Ministerul Educaţiei, Culturii și Cercetării

al Republicii Moldova   
Universitatea Tehnică a Moldovei

FCIM  
  
  
  
  
  
  
  
RAPORT

# LUCRARE DE LABORATOR NR. 2

**LA DISCIPLINA „PROGRAMAREA ORIENTATĂ PE OBIECTE”**

A efectuat: st. gr. TI-211 Popa Cătălin   
  
A verificat: Lisnic Inga

UTM, Chișinău 2022

**Tema:** Constructorul – funcţie de iniţializare a obiectelor clasei

**Scopul lucrării:**

1.Studierea principiilor de definire şi utilizare a constructorilor

2.Studierea principiilor de definire şi utilizare a destructorilor

3.Studierea tipurilor de constructori

Varianta 10

а) Să se creeze clasa *Soft* – fişier, care conţine informaţia despre numele complet al fişierului şi anexa de asociere (doc – Word, psd – Photoshop, etc.), utilizînd memoria dinamică, mărimea, data şi timpul creării. Să se definească toi constructorii. Constructorul de schimbare a tipului are parametrul – numele de fişier. Să se definească funcţiile de redenumire a fişierului, de transfer în alt directoriu şi de modificare a anexei de asociere.

#include <iostream>

using namespace std;

class Soft

{

    string name\_file;

    string anexa;

    int data;

    int time\_ore,time\_minute;

    double size;

    string directoriu;

    public:

    Soft(string name\_file1, string anexa1, int data1, int time\_ore1, int time\_minute1, double size1, string directoriu1)

    {

        name\_file = name\_file1;

        anexa = anexa1;

        data = data1;

        time\_ore = time\_ore1;

        time\_minute = time\_minute1;

        size = size1;

        directoriu = directoriu1;

    }

     void change\_name(string);

     void change\_anexa(string);

     void change\_directoriu(string);

    // Soft(const char\*);

    void citire();

    void afisare();

     ~Soft(){cout <<"\n// Fisierul a fost distrus\\\\\n" <<endl;}

};

// Soft::Soft(char\* name)

// {

//     name\_file = name;

// }

void Soft::afisare()

{

    cout << " \nName | "<<name\_file<< endl;

    cout << " \nAnexa | "<<anexa<< endl;

    cout << " \nData | "<<data<< endl;

    cout << " \nOra | "<<time\_ore<< " : "<<time\_minute<< endl;

    cout << " \nMarimea | "<<size<< endl;

    cout << " \nDirectoriu | "<<directoriu<< endl;

}

void Soft::change\_name(string new\_name)

{

    name\_file = new\_name;

}

void Soft::change\_anexa(string new\_anexa)

{

    anexa = new\_anexa;

}

void Soft::change\_directoriu(string new\_directoriu)

{

    directoriu = new\_directoriu;

}

int main()

{

    string name\_name, anexa1, directoriu1;

    Soft \*a;

    ///Crearea obiectului folosind memoria dinamica si constructor cu parametri

    a = new Soft("test","exe",12,4,21,200,"folder");

    a->afisare();

    cout << "\n////////////////////////" <<endl;

    ///Schimbuarea numelui fisierului

    cout <<"\nDati denumirea noua: \n";

    cin >> name\_name;

    a->change\_name(name\_name);

    a->afisare();

    cout << "\n////////////////////////" <<endl;

    ///Schimbuarea anexeifisierului

    cout <<"\nDati anexa noua: \n";

    cin >> anexa1;

    a->change\_anexa(anexa1);

    a->afisare();

    cout << "\n////////////////////////" <<endl;

    ///Schimbuarea directoriului fisierului

    cout <<"\nDati directoriul nou: \n";

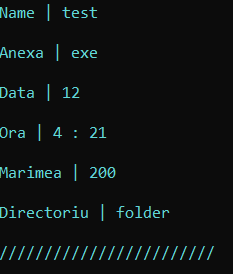
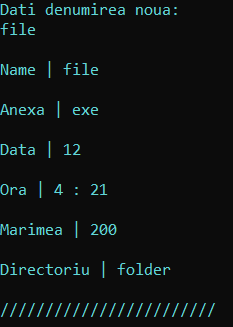
    cin >> directoriu1;

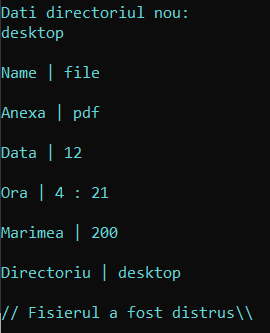
    a->change\_directoriu(directoriu1);

    a->afisare();

    delete a;

}

b) Să se creeze clasa *Matrix –* matrice. Clasa conţine pointer spre *long,* numărul de rînduri şi de coloane şi o variabilă – codul erorii. Să se definească constructorul fără parametri (constructorul implicit), constructorul cu un parametru – matrice pătrată şi constructorul cu doi parametri – matrice dreptunghiulară ş. a. Să se definească funcţiile membru de acces: returnarea şi setarea valorii elementului (i,j). Să se definească funcţiile de adunare şi scădere a două matrice; înmulţirea unei matrice cu alta; înmulţirea unei matrice cu un număr. Să se testeze funcţionarea clasei. În caz de insuficienţă de memorie, necorespondenţă a dimensiunilor matricelor, depăşire a limitei memoriei utilizate să se stabilească codul erorii.

#include <iostream>

using namespace std;

class Matrix

{

    long int \*\*p;

    int randuri;

    int coloane;

    bool error;

public:

    Matrix();

    Matrix(int randuri);

    Matrix(int randuri, int coloane);

    ~Matrix();

    void insert();

    void afisare();

    void suma(Matrix first\_matrix);

    void diferenta(Matrix first\_matrix);

    void multiply\_with\_num(int num);

    void multiply\_matrix(Matrix matrix);

    void multiply(Matrix a);

};

void Matrix::insert()

{

    for (int i = 0; i < randuri; i++)

    {

        for (int j = 0; j < coloane; j++)

        {

            cin >> p[i][j];

        }

    }

}

void Matrix::afisare()

{

    for (int i = 0; i < randuri; i++)

    {

        for (int j = 0; j < coloane; j++)

        {

            cout << p[i][j] << " ";

        }

        cout << "\n";

    }

}

Matrix::~Matrix()

{

    for (int i = 0; i < randuri; i++)

    {

        delete[] \* (p + i);

    }

    delete[] p;

    randuri = 0;

    coloane = 0;

    error = 1;

    cout << "\n Matricea a fost eliberata!!\n";

}

Matrix::Matrix()

{

    randuri = 0;

    coloane = 0;

    p = NULL;

    error = 1;

}

Matrix::Matrix(int num)

{

    if (num > 0)

    {

        randuri = num;

        coloane = num;

        p = new long int \*[num];

        for (int i = 0; i < num; i++)

        {

            p[i] = new long int[num];

        }

        error = 0;

    }

    else

    {

        error = 1;

    }

}

Matrix::Matrix(int randuri1, int coloane1)

{

    if (randuri > 0 && coloane > 0)

    {

        randuri = randuri1;

        coloane = coloane1;

        p = new long int \*[randuri1];

        for (int i = 0; i < randuri1; i++)

        {

            p[i] = new long int[coloane1];

        }

        error = 0;

    }

    else

    {

        error = 1;

    }

}

void Matrix::suma(Matrix first\_matrix)

{

    if (randuri == first\_matrix.randuri && coloane == first\_matrix.coloane)

    {

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

        {

            for (int j = 0; j < coloane; j++)

            {

                p[i][j] = p[i][j] + first\_matrix.p[i][j];

            }

        }

    }

    else

    {

        cout << "\nAdunarea nu poate fi efectuata!!!\n"

             << endl;

    }

}

void Matrix::diferenta(Matrix first\_matrix)

{

    if (first\_matrix.randuri == randuri && first\_matrix.coloane == coloane)

    {

        for (int i = 0; i < first\_matrix.randuri; i++)

        {

            for (int j = 0; j < first\_matrix.coloane; j++)

            {

                p[i][j] = p[i][j] - first\_matrix.p[i][j];

            }

        }

    }

    else

    {

        cout << "\nScaderea nu poate fi efectuata!!!\n"

             << endl;

    }

}

void Matrix::multiply\_with\_num(int n)

{

    for (int i = 0; i < randuri; i++)

    {

        for (int j = 0; j < coloane; j++)

        {

            p[i][j] = p[i][j] \* n;

        }

    }

}

void Matrix::multiply(Matrix M2)

{

    int M[randuri][coloane];

    for (int i = 0; i < randuri; i++)

    {

        for (int j = 0; j < coloane; j++)

        {

            M[i][j] = p[i][j];

        }

    }

    if (coloane == M2.coloane && randuri == M2.randuri)

    {

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

        {

            for (int j = 0; j < M2.coloane; j++)

            {

                p[i][j] = 0;

                for (int k = 0; k < coloane; k++)

                {

                    p[i][j] += ((M[i][k]) \* (M2.p[k][j]));

                }

            }

        }

    }

    else if (coloane != M2.coloane && randuri != M2.randuri)

    {

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

        {

            for (int j = 0; j < M2.coloane; j++)

            {

                p[i][j] = 0;

                for (int k = 0; k < coloane; k++)

                {

                    p[i][j] += ((M[i][k]) \* (M2.p[k][j]));

                }

            }

        }

        coloane = M2.coloane;

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

    {

        for (int j = 0; j < coloane; j++)

        {

            cout << p[i][j] << " ";

        }

        cout << "\n";

    }

    }

}

    ////////////////////////////////////////////////////////////////////////////////////////

    int main()

    {

        int count;

        cout << "\n1.Adunarea a doua matrice: \n";

        cout << "\n2.Scaderea a doua matrice: \n";

        cout << "\n3.Inmultirea a doua matrice: \n";

        cout << "\n4.Inmultirea cu un numar a unei matrice: \n";

        cout << "\nIntroduceti o optiune: \n";

        cin >> count;

        if (count == 1)

        {

            cout << "\nIntroduceti valorile primei matrice(2x2): \n";

            Matrix first\_matrix(2, 2);

            first\_matrix.insert();

            cout << "\nIntroduceti valorile altei matrice(2x2): \n";

            Matrix second\_matrix(2, 2);

            second\_matrix.insert();

            cout << "\nSuma matricelor 2x2 este: \n";

            first\_matrix.suma(second\_matrix);

            first\_matrix.afisare();

        }

        else if (count == 2)

        {

            cout << "\nIntroduceti valorile primei matrice(3x2): \n";

            Matrix first\_matrix(3, 2);

            first\_matrix.insert();

            cout << "\nIntroduceti valorile altei matrice(3x2): \n";

            Matrix second\_matrix(2, 3);

            second\_matrix.insert();

            cout << "\nDiferenta matricelor 4x4 este: \n";

            first\_matrix.diferenta(second\_matrix);

            first\_matrix.afisare();

        }

        else if (count == 3)

        {

            cout << "\nIntroduceti valorile primei matrice(2x2): \n";

            Matrix first\_matrix(3, 2);

            first\_matrix.insert();

            cout << "\nIntroduceti valorile altei matrice(2x2): \n";

            Matrix second\_matrix(2, 3);

            second\_matrix.insert();

            cout << "\nProdusul matricelor 2x2 este: \n";

            first\_matrix.multiply(second\_matrix);

            first\_matrix.afisare();

            cout << "\nttttttttttttttttt";

        }

        else if (count == 4)

        {

            cout << "\nIntroduceti valorile matricei: \n";

            Matrix first\_matrix(2);

            first\_matrix.insert();

            int number;

            cout << "\n\nIntroduceti un numar: \n";

            cin >> number;

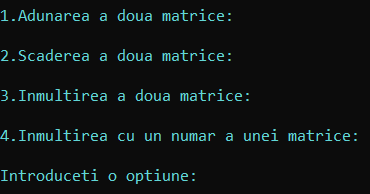
            cout << "\n";

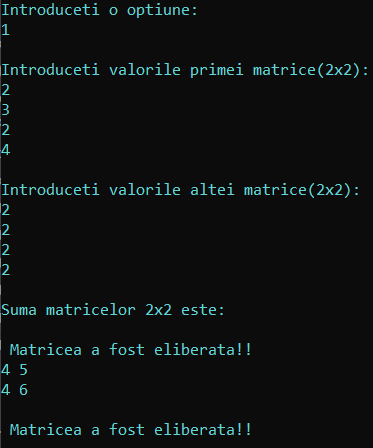
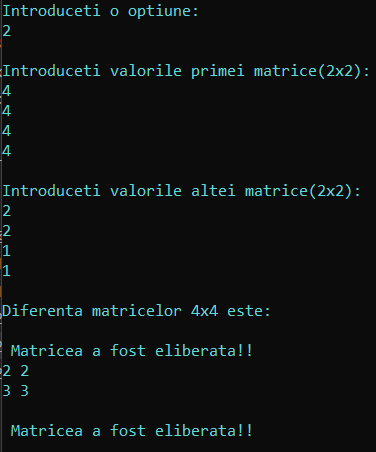
            first\_matrix.multiply\_with\_num(number);

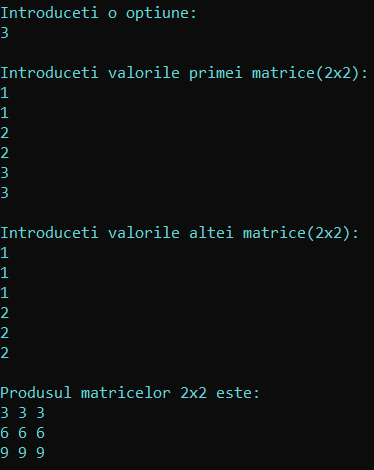
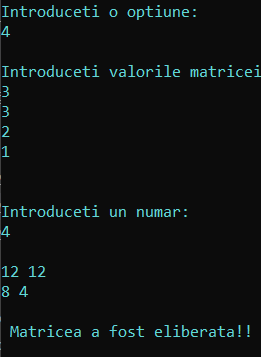
            first\_matrix.afisare();

        }

    }



**Concluzii:**

1.Verificarea ne arata ca rezultatele obtinute sunt corecte si programul lucreaza corect.

2.Au fost obtinute cunoștințe în studierea principiilor de definire şi utilizare a

constructorilor și distructurilor, a constructorilor impliciți, de copiere, citire, de conversie

a tipului și constructor general.