**MINISTERUL EDUCAŢIEI**

**COLEGIUL NAŢIONAL “FRATII BUZESTI” CRAIOVA**

PROIECT

pentru

examenul de atestare a competențelor profesionale a

absolvenților claselor de matematică-informatică

**Subprograme**

Profesor îndrumător:

**LILIANA SCHIOPU**

Absolvent:

**Tanase Elena-Alexandra**

Clasa a XII-a A

- **2023** -

**CUPRINS**

[INTRODUCERE 3](#_Toc1424983470)

[Motivație 3](#_Toc205151968)

[CAPITOLUL 1. Subprograme 4](#_Toc1015525520)

[Definiție 4](#_Toc238936324)

[Avantaje 4](#_Toc59088487)

[Structura 4](#_Toc1660520185)

[Terminologie 6](#_Toc698198565)

[Parametrii unui subprogram 7](#_Toc666378926)

[Clasificare 8](#_Toc1848399427)

[Variabile 8](#_Toc1191417794)

[Transmiterea parametrilor 11](#_Toc637432624)

[Tablouri si funcții 13](#_Toc1699657394)

[CAPITOLUL 2. Aplicație 17](#_Toc1453777230)

[BIBLIOGRAFIE 18](#_Toc1565268782)

## INTRODUCERE

# Motivație

Am ales tema “Subprograme” pentru lucrarea de atestat, deoarece acestea sunt foarte importante in programarea practica C++, iar avantajele oferite de acestea sunt folositoare in programarea de zi cu zi.

Subprogramele, care se mai numesc in limbajul C++ și “funcții”, fac posibila redactarea programului într-un mod mai eficient si mai lizibil/ plăcut estetic, codul sursă arată mult mai aerisit, așadar facilitează citirea și înțelegerea acestuia.

Subprogramele împart o problemă complexă în mai multe probleme mici si simple. Prin intermediul mai multor funcții simple se rezolvă cu ușurință o problema mai complexă.

## CAPITOLUL 1. Subprograme

# Definiție

Blocul este unitatea de bază a unui program C++ și este implementat printr-o instrucțiune compusă, adică nu set de instrucțiuni delimitate de acolade.

Subprogramul este o secvență de instrucțiuni care rezolvă o anumită sarcină și care poate fi descrisă separat de blocul principal și lansată în execuție din cadrul unui bloc, de câte ori este nevoie. În limbajul C++ subprogramele se mai numesc și funcții.

# Avantaje

* Se face economie de memorie internă
* O secvență de instrucțiuni care trebuie să se scrie de mai multe ori într-o aplicație se va scrie o singură dată într-un subprogram și se va executa prin apelarea subprogramului ori de câte ori este nevoie.
* Permite lucrul în echipă
* Fiecare programator scrie subprograme, independent de ceilalți programatori din echipă.
* Depanarea și actualizarea aplicației se fac mai ușor
* Eventualele modificări pentru schimbarea unor cerințe se fac mai simplu la nivelul unui subprogram decât la nivelul întregii aplicații.
* Crește portabilitatea programelor
* Subprogramele dintr-o aplicație pot fi folosite și în alte aplicații in care trebuie rezolvate sarcini similare.

Un program C++ conține cel puțin o funcție **main()** care este apelată la lansarea în execuție a programului, funcții predefinite (de bibliotecă) și funcții construite de programator.

# Structura

Un subprogram sau o funcție are o definiție si atâtea apeluri câte sunt necesare.

**Definiția** unei funcții:

* **Antetul** subprogramului

tip\_rezultat nume\_subprogram (lista\_parametri\_formali)

*lista\_parametri\_formali* este de forma:

tip1 p1, tip2 p2, ..., tipk pk unde tip este tipul parametrului formal (int, char, \*char, void, float, etc.) și p este identificatorul parametrului formal.

*tip\_rezultat* este tipul rezultatului. Poate fi un tip de date sau *void* (dacă nu se returnează un rezultat prin numele subprogramului). În cazul in care tipul rezultatului este diferit de void, corpul funcției trebuie sa conțină cel puțin o instrucțiune return care asigura ieșirea din apel.

* **Corpul** funcției
  + este o instrucțiune compusă care conține declarații și instrucțiuni.
  + apar variabile locale.

{

Instrucțiune 1;

Instrucțiune 2;

(...)

Instrucțiune n;

[return expresie;]

}

**Elemente implicate în utilizarea subprogramelor**

* prototipul subprogramului
  + declara subprogramul
* apelul subprogramului
  + execută subprogramul
* definiția subprogramului
  + descrie antetul si corpul subprogramului
  + antetul subprogramului specifica numele subprogramului si tipul parametrilor si al valorilor returnate
  + corpul subprogramului specifica ceea ce trebuie sa realizeze subprogramul

Ex:

#include <iostream>

using namespace std;

//prototipul funcției

int main(){

//declarații

//instrucțiuni

//apelul funcției

return 0;

}

//definiția funcției

**SAU**

#include <iostream>

using namespace std;

//definiția funcției

int main(){

//declarații

//instrucțiuni

//apelul funcției

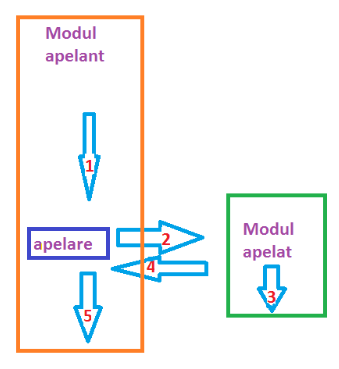
return 0;

}

# Terminologie

Într-o structură modulară în care fiecare modul este descris printr-un subprogram, modulele se clasifică astfel:

* + **Modul apelant –** este modulul care apeleaza alte module
  + **Modul apelat** – este un modul apelat de alt modul



# Parametrii unui subprogram

**Parametri formali** – apar in antetul subprogramului si sunt utilizați de subprogram pentru descrierea abstracta a unui proces de calcul.

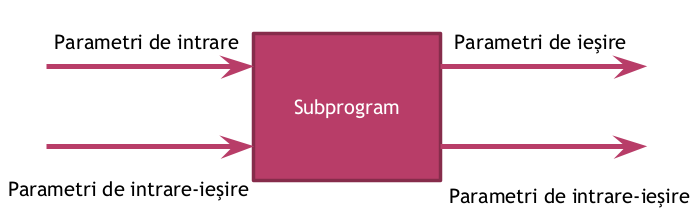
**Parametri actuali –** apar in instrucțiunea de apelare a unui subprogram și sunt folosiți la execuția unui proces de calcul pentru valori concrete.

Parametrii formali nu sunt variabile. O variabila este caracterizată de nume, tip și adresă. Legarea unui parametri formal la o adresă se realizează în timpul execuției instrucțiunii de apelare a subprogramului.

**Parametri de intrare** – sunt date comunicate de modulul apelant si primite de subprogram în vederea prelucrării.

**Parametri de ieșire** - sunt rezultatele obținute de subprogram și sunt primite de modulul apelant după terminarea execuției programului.

**Parametri de intrare-ieșire** - sunt date accesibile modulului apelant și modificate de acestea. Subprogramul le primește la începerea apelului și modulul apelant le primește la terminarea execuției subprogramului.



# Clasificare

Subprogramele se pot clasifica după:

* + Modalitatea de apel
    - Subprograme apelate ca instrucțiuni procedurale (sau funcții fără tip)
    - Subprograme apelate ca operanzi (sau funcții cu tip)
  + Autor
    - Subprograme predefinite sau subprograme standard (de sistem)

Ex: sqrt(), strcpy(), min(), max(), etc.

* + - Subprograme utilizator

# Variabile

Există mai multe zone de memorie in care sistemul alocă spațiu de memorie pentru variabile:

|  |  |
| --- | --- |
| Zona de adrese libere  (HEAP) | Alocare dinamică realizată prin variabile de tip pointer si utilizarea operatorilor **new** si **delete** (alocare explicită) |
| Stiva sistemului (stack) | Variabile locale (alocare implicită) |
| Segment date | Variabile globale (alocare implicită)  Variabile locale statice (alocare cu **static**) |
| Regiștrii procesorului | Variabile locale (alocare cu **register**) |

1) **Variabile cu durată locală**

* + Se numesc si variabile automatice. Sunt variabile create într-o funcție
  + Sunt create si distruse automat de compilator, la începerea apelului funcției, respectiv la sfârșitul apelului.
  + Sunt stocate temporar in stivă (doar pe timpul execuției funcției)
  + La declarare, vor avea o valoare nedefinita. De aceea, acestea trebuie inițializate

2) **Variabile cu durată statică**

* + Sunt variabile globale sau variabile locale statice
  + Sunt stocate in segmentul de date, la compilare, din momentul declarării și până la terminarea execuției funcției
  + La declarare, sunt inițializare cu valoarea zero

3) **Variabile cu durată dinamică**

* + Sunt create in timpul execuției programului si durează cât este necesar
  + Nu sunt inițializate
  + Spațiul de memorie se alocă în timpul execuției programului in HEAP

Ex:

#include <iostream>

using namespace std;

void f(){

int x=20;

//variabila locala x din f

x++; cout << x << ‘ ‘;

}

int main(){

int x=30;

//variabila locala x din main

cout << x << ‘ ’;

f();

int a=40;

//variabila locala a din main

cout << a << ‘ ‘;

return 0;

}

**Se afiseaza: 31 21 40**

#include <iostream>

using namespace std;

void f1(){

int x1=60;

//variabila locala x1 din f1

x1++;

cout << x1 << ‘ ‘;

}

void f2(){

int x2=90;

//variabila locala x2 din f2

x2++;

cout << x2 << ‘ ‘;

}

int main(){

int x=80;

//variabila locala x din main

cout << x << ‘ ‘;

f1();

f2();

f1();

f2();

return 0;

}

**Se afiseaza: 80 61 91 61 91**

**Variabile globale**

* Sunt variabile definite în exteriorul oricărei funcții.
* Sunt vizibile în toate funcțiile care sunt declarate după definirea lor.
* Variabilele declarate în funcția main sunt variabile locale.
* La declarare, variabilele globale sunt inițializate cu valoarea zero.
* Durata de existență este statică, adică pe toată durata execuției programului, din momentul în care au fost declarate.
* Li se alocă spațiu de memorie în segmentul de date.

Ex:

#include<iostream>

using namespace std;

int a;

//variabila globala a

void f(){

int x=20;

//variabila locala x din f

x++; cout<<x<<‘ ‘;

a++; cout<<a<<‘ ‘;

}

int main(){

a=50;

f();

cout<<a<<‘ ‘;

return 0;

}

**Se afiseaza: 21 51 51**

#include<iostream>

using namespace std;

int a;

//variabila globala a

void f(){

int x=20;

//variabila locala x din f

x++; cout<<x<<‘ ‘;

a++; cout<<a<<‘ ‘;

}

int b;

//variabila globala b

int main(){

a=70; b=80;

f();

cout<<a<<‘ ‘;

b++;

cout<<b<<‘ ‘;

f();

return 0;

}

**Se afiseaza: 21 71 71 81 21 72**

# Transmiterea parametrilor

* Transmiterea prin valoare
  + Parametrul prin intermediul căruia se face transmiterea prin valoare se numește **parametru de valoare.**
  + Se folosește pentru parametrii de intrare.
  + La apel, parametrii actuali sunt expresii.
* Transmiterea prin referință
  + Parametrul prin intermediul căruia se face transmiterea prin referință se numește **parametru variabilă** (sau **referință**).
  + Se folosește pentru parametrii de ieșire și parametrii de intrare-ieșire.
  + În definiția funcției, identificatorul parametrului este precedat de operatorul “&”.
  + La apel, parametrii actuali sunt variabile.

Ex: inversul unui număr natural.

* + Definiția funcției

void invers(int n, int &inv){

inv = 0;

while(n){

inv = inv\*10 + n%10;

n /= 10;

}

}

* + Apelul funcției

invers(a, inv1); cout << inv1 << endl;

invers(n, inv2); cout << inv2 << endl;

invers(3497, inv 3); cout << inv3;

* + Explicație:
    - Funcția determină inversul numărului natural transmis prin intermediul parametrului **n** și returnează inversul lui **n** prin intermediul parametrului **inv.**
    - Parametrii funcției **n** este parametru de intrare (transmitere prin valoare), **inv** este parametru de ieșire (transmitere prin referință).

# Tablouri si funcții

* **Vectori și funcții**

Antetul funcției prin care se actualizează un vector, fără a se modifica numărul de elemente (**v** este parametru de intrare-ieșire, **n** este parametru de intrare).

**void f1(int v[100], int n)**

Antetul funcției prin care se actualizează un vector, iar numărul de elemente se modifică (**v** este parametru de intrare-ieșire, **n** este parametru de intrare-ieșire).

**void f2(int v[100], int &n)**

Antetul prin care se parcurge un vector fără a se modifica numărul de elemente (**v** și **n** parametri de intrare).

**void f3 (int v[100], int n)**

Obs. Dacă vectorul este parametru de ieșire sau parametru de intrare-ieșire, numele vectorului nu este precedat de operatorul referință deoarece numele vectorului este pointer constant.

Ex:

Se citește un vector și se efectuează operații asupra sa: afișarea vectorului, suma elementelor și verificarea proprietății de mulțime.

#include<iostream>

#define MAX 100

using namespace std;

void citire(int v[MAX],int &n){

cin>>n;

for(int i=0;i<n;i++)

cin>>v[i];

}

void afisare(int v[MAX],int n){

for(int i=0;i<n;i++)

cout<<v[i]<<‘ ‘;

}

long suma(int v[MAX],int n){

long s=0;

for(int i=0;i<n;i++)

s+=v[i];

return s;

}

int multime(int v[MAX],int n){

for(int i=0;i<n-1;i++)

for(int j=i+1;j<n;j++)

if(v[i]==v[j])

return 0;

return 1;

}

int main(){

int t[MAX],m;

citire(t,m);

afisare(t,m);

cout<<suma(t,m)<<endl;

if(multime(t,m))

cout<<“da”;

else

cout<<“nu”;

return 0;

}

Antetul funcției prin care se actualizează o matrice, fără a se modifica numărul liniilor și coloanelor (**a** este parametru de intrare-ieșire, **m** și **n** sunt parametri de intrare).

**void f1(int a[100][100], int m, int n)**

Antetul funcției prin care se actualizează o matrice, iar numărul liniilor și coloanelor se modifică (**a, m, n** sunt parametrii de intrare-ieșire).

**void f2(int a[100][100], int &m, int &n)**

Antetul funcției prin care se parcurge o matrice, fără a se modifica numărul liniilor si coloanelor (**a, m, n** sunt parametri de intrare).

**void f3(int a[100][100], int m, int n)**

Obs. Dacă matricea este parametru de ieșire sau parametru de intrare-ieșire, numele matricii nu este precedat de operatorul referință deoarece numele matricii este pointer constant.

Ex:

Se citește o matrice cu m linii și n coloane, iar elementele numere întregi și se efectuează operații asupra sa: afișarea matricii, suma elementelor și afișarea elementelor pare.

#include<iostream>

#define MAX 100

using namespace std;

void citire(int a[MAX][MAX],int &m,int &n){

cin>>m>>n;

int i,j;

for(i=0;i<m;i++)

for(j=0;j<n;j++)

cin>>a[i][j];

}

void afisare(int a[MAX][MAX],int m,int n){

int i,j;

for(i=0;i<m;i++){

for(j=0;j<n;j++)

cout<<a[i][j]<<‘ ‘;

cout<<endl;

}

}

long suma(int a[MAX][MAX],int m, int n){

long s=0; int i,j;

for(i=0;i<m;i++)

for(j=0;j<n;j++)

s+=a[i][j];

return s;

}

void pare(int a[MAX][MAX],int m, int n){

int i,j;

for(i=0;i<m;i++)

for(j=0;j<n;j++)

if(a[i][j]%2==0)

cout<<a[i][j]<<endl;

}

int main(){

int t[MAX][MAX],m,n;

citire(t,m,n);

afisare(t,m,n);

cout<<suma(t,m,n)<<endl;

pare(t,m,n);

return 0;

}

## CAPITOLUL 2. Aplicație

Drept aplicație am ales sa creez un program relativ lung, care dacă nu ar fi fost realizat cu ajutorul funcțiilor/ subprogramelor ar fi durat mult mai mult timp să fie scris și să fie simplu de înțeles.

#include <iostream>  
#include <cmath>  
  
using namespace std;  
  
void comenzi(){  
 cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";  
 cout << "1. Suma dintre oglinditul lui n si oglinditul lui m." << endl;  
 cout << "2. Verificare daca numarul x este prim." << endl;  
 cout << "3. Determinarea celui mai mare divizor comun (cmmdc) "  
 "a doua numere n si m" << endl;  
 cout << "4. Sa se afiseze toate numerele din intervalul [1000,99999] \n"  
 "care au proprietatea ca atat ele cat si oglinditul lor sunt prime.\n";  
 cout << "5. Sa se afiseze primii n termeni ai sirului lui Fibonacci.\n";  
 cout << "6. Sa se afiseze primele n (n < 101) numere prime folosind Ciurul lui Eraostene\n";  
 cout << "7. Ordonarea si interclasarea a doi vectori cu n, m elemente.\n"  
 "(cei doi vectori de intrare sunt ordonati crescator)\n";  
 cout << "8. Afiseaza il afiseaza pe Darth Vader din Star Wars.\n";  
 cout << "0. Programul ia sfarsit.\n";  
 cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";  
 cout << "Numarul problemei dorite: ";  
}  
  
int oglindire(int n){  
 int oglindit=0;  
 while(n){  
 oglindit = oglindit\*10 + n%10;  
 n = n/10;  
 }  
 return oglindit;  
}  
  
int prim(int x){  
 bool ok=true;  
 for(int d=2; d<=x/2 && ok; d++)  
 if(x%d == 0)  
 ok=false;  
 return ok;  
}  
  
int cmmdc(int a, int b){  
 while(a != b)  
 if(a > b)  
 a -= b;  
 else  
 b -= a;  
 return a;  
}  
  
void fibonacci(unsigned long n){  
 unsigned long f1=1, f2=1, c;  
 cout << f1 << " " << f2 << " ";  
 for(unsigned long i=3; i<=n; i++){  
 c = f1 + f2;  
 cout << c << " ";  
 f1 = f2;  
 f2 = c;  
 }  
}  
int v[100];  
void Eratostene(int n){  
 v[0] = 1;  
 v[1] = 1;  
 for(int i=2; i<= sqrt(n); i++)  
 if(v[i] == 0)  
 for(int j=2; j<=n/i; j++)  
 v[i\*j] = 1;  
}  
  
void cin\_vector(int x[100], int n){  
 for(int i=0; i<n; i++)  
 cin >> x[i];  
}  
  
void ord\_intr(int a[100], int b[100], int c[200], int n, int m){  
 int i=0, j=0, k=0;  
 while(i<n && j<m){  
 if(a[i] < b[j])  
 c[k++]=a[i++];  
 else  
 c[k++]=b[j++];  
 }  
 if(i<=n){  
 for(int p=i; p<n; p++)  
 c[k++] = a[p];  
 }  
 if(j<=m)  
 for(int p=j; p<m; p++)  
 c[k++] = b[p];  
 for(int p=0; p<k; p++)  
 cout << c[p] << " ";  
}  
  
void darthVader(){  
 cout << ""  
 " .-.\n"  
 " |\_:\_|\n"  
 " /(\_Y\_)\\\n"  
 "|.\\ ( \\/M\\/ )\n"  
 "\\'.\\ \_.'-/'-'\\-'.\_\n"  
 " \\':\\ \_/.--'[[[[]'--.\\\_\n"  
 " \\':\\ /\_' : |::\"| : '.\\\n"  
 " \\':\\ // ./ |ooo| \\.' :\\\n"  
 " \\':\\ \_:'..' \\\_|\_\_\_|\_/ : :|\n"  
 " \\':\\ .' |\_[\_\_\_]\_| :.':\\\n"  
 " [::\\ | : | | : ; : \\\n"  
 " '-' \\/'.| |.' \\ .;.' |\n"  
 " |\\\_ \\ '-' : |\n"  
 " | \\ \\ .: : | |\n"  
 " | \\ | '. : \\ |\n"  
 " / \\ :. .; |\n"  
 " / | | :\_\_/ : \\\\\n"  
 " | | | \\: | \\ | ||\n"  
 " / \\ : : |: / |\_\_| /|\n"  
 " | : : :\_/\_| /'.\_\\ '--|\_\\\n"  
 " /\_\_\_.-/\_|-' \\ \\\n"  
 " '-'"  
 "\n";  
}  
  
int main() {  
 int tasta;  
 int x, n, m, a[100], b[100], c[200];  
  
 cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";  
 cout << "\t Selectati numarul unei probleme pentru a o executa: \n";  
 comenzi();  
  
 do{  
 cin >> tasta;  
 switch (tasta) {  
 case 0:  
 cout << "Sfarsit program..." << endl;  
 break;  
 case 1:  
 cout << "n="; cin >> n;  
 cout << "m="; cin >> m;  
 cout << oglindire(n) + oglindire(m) << endl;  
 comenzi();  
 break;  
 case 2:  
 cout << "x="; cin >> x;  
 if(prim(x))  
 cout << "Numarul " << x << " este numar prim. \n";  
 else  
 cout << "Numarul " << x << " NU este numar prim. \n";  
 comenzi();  
 break;  
 case 3:  
 cout << "n="; cin >> n;  
 cout << "m="; cin >> m;  
 cout << cmmdc(n, m) << endl;  
 comenzi();  
 break;  
 case 4:  
 for(x=1000; x<100000; x++){  
 if(prim(x) && prim(oglindire(x)))  
 cout << x << " ";  
 }  
 cout << endl;  
 comenzi();  
 break;  
 case 5:  
 cout << "n="; cin >> n;  
 fibonacci(n);  
 cout << endl;  
 comenzi();  
 break;  
 case 6:  
 cout << "n="; cin >> n;  
 Eratostene(n);  
 for(int i=1; i<=n; i++)  
 if(!v[i])  
 cout << i << " ";  
 cout << endl;  
 comenzi();  
 break;  
 case 7:  
 cout << "Cate elemente are vectorul A? "; cin >> n;  
 cout << "Elementele vectorului A: "; cin\_vector(a, n);  
 cout << "Cate elemente are vectorul B? "; cin >> m;  
 cout << "Elementele vectorului B: "; cin\_vector(b, m);  
 ord\_intr(a, b, c, n, m);  
 cout << endl;  
 comenzi();  
 break;  
 case 8:  
 darthVader();  
 comenzi();  
 break;  
 default:  
 cout << "Numarul introdus este incorect, incercati din nou." << endl;  
 comenzi();  
 break;  
 }  
 }while(tasta != 0);  
 return 0;  
}

Vom explica acum utilizarea fiecărei funcții:

1. **Funcția comenzi**

void comenzi(){  
 cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";  
 cout << "1. Suma dintre oglinditul lui n si oglinditul lui m." << endl;  
 cout << "2. Verificare daca numarul x este prim." << endl;  
 cout << "3. Determinarea celui mai mare divizor comun (cmmdc) "  
 "a doua numere n si m" << endl;  
 cout << "4. Sa se afiseze toate numerele din intervalul [1000,99999] \n"  
 "care au proprietatea ca atat ele cat si oglinditul lor sunt prime.\n";  
 cout << "5. Sa se afiseze primii n termeni ai sirului lui Fibonacci.\n";  
 cout << "6. Sa se afiseze primele n (n < 101) numere prime folosind Ciurul lui Eraostene\n";  
 cout << "7. Ordonarea si interclasarea a doi vectori cu n, m elemente.\n"  
 "(cei doi vectori de intrare sunt ordonati crescator)\n";  
 cout << "8. Afiseaza il afiseaza pe Darth Vader din Star Wars.\n";  
 cout << "0. Programul ia sfarsit.\n";  
 cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";  
 cout << "Numarul problemei dorite: ";  
}

* + folosind funcția comenzi(); afișăm lista de comenzi după fiecare execuție a unei ramuri din “switch” fără a mai fi nevoie sa scriem toate liniile de cod de fiecare dată, astfel scrierea programului e mai eficientă.

1. **Funcția oglindire**

int oglindire(int n){  
 int oglindit=0;  
 while(n){  
 oglindit = oglindit\*10 + n%10;  
 n = n/10;  
 }  
 return oglindit;  
}

* + prin intermediul funcției oglindire(n); calculăm și returnăm oglinditul numărului n citit de la tastatură.
  + se folosește in instrucțiunea switch de două ori, dar folosind funcția am scris codul o singură dată.

1. **Funcția prim**

int prim(int x){  
 bool ok=true;  
 for(int d=2; d<=x/2 && ok; d++)  
 if(x%d == 0)  
 ok=false;  
 return ok;  
}

* + această funcție determină dacă un număr este prim.

1. **Funcția cmmdc**

int cmmdc(int a, int b){  
 while(a != b)  
 if(a > b)  
 a -= b;  
 else  
 b -= a;  
 return a;  
}

* + determină cel mai mare divizor comun a două numere.

1. **Funcția fibonacci**

void fibonacci(unsigned long n){

unsigned long f1=1, f2=1, c;  
 cout << f1 << " " << f2 << " ";  
 for(unsigned long i=3; i<=n; i++){  
 c = f1 + f2;  
 cout << c << " ";  
 f1 = f2;  
 f2 = c;  
 }  
}

* + afișează primele **n** numere ce se regăsesc în șirul lui Fibonacci.

1. **Funcția Eratostene**

int v[100];  
void Eratostene(int n){  
 v[0] = 1;  
 v[1] = 1;  
 for(int i=2; i<= sqrt(n); i++)  
 if(v[i] == 0)  
 for(int j=2; j<=n/i; j++)  
 v[i\*j] = 1;  
}

* + determină primele **n** numere folosind Ciurul lui Eratostene.

1. **Funcția cin\_vector**

void cin\_vector(int x[100], int n){  
 for(int i=0; i<n; i++)  
 cin >> x[i];

* + ne ajută să citim un vector fără a mai fi nevoiți să scriem de fiecare dată algoritmul de citire al unui vector, scurtând astfel timpul de scriere al algoritmului.

1. **Funcția ord\_intr**

void ord\_intr(int a[100], int b[100], int c[200], int n, int m){  
 int i=0, j=0, k=0;  
 while(i<n && j<m){  
 if(a[i] < b[j])  
 c[k++]=a[i++];  
 else  
 c[k++]=b[j++];  
 }  
 if(i<=n){  
 for(int p=i; p<n; p++)  
 c[k++] = a[p];  
 }  
 if(j<=m)  
 for(int p=j; p<m; p++)  
 c[k++] = b[p];  
 for(int p=0; p<k; p++)  
 cout << c[p] << " ";  
}

* + ordonează și interclasează doi vectori după care afișează vectorul ordonat.

1. **Funcția darthVader()**

void darthVader(){  
 cout << ""  
 " .-.\n"  
 " |\_:\_|\n"  
 " /(\_Y\_)\\\n"  
 "|.\\ ( \\/M\\/ )\n"  
 "\\'.\\ \_.'-/'-'\\-'.\_\n"  
 " \\':\\ \_/.--'[[[[]'--.\\\_\n"  
 " \\':\\ /\_' : |::\"| : '.\\\n"  
 " \\':\\ // ./ |ooo| \\.' :\\\n"  
 " \\':\\ \_:'..' \\\_|\_\_\_|\_/ : :|\n"  
 " \\':\\ .' |\_[\_\_\_]\_| :.':\\\n"  
 " [::\\ | : | | : ; : \\\n"  
 " '-' \\/'.| |.' \\ .;.' |\n"  
 " |\\\_ \\ '-' : |\n"  
 " | \\ \\ .: : | |\n"  
 " | \\ | '. : \\ |\n"  
 " / \\ :. .; |\n"  
 " / | | :\_\_/ : \\\\\n"  
 " | | | \\: | \\ | ||\n"  
 " / \\ : : |: / |\_\_| /|\n"  
 " | : : :\_/\_| /'.\_\\ '--|\_\\\n"  
 " /\_\_\_.-/\_|-' \\ \\\n"  
 " '-'"  
 "\n";  
}

* + folosind codul ASCII afișează portretul lui Darth Vader.

**Funcția main()**

int main() {  
 int tasta;  
 int x, n, m, a[100], b[100], c[200];  
  
 cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";  
 cout << "\t Selectati numarul unei probleme pentru a o executa: \n";  
 comenzi();  
  
 do{  
 cin >> tasta;  
 switch (tasta) {  
 case 0:  
 cout << "Sfarsit program..." << endl;  
 break;  
 case 1:  
 cout << "n="; cin >> n;  
 cout << "m="; cin >> m;  
 cout << oglindire(n) + oglindire(m) << endl;  
 comenzi();  
 break;  
 case 2:  
 cout << "x="; cin >> x;  
 if(prim(x))  
 cout << "Numarul " << x << " este numar prim. \n";  
 else  
 cout << "Numarul " << x << " NU este numar prim. \n";  
 comenzi();  
 break;  
 case 3:  
 cout << "n="; cin >> n;  
 cout << "m="; cin >> m;  
 cout << cmmdc(n, m) << endl;  
 comenzi();  
 break;  
 case 4:  
 for(x=1000; x<100000; x++){  
 if(prim(x) && prim(oglindire(x)))  
 cout << x << " ";  
 }  
 cout << endl;  
 comenzi();  
 break;  
 case 5:  
 cout << "n="; cin >> n;  
 fibonacci(n);  
 cout << endl;  
 comenzi();  
 break;  
 case 6:  
 cout << "n="; cin >> n;  
 Eratostene(n);  
 for(int i=1; i<=n; i++)  
 if(!v[i])  
 cout << i << " ";  
 cout << endl;  
 comenzi();  
 break;  
 case 7:  
 cout << "Cate elemente are vectorul A? "; cin >> n;  
 cout << "Elementele vectorului A: "; cin\_vector(a, n);  
 cout << "Cate elemente are vectorul B? "; cin >> m;  
 cout << "Elementele vectorului B: "; cin\_vector(b, m);  
 ord\_intr(a, b, c, n, m);  
 cout << endl;  
 comenzi();  
 break;  
 case 8:  
 darthVader();  
 comenzi();  
 break;  
 default:  
 cout << "Numarul introdus este incorect, incercati din nou." << endl;  
 comenzi();  
 break;  
 }  
 }while(tasta != 0);  
 return 0;  
}

* în funcția main folosim o instrucțiune “do... while()” pentru a citi tasta corespunzătoare problemei ce va fi rezolvată folosind programul, după care o instrucțiune de tip “switch” care determină numărul citit și il încadrează într-unul din cazuri (număr al problemei).
* în interiorul instrucțiunii switch urmează să fie utilizată fiecare funcție **cel puțin** o dată

**//aplicatia se gaseste si in proiectul atasat ulterior, pentru a putea fi rulata in Visual Studio ☺**

## BIBLIOGRAFIE

1. Alin Burţa, *Informatică - Manual pentru clasa a XII-a*, Editura ALL Educaţional, 2007.
2. Mioara Gheorghe, Monica Tătărâm, Corina Achinca, Ioana Pestriţu, *Informatică-Manual pentru clasa a XII-a*, Editura Corint, 2007.
3. Prof. Alina Cîlțan - *Subprograme Partea 1, 2, 3 și 4.*
4. [Subprograme - Daniela Liliana Nicolae (danielanicolae.com)](https://danielanicolae.com/algoritmi/subprograme/)