**Ministerul Educației al Republicii Moldova**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

**Departamentul Ingineria Software și Automatică.**

**Raport**

Lucrarea de laborator nr.2

Disciplina: Programarea Orientata pe Obiect

Tema: Constructorul – funcţie de iniţializare a obiectelor clasei

Varianta: 5.

**Efectuat**: st.gr. TI-207 Bunescu Gabriel

**Verificat**: Lisnic Inga

Chișinău 2021

**Scopul lucrării:**

* Studierea principiilor de definire şi utilizare a constructorilor
* Studierea principiilor de definire şi utilizare a destructorilor

Studierea tipurilor de constructori

**Sarcina:**

Varianta 5

а) Să se creeze clasa *Document –* document, care conţine informaţia despre denumirea, tema, autorul documentului utilizînd memoria dinamică; numărul de pagini, data şi timpul ultimei redactări. Să se definească toţi constructorii. Constructorul de schimbare a tipului are ca parametru denumirea documentului. Să se definească funcţiile de modificare a temei, datei ultimei redactări ş. a.

b) Să se creeze clasa *Matrix –* matrice. Clasa conţine pointer spre *char*, 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>

#include <string>

using namespace std;

class Document {

string\* name = new string;

string\* theme = new string;

string\* author = new string;

string\* date = new string;

string\* lastEdit = new string;

int\* pages = new int;

public:

Document();

Document(const Document&);

Document(string name) {

\*this->name = name;

}

void setName(string name) { \*this->name = name; }

void setTheme(string theme) { \*this->theme = theme; }

void setAuthor(string author) { \*this->author = author; }

void setDate(string date) { \*this->date = date; }

void setLastEditDate(string lastEdit) { \*this->lastEdit = lastEdit; }

void setPages(int pages) { \*this->pages = pages; }

void show()

{

cout << "\n\n\n\n\n Numele documentului: " << \*this->name;

cout << "\n Tema documentului: " << \*this->theme;

cout << "\n Autorul documentului: " << \*this->author;

cout << "\n Data crearii: " << \*this->date;

cout << "\n Ultima editare: " << \*this->lastEdit;

cout << "\n Numarul de pagini: " << \*this->pages;

cout << "\n\n\n\n\n\n";

}

~Document()

{

delete name;

delete theme;

delete author;

delete date;

delete lastEdit;

delete pages;

}

};

Document::Document() {}

void editDocument(Document\* doc)

{

string tempString;

int tempInt;

cin.ignore();

cout << "Numele documentului: ";

getline(cin, tempString);

doc->setName(tempString);

cout << "Tema: ";

getline(cin, tempString);

doc->setTheme(tempString);

cout << "Autorul: ";

getline(cin, tempString);

doc->setAuthor(tempString);

cout << "Data crearii: ";

getline(cin, tempString);

doc->setDate(tempString);

cout << "Data ultimei editari: ";

getline(cin, tempString);

doc->setLastEditDate(tempString);

cout << "Numarul de pagini: ";

cin >> tempInt; doc->setPages(tempInt);

}

int main()

{

Document\* doc = new Document();

int option = 1;

while (option != 0)

{

cout << "[1] - Editarea documentului \n";

cout << "[2] - Afisarea documentului \n\n";

cout << "[0] - Iesire din program \n\n\n\n";

cin >> option;

switch (option)

{

case 1: editDocument(doc);

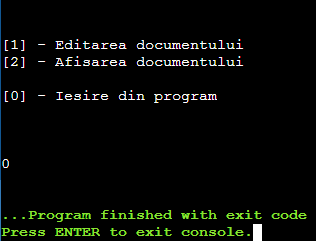
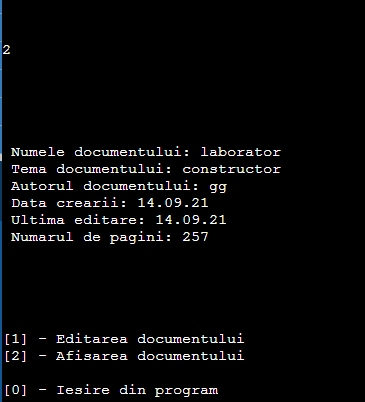
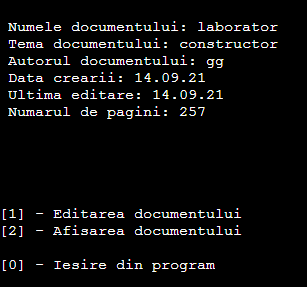
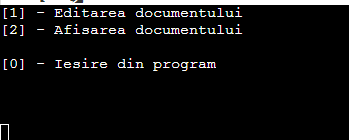
case 2: doc->show();

default: break;

}

}

}



#include <iostream>

#include <string>

#include <vector>

using namespace std;

class Matrix

{

private:

int rows;

int columns;

float\*\* m;

string error;

void createMatrix()

{

m = new float\*[rows];

if (m == NULL)

{

cout << "Memoria nu a fost alocata";

return;

}

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

{

m[i] = new float[columns];

}

}

public: Matrix();

Matrix(int size)

{

this->rows = size;

this->columns = size;

createMatrix();

};

Matrix(int r, int c)

{

this->rows = r;

this->columns = c;

createMatrix();

}

~Matrix()

{

delete m;

}

void set(int x, int y, float value)

{

m[x][y] = value;

}

float get(int x, int y)

{

return m[x][y];

}

void addMatrix(Matrix\* matrix)

{

if (rows != matrix->rows || columns != matrix->columns)

{

cout << "Numarul de randuri si coloane nu coincide";

return; }

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

{

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

{

m[i][n] += matrix->get(i, n);

}

}

}

void decreaseMatrix(Matrix\* matrix)

{

if (rows != matrix->rows || columns != matrix->columns)

{

cout << "Numarul de randuri si coloane nu coincide";

return;

}

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

{

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

{

m[i][n] -= matrix->get(i, n);

}

}

} void multiplyMatrix(int number)

{

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

{

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

{

m[i][n] \*= number;

}

}

}

void multiplyMatrix(Matrix\* matrix)

{

if (rows != matrix->columns || columns != matrix->rows)

{

cout << "Numarul de randuri si coloane nu coincide";

return;

}

cout << "\n\n\n";

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

{

for (int n = 0; n < matrix->columns;n++)

{

int value = 0;

for (int i2 = 0; i2 < rows; i2++)

{

value += m[i][i2] \* matrix->get(i2, i);

}

cout << value << " ";

}

cout << "\n"; } }

void show()

{

cout << "\n";

for (int i = 0;

i < rows; i++)

{

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

{

cout << get(i, n) << " ";

}

cout << "\n";

}

cout << "\n\n\n\n";

}

};

Matrix::Matrix() {}

vector<Matrix\*> matrixes;

void createNewMatrix()

{

int rows,

columns, value;

cout << "Numarul de coloane: ";

cin >> columns;

cout << "Numarul de randuri: ";

cin >> rows;

Matrix\* m = new Matrix(rows, columns);

matrixes.push\_back(m);

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

{

for (int m = 0; m < columns; m++)

{

cout << "Elementul din coloana " << m+1 << ", randul " << i+1 << ": ";

cin >> value; matrixes.back()->set(i, m, value);

}

}

}

void showAllMatrixes()

{

for (int i = 0; i < matrixes.size();i ++)

{

cout << "\n\nMatricea #" << i + 1 << endl;

matrixes[i]->show();

}

if (matrixes.size() == 0)

{

cout << "Nu este nici o matrice creata!";

}

}

void addMatrixes() {

int first,

second; showAllMatrixes();

cout << "Indicati numarul matricei de baza: ";

cin >> first;

cout << "Incati numarul matricei care va fi adunata: ";

cin >> second;

first--;

second--;

if (first < 0 || second < 0 || first >= matrixes.size() || second >= matrixes.size())

{

cout << "Numarul matricelor este incorect!";

}

else

{

matrixes[first]->addMatrix(matrixes[second]);

cout << "\nRezultatul:" << endl;

matrixes[first]->show();

}

} void decreaseMatrixes()

{

int first,second; showAllMatrixes();

cout << "Indicati numarul matricei de baza: ";

cin >> first;

cout << "Incati numarul matricei care va fi adunata: ";

cin >> second; first--;

second--;

if (first < 0 || second < 0 || first >= matrixes.size() || second >= matrixes.size())

{

cout << "Numarul matricelor este incorect!";

}

else

{

matrixes[first]->decreaseMatrix(matrixes[second]);

cout << "\nRezultatul:" << endl;

matrixes[first]->show();

}

}

void multiplyMatrixWithNumber()

{

int first, second;

showAllMatrixes();

cout << "Indicati numarul matricei de baza: ";

cin >> first;

cout << "Incati numarul cu care matrice va fi inmultita: ";

cin >> second;

first--;

if (first < 0 || first >= matrixes.size())

{

cout << "Numarul matricei este incorect!";

}

else

{

matrixes[first]->multiplyMatrix(second);

cout << "\nRezultatul:" << endl;

matrixes[first]->show();

}

}

void multiplyMatrixWithMatrix()

{

int first, second;

showAllMatrixes();

cout << "Indicati numarul matricei de baza: ";

cin >> first;

cout << "Incati numarul matricei care va fi adunata: ";

cin >> second;

first--;

second--;

if (first < 0 || second < 0 || first >= matrixes.size() || second >= matrixes.size()) {

cout << "Numarul matricelor este incorect!";

return;

}

matrixes[first]->multiplyMatrix(matrixes[second]);

}

void showMenu()

{

int option = 1;

while (option != 0)

{

cout << "\n\n"; cout << "[1] - Crearea unei matrice" << endl;

cout << "[2] - Afiseaza toate matricele" << endl;

cout << "[3] - Adunarea a 2 matrice" << endl;

cout << "[4] - Scaderea a 2 matrice" << endl;

cout << "[5] - Inmultirea a 2 matrice" << endl;

cout << "[6] - Inmultirea matricei cu un numar" << endl;

cout << "\n[0] - Iesire din program \n\n\n"; cin >> option;

system("CLS");

switch (option)

{

case 1:

createNewMatrix();

break;

case 2:

showAllMatrixes();

break;

case 3:

addMatrixes();

break;

case 4:

decreaseMatrixes();

break;

case 5:

multiplyMatrixWithMatrix();

break;

case 6:

multiplyMatrixWithNumber();

break;

}

}

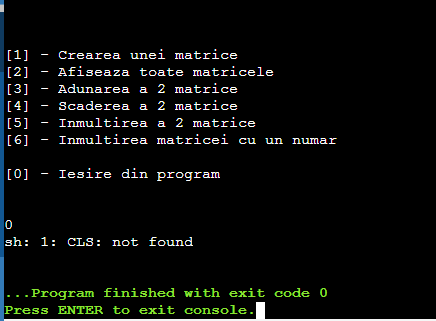
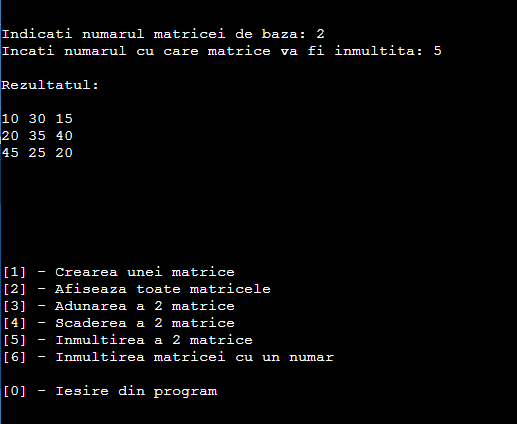
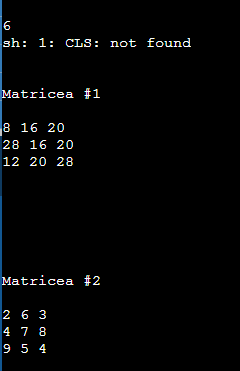
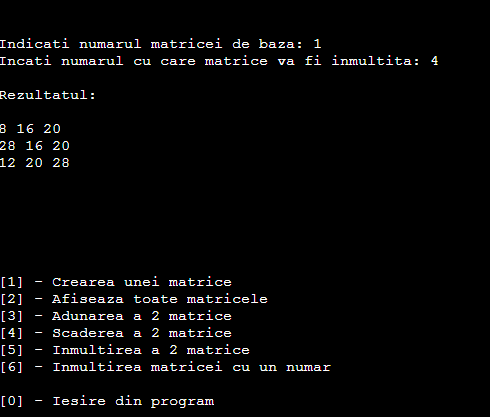
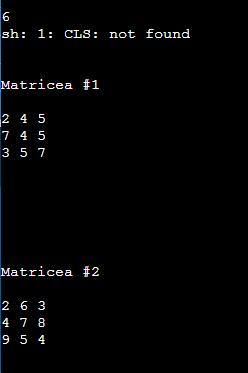
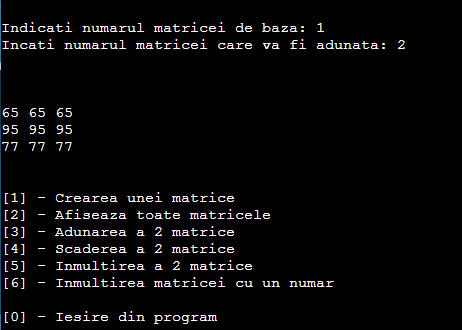
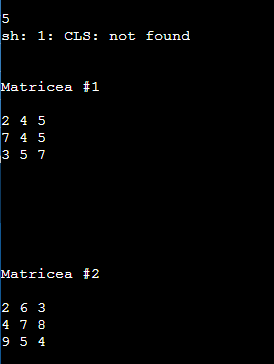
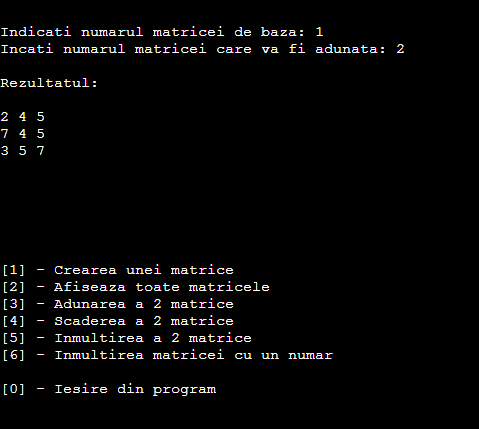
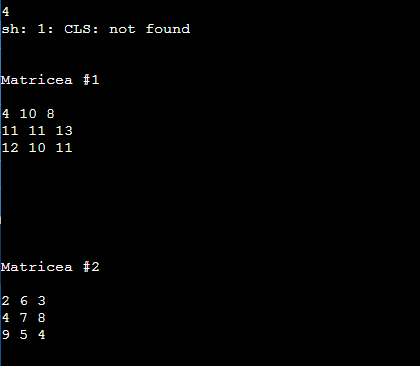
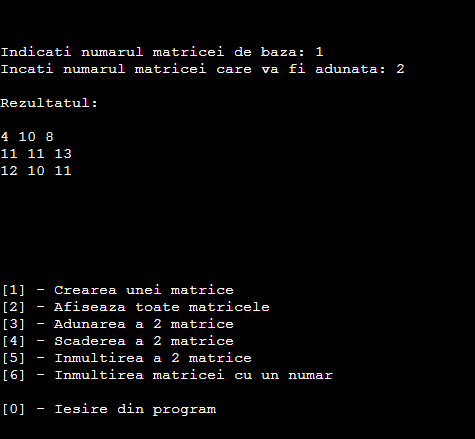
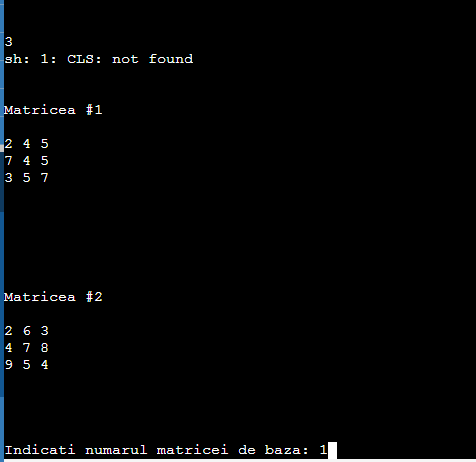
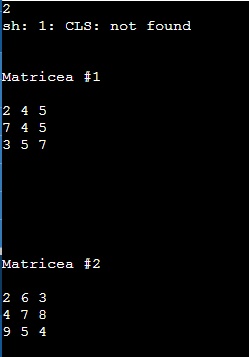
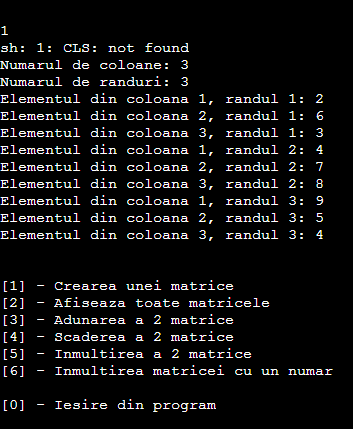
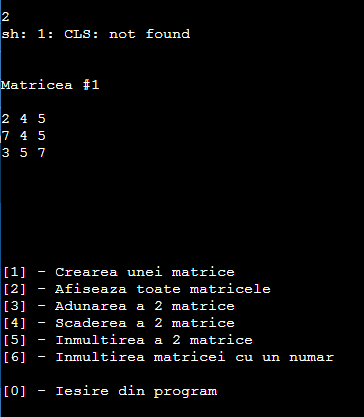
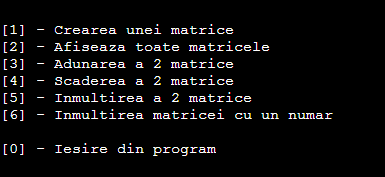
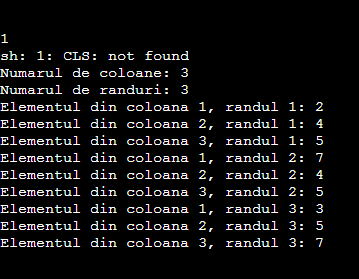
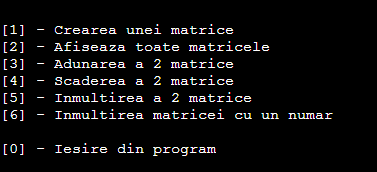
}

int main()

{

showMenu();

}



**Concluzie:**

Pe parcursul elaborării acestei lucrări de laborator eu am studiat programarea prin abstractizarea datelor, regulilor de definire şi utilizare a structurilor de date si cum se creaza variabilelor de tip structură si accesarea componentelor unei structure.Am studiat despre principiilor de definire şi utilizare a constructorilor, principiilor de definire şi utilizare a destructorilor si tipurilor de constructori. Si miam adus aminte de matrici, adunarea,scaderea,inmultirea a doua matrici si inmultirea unei matrici cu un numar oarecare.