INFORMATICĂ

MATEMATICĂ

TOATĂ MATERIA STRUCTURATĂ

Învață cu succes

TOATĂ MATERIA PENTRU
BACALAUREAT LA INFO-MATE

ALL RIGHTS RESERVED ®
EDITURA ÎNVATĂ CU SUCCES

Ciobanu Timofi Flavio

Toată materia structurată la Informatică-Matematică

BACALAUREAT

Editura Învață cu Succes

Sfântu Ilie, 2024

Descrierea CIP a Bibliotecii Naționale a României
CIOBANU TIMOFI, FLAVIO
Toată materia structurată la Informatică-Matematică : bacalaureat / Ciobanu Timofi Flavio. - Sfântu Ilie : Învață cu Succes, 2024
ISBN 978-630-6641-02-4

004
51

Cuprins

1. Algoritmi Pseudocod / C++ .....pg. 5

- Noțiunea de algoritm, caracteristici
- Date, variabile, expresii și operații
- Structuri de bază (liniară, alternativă și repetitivă)
- Descrierea algoritmilor (programare pseudocod)
- Exerciții specifice

2. Elementele de bază ale limbajului de programare C++ .....pg. 13

- Vocabularul limbajului
- Noțiunea de tip de dată. Operatori aritmetici, logici, relaționali
- Variabile. Declararea variabilelor
- Structura programelor. Comentarii
- Expresii. Instrucțiunea de atribuire
- Citirea / scrierea datelor
- Exerciții specifice

3. Subprograme predefinite.....pg. 16

- Subprograme. Mecanisme de transfer prin intermediul parametrilor
- Proceduri și funcții predefinite

4. Tipuri structurate de date.....pg. 17

- Tipul tablou (vectori și matrici)
- Tipul șir de caractere
- Funcții predefinite pentru șirurile de caractere
- Tipul înregistrare (structuri)
- Exerciții specifice

5. Fișiere text.....pg. 28

- Fișiere text. Proceduri și funcții predefinite pentru fișiere text

6. Algoritmi elementari.....pg. 29

- Probleme care operează asupra cifrelor unui număr
- Divizibilitate. Numere prime. Algoritmul lui Euclid
- Determinare minim / maxim
- Metode de ordonare (metoda bulelor, inserției, selecției, numărării)
- Interclasare
- Metode de căutare (secvențială, binară)

- Analiza complexității unui algoritm (eficiență din punct de vedere al timpului de executare și memoriei utilizate)

7. Subprograme definite de utilizator.....pg. 38

- Proceduri și funcții (declarare și apel)
- Parametri formali și efectivi
- Parametri transmiși prin valoare și prin referință
- Variabile globale și locale
- Exerciții specifice

8. Recursivitate.....pg. 43

- Prezentare generală
- Proceduri și funcții recursive
- Exerciții specifice

9. Metoda backtracking.....pg. 46

- Prezentare generală
- Exerciții specifice

10. Generarea elementelor combinatoriale.....pg. 47

- Permutări, aranjamente, combinări
- Produs cartezian, submulțimi
- Exerciții specifice

11. Grafuri.....pg. 49

- Grafuri neorientate (terminologii, proprietăți, metode de reprezentare)
- Grafuri orientate (terminologii, proprietăți, metode de reprezentare)
- Arbori (terminologii, metode de reprezentare în memorie)
- Exerciții specifice

12. Subiecte de bacalaureat.....pg. 58

- Rezolvarea subiectului de bacalaureat la informatică, sesiunea iunie-iulie 2022
- Rezolvarea subiectului de bacalaureat la informatică, sesiunea august-septembrie 2022

Algoritmi Pseudocod / C++

- Noțiunea de algoritm, caracteristici
- Date, variabile, expresii și operații
- Structuri de bază (liniară, alternativă și repetitivă)
- Descrierea algoritmilor (programare pseudocod)

- Noțiunea de algoritm, caracteristici

1. Ce este un algoritm?

Un algoritm este o înșiruire de instrucțiuni care, îmbinate și executate corect, vor rezulta un anume lucru pe care chiar noi îl determinăm să-l facă.

Spre exemplu, învățatul pentru bacalaureatul la informatică necesită mai mulți pași pentru a putea fi eficient și corect, precum:

- deschizi cartea ÎnvațăCuSucces;
- alegi captiolul pe care doreești să-l studiezi;
- începi să scrii pe un caiet câteva notițe;
- iei o pauză de 10 minute;
- rezolvi câteva exerciții;
- ai terminat de învățat, acum ești gata să iei nota 10.

Pare puțin alambicat, nu? Hai să-ți dau un exemplu mai matematic.

Să presupunem că ai două necunoscute, x și y și dorești să le aduni în variabila z. (Ce este o variabilă? O să afli imediat). Cum faci? Ei bine...

- mai întâi afli cine este x;
- apoi cine este y;
- adunzi în z pe x și pe y;
- îl afișezi pe z;
- BOOM! Ai terminat algoritmul. Greu?

2. Caracteristicile unui algoritm:

Pentru ca un algoritm să fie definit corect și să poată funcționa, acesta trebuie să îndeplinească câteva condiții de bază, dintre care cele mai importante sunt:

- generalitatea: algoritmul trebuie să rezolve o anumită problemă, pe scurt, să facă ceva concret și util;

- claritate: un algoritm trebuie să fie clar și coerent, să nu aibă greșeli de scriere / exprimare (oricum are avea erori), dar nici să nu fie alambicat. Gândește-te la parcursul vieții. Una este să spui că mori, te căsătorești și te naști, iar alta că te naști, te căsătorești și mai apoi mori;

- finitudinea: algoritmul trebuie să se și termine, să nu ruleze infinit, căci asta o definim ca „buclă".

Hai că nu este chiar atât de greu. Ce zici?

- Date, variabile, expresii și operații

1. Tipuri de date:

Datele pot să fie de două tipuri. Date de intrare și date de ieșire.

Datele de intrare: Ce sunt acestea? Ei bine, datele de intrare sunt acele informații pe care noi le oferim calculatorului pentru a ne rezulta ceva ce noi ne dorim într-un final. Îți vorbeam adineauri despre cele două necunoscute, x și y. Ei bine, acestea sunt date de intrare. Cum așa? Păi, simplu. Tu cunoști pe x și pe y, întrucât tu le tastezi la calculator. Iar z? Da, ai ghicit, z este un tip de date de ieșire.

Datele de ieșire: Aceste tipuri de date sunt, de fapt, rezultatul pe care noi ni-l dorim, respectiv ceea ce ne rezultă algoritmul. În cazul de mai sus, z este dată de ieșire, căci calculatorul adună cele două numere x și y, ulterior afișând rezultatul prin intermediul variabilei z.

2. Variabile:

Variabilele reprezintă, în informatică, o mică „pârticică" de memorie care are ca rol menținerea de valori, valori ce pot fi ulterior modificate pe tot parcursul întregului program. Variabilele sunt caracterizate printr-un nume unic, alcătuit din litere mici și mari ale alfabetului englez, cifre, dar și unele simboluri.

Tipuri de variabile. Variabilele pot fi de mai multe tipuri, dintre care:

- reale: `float`, `double`, `long double`; (3.9, 34.89, 34546.23958)
- întregi: `int`, `long`, `long int`, `long long int`; (-39, 3489, -345462398)
- naturale: `unsigned int`, `unsigned long int`; (39, 3489, 345462398)
- șir de caractere: `char`; („acesta este un exemplu")
- booleane: `true`, `false`. (1 / 0)

Aceste variabile sunt cele mai de întâlnite și folosie în limbajul uzual C++. În pseudocod, lucrurile nu sunt așa complicate, astfel că le spunem lucrurilor pe nume. Aici există variabile: reale, întregi, dar și șir de caractere. Nimic mai simplu.

3. Expresii și operații:

Pentru a putea determina anumite calcule, spre exemplu, calculatorul efectuează anumite expresii, precum adunare, scădere, înmulțire, împărțire și așa mai departe. Aceste expresiiu se realizează cu ajutorul operatorilor aritmetici, aceștia fiind următorii:

- \+ pentru adunare
- – pentru scădere
- * pentru înmulțire
- / pentru împărțire

- % (`modulo`) pentru restul împărțirii
+ altele pe care le vom discuta în lecțiile viitoare

Deci, dacă doresc să adun două numere, voi folosi semnul +, astfel: 2+5, iar dacă doresc să împart două numere, voi folosi semnul /, astfel: 12/6.

Ce este așa special cu %? Ei bine, în C++, % reprezintă restul împărțirii numărului x la y, de fel un rezultat întreg.

Să presupunem că x și y sunt variabile de tip întreg, adică 13, -56, 24, șamd, iar tu vrei să afli restul împărțirii lui 13 la 10. Ei bine, știm că 13/10 este 1.3, iar noi dorim restul împărțirii, adică 3. Dar în cazul în care aș vrea să calculez 6%3? Păi 6/3 este 2, deci restul este 0. Dar 17%3? Restul împărțirii se calculează mai diferit, hai să-ți arăt cum:

- 3 de câte ori intră în 17? De 5 ori (5 * 3 = 15)
- 17 - 15 = 2, deci restul devine 2, restul nu ne mai interesează

Acum ia spune-mi, cât fac 18%4? Dar 101%5?

În C++, anumite simboluri din pseudocod se traduc, întrucât nu poți reda ≤ sau ≥ în C++, astfel că este nevoie de simbolurile <= și >=.

Pseudocod:	C++:
< - mai mic	< - mai mic
> - mai mare	> - mai mare
≤ - mai mic sau egal	<= - mai mic sau egal
≥ - mai mare sau egal	>= - mai mare sau egal
`și` - și logic	`&&` - și logic
`sau` - sau logic	`||` - sau logic
`NOT` - negare	`!` - negare

Negările se realizează prin intermediul expresiei NOT (în pseudocod) și ! (în C++). Ce este aceea o negare? Negările reprezintă opusul expresiilor deja date.

Exemplu de negări?

Pseudocod:	C++:
NOT(x și y) -> x sau y	!(x && y) -> x || y
NOT(x < y) -> x ≥ y	!(x < y) -> x >= y
NOT(x ≥ y) -> x < y	!(x >= y) -> x < y
NOT(x sau y) sau z -> (x sau y) sau z	!(x || y) || z -> (x || y) || z

Pentru a evalua o expresie în C++, vom folosi semnul ==, spre exemplu pentru verificarea unui număr dacă este par: x % 2 == 0? Dacă da, atuunci x este par, altfel, x este impar.

- Structuri de bază (liniară, alternativă și repetitivă)

1. Structura liniară:

Structura liniară reprezintă o înșiruire de atribuiri / expresii / afișări care pot avea sau nu legătura una cu alta.

Exemplu Pseudocod:

```
întreg a, b, c
a ← 5
citește b
c ← (a + b) / 2
scrie c
```

Exemplu C++:

```
int a, b, c;
a = 5;
cin >> b;
c = (a +b ) / 2;
cout << c;
```

2. Structura alternativă (dacă / if):

Această structură presupune executarea unor instrucțiuni / operații dacă și numai dacă condiția este îndeplinită. În cazul în care condiția nu este îndeplinită, vom trece la următoarea operație, dacă există, altfel, se părăsește complet structura.

Exemplu Pseudocod:

```
întreg a, b
citește a, b
┌dacă a % 2 = 0 și b % 2 = 1, atunci
│     scrie "a este par, b este impar"
│  altfel
│     scrie "altceva"
└
```

Exemplu C++:

```
int a, b;
cin >> a >> b;
if(a % 2 == 0 && b % 2 == 1)
    cout << "a este par, b este impar";
  else
    cout << "altceva";
```


3.1. Structura repetitivă cu test inițial (cât timp... execută / while)

Această structură face ca setul de instrucțiuni / operații să se execute cât timp o condiție este îndeplinită. În cazul în care condișia nu mai este îndeplinită, atunci pur și simplu se oprește structura și se trece mai departe.

Exemplu Pseudocod:

```
întreg n, i, S
i ← 1; S ← 0
citește n
┌cât timp i ≤ n execută
│     S ← S + i
│     i ← i + 1
└
scrie S
```

Exemplu C++:

```
int n, i, S;
i = 1; S = 0;
cin >> n;
while(i <= n)
{   S = S + i;
    i = i + 1;
}
cout << S;
```

3.2. Structura repetitivă cu test inițial (pentru... execută / for)

Acest tip de structură este mult mai special decât ai crede, întrucât conține o varibilă care pornește dintr-un punct și ajunge într-un altul, fie de la x la y, fie de la y la x, unde x ≤ y. Atunci când se ajunge ca x să fie strict > y, structura se va sfârși.

Exemplu Pseudocod:

```
întreg n, i, S
S ← 0
citește n
┌pentru i ← 1, n, execută SAU pentru i ← n, 1, -1, execută
│     S ← S + i
└
scrie S
```

Exemplu C++:

```
int n, i, S;
S = 0;
cin >> n;
for(i = 1; i <= n; i++) SAU for(i = n; i >= 1; i--)
```

```
    S = S + i;
cout << S;
```

3.3. Structura repetitivă cu test final (`execută... cât timp / do... while`)

Această structură permite algoritmului ca mai întâi să execute cel puțin o dată setul de instrucțiuni / operații, iar mai apoi să-ți pună întrebarea dacă condiția este îndeplinită. În caz afirmativ, algoritmul se repetă, altfel se trece la următorul pas al algoritmului.

Exemplu Pseudocod:

```
întreg n, i, S
i ← 1; S ← 0
citește n
┌execută
│    S ← S + i
│    i ← i + 1
└cât timp i ≤ n
scrie S
```

Exemplu C++:

```
int n, i, S;
i = 1; S = 0;
cin >> n;
do
{   S = S + i;
    i = i + 1;
}while(i <= n);
cout << S;
```

3.4. Structura repetitivă cu test inițial (`repetă.... până când`)

Ultima structură repetitivă este întâlnită numai în limbajul pseudocod, întrucât în C++ se consideră drept do... while. Se comporta ca structura anterioară, însă vom repeta până când condiția nu se mai execută.

Exemplu Pseudocod:

```
întreg n, i, S
i ← 1; S ← 0
citește n
┌repetă
│    S ← S + i
│    i ← i + 1
└până când i > n
scrie S
```

La folosirea acestei expresii este importantă NEGAREA. Aici punem în aplicare ceea ce am învățat la expresii și operații.

- Descrierea algoritmilor (programare pseudocod

După cum probabil ați sesizat și mai sus, programarea pseudocod are câteva aspecte importante:

┌ reprezintă semnul de deschidere al unei instrucțiuni, indiferent dacă are una sau mai multe expresii / operații

│ reprezintă continuarea structurii, adică tot ceea ce regăsim în structura de bază (expresii, operații sau chiar alte structuri)

└ reprezintă finalul structurii

← reprezintă semnul de atribuire a unei valori către o variabilă

Programarea pseudocod este una simplă și nu necesită multe lucruri de învățat până ca totul să prindă mai mult sens. Mai jos veți regăsi un program complet împreună cu cerința acestuia. Studiați algoritmul cât mai amănunțit și vedeți care sunt principalele caracteristici ale programării pseudocod.

Cerință: Să se citească n numere naturale și să se afișeze pe ecran câte dintre ele sunt pare.
Rezolvare:

```
întreg n, număr, i, contor
citește n
contor ← 0
┌pentru i ← 1, n execută
│    citește număr
│    ┌dacă număr % 2 = 0 atunci
│    │ contor ← contor + 1
│    └
└
scrie contor
```

Ce am făcut mai sus? Ei bine, am citit un număr n care reprezintă numărul efectiv de numere, apoi fiecare număr în parte. În cazul în care restul împărțirii numerelor citite la 2 este 0, adică este par, în contor am adăugat numărul 1, deci am contorizat câte numere pare există. La final, am afișat variabila contor care, evident, avea memorat număr de numere pare.

- Exerciții specifice

1.

SUBIECTUL al II-lea **(40 de punc**

1. **Algoritmul alăturat este reprezentat în pseudocod.** S-a notat cu `a%b` restul împărțirii numărului natural `a` la numărul natural nenul `b` și cu `[a]` partea întreagă a numărului real `a`.

   a. Scrieți ce se afișează în urma executării algoritmului dacă se citesc, în această ordine, numerele `6, 16, 4273, 1679, 3165, 16, 200, 167`. **(6p.)**

   b. Dacă primele două numere citite sunt, în această ordine, `2` și `22`, scrieți o secvență de numere distincte din intervalul $[10^3, 10^4)$ care pot fi citite în continuare astfel încât, în urma executării algoritmului, să se afișeze valoarea `2`. **(6p.)**

   c. Scrieți programul C/C++ corespunzător algoritmului dat. **(**

   d. Scrieți în pseudocod un algoritm, echivalent cu cel dat, înlocuind adecvat prima struc `cât timp...execută` cu o structură repetitivă de tip `pentru...execută`. **(**

```
citește n,x (numere naturale nenul
nr←0; i←1
┌cât timp i≤n execută
│ citește y (număr natural)
│ ┌cât timp y≠0 și y%2≠x%2 exec
│ │ y←[y/10]
│ └■
│ ┌dacă y=x atunci nr←nr+1
│ └■
│ i←i+1
└■
scrie nr
```

- Simulare Bacalaureat 2022

Algoritmul proiectat determină câte numere încep cu cifrele 1 și 6.

a. 3

b. Orice număr de forma 22xy, unde x și y sunt cifre impare.

c.

```
#include <iostream>
using namespace std;
int main()
{
    int n, x, y, nr, i;
    cin >> n >> x;
    nr = 0; i = 1;
    while(i <= n)
    {
        cin >> y;
        while(y != 0 && y % 2 != x % 2)
            y = y / 10;
        if(y == x)
            nr++;
        i++;
    }
    cout <<nr;
    return 0;
}
```

d.

```
citește n,x (numere naturale nenule)
nr<-0; i<-1
┌pentru i<-1,n execută
│   citește y
│   ┌cât timp y≠0 și y%2≠x%2 execută
│   │   y<-[y/10]
│   └■
│   ┌dacă y=x atunci nr<-nr+1
│   └■
└■
scrie nr
```


2. Elementele de bază ale limbajului de programare C++

- Vocabularul limbajului
- Noțiunea de tip de dată. Operatori aritmetici, logici, relaționali
- Variabile. Declararea variabilelor
- Structura programelor. Comentarii
- Expresii. Instrucțiunea de atribuire
- Citirea / scrierea datelor

- **Vocabularul limbajului**

Limbajul pe care noi îl studiem este C++, iar pentru ca algoritmii noștii să funcționeze, aceștia trebuie să fie și corecți.

Cum începe de fapt un program C++? Ei bine, începe prin a include un fișier standard sau definit de utilizator. Noi vom folosi doar fișiere standard, anume `#include <iostream>`. Vom putea adăuga și alte fișiere, sau mai bine zis biblioteci, însă vom discuta despre ele puțin mai târziu.

După acest lucru, va trebui să adăugăm `using namespace std`. La ce ne ajută asta? Ei bine, când efectuăm un apel la utilizarea spațiului de nume, toate acele simboluri din spațiu vor deveni invizibile, fără a fi nevoie să adăugăm prefixul spațiului de nume.

Apoi vom adăuga funcția principală, adică `int main(){` pe care o deschidem, iar la final o închidem, nu înainte de a da `return 0;` pentru ca programul nostru să nu ruleze infinit sau alte erori de genul.

Deci, pe scurt, cum arată un program?

```
#include <iostream>
using namespace std;
int main()
{
     //programul tău
     return 0;

}
```

- Noțiunea de tip de dată. Operatori aritmetici, logici, relaționali

Ca și în pseudocod, lucrurile stau la fel. Tipurile de date sunt destul de numeroase, cele mai des întâlnite fiind `int`, `float`, `char` și `unsigned int`. Negările se realizează cu ajutorul simbolului !.

- Structura programelor. Comentarii

Un program este corect dacă și numai dacă îndeplinește câteva condiții de bază. Vorbind despre un program care se introduce în calculator, acesta trebuie să fie corect tocmai pentru a nu exista erori. Până și simbolul ; uitat sau pus random, poate da peste cap întreg algoritmul.

Cum putem preveni acest lucru? Ei bine, simplu. Programul trebuie să aibă structura prezentată mai sus, să nu existe greșeli de scriere, să existe simbolul ; după fiecare instrucțiune / operație și să respecte cerința.

Comentariile se realizează cu ajutorul simbolului `//` urmat de text. Acesta permite scrierea a ce dorești fără a fi luat în seamă de calculator.

Atenție, acest simbol funcționează și este valabil doar pe o linie, iar orice este scris după nu se poate transforma în cod.

Pentru a pune comentarii pe mai multe rânduri este necesar folosirea simbolurilor `/*` și `*/`, între ele scriindu-se comentariul dorit, ca în exemplul de mai jos.

```
#include <iostream>
using namespace std;
int main()
{
    //comentariu 1
    int a;
    /*
    comentariu 2
    comentariu 3
    */
    return 0;
}
```

- Expresii. Instrucțiunea de atribuire

Pentru a atribui o valoare uni variabile, se folosește semnul =. Spre exemplu, dacă vrei ca variabila x să memoreze valoarea 5, este nevoie să scrii doar `x = 5;` și gata, acum variabila x are memorată valoarea 5.

Pentru a verifica o expresie, se folosește simbolul ==. Spre exemplu, dacă vrei să verifici dacă un număr este par, vei scrie `numar % 2 == 0`, nu = `0`, căci nu îi atribui valoarea 0 lui `numar % 2`. Atât de simplu.

În general, pentru verificarea acestor instrucțiuni se folosesc variabilele booleane true și false. Spre exemplu, `1 == 3` este `false`, pentru că 1 clar nu este egal cu 3, dar `3 >= 3` este `true`, căci 3 este evident ≥ decât 3.

- Citirea / scrierea datelor

Citirea și scrierea datelor se face extrem de simplu. Pentru a citi variabile se folosește funcția cin urmată de simbolul >>. Spre exemplu, `cin >> nr;`. Dacă se dorește citirea a mai multor variabile, se va continua cu semnul >> până ce vom termina ce dorim să citim, spre exemplu: `cin >> nr1 >> nr2 >> nr3 >> nrn;`.

La afișare se procedează exact la fel, întrucât nu necesită schimbări majore, decât utilizarea unei alte funcții, anume `cout`. Spre exemplu, `cout << nr` sau `cout << nr1 << nr2 << nr3 << nrn;`.

Pentru a afișa un spațiu, putem folosi simbolul `' '` sau `" "`, spre exemplu `cout << nr1 << ' ' << nr2;` sau `cout << nr1 << " " << nr2;`.

- Exerciții specifice

1.

1. Variabila `x` este de tip întreg. Indicați o expresie care are valoarea `1` dacă și numai dacă expresia C/C++ alăturată are valoarea `1`. | `x<=20 || x>22`

a. `!(x>20) || !(x<=22)`
b. `!(x>=20) && !(x>=22)`
c. `!(x<20 || x<=22)`
d. `!(x<20 && x<22)`

- Model Bacalaureat 2022

Vom lua fiecare variantă și vom rezolva negările:

a. `x <= 20 || x > 22`
b. `x < 20 && x < 22`
c. `x >= 20 && x > 22`
d. `x >= 20 || x >= 22`

Astfel varianta corectă rămâne a.

2.

1. Variabila `x` este de tip întreg. Indicați o expresie care are valoarea `1` dacă și numai dacă expresia C/C++ alăturată are valoarea `1`. | `x<=3 || x>30`

a. `!(x<3) && x<30`
b. `!(x>=3 && x<=30)`
c. `!(x<3 || x<=30)`
d. `!(x>3) || !(x<=30)`

- Simulare Bacalaureat 2022

Vom face la fel ca mai sus:

a. `x >=3 && x < 30`
b. `x < 3 || x > 30`
c. `x >= 30 && x > 30`
d. `x <= 3 || x > 30`

Astfel, varianta corectă rămâne d.

3. Subprograme predefinite

- Subprograme. Mecanisme de transfer prin intermediul parametrilor
- Proceduri și funcții predefinite

- Subprograme. Mecanisme de transfer prin intermediul parametrilor

Pentru început, trebuie să înțelegem ce este acela un subprogram. Ei bine, un subprogram reprezintă o funcție deja definită de calculator sau de utilizator care ne ajută să nu fim nevoiți în a repeta aceeași parte de două sau mai multe ori într-un program. Pe scurt, putem creea, spre exemplu, un subprogram care determină maximul dintre două numere, putând să-l apelăm ori de câte ori este nevoie, fără a scrie din nou întreaga structură alternativă. Apelul se face prin intermediul parametrilor, aceștia fiind de două tipuri: de intrare / de ieșire, însă, în cazul funcțiilor predefinite vom folosi doar parametrii de intrare, adică ceea ce deja știm.

- Proceduri și funcții predefinite

Ce funcții predefinite cunoaștem? Ei bine, sunt o grămadă, iar cele pe care le folosim noi sunt, majoritatea, funcțiile șirurilor de caractere pe care ți le voi prezenta în lecția viitoare. Însă, până atunci, îți voi prezenta câteva funcții predefinite care ne pot ajuta în rezolvarea problemelor.

`cout << max(a, b);` - afișează maximul dintre a și b

`cout << min(a, b);` - afișează minimul dintre a și b

`swap(a, b);` - inversează valorile dintre a și b (a devine b, iar b devine inițialul a)

`break;` - oprește prima instrucțiune repetitivă pe care o întâlnește, indiferent dacă se mai poate sau nu îndeplini condiția de bază

(!) Nu se recomandă utilizarea lor în examenul de bacalaureat, întrucât sunt niște funcții mai speciale, iar unii profesori pot considera că v-a fost lene să faceți algoritmul complet.

Funcții matematice predefinite:

Pentru a utiliza fără greșeli funcțiile matematice, este nevoie să introducem bibliotecha `#include <cmath>` care conține funcțiile matematice, cele mai importante și întâlnite fiind:

`pow(x, y);` - ridici x la puterea y

`sqrt(x);` - radical din x

`abs(x);` - absolut / modul din x

`ceil(x);` - rotunjești pe x la cel mai apropiat întreg, mai mare decât x

`floor(x);` - rotunjești pe x la cel mai apropiat întreg, mai mic decât x

Cum scriem corect o funcție matematică? Simplu. Ținem cont de funcțiile predefinite și punem parantele acolo unde trebuie.

Exemplu, pentru expresia alăturată: $\frac{x^2+y}{\sqrt{z}+5}$.

```
((pow(x,2) + y) / (sqrt(z) + 5));
```

Simplu, scurt și la obiect. Numărătorul și numitorul se pun în paranteze pentru a nu exista ulterior erori, iar restul parantezelor vin de la sine, după caz.

4. Tipuri structurate de date

- Tipul tablou (vectori și matrici)
- Tipul șir de caractere
- Funcții predefinite pentru șirurile de caractere
- Tipul înregistrare (structuri)
- Exerciții specifice

- Tipul tablou (vectori și matrici)

1. Vectori (Tablouri Unidimensionale):

Ce sunt vectorii? Ei bine, acestea sunt structuri de date bine definite ce pot păstra în memorie mai multe variabile de diferite tipuri (întregi, reale, char, șamd) ce pot oricând utilizate, modificate sau șterse.

Cum declarăm un vector?

```
tip_dată nume_vector [dimensiune_ maximă];
```

Spre exemplu, `int V[100]`, sau chiar `float Vector[15]`.

De asemenea, putem defini un vector ca fiind gol folosind `int V[100] = {0};` sau dacă știm anumite valori îl putem defini astfel `int V[5] = {15, -3, 5, 0, 25};`, astfel valoarea 15 se va afla pe poziția 1, valaorea -3 pe poziția 2 și așa mai departe.

Vectrorii sunt extrem de utili, întrucât ne ajută să gestionăm mai bine anumite valori pe care le folosim pe moment sau mai târziu, fără ca aceste valori să fie afectate și fără să fim nevoiți să folosim multe variabile, evident inutile.

Cum citim un vector? Ei bine, fiecare valoare memorată într-un vector are o poziție alocată, începând cu 0 până la dimensiunea maximă - 1.

Dacă vom considera vectorul de mai sus, pe poziția 0 se poate afla o valoare, pe poziția 1 o altă valoare și așa mai departe. De asemenea, nu este necesar să începem din 0, căci de fel se pornește din 1, însă dacă începem din 1, vom defini vectorul cu dimensiunea maximă + 1 valori.

Spre exemplu, int V[101] va avea 100 de valori, dacă noi dorim să memorăm prima valoare din 1.

Dec, cum facem citirea?

```
int V[101], i;
for(i = 1; i <= 100; i++)
     cin >> V[i];
```

Atât. Simplu, nu?

Dar afișarea cum se face? Ei bine, la fel de simplu.

```
for(i = 1; i <= 100; i++)
     cout << V[i] << " ";
```

Totuși să nu uităm și de spațiu, este foarte important.

Ce este un vector de frecvență? Ei bine, un vector de frecvență este acela a cărei valoare de pe poziția i arată numărul de apariții ale lui i într-o anumită entitate.

Spre exemplu, pentru numărul 14568892784, vectorul de poziții poate arăta așa :

i	0	1	2	3	4	5	6	7	8	9
Nr_apar.	0	1	1	0	2	1	1	1	3	0

Ce putem face totuși cu vectorii ? O grămadă de lucruri. Aruncă o privire la următoarea lecție (după fișiere text) și vei vedea acolo algoritmii elementari.

2. Matrici (Tablouri Bidimensionale):

Ce sunt matricile? Matricile sunt ceva mai speciale, întrucât pot memora valori pe linii și pe coloană. Practic, matricile suntr o în șiruire de mai multe tablouri unidimensionale (vectori) pe două direcții (orizontal și vertical).

Știa că sudoku este, de fapt, o matrice?

Cum declarăm o matrice?

```
tip_dată
nume_matrice[dimensiune_maximă_linii][dimensiune_
maximă_coloane];
```

Spre exemplu, `int M[100][100];` sau chiar `float Matrice[50][30];`

Putem defini o matrice goală folosind `int M[100][100] = {0}`, spre exemplu, iar dacă știm liniile și coloanele, putem face astfel:

```
int M[3][3] = {
      {1,2,3},
      {2,3,4},
      {3,4,5}
};
```

Cum citim o matrice? Ei bine, este puțin mai complicat, căci necesită două structuri repetitive, una pentru linie și una pentru coloană. Citirea se face linie cu linie, de la stânga la dreapta, astfel:

```
int M[101][101], i, j;
for(i = 1; i <= 100; i++)
     for(j = 1; j <= 100; j++)
          cin >> M[i][j];
```

Atât de simplu. Dar afișarea? Ei bine, aici este ceva mai complicat, întrucât trebuie să afișăm linie cu linie și coloană cu coloană, liniile să fie afișate pe rânduri diferite, iar coloanele cu spații între ele, astfel:

```
for(i = 1; i <= 100; i++)
{
     for(j = 1; j <= 100; j++)
          cout << M[i][j] << " ";
     cout << endl;
}
```

Matricile sunt puțin mai complexe, astfel că este nevoie să discutăm și despre diagonale și elementele aflate între acestea.

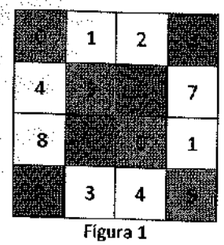


Figura 1

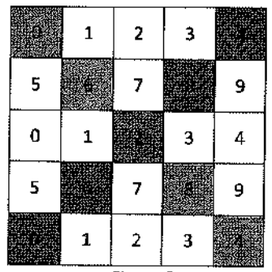


Figura 2

În Figura 1 și 2 cu verde este prezentată diagonala principală, iar cu albastru diagonala secundară. Se observă în Figura 2 căsuța roșie care reprezintă atât diagonala principală cât și secundară, specifică matricilor pătratice de ordin impar.

(!) Aceste diagonale se pot determina numai la matricile pătratice.

Cum determinăm diagonala principală? Ei bine, aici avem un singur caz.

`i == j`

Dacă indicii liniei și coloanei sunt identici, atunci ne aflăm pe o diagonală principală indiferent dacă pornim din 0 sau 1.

Cum determinăm diagonala secundară? Aici lucrurile se complică, având 2 cazuri posibile:

`i + j == n + 1` - pentru atunci când pornim cu indicii din 1

`i + j == n - 1` - pentru atunci când pornim cu indicii din 0

unde n este ordinul matricei

Mai există și posibilitatea de a determina elementele strict deasupra diagonalei principale:

`i < j`

sau elementele strict dedesubtul diagonalei principale:

`i > j`

sau elementele strict deasupra diagonalei secundare:

`i + j < n + 1` - pentru atunci când pornim cu indicii din 1

`i + j < n - 1` - pentru atunci când pornim cu indicii din 0

sau elementele strict dedesubtul diagonalei secundare:

`i + j > n + 1` - pentru atunci când pornim cu indicii din 1

`i + j > n - 1` - pentru atunci când pornim cu indicii din 0

De asemenea, putem determina cu ușurință și elementele dintre diagonale, partea de sus:

`i < j && i + j < n + 1` - pentru atunci când pornim cu indicii din 1

`i < j && i + j < n - 1` - pentru atunci când pornim cu indicii din 0

sau elementele dintre diagonale, partea de jos:

`i > j && i + j > n + 1` - pentru atunci când pornim cu indicii din 1

`i > j && i + j > n - 1` - pentru atunci când pornim cu indicii din 0

- Tipul șir de caractere

Ce sunt șirurile de caractere mai exact? Ei bine, șirurile de caractere reprezintă, de fapt, niște tablouri unidimensionale care conțin caractere în loc de numere.

Pe scurt, stocăm niște cuvinte / propoziții într-un vector care memorează rând pe rând anumite litere, cifre, dar și caractere.

Trebuie să știți că la șirurile de caractere, numerotarea începe de fiecare dată de la 0, deci nu există excepții în acest caz.

Fiecare șir de caractere se termină cu caracterul `\0` sau așa zisul caracter `NULL` ce permite calculatorului să știe când trebuie să se oprească.

Un alt lucru important este de reținut că trebuie să folosim bibioteca `#include <cstring>` atunci când folosim șiruri de caractere, pentru a putea folosi funcțiile predefinite acestui șir.

Și totuși, cum definim un șir de caractere? Simplu, însă există mai multe tipuri de șiruri de caractere.

1. Un simplu cuvânt. Acesta se definește astfel: `tip_dată nume[dimensiune_maximă];`, spre exemplu `char cuvânt[15];`, adică am definit un șir de caractere care memorează un cuvânt ce are numele cuvânt cu maxim 15 caractere.

2. Un șir format din mai multe caractere. Acesta se definește ca la vectori, atunci când știm exact ce caractere dorim să folosim.

(!) Atenție! Caracterele se pun în apostrof (' '), iar cuvintele în ghilimele (" ").

Deci, îl definim astfel: tip_dată nume[număr_maxim_elemente] = {'a', 'b', 'c', 'z'};, spre exemplu `char caractere[7] = {'a', 'b', 'c', 'd', 'e', 'f', 'g'};`, adică am definit un șir de caractere ce conține 7 caracte=ere, cel de pe poziția 0 fiind a, cel de pe poziția 1 fiind b și așa mai departe.

3. Un șir de caractere format doar din cuvinte. Acesta este cu totul special, întrucât se transformă la aspect ca o matrice, însă este ceva mai special de atât. Aici sunt necesare două lucruri: numărul maxim de cuvinte și numărul maxim de caractere ale cuvintelor, deci îl vom defini astfel: tip_dată nume[număr_maxim_cuvinte][caractere_maxime_cuvinte] = {"cuvant1", "cuvant2", "cuvantn"};, deci, pe scurt, va arăta așa: `char cuvinte[3][7]={ "Invata", "cu", "Succes"};`, astfel că am definit un șir de caractere cu 3 cuvinte, iar fiecare dintre aceste 3 cuvinte având maxim 7 caractere (în cazul de față 6 + caracterul NULL = 7), unde Invata este cuvântul de ordin 0, cu este cuvântul de ordin 1, iar Succes cuvântul de ordin 2.

Greu? Destul de simplu, zic eu. Ce mai trebuie să știi aici este codul ASCII. Acest reprezintă o codificare a fiecărui caracter aflată în spatele a ce vedem pe ecran. Ce trebui să știi aici este că majusculele sunt înainte de minuscule, litera A având codul 65, iar liter a având codul 97, astfel că diferența `'a' - 'A'` devine 32.

Totodată, cifra 0 are codul 48, iar cifra 9 are codul 57.

- Funcții predefinite pentru șirurile de caractere

Funcțiile predefinite sunt cele care ne ușurează munca de cele mai multe ori. Ai surprins să afli că la examenul de bacalaureat, cele mai dese funcții predefinite pe car chiar avem voie să le folosim sunt cele ale șirurilor de caractere.

1. Citirea unui șir de caractere. Aici există 2 posibilități.

1.1 Dacă dorim să citim un simplu cuvânt, atunci vom folosi funcția cin, deci `cin >> sir;`

1.2 Dacă dorim să citim un șir care are mai multe cuvinte, inclusiv spații, vom folosi funcția cin.get(nume_șir, număr_maxim_caractere_pe_care_le_citim); astfel: `cin.get(sir, 100);`, deci am citit șirul cu numele sir, însă maxim 100 de caractere din totalul pe care îl are (acesta putând să fie 100 sau mai mult, dar niciodată mai puțin)

2. Afișarea unui șir de caractere. Scurt, simplu și la obiect, `cout << nume_șir;`, nimic mai greu

3. Lungimea unui șir: `strlen(nume_șir);`

4. Transformarea din literă mică / mare în literă mare / mică. Adineauri îți vorbeam de codul ASCII. Știm că 'a' are codul 97, iar 'A' are codul 65, deci dacă am face o simplă scădere, anume 'a' - 32, am ajunge la 'A'. Atât de simplu. La fel se procedează și invers, nimic mai greu.

5. Coplerea unui șir într-altul. Pentru a face acest lucru se utilizează funcția strcpy(nume_șir, noul_șir);, putând copia un alt șir deja existent sau un șir de caractere predefinit de noi). Spre exemplu:

`strcpy(sir1, sir2);` sau `strcpy(sir, "Informatica");`

(!) Mare atenție, o dată copiat un șir peste celălalt, inițialul va dfispărea complet.

6. Lipirea unui șir de altul, sau cum se mai spune, procedeul de concatanare. Pentru acestul lucru vom folosi funcția strcat, care funcționează ca mai sus, însă fără a dispărea cuvântul inițial, ci doar lipindu-l, deci:

`strcat(sir1, sir2);` sau `strcat(sir1, "Informatica");`

7. Compararea a două șiruri. Ce mai înseamnă și asta? Ei bine, putem compara șirurile de caractere tocmai pentru că, având coduri, calculatorul știe dacă un cuvânt este mai „mic" sau mai „mare" decât celălalt. Ana este evident mai „mic" decât Bianca, întrucât A < B. Funcția este:

`strcmp(sir1, sir2);` și nu face diferența dintre majuscule și minuscule

dacă se dorește această diferență, se va folosi

`stricmp(sir1, sir2);`

8. Transformarea unui întreg șir format din litere mici în litere mari:

`strupr(sir);`

9. Transformarea unui întreg șir format din litere mari în litere mici:

`strlwr(sir);`

10. Inversarea unui șir. Asta înseamnă că dacă eu am cuvântul informatica, inversul acestuia este acitamrofni.

`strrev(sir);`

Funcții predefinite speciale: Am ajuns la această parte unde regăsim cele mai importante funcții predefinite pe care le folosim în exerciții. Înainte de acestea, trebuie să știi ce sunt pointerii.

Pointerii reprezintă date ale căror valori sunt adrese de memorie. Aceștia se definesc astfel: tip_dată * nume, deci putem găsi, spre exemplu, `int * pointer;`.

1. Căutarea unui caracter într-un șir de caractere:

`strchr("caractere", sir);`

Această funcție întoarce, pe scurt, un pointer către prima apariția al unuia dintre caracterele aflate între ghilimele în cazul în care se găsesc în șirul dat.

`strrchr("caractere", sir);`

Această funcție întoarce, pe scurt, un pointer către ultima apariția al unuia dintre caracterele aflate între ghilimele în cazul în care se găsesc în șirul dat.

2. Căutarea unui șir de caractere într-un alt șir de caractere:

Dacă dorim să căutăm un șir, nu un caracter, într-un alt șir, această funcție ne va fi de folos, întrucât funcția întoarce un pointer către prima apariție a șiruiui pe care îl căutăm în șirul în care caut:

`strstr(sir_in_care_caut, sir_pe_care_il_caut);`

3. Furnizarea unui pointer drept delimitator:

Dacă dorim să preluăm fiecare cuvânt dintr-un întreg șir, rând pe rând, această funcție ne este de folos. De fel, în exercițiile de la examen avem propoziții normale, deci alcătuite din cuvinte și spații. Această funcție întoarce un pointer către prima apariție a caracterului pe care îl căutăm în șir.

`pointer = strtok(sir, caracter_pe_care_il_caut);`

Pentru a trece mai departe, trebuie să îi atribui pointerului prima apariție caracterului pe care îl caut în șirul NULL, întrucât, la prima apariție, întreg șirul de dinaint cuvântului pe care l-am găsit, dispare, astfel:

```
pointer = strtok(NULL, caracter_pe_care_il_caut);
```

- Tipul înregistrare (structuri)

Ce sunt structurile de date? Ei bine, structurile de date reprezintă o zonă de memor care poate, însă nu este obligatoriu să primească un nume și în care se pot memora t felul de variabile de tot tipul.

Cum se definesc structurile? Ei bine, acestea sunt destul de ușor de stăpânit da respectăm structura cu strictețe:

```
struct nume_structură(opțional)
{
   tip_dată, nume;
}nume_variabilă_prin_care_apelăm_câmpurile;
```

Câmpurile sunt chiar cele pe care le definim în interiorul structurii.

Cum funcționează totuși structurile? Ei bine, prin intermediul variabilelor noi pute apela câmpurile deja existente. Am să dau un exemplu concret.

```
struct liceulTeoretic
{
     char numeDirector;
     int numarElevi, numarProfesori;
     float mediaBac;
}lic;
```

Mai sus am definit o structură ce are denumirea liceulTeoretic, în care găsim mai mult câmpuri de diferite tipuri, toate aceste câmpuri putând fi accesate prin intermediul variabile lic.

Cum apelăm acele câmpuri? Ei bine, sintaxa arată așa: nume_variabilă.nume_câmp deci:

`lic.numeDirector;` sau `lic.numarElevi;`, sau `lic.numarProfesori;` sa `lic.mediaBac;`. Simplu, nu?

Pentru a citi / afișa aceste câmpuri, se procedează la fel ca în celelalte cazuri, spr exemplu:

`cin.get(numeDirector, 10);` sau `cin >> numarElevi;`

respectiv

`cout << numeDirector;` sau `cout << numarElevi;`

Ce mai trebuie să știi este că pot exista și structuri în structuri, numele acestora fiind tructuri imbricate. Pur și simplu se definește o structură în alta. Iată un exemplu:

```
struct liceulTeoretic
{
     char numeDirector;
     int numarElevi, numarProfesori;
     float mediaBac;
     struct clasaIX
     {
          char sefClasa;
          int mediaClasa;
     }clasa;
}lic;
```

Mai sus am adăugat structura clasaIX care conține câmpurile sefClasa și mediaClasa, acestea putând fi apelate prin intermediul variabilei clasa, dar nu înainte ca toate acestea să fie apelate prin intermediul variabilei lic. Cam asta înseamnă o structură imbricată. Nu merge de capul ei, pe scurt.

`clasa.sefClasa;` este greșit, în schimb ce `lic.clasa.sefClasa;` este corect.

Putem să ne gândim mai bine la faptul că respectiva clasă nu poate funcția, evident, decât dacă se află într-un liceu.

De asemenea, se pot folosi și tablouri unidimensionale pentru structuri, precum lic[15], dacă vorbim, spre exemplu, de 15 licee. Apelurile făcându-se astfel:

`lic[0].numeDirector;`, `lic[1].numeDirector;` și așa mai departe

- Exerciții specifice

1.

2. Scrieți un program C/C++ care citește de la tastatură un număr natural, n ($n \in [2, 10^3]$), și un șir de n numere naturale din intervalul $[10, 10^3]$, elemente ale unui tablou unidimensional. Programul afișează pe ecran un număr natural k, dacă toate numerele din șir au câte k cifre, sau mesajul NU, în caz contrar.
   Exemplu: pentru $n=5$ și tabloul (100, 712, 310, 130, 490), se afișează pe ecran 3
   pentru $n=5$ și tabloul (1000, 712, 310, 130, 490), se afișează pe ecran mesajul NU (10p.)

- Model Bacalaureat 2022 (Științele-Naturii)

```
#include <iostream>
using namespace std;
int main()
{
    int V[1000], n, i, nrCifrePrimulNr = 0, nrCifre, k; bool
verificare = true;
    cin >> n;
    for(i = 0; i < n; i++)
        cin >> V[i];
    while(V[0] != 0)
```

```
{
    nrCifrePrimulNr++;
    V[0] = V[0] / 10;
}
k = nrCifrePrimulNr;
for(i = 1; i < n; i++)
{
    nrCifre = 0;
    while(V[i] != 0)
    {
        nrCifre++;
        V[i] = V[i] / 10;
    }
    if(nrCifrePrimulNr != nrCifre)
        verificare = false;
}
if(verificare == true)
    cout << k;
else
    cout << "NU";
return 0;
}
```

2.

2. Suprafața unei scene a fost împărțită în zone dispuse ca elementele unui tablou bidimensional c egal de *linii* și coloane. Fiecare instrument are alocată o zonă și este identificat printr-un numă Dirijorul hotărăște eliminarea tuturor instrumentelor plasate pe diagonala principală a tab rearanjarea celor păstrate, prin deplasarea unora dintre acestea cu câte o poziție spre stânga, a să nu existe zone intermediare libere, ca în exemplu.
Scrieți un program C/C++ care citește de la tastatură numere naturale din intervalul $[2,10^2]$, și a elementele unui tablou bidimensional cu `n` linii și `n` coloane, reprezentând numerele de identi instrumentelor, în ordinea plasării lor inițiale pe scenă. Programul modifică apoi tabloul în corespunzător hotărârii dirijorului și afișează pe ecran tabloul obținut, fiecare linie a tabloului pe c a ecranului, elementele de pe aceeași linie fiind separate prin câte un spațiu.

Exemplu: pentru `n=4` și tabloul

20	30	80	52
71	93	64	51
62	81	55	49
90	65	32	22

se obține tabloul

30	80	52
71	64	51
62	81	49
90	65	32

- Sesiunea Specială 202

```
#include <iostream>
using namespace std;
int main()
{
    int Scena[100][100], n, i, j;
    cin >> n;
    for(i = 0; i < n; i++)
        for(j = 0; j < n; j++)
            cin >> Scena[i][j];
    for(i = 0; i < n; i++)
```



```
        for(j = i; j < n - 1; j++)
            Scena[i][j] = Scena[i][j+1];
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < n - 1; j++)
            cout << Scena[i][j] << " ";
        cout << endl;
    }
    return 0;
}
```

3.

2. Un text are cel mult 100 de caractere și este format din cuvinte, numere naturale și spații. Textul conține cel puțin un număr, iar cuvintele sunt formate numai din litere mici ale alfabetului englez. Cuvintele și numerele sunt separate prin câte un spațiu, ca în exemplu.
Scrieți un program C/C++ care citește de la tastatură un text de tipul precizat și afișează pe ecran un număr natural `k`, dacă toate numerele care intră în componența sa au câte `k` cifre, sau mesajul `NU`, în caz contrar.
Exemplu: pentru textul `la zoo sunt 100 de cocori si 120 de pasari flamingo`
sau pentru textul `la zoo sunt 100 de cocori`
se afișează `3`
iar pentru textul `la zoo sunt 100 de cocori si 10 pasari flamingo`
se afișează `NU` (10p.)

- Model Bacalaureat 2022

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char text[100], *pointer; int k; bool verificare1 =
false, verificare2 = true;
    cin.get(text, 100);
    pointer = strtok(text, " ");
    while(pointer != 0)
    {
        if(strchr("0123456789", pointer[0]) != NULL)
            if(verificare1 == false)
            {
                verificare1 = true;
                k = strlen(pointer);
            }
            else if(k != strlen(pointer))
                verificare2 = false;
        pointer = strtok(NULL, " ");
    }
    if(verificare2 == true) cout << k;
    else cout << "NU";
    return 0;
```

}

4.

3. Pentru o sală de spectacole se memorează următoarele informații specifice: numărul de locuri disponibile și prețul biletului pentru un loc. Variabilele `i` și `s` sunt întregi, iar variabila `t` memorează informațiile specifice pentru fiecare dintre cele 15 săli de spectacole dintr-un oraș.

```
struct sal
{ int nrLo
  int pret
} t[15];
```

Scrieți o secvență de instrucțiuni în urma executării căreia variabila `s` să memoreze suma totală
urma vânzării biletelor pentru un festival de teatru care s-ar desfășura simultan în toate sălile, iar
fi ocupate complet.

- Sesiunea Specială 20

```
for(i = 0; i < 15; i++)
    s = s + t[i].pret * t[i].nrLocuri;
```

5. Fișiere text

- Fișiere text. Proceduri și funcții predefinite pentru fișiere text

Iată că am ajuns și la ultima bibliotecă pe care trebuie să o studiem. Pe lângă cmath, string, trebuie să știți și de biblioteca `#include <fstream>` care permite utilizare fișierelor pentru a citi / afișa din / în ele.

Și deci, cum citesc din fișier? Ei bine, până acolo mai este. Mai întâi trebuie să definit un fișier pentru a-l putea utiliza. Există mai multe metode, însă noi vom folosi una și simpl pentru a putea citi:

ifstream cod_fișier("nume_fișier");, unde codul îl vom folosi pentru a utiliza fișierul. U exemplu bun ar fi:

```
ifstream fin("fisier.txt");
```

fin reprezintă doar numele, astfel că el poate să ia denumirea până și de IONEL. D chiar IONEL.

Pentru a citi din fișier este necesar doar să citim din... IONEL. Ok, lăsăm gluma la parte. Totuși, cam așa se face, sintaxa fiind:

fin >> numele_variabilei_pe_care_dorești_să_o_citești, deci ar arăta cam așa:

```
fin >> variabila1 >> variabila2;
```

Pentru a afișa în fișier trebuie să declarăm altfel fișierul, ceva asemănător cu ce a făcut mai sus:

```
ofstream fout("fisier.txt");
```


fout fiind evident, numele celui de-alș doilea șir. Deci, ca să afișăm în acesta, vom folosi intaxa:

out << numele_variabilei_pe_care_dorești_să_îl_afișezi, deci ar arăta cam așa:

```
fout << variabila1 >> variabila2;
```

(!) În loc de "fișier.txt" se poate adăuga adresa exactă a fișierului din care dorim să citim, sau pe cel în care dorim să afișăm.

6. Algoritmi elementari

- Probleme care operează asupra cifrelor unui număr
- Divizibilitate. Numere prime. Algoritmul lui Euclid
- Determinare minim / maxim
- Metode de ordonare (metoda bulelor, inserției, selecției, numărării)
- Interclasare
- Metode de căutare (secvențială, binară)
- Analiza complexității unui algoritm (eficiență din punct de vedere al timpului de executare și memoriei utilizate)

(!) Programele de mai jos vor fi scrise într-un mod cât mai explicit, fiind folosite nume ale variabilelor cât mai clare. În general, nu se recomandă scrierea lor chiar în acest tip.

- Probleme care operează asupra cifrelor unui număr

1. Afișarea numărului de cifre al unui număr:

```
#include <iostream>
using namespace std;
int main()
{
    int numar, nr_cifre = 0;
    cin >> numar;
    while(numar != 0)
    {
        nr_cifre++; //nr_cifre = nr_cifre + 1;
        numar = numar / 10;
    }
    cout << "Numarul de cifre al numarului este " << nr_cifre;
```

```
    return 0;
}
```

2. Afișarea numărului de cifre pare / impare ale unui număr:

```
#include <iostream>
using namespace std;
int main()
{
    int numar, nr_cifre_pare = 0, nr_cifre_impare = 0;
    cin >> numar;
    while(numar != 0)
    {
        if(numar % 2 == 0)
            nr_cifre_pare++;
        else
            nr_cifre_impare++;
        numar = numar / 10;
    }
    cout << "Numarul de cifre pare este " << nr_cifre_pare
<< endl;
    cout << "Numarul de cifre impare este "
nr_cifre_impare;
    return 0;
}
```

3. Determinarea cifrei maxime / minime al unui număr:

```
#include <iostream>
using namespace std;
int main()
{
    int numar, cifra, cifra_maxima = 0, cifra_minima = 9;
    cin >> numar;
    while(numar != 0)
    {
        cifra = numar % 10;
        if(cifra > cifra_maxima)
            cifra_maxima = cifra;
        if(cifra < cifra_minima)
            cifra_minima = cifra;
        numar = numar / 10;
    }
    cout << "Cifra maxima este " << cifra_maxima << endl;
    cout << "Cifra minima este " << cifra_minima;
    return 0;
}
```


4. Determinarea aparițiilor cifrelor într-un număr:

```
#include <iostream>
using namespace std;
int main()
{
    int V_de_aparitii[10] = {0}, numar, cifra, indice;
    cin >> numar;
    while(numar != 0)
    {
        cifra = numar % 10;
        V_de_aparitii[cifra]++;
        numar = numar / 10;
    }
    for(indice = 0; indice <= 9; indice++)
        cout << "Cifra " << indice << " apare de " <<
V_de_aparitii[indice] << " ori." << endl;
    return 0;
}
```

- **Divizibilitate. Numere prime. Algoritmul lui Euclid**

1. Verificarea dacă un număr este divizibil cu o cifră / altr număr:

```
#include <iostream>
using namespace std;
int main()
{
    int numar, cifra_sau_alt_numar;
    cin >> numar >> cifra_sau_alt_numar;

    if(numar % cifra_sau_alt_numar == 0)
        cout << "Numarul citit este divizibil cu cifra /
celalalt numar citit.";
    else
        cout << "Numarul citit nu este divizibil cu cifra
/ celalalt numar citit.";
    return 0;
}
```

2. Verificarea dacă un număr este sau nu prim:

```
#include <iostream>
using namespace std;
int main()
{
    int numar, divizor; bool verificare = true;
    cin >> numar;
    for(divizor = 2; divizor <= numar / 2; divizor++)
        if(numar % divizor == 0)
```

```
            verificare = false;
    if(verificare == false)
        cout << "Numarul nu este prim.";
    else
        cout << "Numarul este prim.";
    return 0;
}
```

3.1. Algoritmul lui Euclid (CMMDC prin scăderi repetate):

```
#include <iostream>
using namespace std;
int main()
{
    int numar1, numar2, CMMDC;
    cin >> numar1 >> numar2;
    while(numar1 != numar2)
        if(numar1 > numar2)
            numar1 = numar1 - numar2;
        else
            numar2 = numar2 - numar1;
    CMMDC = numar1; //CMMDC = numar2;
    cout << "CMMDC dintre cele doua numere este " << CMMDC;
    return 0;
}
```

3.2. Algoritmul lui Euclid (CMMDC cu împărțiri):

```
#include <iostream>
using namespace std;
int main()
{
    int numar1, numar2, rest, CMMDC;
    cin >> numar1 >> numar2;
    while(numar2 != 0)
    {
        rest = numar1 % numar2;
        numar1 = numar2;
        numar2 = rest;
    }
    CMMDC = numar1;
    cout << "CMMDC dintre cele doua numere este " << CMMDC;
    return 0;
}
```

(!) Cel mai mic multiplu comun se realizează cu ajutorul formulei:

`(numar1 * numar2)/CMMDC`, unde CMMDC este CMMDC dintre numar1 și numar2.

- **Determinare minim / maxim**

1. Determinarea minimului / maximului dintre mai multe numere:

```
#include <iostream>
using namespace std;
int main()
{
    int Vector[15], numar_de_elemente, maxim, minim,
indice;
    cin >> numar_de_numere;
    for(indice = 0; indice < numar_de_elemente; indice++)
        cin >> Vector[indice];
    maxim = minim = Vector[0];
    for(indice = 1; indice < numar_de_elemente; indice++)
    {
        if(Vector[indice] > maxim)
            maxim = Vector[indice];
        if(Vector[indice] < minim)
            minim = Vector[indice];
    }

    cout << "Cel mai mare numar dintre cele citite este "
<< maxim << endl;
    cout << "Cel mai mic numar dintre cele citite este "
<< minim;
    return 0;
}
```

- **Metode de ordonare (metoda bulelor, inserției, selecției, numărării)**

1. Metoda bulelor:

```
#include <iostream>
using namespace std;
int main()
{
    int Vector[100], numar_de_elemente, verificare,
auxiliar, indice;
    cin >> numar_de_elemente;
    for(indice = 0; indice < numar_de_elemente; indice++)
        cin >> Vector[indice];
    do
    {
        verificare = 1;
        for(indice = 0; indice < numar_de_elemente - 1;
indice++)
            if(Vector[indice] > Vector[indice + 1]) //
Vector[indice] < Vector[indice + 1]
```

```
            {
                auxiliar = Vector[indice];
                Vector[indice] = Vector[indice + 1];
                Vector[indice + 1] = auxiliar;
                verificare = 0;
            }
        }while(verificare == 0);
        for(indice = 0; indice < numar_de_elemente; indice++)
            cout << Vector[indice] << " ";
        return 0;
}
```

2. Metoda *inserției*:

```
#include <iostream>
using namespace std;
int main()
{
    int Vector[100], numar_de_elemente, auxiliar, indice,
indice2;
    cin >> numar_de_elemente;
    for(indice = 1; indice <= numar_de_elemente; indice++)
        cin >> Vector[indice];
    for(indice = 2; indice <= numar_de_elemente; indice++)
    {
        indice2 = indice - 1;
        auxiliar = Vector[indice];
        while(indice2 >= 1 && Vector[indice2] > auxiliar)
        {
            Vector[indice2 + 1] = Vector[indice2];
            indice2--;
        }
        Vector[indice2 + 1] = auxiliar;
    }
    for(indice = 1; indice <= numar_de_elemente; indice++)
        cout << Vector[indice] << " ";
    return 0;
}
```

3. Metoda *selecției*:

```
#include <iostream>
using namespace std;
int main()
{
    int Vector[100], numar_de_elemente, verificare,
auxiliar, indice, indice2;
    cin >> numar_de_elemente;
    for(indice = 0; indice < numar_de_elemente; indice++)
```



```
        cin >> Vector[indice];
    for(indice = 0; indice < numar_de_elemente - 1;
indice++)
        for(indice2 = indice + 1; indice2 <
numar_de_elemente; indice2++)
            if(Vector[indice] > Vector[indice2]) //
Vector[indice] < Vector[indice2]
            {
                auxiliar = Vector[indice];
                Vector[indice]= Vector[indice2];
                Vector[indice2] = auxiliar;
            }
    for(indice = 0; indice < numar_de_elemente; indice++)
        cout << Vector[indice] << " ";
    return 0;
}
```

4. Metoda *numărării*:

```
#include <iostream>
using namespace std;
int main()
{
    int Vector1[100], Vector2[100], Vector3[100] = {0},
numar_de_elemente, indice, indice2;
    cin >> numar_de_elemente;
    for(indice = 0; indice < numar_de_elemente; indice++)
        cin >> Vector1[indice];
    for(indice = 0; indice < numar_de_elemente - 1;
indice++)
        for(indice2 = indice + 1; indice2 <
numar_de_elemente; indice2++)
            if(Vector1[indice] > Vector1[indice2])
                Vector3[indice]++;
            else
                Vector3[indice2]++;
    for(indice = 0; indice < numar_de_elemente; indice++)
        Vector2[Vector3[indice]] = Vector1[indice];

    for(indice = 0; indice < numar_de_elemente; indice++)
        cout << Vector2[indice] << " ";
    return 0;
}
```

- **Interclasare**

```
#include <iostream>
using namespace std;
int main()
```

```
    {
        int Vector1[100], Vector2[100], Vector3[100],
numar_de_elemente1, numar_de_elemente2, indice, indice2,
contor;
        cin >> numar_de_elemente1 >> numar_de_elemente2;
        for(indice = 0; indice < numar_de_elemente1;
indice++)
            cin >> Vector1[indice];
        for(indice2 = 0; indice2 < numar_de_elemente2;
indice2++)
            cin >> Vector2[indice2];
        indice = indice2 = contor = 0;
        while(indice < numar_de_elemente1 && indice2 <
numar_de_elemente2)
            if(Vector1[indice] < Vector2[indice2])
                Vector3[contor++] = Vector1[indice++];
            else
                Vector3[contor++] = Vector2[indice2++];
        while(indice < numar_de_elemente1)
            Vector3[contor++] = Vector1[indice++];
        while(indice2 < numar_de_elemente2)
            Vector3[contor++] = Vector2[indice2++];
        for(indice = 0; indice < contor; indice++)
            cout << Vector3[indice] << " ";
        return 0;
    }
```

- Metode de căutare (secvențială, binară)

1. Metoda de căutare secvențială:

```
#include <iostream>
using namespace std;
int main()
{
    int Vector[100], numar_de_elemente, element_cautat,
indice, pozitie, verificare = 0;
    cin >> numar_de_elemente;
    for(indice = 0; indice < numar_de_elemente; indice++)
        cin >> Vector[indice];
    cin >> element_cautat;
    for(indice = 0; indice < numar_de_elemente; indice++)
        if(Vector[indice] == element_cautat)
        {
            verificare = 1;
            pozitie = indice;
        }
    if(verificare == 1)
```



```
        cout << "Elementul cautat se gaseste pe pozitia "
<< pozitie + 1;
    else
        cout << "Elementul cautat nu se regaseste in
tablou.";
    return 0;
}
```

- **2. Metoda de căutare binară (Se aplică doar vectorilor ordonați):**

```
#include <iostream>
using namespace std;
int main()
{
    int Vector[100], numar_de_elemente, element_cautat,
indice, stanga, mijloc, dreapta, pozitie;
    cin >> numar_de_elemente;
    for(indice = 1; indice <= numar_de_elemente;
indice++)
        cin >> Vector[indice];
    cin >> element_cautat;
    stanga = 1; dreapta = numar_de_elemente; pozitie = 0;
    while(stanga <= dreapta && pozitie == 0)
    {
        mijloc = (stanga + dreapta) / 2;
        if(Vector[mijloc] == element_cautat)
            pozitie = mijloc;
        else if(Vector[mijloc] < element_cautat)
            stanga = mijloc + 1;
        else
            dreapta = mijloc - 1;
    }
    if(pozitie != 0)
        cout << "Elementul cautat se gaseste pe pozitia "
<< pozitie;
    else
        cout << "Elementul nu a fost gasit in tablou.";
    return 0;
}
```

- **Analiza complexității unui algoritm (eficiență din punct de vedere al timpului de executare și memoriei utilizate)**

1. Complexitate din punct de vedere al timpului de executare:

Cu cât folosim mai multe structuri, cu atât timpul de executare al programului nostru crește, fie că este sau nu sesizabil. Pentru a avea un program cât mai eficient posibil (se cere la subiectul al III-lea, exercițiul 3) trebuie ca acesta să îndeplinească anumite condiții.

Cel mai principal și de bază lucru este ca programul să nu conțină structuri repetitive una într-alta. Un program liniar va avea o complexitate constantă, însă unul mai complex poate avea complexitatea pătratică, logaritmică și așa mai departe. Spre exemplu, algoritmul de căutare secvențială are o complexitate O(n), întrucât parcurgem o singură dată tablou unde n reprezintă numărul_de_elemente, însă algoritmul de interclasare are complexitate O(n1 + n2), unde n1 reprezinta numărul_de_elemente1, iar n2 reprezintă numărul_de_elemente2. De asemenea, algoritmul de sortare prin metoda selecției are complexitatea $O(n^2)$, unde n reprezintă și el numărul_de_elemente, însă ridicat la pătrat.

2. Complexitatea din punct de vedere al memoriei utilizate:

Aici nu sunt prea multe de spus. La examenul de bacalaureat, în cazul în care se cere rezolvarea acestei cerințe, tot ce trebuie să faceți este să nu utilizați tablouri. Nici măcar u tablou cu 10 elemente. Ideea constă în faptul că, cu cât ai mai multe variabile și cu cât stoche cât mai multe date, cu atât programul este mai „încărcat". Astfel că, tot ce îți rămâne, este s utilizezi alte variabile în rezolvarea problemelor, fără a introduce niciun tip de tablou, fie u uni sau bidimensional.

7. Subprograme definite de utilizator

- Proceduri și funcții (declarare și apel)
- Parametri formali și efectivi
- Parametri transmiși prin valoare și prin referință
- Variabile globale și locale

- Proceduri și funcții (declarare și apel)

Acum câteva lecții, mai pe la început, am vorbit de funcții predefinite. Ei bine, dup cum le spune numele, acelea sunt predefinite, însă noi putem să definim prorpiile funcți Putem creea orice funcție, să facă ce ne dorim noi și să ne ajute în orice program pe care utilizăm.

La ce ne ajută funcțiile / subprogramele? Ei bine, să presupunem că ai o problemă în care trebuie să faci suma tuturor elementelor de pe diagonala principală a n matrici citite de la tastatură. Cel mai bine faci un subprogram care rezolvă suma pentru o matrice, ulterio apelându-l de câte ori este nevoie, în cazul nostru de n ori. Simplu, nu?

Cum definim un subprogram? Acesta are o sintaxă care trebuie respectată de la sine

tip_dată_funcție nume(tip_dată_parametru nume_variabilă)

Ce vedeți mai sus se numește antetul funcției, el putând arăta, ca un exemplu, așa:

int subprogram(int număr, float media)

Important este de precizat că subprogramele pot fi de două tipuri: cele care returnează și cele care furnizează / afișează. Subprogramele care returnează vor avea drept tip_dată un tip de dată cunoscut, precum int, float, char, și așa mai departe, iar cele care furnizează sau afișează, vor avea în loc de tip_dată, cuvântul void.

Totuși, cum arată un subprogram complet? Ei bine, un subprogram complet arată ca ce făceam noi până acum, însă este cuprins între cele două acolade ale antetului funcției. Mai jos voi prezenta un subprogram care citește n numere, calculează și returnează suma acestora:

```
int suma(int numar_de_elemente)
{
    int numar, indice, Suma = 0;
    for(indice = 1; indice <= numar_de_elemente; indice++)
    {
        cin >> numar;
        Suma = Suma + numar;
    }
    return Suma;
}
```

Ce este totuși acel numar_de_elemente? De ce se declară alte variabile în interiorul funcției? Ce este cu acel return? Da, sunt multe întrebări, dar haideți să le luăm pe rând.

Acel numar_de_elemente reprezintă parametrul de intrare al funcției. Atunci când apelăm funcția, o vom apela prin intermediul unei variabile al cărei valori se va transfera în noua variabilă a subprogramului, respectiv numar_de_elemente.

Variabilele din interior sunt variabile locale, pe care le vom discuta mai jos.

Cum se face acest apel? Simplu: nume(variabilă1, variabilă2, variabilăn);, spre exemplu:

`suma(numar);`, unde numar_de_elemente va lua valoarea lui numar.

Nu se poate apela o funcție care are un număr de n parametrii de intrare prin intermediul a m variabile, unde n și m sunt numere diferite. Pe scurt, apelul de mai jos nu poate avea loc:

`suma(numar1, nuamar2);`, întrucât funcția noastă are doar un parametru de intrare, anume numar_de_elemente.

- Parametri formali și efectivi

Îți voi mai da drept exemplu o funcție plus un apel posibil pentru a înțelege mai bi lecția actuală:

```
int subprogram(int formal1, int formal2)
{
     // continut
}
int efectiv1, efectiv2;
subprogram(efectiv1, efectiv2);
```

În cazul nostru, formal1 și formal2 sunt parametri formali, în timp ce efectiv1 efectiv2 sunt parametri efectivi. Aceștia trebuie să coincidă la tipul de dată, alt programul va produce diferite erori.

- Parametri transmiși prin valoare și prin referință

Ce trebuie să mai știi despre subprograme este că parametrii, pe lângă cei prezenta mai sus, pot fi transmiși și prin referință.

În cazul parametrilor transmiși prin valoare, aceșia revin la forma lor inițială dup terminarea algoritmului. Să presupunem că avem un program care conține u subprogram ce calculează suma cifrelor unui număr:

```
#include <iostream>
using namespace std;
int sumaCifrelor(int numar)
{
     int Suma = 0;
     while(numar != 0)
     {
          Suma = Suma + numar % 10;
          numar = numar / 10;
     }
     return Suma;
}
int main()
{
     int nr;
     cout << sumaCifrelor(nr);
     return 0;
}
```

Ca să recapitulăm, numar este parametru formal, iar nr parametru efectiv, ambii fiin de tip întreg. Cu toate că în subprogram valoarea atribuită lui nr intră în numar și ajung să devină 0, la finalul subprogramului, după returnarea sumei, acesta revine la valoare inițială. De fapt, nici nu se modifică, astfel că subprogramul nu va afecta cu nimi programul întreg.

Și totuși, dacă am nevoie să rămână salvată valoarea? Aici intervin subprogramele ce conțin parametrii prin referință. Referința se notează cu & și este valabilă doar la subprogramele de tip void, adică cele care afișează / furnizează valori.

Mai jos voi lăsa un alt exemplu, respectiv un program ce conține un subprogram care taie ultima cifră a tuturor numerelor citite:

```
#include <iostream>
using namespace std;
void taie(int &numar)
{
   numar = numar / 10;
}
int main()
{
   int Vector[100], numar_de_elemente, indice;
   cin >> numar_de_elemente;
   for(indice = 1; indice <= numar_de_elemente; indice++)
   {
      cin >> Vector[indice];
      taie(Vector[indice]);
   }
   return 0;
}
```

Pe scurt, subprogramul alăturat, prin acel simbol de referință, respectiv & face ca valorile modificare să rămână salvate și la ieșirea din subprogram, adică, dacă elementele mele sunt 11, 22, 33, 44 și 55, la final vor deveni 1, 2, 3, 4 și 5.

Apelul la subprogramele prin referință se face strict cum a fost precizat mai sus, la subprogramele care returnezză fiind necesar un cout în față în cazul în care se dorește afișarea valorii returnate.

Nu la toate subprogramele este necesară referința. Pot exista subprograme de tip void care afișează și nu necesită această referință, sau pot exista subprograme care au 3 parametri formali, însă doar unul cu referință.

- Variabile globale și locale

Pentru a înțelege cel mai bine acest concept, voi atașa un program care citește un anumit număr de elemente și cu ajutorul unei funcții returnează suma celor pare:

```
#include <iostream>
using namespace std;
int Suma;
int SumaPare(int numar)
{
   Suma = Suma + numar;
}
```

```
int main()
{
    int Vector[100], numar_de_elemente, indice;
    cin >> numar_de_elemente;
    for(indice = 1; indice <= numar_de_elemente; indice++)
    {
        cin >> Vector[indice];
        if(Vector[indice] % 2 == 0)
            SumaPare(Vector[indice]);
    }
    cout << Suma;
    return 0;
}
```

Singura variabilă globală prezentă în subprogram este Suma și trebuie să știți c variabilele globale sunt, de fapt, cele definite în afara funcțiilor, de regulă înainte de fucti int main(). Aceste variabile sunt automat inițializate cu valorea 0, tocmai de acee nemaifiind nevoie să inițializăm noi variabila Suma.

Restul variabilelor (numar, Vector, numar_de_elemente, indice) sunt variabile locale Da, până și variabilele din cadrul funcțiilor, cu toate că sunt înainte de funcția int main sunt și ele variabile locale. Acestea nu sunt inițializate cu 0, ci cu o valoare random di memorie, deseori un număr lung și fără sens.

- Exerciții specifice

1.

1. Subprogramul **rest** are patru parametri:
   - **x**, **y** și **n**, prin care primește câte un număr natural din intervalul **[1,10⁶]**, **x<y<n**;
   - **k**, prin care furnizează cea mai mare valoare naturală din intervalul **[1,n]** pentru care at împărțirii la **x**, cât și restul împărțirii la **y**, sunt egale cu **2**, sau **0** dacă nu există o astfel de valoar

   Scrieți definiția completă a subprogramului.
   Exemplu: pentru **x=10**, **y=101** și **n=3000**, subprogramul returnează numărul **2022** (pentru num **1012** și **2022** atât restul împărțirii la **10**, cât și restul împărțirii la **101**, este **2**).

- Simulare Bacalaureat 202

```
void rest(int x, int y, int n, int &k)
{
    int i; bool verificare = true;
    for(i = n; i >= 1 && verificare == true; i--)
        if(i % x == 2 && i % y == 2)
        {
            verificare = false; k = i;
        }
}
```


2.

1. Subprogramul **patrate** are trei parametri:
   - **n**, prin care primește un număr natural (**n∈[2,10⁹]**);
   - **x** și **y**, prin care furnizează câte un număr natural cu proprietatea că **x²+y²=n** și **2≤x<y** sau valoarea **0**, prin fiecare dintre aceștia, dacă nu există două astfel de numere. Dacă sunt mai multe astfel de valori, se furnizează cele corespunzătoare unei valori minime a lui **x**.

   Scrieți definiția completă a subprogramului.
   Exemplu: pentru **n=400**, după apel, **x=2** și **y=10**, iar pentru **n=16** sau **n=24**, după apel, **x=0** și **y=0**. **(10p.)**

- Sesiunea Specială 2022

```
void patrate(int n, int &x, int &y)
{
    x = y = 0; bool verificare = true;
    for(x = 2; x < n / 2 && verificare==true; x++)
        for(y = x + 1; y < n / 2 && verificare==true;
y++)
            if((x * x * y * y) == n) verificare = false;
    x--; y--; }
```

8. Recursivitate

- **Prezentare generală**
- **Proceduri și funcții recursive**

- **Prezentare generală**

Ce reprezintă recursivitatea? Se pare că încă nu am scăpat de capitolul funcții / subprograme. Recursivitatea constă în procesul de autoapelare a unei funcții deja existente până la un punct, atunci când se reîntoarce și returnează sau afișează un anume rezultat.

Un exemplu potrivit pentru a înțelege mai bine acest aspect, este determinarea cifrei minime al unui număr prin metodă recursivă:

```
int CifraMinima(int numar)
{
    int cifra;
    if(numar == 0) return 9;
```

```
    cifra = CifraMinima(numar / 10);
    if(numar % 10 < cifra)
        return numar % 10;
    return cifra;
}
```

Acest subprogram se autoapelează până ce va ajunge ca numărul să devină 0, iar ap pleacă cu valoarea 9, înapoi, până va ajunge să se returneze cea mai mică valoare, în ca nostru cea mai mică cifră. Ca să înțelegi subprogramele trenuie să te gândești la niște sc Gândește-te că tu cobori până într-o magazie pentru a lua ceva, apoi te întorci cu el de un ai venit.

Ce este important la funcțiile recursive este modul de executare, anume cum gând calculatorul când vede apelul. Ei bine, o dată întâlnit un apel, acesta are prioritate, iar da urmează alte instrucțiuni în cadrul funcției (spre exemplu condiția alternativă și returnarea la final), acestea se vor executa abia după ce se termină apelul inițial.

- Proceduri și funcții recursive

1. Proceduri:

Pentru a înțelege mai bine funcțiile recursive trebuie să ne dăm seama c funcționează, de fapt, autoapelările. Pentru funcțiile care returnează, autoapelul se fa cum deja îl cunoaștem însă în interiorul funcției deja existente. Pentru funcțiile ca afișează / furnizează, autoapelul se face printr-o simplă isntrucțiune de atribuire.

2. Funcții recursive:

2.1.1. CMMDC recursiv (prin scăderi repetate):

```
int CMMDC(int numar1, int numar2)
{
    if(numar1 == 0 || numar2 == 0)
        return 0;
    else if(numar1 == numar2)
        return numar1;
    else if(numar1 > numar2)
        return CMMDC(numar1 - numar2, numar2);
    else
        return CMMDC(numar1, numar2 - numar1);
}
```

2.1.2. CMMDC recursiv (prin resturi):

```
int CMMDC(int numar1, int numar2)
{
    if(numar2 == 0)
        return numar1;
    return CMMDC(numar2, numar1 % numar2);
}
```


2.2. Factorialul unui număr:

```
long long factorial(int numar)
{
  if(numar == 1)
      return 1;
  return numar * factorial(numar - 1);
}
```

- Exerciții specifice

1.

2. Subprogramul `f` este definit alăturat.

```
int f(int st,int dr,int v[])
{ if(st==dr) if(v[st]==st)return 1;
             else return 0;
  else return f(st,(st+dr)/2,v) + f(1+(st+dr)/2,dr,v);
}
```

Știind că variabila `x` memorează un tablou unidimensional cu elementele `(7,1,2,5,4)`, în această ordine, numerotate de la 0 la 4, indicați valoarea `f(0,4,x)`.

a. 2 b. 3 c. 4 d. 5

- Simulare Bacalaureat 2022

Răspuns: b. 3

Explicație: Ideea exercițiului constă mai mult în recunoașterea algoritmului de căutare binară recursivă, într-o formă recursivă. În cazul în care acest algoritm nu reiese, se poate face verificarea pas cu pas. Secvența determină câte numere se află pe pozițile numerotate cu valaorea lor. (1 se află pe poziția 1, 2 pe poziția 2 și 4 la fel), acestea fiind numerotate începând cu 0.

2.

2. Subprogramul `f` este definit alăturat. Indicați valoarea parametrului `x` pentru care `f(x,3)` are valoarea 2022.

```
int f(int n,int p)
{ if(n==0) return 0;
  else if(p==0) return n*10+2;
       else return f(n/10,p-1);
}
```

a. 120234 b. 122023 c. 202345 d. 220223

- Sesiunea Specială 2022

Răspuns: c. 202345

Explicație: Printr-o simplă verificare a soluțiilor de mai sus, se poate observa cu ușurință că algoritmul preia ultimele 3 cifre de la dreapta la stânga și adaugă un 2 la final, astfel că numere formate ar fi următoarele:
a. 0212, deci 212
b. 2212
c. 2022
d. 0222, deci 222

9. Metoda backtracking

- Prezentare generală
- Exerciții specifice

- Prezentare generală

Ce reprezintă, de fapt, metoda backtracking?

Backtracking-ul este o metodă de rezolvare a unor probleme. Acesta poate fi aplicat î
găsirea soluțiilor unor cerințe dacă se respectă anumite condiții, printre care soluția să poat
fi reprezentată cu ajutorul unui tablou unidimensional.

De asemenea, când ne gândim la elemente de combinatorică / backtracking, trebuie s
avem în minte și aranjamentele și combinările, însă stai liniștit căci nu avem nevoie de vre
formulă pentru ele.

Un exemplu concret ar reprezenta mulțimea {1,2,3,4,5}. Să presupunem că vrem s
combinăm aceste numere din mulțime, luate câte 3. Deci, pe scurt, avem aranjamente de
luate câte 3.

Care ar fi prima soluție? {1,2,3}, corect. Dar următoarea? {1,2,4}, {1,2,5}, și așa ma
departe. Mai jos am să las chiar toate submulțimile:

{1,2,3}, {1,2,4}, {1,2,5}, {1,3,4}, {1,3,5}, {1,4,5}, {2,3,4}, {2,3,5}, {3,4,5}

- Exerciții specifice

1.

3. Utilizând metoda backtracking, se generează toate posibilitățile de a expune câte trei mașini produ
România, din gama Dacia, din mulțimea (`Dokker` (negru), `Duster` (negru), `Lodgy` (gri), `Logan` (
`Sandero` (roșu)), astfel încât să nu fie expuse simultan două mașini de aceeași culoare, Sandero să
apară pe prima poziție, iar Logan să NU fie înainte de Dokker sau Duster. Două soluții sunt distincte d
au cel puțin o mașină diferită sau dacă ordinea mașinilor este diferită. Primele cinci soluții generate s
în această ordine (`Dokker, Lodgy, Logan`), (`Dokker, Lodgy, Sandero`), (`Dokker, Logan, Lod
(`Dokker, Logan, Sandero`), (`Dokker, Sandero, Lodgy`). Indicați penultima soluție, în ordinea gene
acestora.

a. `Logan, Lodgy, Sandero`

b. `Logan, Sandero, Lodgy`

c. `Lodgy, Dokker, Logan`

d. `Lodgy, Logan, Dokker`

- Sesiunea Specială 202

Răspuns: `a. Logan, Lodgy, Sandero`

Explicație: Primul lucru pe care majoritatea îl fac este să gândească că, pentru penultim
soluție trebuie să o luăm de la dreapta la stânga cu mașinile, astfel că prima variantă devin
Sandero. Teoretic, este corect, însă nouă ni se specifică faptul că Sandero nu poate fi pe prim

poziție. Deci, prima mașină ar veni Logan, apoi Sandero și Lodgy, însă aceasta este doar ultima soluție. După aceasta ar veni Logan, Sandero, Duster / Logan, Sandero, Dokker, ceea ce nu este posibil, deci vom trece Lodgy în loc de Sandero și vom avea Logan, Lodgy, iar următoarea mașină este, în ordine inversă, Sandero. Deci, varianta finală: Logan, Lodgy, Sandero.

10. Generarea elementelor combinatoriale

- Permutări, aranjamente, combinări
- Produs cartezian, submulțimi
- Exerciții specifice

- Permutări, aranjamente, combinări

Haide totuși să ne areamintim formulele:

Permutări: $n!$

Aranjamente de n luate câte k: $A_n^k = \frac{n!}{(n-k)!}$

Combinări de n luate câte k: $C_n^k = \frac{n!}{k!\cdot(n-k)!}$

La ce ne ajută totuși aceste lucruri? Ei bine, majoritatea problemelor se rezolvă cu ajutorul permutărilor, aranjamentelor sau combinărilor. De fel, cerința este una exactă care respectă un anumit criteriu pe care noi trebuie, de asemenea, să-l respectăm. Spre exemplu, ni se poate cere să formăm submulțimi ale unei mulțimi care să înceapă cu un anumit element, sau să aranjăm niște elemente după anumite reguli.

- Produs cartezian, submulțimi

Produsul cartezian este unul dintre cele mai frumoase, după mine, căci este ca o joacă. Trebuie pur și simplu să formăm tot felul de grupulețe din două sau mai multe grupuri mai mari, după exemplu:

A: {1,2,3,4}, B: {1,2,3}, C{1,2}

Cum vom creea grupulețele? Luând pe rând, în ordine, fiecare element din A, urmat de un altul din B, apoi la final C și așa mai departe. Am să las câteva soluții generate mai jos:

{1,1,1}, {1,1,2}, {1,2,1}, {1,2,2}, {1,3,1}, {1,3,2}, {2,1,1}, {2,1,2}, {2,2,1}, {2,2,2} . . .

Ce ați văzut mai sus reprezină niște submulțimi dintr-o mare de mulțimi. Sp
exemplu, dacă am mulțimea A: {1,2,3,4,5} și voi lua doar {1,2,3}, aceasta se poate num
de fapt, o submulțime.

- Exerciții specifice

1.

3. Utilizând metoda backtracking, se generează toate posibilitățile de a forma cutii cu dulciur
distincte din mulțimea {`bomboane`, `drajeuri`, `jeleuri`, `praline`}. Într-o cutie sunt cel puțin
de dulciuri, dar nu pot fi și jeleuri și praline simultan. Două cutii sunt distincte dacă ele conțin ce
tip diferit de dulciuri. Primele patru soluții generate sunt, în această ordine, (`bomboane`, `dr`
(`bomboane`, `drajeuri`, `jeleuri`), (`bomboane`, `drajeuri`, `praline`), (`bomboane`, `jeleuri`
a șasea soluție, în ordinea generării acestora.

a. (`bomboane`, `praline`)
b. (`drajeuri`, `jeleuri`)
c. (`drajeuri`, `praline`)
d. (`jeleuri`, `praline`)

- Model Bacalaureat 202

Răspuns: `b. (drajeuri, jeleuri)`

Explicație: Următoarea soluție generată este (bomboane, jeleuri, praline), ceea ce es
greșit, apoi (bomboane, praline), iar într-un final, a șasea soluție, (drajeuri, jeleuri).

2.

3. Un sport se desfășoară pe o pistă înghețată, cu o echipă alcătuită din 4 jucători, care sunt,
ordine, lider, secund, vice-skip și skip. Lotul este format din jucătorii din mulțimea {`Alin`,
`Cristian`, `Răzvan`, `Ștefan`, `Valentin`}; `Cristian` poate fi doar lider, `Alin` și `Valent`
doar lider sau secund, `Bogdan` poate fi doar secund, iar ceilalți doi jucători pot fi doar vice-skip
Două echipe sunt distincte dacă există cel puțin un rol care, în cele două echipe, este atri
jucători diferiți. Pentru antrenamente, s-au format echipe în toate formulele posibile, primele s
generate fiind, în această ordine: {`Alin`, `Bogdan`, `Răzvan`, `Ștefan`}, {`Alin`, `Bogdan`,
`Răzvan`}, {`Alin`, `Valentin`, `Răzvan`, `Ștefan`}, {`Alin`, `Valentin`, `Ștefan`,
{`Cristian`, `Alin`, `Răzvan`, `Ștefan`}.{`Cristian`, `Alin`, `Ștefan`, `Răzvan`}.
Indicați soluția generată imediat după {`Cristian`, `Valentin`, `Ștefan`, `Răzvan`}.

a. {`Cristian`, `Bogdan`, `Răzvan`, `Ștefan`}
b. {`Cristian`, `Bogdan`, `Ștefan`, `Răzva`
c. {`Valentin`, `Alin`, `Răzvan`, `Ștefan`}
d. {`Valentin`, `Bogdan`, `Ștefan`, `Răzva`

- Simulare Bacalaureat 202

Răspuns: `c. {Valentin, Alin, Răzvan, Ștefan}`

Explicație: Tot ce trebuie să facem aici este doar să fim atenți. Variantele a și b sunt de
generate înainte de ceea ce ni se cere, deci rămân c și d. La o simplă privire, c este
variantă generată înainte de d, astfel că aceasta rămâne și soluția corectă.

11. Grafuri

- Grafuri neorientate (terminologii, proprietăți, metode de reprezentare)
- Grafuri orientate (terminologii, proprietăți, metode de reprezentare)
- Arbori (terminologii, metode de reprezentare în memorie)
- Exerciții specifice

- Grafuri neorientate (terminologii, proprietăți, metode de reprezentare)

1. Ce sunt grafurile neorientate?

Un graf neorientat reprezintă o pereche ordonată de mulțimi pe care o putem nota G = (N,M), unde N reprezintă mulțimea tuturor nodurilor, iar M mulțimea tuturor muchiilor.

2. Ce poate avea un graf neorientat?

- nod (vârf): nodul reprezintă un element al mulțimii N care este reprezentat printr-un cerc; Dacă două noduri sunt unite, putem spune despre acelea că sunt noduri adiacente;
- muchie: muchia reprezintă uniunea dintre două noduri, fără a se ține cont de sens (legătura nod1 - nod2 este aceeași cu legătura nod2 - nod1); Dacă o muchie care unește două noduri se numește muchie incidentă;
- grad: gradul unui nod este reprezentat de numărul de muchii care intră / iese din el; Dacă nu intră / iese nicio muchie din acesta, îl vom numi nod izolat; Dacă intră / iese doar o muchie din el, îl vom numi nod terminal;
- lanțuri:
  - elementare: reprezintă o conexiune care pornește dintr-un nod și ajunge într-altul, prin legături existente, fără a trece de două sau mai multe ori prin același nod;
  - neelementare: reprezintă o conexiune care pornește dintr-un nod și ajunge într-alul, prin legături deja existente, cu posibilitatea de a trece de două sau mai multe ori prin același nod
- cicluri:
  - elementare: reprezintă o conexiune care pleacă și ajunge la același nod, prin legături deja existente, fără a trece de două sau mai multe ori prin același nod;
  - neelementare: reprezintă o conexiune care pleacă și ajunge la același nod, prin legături deja existente, cu posibilitatea de a trece de două sau mai multe ori prin același nod;
  - euleriene: reprezintă o conexiune care trece prin toate muchiile grafului; Într-un ciclu eulerian, fiecare nod trebuie să aibă un grad par; Numim graf eulerian acel graf care are cel puțin un ciclu eulerian;
  - hamiltoniene: reprezintă o conexiune care trece prin toate vârfurile grafului; Numim graf hamiltonian, acel graf care are cel puțin un ciclu hamiltonian;
- subgraf: subgraful reprezintă un graf ce se obține din graful inițial doar prin eliminarea a unuia sau mai multor noduri, fără a afecta muchiile, decât cele care erau unite cu nodurile care dispar;

- graf parțial: graful parțial reprezintă un graf care se obține din graful inițial pr eliminarea uneia sau mai multor muchii, fără a afecta nodurile, ele putând să rămână chi izolate;

- graf aciclic: graful aciclic este acel graf care nu conține niciun ciclu;

- graful conex: graful conex este acel graf care nu are noduri izolate, adică toa nodurile sunt interconectate între ele, cu cel puțin măcar o muchie;

- componentă conexă: componenta conexă reprezintă un subgraf conex.

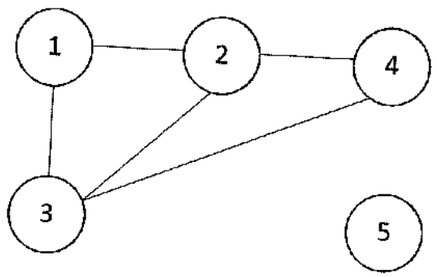
(!) Lungimea unui lanț / ciclu este determinată de numărul de muchii de la un capăt celălalt.

3. Formule pentru ușurarea rezolvării exercițiilor:

- suma tuturor gradelor nodurilor = 2 * numărul de muchii;
- numărul de muchii în graful neorientat: C_n^2;
- numărul grafurilor neorientate cu v vârfuri: $2^{C_n^2}$;
- numărul grafurilor parțiale știind numărul m de muchii: 2^m.

4. Reprezentare:

Pentru a înțelege mai bine cum se reprezintă, am să las mai jos o imagine cu un gra neorientat:



4.1. Lista de adiacență:

Vom scrie numărul nodului și nodurile cu care este adiacent:

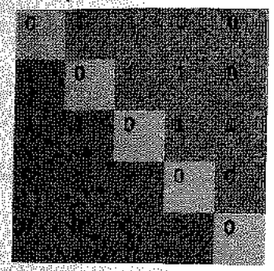
1: 2,3
2: 1,3,4
3: 1,4
4: 2,3
5: -

4.2 Matricea de adiacență:

Vom face o matrice și vom pune pe fiecare linie și coloană elementul 1 dacă există muchie între cele două noduri sau 0 în caz contrar.

(!) Diagonala principală mereu va avea 0, întrucât nu poate exista muchie de la 1 la 1, de la 2 la 2 și așa mai departe.

Exemplu:



Se poate vedea o simetrie între partea superioară a diagonalei principale și partea inferioară a acesteia.

- Grafuri orientate (terminologii, proprietăți, metode de reprezentare)

1. Ce sunt grafurile orientate?

Un graf orientat reprezintă o pereche ordonată de mulțimi pe care o putem nota G = (N,A), unde N reprezintă mulțimea tuturor nodurilor, iar A mulțimea tuturor arcelor.

2. Ce poate avea un graf orientat?

- nod (vârf): nodul reprezintă un element al mulțimii N care este reprezentat printr-un cerc; Dacă două noduri sunt unite, putem spune despre acelea că sunt noduri adiacente;
- arc: arcul reprezintă uniunea dintre două noduri, însă ținându-se cont de sens (legătura nod1 - nod2 nu este aceeași ca și legătura nod2 - nod1); Dacă o muchie care unește două noduri se numește muchie incidentă;
- grad:
  - interior: acesta este notat cu $d^-(N)$ și reprezintă numărul de arce care intră în nodul N;
  - exterior: acesta este notat cu $d^+(N)$ și reprezintă numărul de arce care iese din nodul N;
  - dacă noul are gradul 0, acesta se numește nod izolat;
- drumuri:
  - elementare: reprezintă o conexiune care pornește dintr-un nod și ajunge într-altul, prin legături existente, fără a trece de două sau mai multe ori prin același nod;
  - neelementare: reprezintă o conexiune care pornește dintr-un nod și ajunge într-alul, prin legături deja existente, cu posibilitatea de a trece de două sau mai multe ori prin același nod
- circuite:
  - elementare: reprezintă o conexiune care pleacă și ajunge la același nod, prin legături deja existente, fără a trece de două sau mai multe ori prin același nod;

- neelementare: reprezintă o conexiune care pleacă și ajunge la același no
prin legături deja existente, cu posibilitatea de a trece de două sau mai multe ori pr
același nod;

- subgraf: subgraful reprezintă un graf ce se obține din graful inițial doar pr
eliminarea a unuia sau mai multor noduri, fără a afecta arcele, decât cele care erau uni
cu nodurile care dispar;

- graf parțial: graful parțial reprezintă un graf care se obține din graful inițial pr
eliminarea uneia sau mai multor arce, fără a afecta nodurile, ele putând să rămână ch
izolate;

- graful tare conex: graful tare conex este acel graf în care, pentru fiecare nod distin
există cel puțin un drum;

- componentă conexă: componenta conexă reprezintă un subgraf tare conex.

(!) Lungimea unui lanț / ciclu este determinată de numărul de muchii de la un capăt
celălalt.

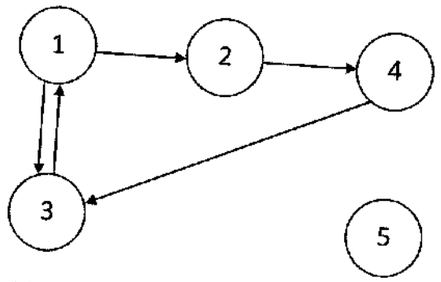
3. Formule pentru ușurarea rezolvării exercițiilor:

- suma gradelor $d^-(N)$ = suma gradelor $d^+(N)$ = numărul de arce

- numărul grafurilor orientate complete cu v vârfuri: $3^{C_n^2}$;

4. Reprezentare:

Pentru a înțelege mai bine cum se reprezintă, am să las mai jos o imagine cu un gr
neorientat:



4.1. Lista de adiacență:

Vom scrie numărul nodului și nodurile cu care este adiacent:

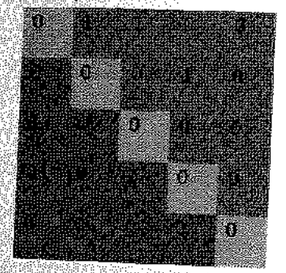
1: 2,3
2: 4
3: 1
4: 3
5: -

4.2 Matricea de adiacență:

Vom face o matrice și vom pune pe fiecare linie și coloană elementul 1 dacă există arc între cele două noduri sau 0 în caz contrar.

(!) Diagonala principală mereu va avea 0, întrucât nu poate exista muchie de la 1 la 1, de la 2 la 2 și așa mai departe.

Exemplu:



Se poate vedea că nu avem o simetrie între partea superioară a diagonalei principale și partea inferioară a acesteia, față de graful neorientat.

- Arbori (terminologii, metode de reprezentare în memorie)

1. Ce sunt arborii?

Arborii reprezintă niște grafuri (neorientate) conexe aciclice.

2. Ce poate avea un arbore?

- nod (vârf);
- n-1 muchii;
- între oricare două noduri există un lanț elementar;
- rădăcină: rădăcina reprezintă nodul de la care pornește arborele;
- descendent: descendentul reprezintă un nod care se află pe un lanț elementar; rădăcina nu poate fi descendent;
- fiu: fiul reprezintă un descendent al unui nod;
- frunza: reprezintă nodurile de la finalul arborelui ce nu au descendenți;
- înălțime: înălțimea este determinată de numărul de muchii de la o rădăcina la o frunză.

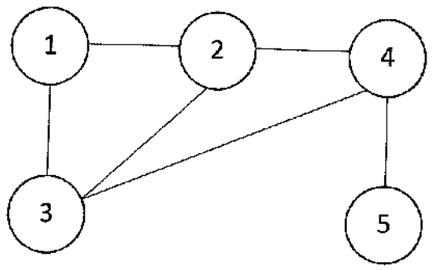
(!) Arborele binar reprezină un arbore ale căror noduri au cel mult doi fii.

3. Formule pentru ușurarea rezolvării exercițiilor:

- într-un graf neorientat cu n noduri și m muchii, numărul de muchii ce trebuie eliminate pentru ca graful să devină arbore este m - n + 1.

4. Reprezentare:

Pentru a înțelege mai bine cum se reprezintă, am să las mai jos o imagine cu un graf neorientat pe care îl transformăm în arbore (Presupunem rădăcina ca fiind 2):



4.1. Lista de adiacență:

Vom scrie numărul nodului și nodurile cu care este adiacent:

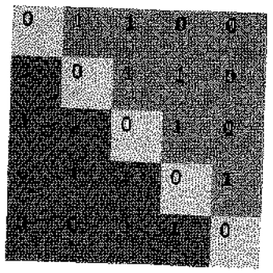
1: 2,3
2: 1,3,4
3: 1,4
4: 2,3,5
5: 4

4.2. Matricea de adiacență:

Vom face o matrice și vom pune pe fiecare linie și coloană elementul 1 dacă există muchie între cele două noduri sau 0 în caz contrar.

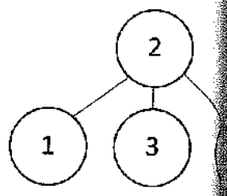
(!) Diagonala principală mereu va avea 0, întrucât nu poate exista muchie de la 1 la 1 de la 2 la 2 și așa mai departe.

Exemplu:



Se poate vedea o simetrie între partea superioară a diagonalei principale și partea inferioară a acesteia.

4.3. Vectorul de tați:



Acesta se realizează scriind între paranteze pe fiecare poziție, începând de la 1 la numărul de noduri, 5 în cazul nostru, indicele tatălui, iar rădăcina fiind notată cu 0, ca în exemplu:

(2,0,2,2,4)

- Exerciții specifice

1.

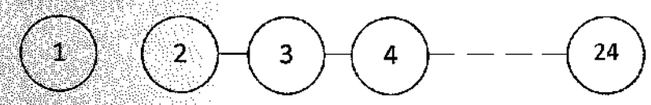
4. Indicați numărul de noduri ale unui graf neorientat fără cicluri, cu 22 de muchii și două componente conexe.
a. 11 b. 23 c. 24 d. 44

- Model Bacalaureat 2022

Răspuns: c. 24

Explicație: O componentă conexă poate rămâne goală (un nod), iar celelalte 22 de muchii sunt legate în șir ca un șarpe cu ajutorul a 23 de noduri, deci astfel rămân 23 + 1 = 24.
Nu este ogligatoriu acest lucru, însă s-ar ajunge la același răspuns.

Desen:



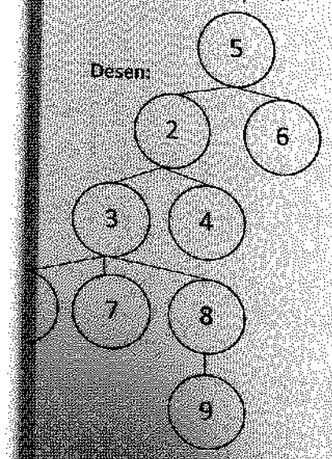
2.

2. Un arbore cu 9 noduri, numerotate de la 1 la 9, este reprezentat prin vectorul de „tați”
(3, 5, 2, 2, 0, 5, 3, 3, 8). Scrieți numerele asociate nodurilor care sunt descendenți direcți („fii”) ai nodului cu eticheta 3. (6p.)

- Model Bacalaureat 2022

Răspuns: 1, 7 și 8

Desen:



3.

5. Un graf neorientat complet are 4 noduri. Indicați o valoare care NU poate fi numărul de muchii al unui subgraf al acestui graf.

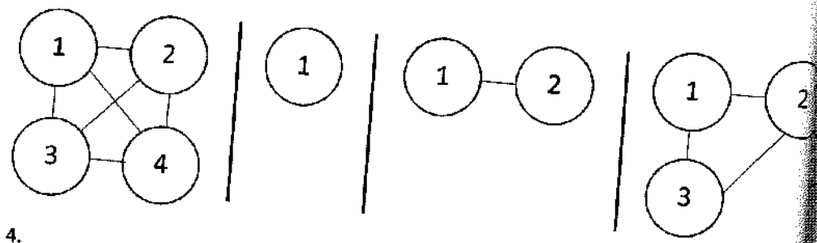
a. 0 b. 1 c. 2 d. 3

- Simulare Bacalaureat 20

Răspuns: c. 2

Explicație: Un graf neorientat complet cu 4 noduri are n(n-1)/2 muchii, adica 6.

Desen:



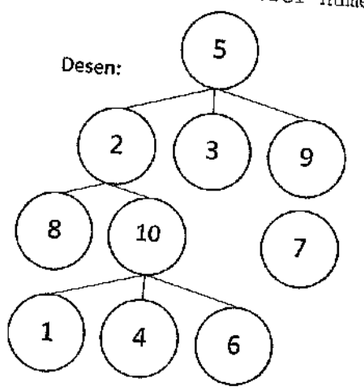
4.

2. Într-un arbore cu rădăcină un nod se află pe nivelul x dacă lanțul elementar care are o extremitate în nodul respectiv și cealaltă extremitate în rădăcina arborelui are lungimea x. Pe nivelul 0 se află un singur nod (rădăcina). Un arbore cu rădăcină are 10 noduri și este reprezentat prin vectorul de tați (10,5,5,10,0,10,9,2,5,2). Enumerați trei dintre frunzele situate în arbore pe niveluri impare.

- Simulare Bacalaureat 20

Răspuns: Oricare trei numere dintre 1, 3, 4 sau 6

Desen:



5.

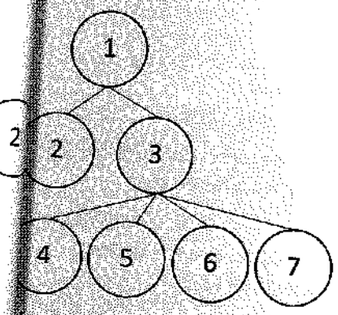
5. Într-un arbore cu 7 noduri, numerotate de la 1 la 7, nodul 1 este rădăcină și pentru orice nod numerotat cu i (i∈[1,7]), dacă se notează cu F(i) numărul de descendenți direcți („fii") ai săi, atunci F(i)=0, dacă i este „frunză", altfel F(i)=i+1. Indicați numărul maxim de „frați" ai nodului 7.

a. 4 b. 3 c. 2 d. 1

- Sesiunea Specială 2022

Răspuns: b. 3

Desen:



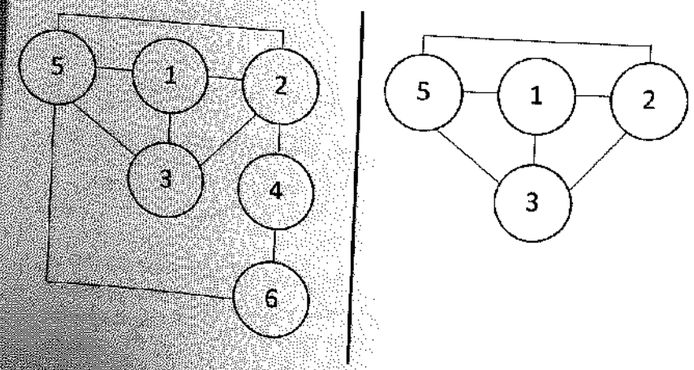
6.

2. Un graf neorientat cu 6 noduri, numerotate de la 1 la 6, are muchiile [1,2], [1,3], [1,4], [1,5], [2,3], [2,4], [2,5], [3,5], [4,6], [5,6]. Scrieți mulțimea muchiilor unui subgraf al acestui graf, care să fie complet și să aibă un număr maxim de noduri. (6p.)

- Sesiunea Specială 2022

Răspuns: [1,2], [1,3], [1,5], [2,3], [2,5], [3,5]

Desen:



12. Subiecte de bacalaureat

- Rezolvarea subiectului de bacalaureat la informatică, sesiunea iunie-iulie 2022
- Rezolvarea subiectului de bacalaureat la informatică, sesiunea august-septembr 2022

- Rezolvarea subiectului de bacalaureat la informatică, sesiunea iunie-iulie 2022

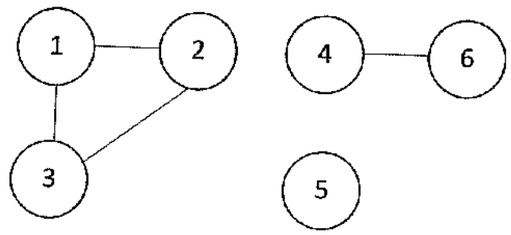
Subiectul I:

1. c (10)
Explicație: 22/20 = 1; 1*20 = 20; 20/2 = 10, deci răspunsul este 10

2. d (`valoare=2*x.cantitate*x.pret;`)
Explicație: Variabila v este de tip struct ce poate apela doar câmpurile denumire, cantitate și pret. produs este hnumele structurii, deci nu se pooate lua în calcul

3. a (`{volei, handbal}`)
Explicație: Se poate observa cu ușurință că varianta b este greișită, iar celelalte două sunt după prima variantă, astfel că rămâne doar prima viariantă corectă

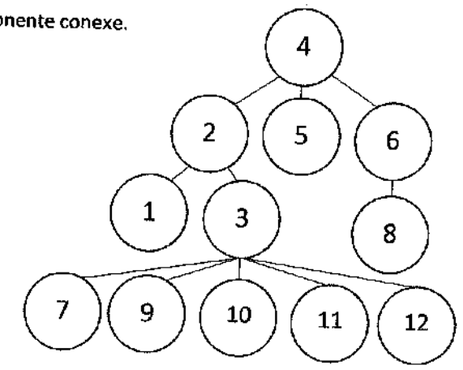
4. b (3)
Desen:



În desen sunt 3 componente conexe.

5. c (`x=3, y=8`)
Desen:

Subiectul al II-lea:



1.
a. 4
b. Orice număr prim din intervalul dat.
c.

```
#include <iostream>
using namespace std;
int main()
{
    int n, i, k;
    cin >> n;
    i = 2; k = 0;
    while(n >= i)
    {
        while(n % i == 0)
        {
            k++;
            n = n / i;
        }
        if(i == 2)
            i++;
        else
            i = i + 2;
    }
    cout << k;
    return 0;
}
```

d.

```
citește n (număr natural)
i<-2; k<-0
┌cât timp n>i execută
│  ┌dacă n%i=0 atunci
│  │  ┌execută
│  │  │  k<-k+1
│  │  │  n<-[n/i]
│  │  └cât timp n%i=0
│  └
│  ┌dacă i=2 atunci i<-i+1
│  │  altfel i<-i+2
│  └
└
 scrie k
```

2. Oricare două numere dintre: 2022, 2023, 2024, 2025.
Explicație: dacă valaorea lui a este strict mai mică sau egală decât cea a lui b, atunci se va returva valoarea lui a, respectiv 2022. În restul cazurilor, subprogramul v-a avea alte afișări.

3.

```
int i;
for(i = 0; i < strlen(s); i++)
    if(s[i] == ' ')
        strcpy(id, s + i + 1);
strcat(id, "2022");
```

Subiectul al III-lea:

1.

```
void secventa(long &n)
{
    long aux = 0, p = 1;
    while(n)
    {
        if(n % 10 == 2 && (n / 10) % 10 == 2)
        {
            p = p * 10;
            aux = aux + p * 2;
            p = p * 10;
            n = n / 100;
        }
        else
        {
            aux = aux + p * (n % 10);
            p = p * 10;
            n = n / 10;
        }
    }
    n = aux;
}
```

2.

```
#include <iostream>
using namespace std;
int main()
{
    int M[100][100], m, n, i, j, k = 0, min;
    cin >> m >> n;
    for(i = 1; i <= m; i++)
        for(j = 1; j <= n; j++)
            cin >> M[i][j];
    for(i = 0; i <= m + 1; i++)
        for(j = 0; j <= n + 1; j++)
            if(i == 0 || j == 0 || i == m + 1 || j == n + 1
                M[i][j] = 0; // creem un contor nul
                // pentru a putea afla mai usor minimul
    for(i = 1; i <= m; i++)
```



```
        for(j = 1; j <= n; j++)
        {
            min = 11;
            if(M[i-1][j] != 0 && M[i-1][j] < min)
                min = M[i-1][j];
            if(M[i+1][j] != 0 && M[i+1][j] < min)
                min = M[i+1][j];
            if(M[i][j-1] != 0 && M[i][j-1] < min)
                min = M[i][j-1];
            if(M[i][j+1] != 0 && M[i][j+1] < min)
                min = M[i][j+1];
            if(M[i][j] < min)
                k = k + (min - M[i][j]);
        }
    cout << k;
    return 0;
}
```

3.

a.

Algoritmul proiectat afișează pe ecran numărul de valori distincte din șirul aflat pe a doua linie a fișierului care aparțin intervalului [x, y] folosindu-se de o variabilă ajutătoare ce permite mereu salvarea numărului actual din fișier.

Algoritmul proiectat este eficient din punct de vedere al timpului de executare, întrucât acesta este unul liniar.

Algoritmul proiectat este eficient din punct de vedere al memoriei utilizate, întrucât acesta utilizează doar câteva variabile și nici măcar un tablou.

b.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ifstream f("bac.txt");
    int x, y, nr1, nr2 = 0, k = 0;
    f >> x >> y;
    while(f >> nr1)
    {
        if(nr1 >= x && nr1 <= y && nr1 != nr2){
            nr2 = nr1;
            k++;
        }
    }
    cout << k;
```

```
    f.close();
    return 0;
}
```

- Rezolvarea subiectului de bacalaureat la informatică, sesiunea august-septembrie 2022

Subiectul I:

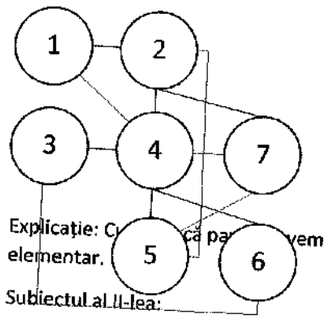
1. b (!(x<3 || x>=10))
Explicație: ! x<3 este x >= 3; ! || este &&; ! x>=10 este x < 10

2. a (y + 2)
Explicație: La o scurtă analiză a subprogramului, se poate observa că singura variantă corectă este a. Scăderea, adunarea sau împărțirea cu 2 ar putea duce la erori, rulări infinite sau răspunsuri greșite ale apelului.

3. d ((prună, caisă, piersică))
Explicație: Varianta (gutuie, prună, piersică) nu este acceptată de cerință, astfel continuăm cu prima variantă posibilă, anume prună, urmată evident de prună și piersică.

4. c (79)
Explicație: Dacă indicii pornesc din 0, avem relația i + j == n - 1. Cum i = 20, n = 100, rămâne 20 + j == 99, deci j = 79

5. b ([1,4] și [5,7])
Desen:



Explicație: C[illegible] ca pa[illegible]vem un drum mai lung, acesta ar exista doar dacă nu ar fi elementar.

Subiectul al II-lea:

1.
a. 45
b. Oricare două numere din intervalul [10,14].
c.

```
#include <iostream>
using namespace std;
int main()
{
    int m, n, x, y;
```

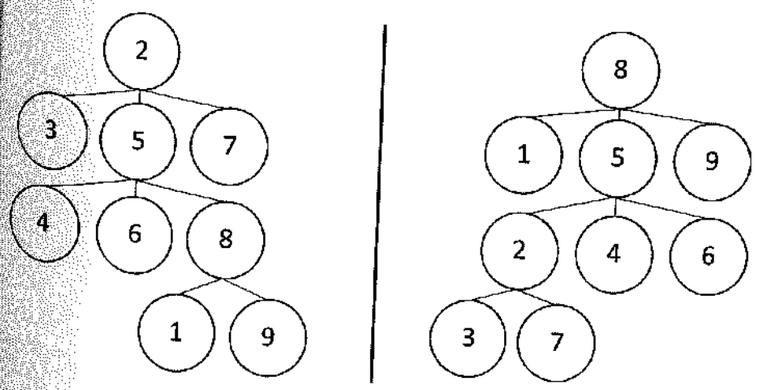


```
    cin >> m >> n;
    x = 1;
    while(x == 1)
    {
        x = m; y = n; n--;
        while(x != y)
            if(x > y)
                x = x - y;
            else
                y = y - x;
    }
    cout << n + 1;
    return 0;
}
```

d.

```
citește m,n (numere naturale nenule, m<n)
x<-1
┌dacă x=1 atunci
│  ┌execută
│  │   x<-m; y<-n; n<-n-1
│  │  ┌cât timp x≠y execută
│  │  │  ┌dacă x>y atunci x<-x-y
│  │  │  └altfel y<-y-x
│  │  └■
│  └■cât timp x=1
└■
scrie n+1
```

2. 2 și 8
Desen:



3.

```
if(x.temperatura > 11)
    cout << "CALDUROS";
else if(x.temperatura < 10)
    cout << "RACOROS";
else
    cout << "NORMAL";
```

Subiectul al III-lea:

1.

```
void schimb(int &n, int x, int p)
{
    int aux, poz = 0, prod = 1; aux = n, n = 0;
    while(aux)
    {
        if(poz != p)
        {
            n = n + (aux % 10) * prod;
            prod = prod * 10;
        }
        else
        {
            n = n + x * prod;
            prod = prod * 10;
        }
        aux = aux / 10; poz++;
    }
}
```

2.

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char sir[50], voc; int i;
    cin >> sir;
    for(i = 0; i < strlen(sir); i++)
        if(strchr("aeiou", sir[i]) == NULL)
        {
            if('a' < sir[i]) voc = 'a';
            if('e' < sir[i]) voc = 'e';
            if('i' < sir[i]) voc = 'i';
            if('o' < sir[i]) voc = 'o';
            if('u' < sir[i]) voc = 'u';
            sir[i] = voc;
        }
    cout << sir;
```



```
    return 0;
}
```

3.

a.

Algoritmul proiectat afișează pe ecran numărul secvențelor paritare de lungime maximă precum și această lungimea maximă folosindu-se de mai multe variabile ajutătoare care permit calcularea lungimii actuale, lungii maxime, dar și numărului de secvențe.

Algoritmul proiectat este eficient din punct de vedere al timpului de executare, întrucât acesta este unul liniar.

Algoritmul proiectat este eficient din punct de vedere al memoriei utilizate, întrucât acesta utilizează doar câteva variabile și nici măcar un tablou.

b.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ifstream f("bac.txt");
    int nr1, nr2, secv = 0, secvTotala = 0, l = 1, lmax = 0;
    f >> nr1;
    while(f >> nr2)
    {
        if(nr1 % 2 == nr2 % 2)
        {
            l++;
            if(l > lmax)
            {
                lmax = secv = l; secvTotala = 0;
            }
            if(l == lmax && secv == lmax)
                secvTotala++;
        }
        else
            l = 1;
        nr1 = nr2;
    }
    cout << secvTotala << " " << lmax;
    f.close();
    return 0;
}
```