex = Există lanț (drum) între oricare două orașe.

Graf neorientat = accesul între două orașe se face în ambele sensuri.

Muchiilor li se pot asocia costuri; fiecare cost reprezintă distanța între două orașe.

Lanț de cost minim = drumul cel mai scurt dintre două orașe.

Cerințe:

Pornind de la Harta rutieră:

- formulați probleme care se pot rezolva prin aplicarea elementelor de teoria grafurilor;
- construiți și caracterizați graful asociat hărții rutiere.

2. Rute aeriene

Nod = oraș.

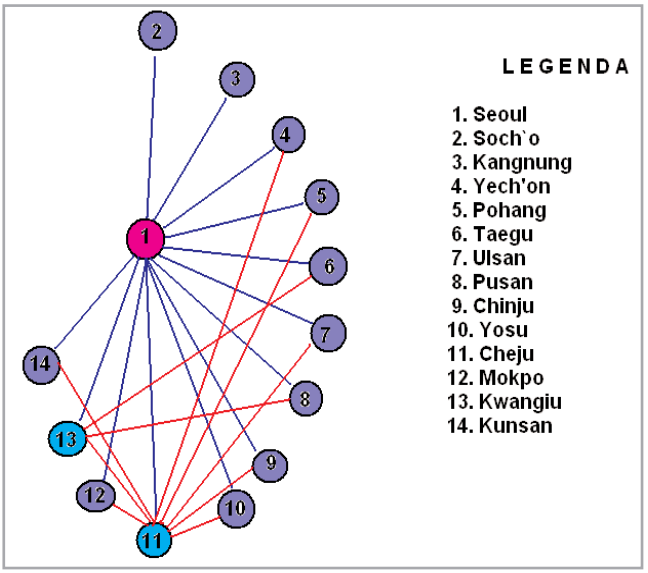
Muchie = linie aeriană.

Graf conex = Există lanț (traseu aerian) între oricare două orașe.

Graf neorientat = accesul între două orașe se face în ambele sensuri de zbor.

Graf asociat rutelor aeriene

Muchiilor li se pot asocia costuri; fiecare cost reprezintă prețul zborului între două orașe. Lanț de cost minim = prețul minim al zborului între două orașe.

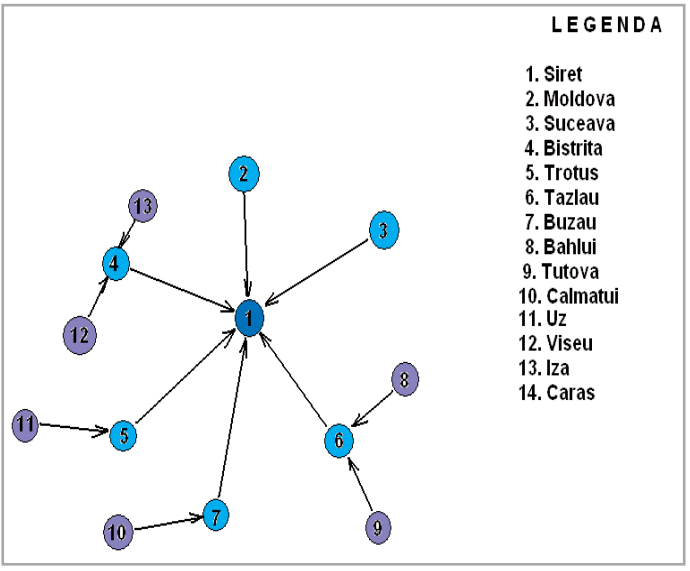


Cerințe:

- Pornind de la harta rutelor aeriene, formulați probleme care se pot rezolva prin aplicarea elementelor de teoria grafurilor.
- Caracterizați graful asociat hărții rutelor aeriene.

3. Harta râurilor

Graful asociat râurilor



Cerințe:

- Caracterizați graful asociat hărții râurilor.
- Construiți matricea vârfuri-arce corespunzătoare acestui graf.

D. ***Clase de echivalență***

Pentru elementele unei mulțimi se definește relația de echivalență a~b cu proprietățile: reflexivitate (a~a) și tranzitivitate (a~b, b~c implică a~c). Pentru o mulțime dată, să se determine clasele de echivalență cunoscându-se relațiile directe de echivalență.

Exemplu:

X={1,3,5,7,9}

Pentru echivalențe: 1~3 ; 5~9 ; 3~7 vor exista 2 clase de echivalență:

C1={1,3,7} și C2={5,9} .

Cerințe:

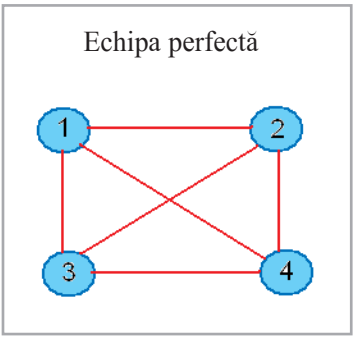
- Asociați problemei un graf și precizați semnificația elementelor.
- Cum poate fi recunoscută proprietatea de tranzitivitate?
- Care este semnificația claselor de echivalență?
- Realizați un program pentru determinarea claselor de echivalență și a elementelor din fiecare clasă.

E. *Sociograma*

1. Realizați un „sondaj de opinie" pentru determinarea relațiilor de afectivitate, simpatie, colaborare, respingere dintre elevii unei clase. Pentru claritatea și concizia datelor ce vor fi prelucrate, formulați o singură întrebare (exemplu: „Cu cine ai dori să mergi în excursie?").

2. Codificați de la 1 la n membrii grupului participanți la „sondaj".

3. Întocmiți sociograma: reprezentare grafică prin puncte și săgeți cu semnificația A⇒B: A îl simpatizează pe B (dacă prin sondaj s-au urmărit relații de simpatie). Dacă sondajul s-a aplicat unui grup mare (n>30), folosiți, pentru reprezentare, hârtia milimetrică.



- Completați următorul enunț: *Sociograma este un model .....format din ..... și ..... reprezentând..... .*
- Ce model de reprezentare a datelor poate fi asociat sociogramei?
- Care ar fi semnificația sociogramei dacă i-am asocia modelul unui graf neorientat? De ce nu se poate folosi acest model?
- Construiți matricea de adiacență pentru sociograma realizată în urma „sondajului".
- Analizați matricea de adiacență.
- Care este semnificația elementelor de pe diagonala principală?
- Care este semnificația sumei elementelor de pe o linie oarecare *i*?
- Ce reprezintă suma elementelor de pe o coloană oarecare *j*?
- Ce semnificație are o linie „zero"?
- Ce semnificație are o coloană „zero"?
- Concretizați analiza matricei de adiacență pentru grupul de elevi pe care ați efectuat sondajul.

4. Analiza coeziunii grupului:

- Ce semnificație are noțiunea de lanț într-un graf orientat?
- Completați enunțul: *Un lanț în care toate arcele au aceeași orientare se numește..... .*
- Ce semnificație are proprietatea de conexitate într-un graf orientat?
- Cum putem determina numărul de „subgrupuri " dintr-un grup?
- Cum determinăm liderul grupului?
- Calculați indicele de coeziune a grupului:

IC=(2*(numar_relatii_reciproce))/(n*(n-1)).

PROBLEME RECAPITULATIVE

Pentru fiecare dintre următoarele probleme, stabiliți structurile de date necesare si rezolvați cerințele prin programe modularizate în variantă iterativă și recursivă, după caz.

1. Să se afișeze linie cu linie o matrice rară; se cunoaște lista cu valorile nenule și poziția acestora în matrice.

2. Se dă o greutate G și alte *n* greutăți gi, i≠1, n, n<= 50. Să se stivuiască greutățile în ordinea de la1 la *n* și să se compare suma lor cu G.

3. Se înregistrează, într-o listă L1, datele (numele și media generală) depre elevii programați la un interviu pentru un job de vacanță. Să se formeze două liste noi: L2 numai cu fete și L3 numai cu băieți. Toate numele fetelor se termină cu litera '*a*'.

4. Se cunosc relațiile de prietenie dintre toți elevii unei clase. Să se determine dacă toți elevii clasei sunt prieteni între ei și, dacă este cazul, numărul de grupuri de prieteni din clasă.

5. Se cunosc căile directe de acces dintre cele *n* localități dintr-un teritoriu. Să se determine dacă se poate ajunge din orice localitate în oricare alta, nu neapărat prin legături directe.

6. Să se determine toate modalitățile de descompunere a unui număr natural *n* în sumă de numere naturale distincte.

7. Pe o suprafață formată din *n*n* puncte se află un singur punct colorat distinct. Să se determine coordonatele acestui punct printr-un număr minim de căutări.

8. Să se transforme un număr natural *n* în sumă de alte numere naturale, fiecare dintre ele nedepășind o valoare *m* specificată, corespunzător relației:

$$S(n,m) = \begin{cases} 1 \text{ dacă } n=1 \text{ sau } m=1 \\ 1 + S(n,n-1) \quad \text{dacă } n<=m \\ S(n,n-1) + S(n-m,m) \quad \text{dacă } n>m \end{cases}$$

9. Să se determine valoarea expresiei: E = 1+ 1*3 + ...+1*3*....*(2n-1)

10. Să se determine dacă două cuvinte introduse de la tastatură sunt anagrame între ele. Exemplu: cuvintele *apolodor* și *radopolo* sunt anagrame.

11. Un curier trebuie să transmită zilnic corespondența către cei *n* clienți. În prima zi de lucru, curierul primește cele *n* adrese și harta orașului.
După ce a găsit pe hartă toate modalitățile de a ajunge de la o adresă la alta, el își propune să caute un traseu prin care să ajungă la toate adresele fără să treacă de mai multe ori pe la un client.

12. Să se determine toate numerele naturale care se pot forma din *k* cifre, k<=5 introdus de la tastatură.

13. O bancă are *n* clienți; se cunosc relațiile băncii cu fiecare dintre clienți: un client are împrumut de la bancă; banca păstrează economiile unui client.
Un client se poate afla, față de bancă, în oricare dintre cele două relații.
Să se determine:
a) toți clienții care au împrumut la bancă;
b) toți clienți care au economii la bancă;
c) valoarea celui mai mare împrumut;
d) clienții al căror împrumut depășește jumătate din valoarea economiilor.

14. Se cunosc relațiile de colaborare dintre cei *n* membri ai unei oranizații.
Să se determine:
a) membrul sau membrii cu cel mai mare număr de colaboratori;
b) membrul sau membrii care nu au participat la nicio activitate colectivă.

15. Într-un grup de *n* copii, există *x* copii care spun întotdeauna adevarul și *y* copii mincinoși.
Să se formeze toate grupurile de câte *z* copii astfel încât, în fiecare grup, să existe cel mult un mincinos; acesta trebuie așezat între doi copii care spun adevărul. Valorile pentru *n, x, y* și *z* se citesc de la tastatură.

16. Un copil primește *n* cuburi. Se cunosc laturile și culoarea pentru fiecare dintre cele *n* cuburi. Există cuburi roșii, galbene, albe și verzi.
Să se determine toate modalitățile de a construi turnuri stabile de înălțime *h* în care să nu existe două cuburi învecinate colorate la fel.
Datele problemei se citesc din fișirul *turn.in* cu structura: pe prima linie, valorile *n* și *h* separate prin spațiu; pe fiecare dintre următoarele *n* linii, latura și culoarea unui cub. Soluțiile problemei se scriu în fișierul *turn.out*; fiecare soluție pe câte o linie.

17. Să se determine toate elementele unui șir de lungime *n* definit astfel:
1, 21, 123, 4321, 12345, ...

18. Să se determine toate cuvintele de *n* litere (3<=n<=5) care se pot forma cu literele mulțimii {a, e, f, g, h, i, m, n, t, u} astfel încât să fie respectate următoarele restricții:
a) fiecare cuvât să înceapă și să se termine cu o vocală;
b) să nu existe mai mult de două consoane alăturate;
c) să nu conțină grupul *ghi.*

19. Să se determine toate elementele unui șir de lungime *n* definit astfel:
1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5.....

20. Să se determine toate modalitățile de a urca o scară cu *n* trepte, știind că scara poate fi urcată treptă cu treaptă sau câte două trepte o dată.

ANEXA 1

PROBLEMA OPRIRII

Această problemă a fost formulată în anul 1946 şi publicată în articolul "A variant of a recursively unsolvable problem", în volumul 52 al *Bulletin of the American Mathematical Society.* Fiind mai simplă decât problema opririi programelor, problema corespondenței Post este frecvent folosită pentru demonstrarea – prin reducere – a nedecidabilității unor probleme din informatică.

Problema Opririi cere să se decidă algoritmic (după un număr finit de paşi) dacă un program arbitrar se opreşte sau nu. Matematicianul Alan Turing[1] a arătat, în 1936, că această problemă este nerezovabilă algoritmic: nu există niciun algoritm capabil să rezolve *Problema Opririi*. Există programe care rezolvă parțial *Problema Opririi* (pentru clase de programe), dar nu există niciun program care să rezolve *Problema Opririi* pentru orice program.

Vom prezenta demonstrația propusă de matematicianul Gregory Chaitin[2]. Vom lucra numai cu programe care manipulează numere naturale.

Presupunem, prin absurd, că există un program – numit programul de oprire – care decide dacă un program arbitrar se opreşte sau nu. În memorie, programele apar ca secvențe de biți corespunzătoare codului intern al instrucțiunilor. Rezultă că fiecare program are în memorie o lungime exprimată în biți. Considerăm programul:

Pasul 1. Se citeşte numărul natural **n**.

Pasul 2. Se listează toate programele care în binar se scriu cu cel mult **n** biți.

Pasul 3. Se aplică programul de oprire şi se elimină din listă toate programele care nu se opresc.

Pasul 4. Se execută programele rămase în listă până când toate programele se opresc; rezultatul este cel mai mic număr natural, diferit de toate rezultatele produse de fiecare program din listă.

Programul de mai sus se opreşte pentru fiecare intrare **n**. Dacă scriem acest program în binar, vom avea nevoie de aproximativ log (**n**) plus o constantă constând din numărul de biți (singura parte variabilă a programului este **n**). Dacă executăm programul cu o intrare **n** foarte mare, atunci **n** va fi mai mare decât numărul de biți necesari pentru a scrie programul: programul se va genera pe sine şi la **Pasul 4** vom obține o contradicție.

---

[1] **Alan M. TURING** (1912–1954), matematician englez, considerat părintele informaticii. A introdus noțiunea de „maşina Turing", unul din principalele modele teoretice de calcul discret. A inventat „testul Turing" care a fost folosit în inteligența artificială pentru a decide dacă există maşini care gândesc.

[2] **Gregory CHAITIN** (n. 1947), matematician şi informatician american la IBM, New York, unul dintre întemeietorii teoriei algoritmice a informației. A obținut rezultate importante privind şirurile aleatoare şi limitele formale ale matematicii.

Anexa 2

FUNCȚII PREDEFINITE ÎN PASCAL

Nr	Nume	Parametri	Tipul funcției	Returnează
1	**odd**	un întreg	**boolean** valoarea **True**, dacă parametrul este impar	**odd**(7) = **True**; **odd**(8) = **False**
2	**abs**	un real sau un întreg	**real** sau **integer** valoarea absolută a parametrului	**abs**(3) = 3; **abs**(-3) = 3
3	**sqr**	un real sau un întreg	**real** sau **integer** pătratul parametrului	**sqr**(2.5) = 6.25
4	**sqrt**	un real sau un întreg	**real** rădăcina pătrată a parametrului	**sqrt**(2.56) = 1.6
5	**exp**	un real sau un întreg	**real** exponențiala de bază **e** a parametrului	**exp**(1) = e; **exp**(0) = 1 **exp**(5***ln**(2)) = 2^5
6	**ln**	un real sau un întreg	**real** logaritmul natural al parametrului	**ln**(e) = 1; **ln**(1) = 0; **ln**(**exp**(5.6)) = 5.6
7	**random**	un întreg pozitiv	n **integer** un număr aleator cuprins între 0 și n-1	***random***(18) = un întreg între 0 și 17
8	**random**	fără parametru **real**	un real din intervalul [0, 1)	
9	**sin**	un real sau un întreg	**real**	sinusul trigonometric al parametrului
10	**cos**	un real sau un întreg	**real**	cosinusul trigonometric al parametrului
11	**arctan**	un real sau un întreg	**real**	arctangenta trigonometrică a parametrului
12	**ord**	un tip ordinal	**integer** poziția parametrului în interiorul tipului	**ord**('a') = 97; **ord**('A') = 65

13	**chr**	un întreg	**char** caracterul corespunzător	**chr**(33) = '!'; **chr**(**ord**('x')) = 'x'; **ord**(**chr**(33)) = 33
14	**trunc**	un real	**integer** partea întreagă a parametrului	**trunc**(3.9) = 3; **trunc**(-4.2) = -4
15	**round**	un real	**integer** cel mai apropiat întreg de valoarea parametrului	**round**(3.9)=4; **round**(-4.2)= -5;

Atenție

Subprogramele predefinite se supun unor reguli diferite de cele care se referă la subprogramele definite de utilizatori; aceste subprograme sunt doar apelate (declararea și definiția lor nu trebuie inclusă în corpul programului apelant), numărul și tipul parametrilor poate varia de la un apel la altul etc.

ANEXA 3

PROCEDURI PREDEFINITE ÎN PASCAL

Nume	Parametri	Efect
write	oricâți și de orice tip	afișează valorile parametrilor pe linia curentă
writeln	oricâți și de orice tip	afișează valorile parametrilor pe linia curentă și o încheie, determinând saltul la o linie nouă
read	oricâți și de orice tip	citește valorile parametrilor de pe linia curentă
readln	oricâți și de orice tip	citește valorile parametrilor de pe linia curentă și la final o abandonează
randomize	niciun parametru	inițializează variabila de pornire a generatorului de numere aleatoare

Anexa 4

FUNCȚIILE PREDEFINITE ÎN C/C++

Funcțiile standard din ***C/C++*** se află în bibliotecile **stdio.h, conio.h, iostream.h, ctype.h, stdlib.h, math.h, complex.h, mem.h** și în alte aproximativ 30 de fișiere cu extensia **.h** .

Nr	Nume	Parametri	Biblioteca	Efect
1	**scanf**	oricâți și de orice tip	**stdio.h**	citește valorile parametrilor de pe linia curentă
2	**printf**	oricâți și de orice tip	**stdio.h**	afișează valorile parametrilor pe linia curentă
3	**random**	un întreg	**stdio.h**	returnează un intreg pozitiv strict mai mic decât parametrul
4	**pow**	doi reali sau doi complecși	**math.h** sau **complex.h**	ridică primul parametru la o putere dată de al doilea parametru
5	**sqrt**	un real sau un complex	**math.h** sau **complex.h**	returnează rădăcina pătrată a parametrului
6	**exp**	un real sau un complex	**math.h** sau **complex.h**	returnează exponențiala de bază **e** a parametrului
7	**ln**	un real sau un complex	**math.h** sau **complex.h**	returnează logaritmul natural al parametrului
8	**abs**	un întreg sau un complex	**stdlib.h, math.h** sau **complex.h**	returnează valoarea absolută a parametrului
9	**min, max**	doi întregi	**stdlib.h**	returnează minimum, maximum dintre parametri
10	**poly**	un întreg și cel puțin doi reali	**math.h**	calculează valoarea unui polinom de o variabilă, de grad și coeficienți dați

Anexa 5

SUBPROGRAME PREDEFINITE PENTRU PRELUCRAREA ȘIRURILOR DE CARACTERE ÎN PASCAL

Operația	*Funcția*	*Definiția funcției*
Lungimea șirului	***length***	**function length** (sursa: **string**): **integer;**
Concatenarea	***concat***	**function concat** (s1[,s2,…,sn]: **string**): **string;**
Căutarea unui subșir într-un sir de caractere	***pos***	**function pos (**subsir**: string;** sir**: string): byte;**
Extragerea unui subșir dintr-un șir de caractere	***copy***	**function copy** (sursa: **string**; pozitia: **integer**; numar: **integer**): **string**;
Inserarea unui subșir într-un șir de caractere	***insert***	**procedure insert (**sursa**: string; var** destinatia**: string;** pozitia**: integer);**
Ștergerea unui subșir dintr-un șir de caractere	***delete***	**procedure delete (var** sursa**: string;** pozitia**: integer;** numar**: integer);**
Conversia unei valori numerice într-o secvență de caractere	***str***	**procedure str (**numar:i; **var** sursa:**string**);
Conversia unei secvențe de caractere într-un număr	***val***	**procedure val (**sursa**; var** v: **integer; var** cod**: integer);**
Conversia literelor mici în majuscule	***upcase***	**function upcase (**car**: char): char;**

Anexa 6

FUNCȚII PREDEFINITE PENTRU PRELUCRAREA ȘIRURILOR DE CARACTERE ÎN C/C++

Operația	*Funcția*	*Definiția*
Atribuirea	***strcpy***	**char *strcpy** (**char *** destinație, **char *** sursa);
Compararea	***strcmp***	**int strcmp (const char *** sir1**, const char *** sir2**);**
Lungimea șirului	***strlen***	size_t **strlen** (**char *** sursa);
Concatenarea	***strcat***	**char *strcat** (**char *** destinație, **char *** sursa);
Inversarea ordinii caracterelor într-un șir	***strrev***	**char *strrev (char *** s**);**
Căutarea unui subșir într-un șir de caractere	***strstr***	**char *strstr (char *** sir**, char *** subsir**);**
Conversia unei valori numerice într-o secvență de caractere	***itoa***	**char *itoa (int** valoare, **char *** sir, **int** baza**);**
Conversia unei secvențe de caractere într-o valoare numerică	***atol***	**long atol (char *** sursa**);**
Conversia literelor mici în majuscule	***strupr***	**char *strupr (char *** sursa**);**
Conversia literelor mari în litere mici	***strlwr***	**char *strlwr (char *** s**);**

Anexa 7

TABELA CODURILOR ASCII

Zecimal	Hexazecimal	Caracter	Zecimal	Hexazecimal	Caracter	Zecimal	Hexazecimal	Caracter	Zecimal	Hexazecimal	Caracter
0	0	NUL	32	20		64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	“	66	42	B	98	62	b
3	3	ETX	35	23	-	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	‘	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	TAB	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	FF	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SO	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	„	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	“	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	

BIBLIOGRAFIE SELECTIVĂ

CRISTEA V., ATANASIU I., KALISY E., IORGA V. — *Tehnici de programare*, Ed. Teora 1999.

GREGORY J. CHAITIN — *Information, Randomness and Incompleteness*, World Scientific, 1987.

CRISTIAN J. CLAUDE — *Information and Randomness*, Springel Verlag, Berlin, 2002.

CRISTIAN A. GIUMALE — *Introducere în analiza algoritmilor*, Ed. Polirom 2004.

IOSIF IGNAT, CLAUDIA-LAVINIA IGNAT — *Programarea calculatoarelor, descrierea algoritmilor și fundamentele limbajului C/C++*, Editura Albastră, Cluj-Napoca, 2002.

CORNELIA IVASC, MONA PRUNĂ — *Bazele Informaticii* (Grafuri și elemente de combinatorică) manual pentru clasa a X-a, Editura Petrion, București, 1995.

KNUTH, D.E. — *Arta programării calculatoarelor* vol. I, Ed. Teora, 1999.

L. LIVOVSCHI, H. GEORGESCU — *Bazele informaticii, algoritmi* – U.B. 1985.

PĂTRĂȘCOIU O., MARIAN G., MITROI N. — *Elemente de grafuri și combinatorică*, Ed. All, 1994.

GHEORGHE PĂUN — *Din spectacolul matematicii*, Editura Albatros, București, 1983.

DORIAN STOILESCU — *Culegere de **C/C++;*** Editura Radial, Galați, 1998.

I. TOMESCU — *Bazele informaticii*, Manual pentru licee/clase de Informatică clasa a X-a, Editura Didactică și Pedagogică, București, 1996.

GREGORY F. WETZEL, WILLIAM G. BULGREN — *Pascal and Algorithms; An Introduction to Problem Solving*, Science Research Associates Inc., Chicago, 1987.

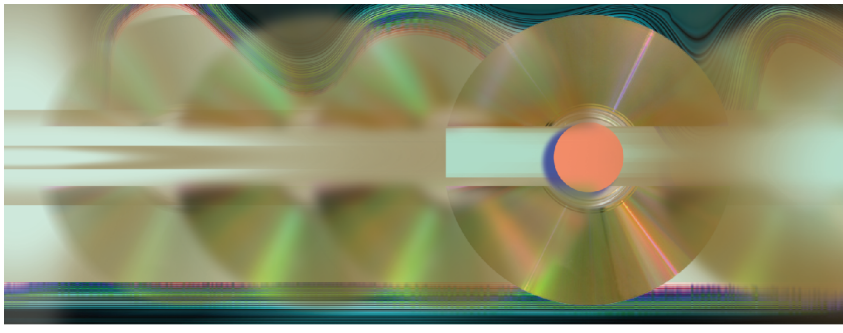
CUPRINS

Mioara Gheorghe
(coordonator)

Monica Tătărâm
Corina Achinca
Constanța Năstase

Informatică

Manual pentru clasa a XI-a



Filiera teoretică, profil real, specializarea matematică-informatică
Filiera vocațională, profil militar MApN, specializarea matematică-informatică

CORINT EDUCAȚIONAL

ISBN: 978-973-135-353-1