

**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII**

**AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Ingineria Software și Automatica**

**Raport**

**pentru lucrarea de laborator Nr. 2**

***la cursul de “Programarea orientată pe obiecte”***

A efectuat:

**Cucerenco Maxim, SI-191**

A verificat:

**Mititelu Vitalii**

**Chișinău – 2020**

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

**Scopul lucrării:**

- Studierea principiilor de definire şi utilizare a constructorilor

- Studierea principiilor de definire şi utilizare a destructorilor

- Studierea tipurilor de constructori

**Sarcini (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.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.

**Codul:**

**Soft:**

#include <iostream>

#include <cstring>

#include <chrono>

#include <stdint.h>

typedef std::chrono::system\_clock Clock;

class Soft {

public:

Soft();

Soft(Soft &new\_obj);

Soft(char \*SoftName);

~Soft();

void initSoft(char \*SoftName = NULL);

void renameTo(char \*newName);

void moveTo(char \*newPath);

void setExtension(char \*extension);

void setCurrentPath();

char\* getTime();

void displaySoftData();

private:

char \*\_SoftName;

char \*\_extension;

char \*\_path;

char \*\_creationTime;

int \_size;

};

Soft::Soft() {

char \*name = new char[9];

strcpy(name, "Soft.txt");

initSoft(name);

}

Soft::Soft(Soft &obj) {

this->\_SoftName = new char[strlen(obj.\_SoftName) + 7];

strcpy(this->\_SoftName, obj.\_SoftName);

strcat(this->\_SoftName, "(copy)");

this->\_extension = new char[strlen(obj.\_extension) + 1];

strcpy(this->\_extension, obj.\_extension);

this->\_path = new char[strlen(obj.\_path) + 1];

strcpy(this->\_path, obj.\_path);

this->\_creationTime = new char[strlen(obj.\_creationTime) + 1];

strcpy(this->\_creationTime, obj.\_creationTime);

this->\_size = obj.\_size;

}

Soft::Soft (char \*SoftName) {

initSoft(SoftName);

}

Soft::~Soft() {

std::cout << "\"" << this->\_SoftName << "." << this->\_extension;

std::cout << "\" S-a distrus" << std::endl;

delete[] this->\_SoftName;

delete[] this->\_extension;

delete[] this->\_path;

delete[] this->\_creationTime;

}

void Soft::initSoft(char \*fullName) {

char \*dotPosition = strstr(fullName, ".");

this->\_extension = new char[strlen(dotPosition+1)+1];

strcpy(this->\_extension, dotPosition+1);

\*dotPosition = '\0';

this->\_SoftName = new char[strlen(fullName)+1];

strcpy(this->\_SoftName, fullName);

this->\_path = new char[3];

strcpy(this->\_path, "~/");

this->\_size = 0;

char \*ptrr = getTime();

this->\_creationTime = new char[strlen(ptrr)+1];

strcpy(this->\_creationTime, ptrr);

std::cout << "Fisierul \"" << this->\_SoftName << ".";

std::cout << this->\_extension << "\" a fost creat" << std::endl;

delete ptrr;

delete fullName;

}

char\* Soft::getTime() {

auto now = Clock::now();

std::time\_t now\_c = Clock::to\_time\_t(now);

struct tm \*parts = std::localtime(&now\_c);

std::string date = std::to\_string(1900+parts->tm\_year) +"."

+ std::to\_string(1+parts->tm\_mon) +"."+ std::to\_string(parts->tm\_mday) +" "

+ std::to\_string(parts->tm\_hour) +":"+ std::to\_string(parts->tm\_min) +":"

+ std::to\_string(parts->tm\_sec);

char \*toreturn = new char[date.size() + 1];

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

toreturn[i] = date.at(i);

}

return toreturn;

}

void Soft::moveTo(char \*newPath) {

delete this->\_path;

int newLen = strlen(newPath);

this->\_path = new char[newLen + 1];

strcpy(this->\_path, newPath);

std::cout << "Soft-ul a fost mutat in \"" << this->\_path << "\"" << std::endl;

}

void Soft::renameTo(char \*newName) {

int newNameLen = strlen(newName);

if (newNameLen == 0) {

std::cout << "Numele nu a fost schimbat" << std::endl;

return;

}

delete this->\_SoftName;

this->\_SoftName = new char[newNameLen + 1];

strcpy(this->\_SoftName, newName);

std::cout << "Numele a fost modificat in \"" << this->\_SoftName << "\"";

std::cout << std::endl;

}

void Soft::setExtension(char \*extension) {

int newExtLen = strlen(extension);

if (newExtLen == 0) {

std::cout << "Extensia nu a fost schimbata" << std::endl;

return;

}

delete this->\_extension;

this->\_extension = new char[newExtLen + 1];

strcpy(this->\_extension, extension);

std::cout << "Extensia a fost modificata in \"" << this->\_extension << "\"";

std::cout << std::endl;

}

void Soft::displaySoftData() {

std::cout << std::endl;

std::cout << "Name : " << this->\_SoftName << std::endl;

std::cout << "Extension : " << this->\_extension << std::endl;

std::cout << "Size : " << this->\_size << std::endl;

std::cout << "Path : " << this->\_path << std::endl;

std::cout << "Creation : " << this->\_creationTime << std::endl;

}

=======================================================================**Codul programului:**

#include <iostream>

#include "Soft.cpp"

int main() {

Soft unu;

Soft doi;

char temp[30];

strcpy(temp, "lab1A");

unu.renameTo(temp);

strcpy(temp, "cpp");

unu.setExtension(temp);

strcpy(temp, "~/UTM/POO/Lab1");

unu.moveTo(temp);

Soft unuCopy(unu);

strcpy(temp, "raportLab2");

doi.renameTo(temp);

strcpy(temp, "odt");

doi.setExtension(temp);

strcpy(temp, "~/UTM/LFA/Lab2");

doi.moveTo(temp);

unu.displaySoftData();

doi.displaySoftData();

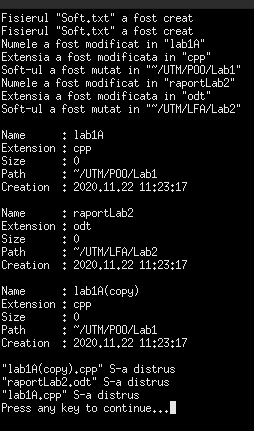
unuCopy.displaySoftData();

std::cout << std::endl;

return 0;

}

**Output-ul:** a



**Codul:** b

**Matrix**

#include <iostream>

#include <stdlib.h>

class Matrix {

public:

Matrix();

Matrix(long row, long col = 0);

~Matrix();

long &elAt(long i, long j);

void sum(Matrix &second);

void substract(Matrix &second);

void sumAndDiv(Matrix &second, int sign);

void multiplication(Matrix &second);

void multiplication(long element);

void resizeTo(long row, long col);

void afiseaza();

private:

long \_rows;

long \_cols;

long \_error;

long \*\*\_matrix;

};

Matrix::Matrix() {

this->\_rows = 0;

this->\_cols = 0;

this->\_matrix = NULL;

}

Matrix::Matrix(long row, long col) {

this->\_rows = row;

if (col == 0) this->\_cols = this->\_rows;

else this->\_cols = col;

this->\_matrix = (long\*\*)malloc(sizeof(long\*) \* this->\_rows);

for (long i = 0; i < this->\_rows; i++)

this->\_matrix[i] = (long\*)calloc(sizeof(long), this->\_cols);

}

Matrix::~Matrix () {

if (this->\_matrix != NULL) {

for (long i = 0; i < this->\_rows; i++)

free(this->\_matrix[i]);

free(this->\_matrix);

}

}

long &Matrix::elAt(long i, long j) {

return this->\_matrix[i][j];

}

void Matrix::sumAndDiv(Matrix &second, int sign) {

if (this->\_rows == 0 && this->\_rows == 0) {

std::cout << "Matricea principala este nula" << std::endl;

return;

}

if (second.\_rows == 0 && second.\_rows == 0) {

std::cout << "Matricea secundara este nula" << std::endl;

return;

}

if (this->\_rows != second.\_rows && this->\_cols != second.\_cols) {

this->\_error = 2;

std::cout << "Codul de eroare 2 - marimea matricilor nu corespund";

return;

}

for (long row = 0; row < this->\_rows; row++) {

for (long col = 0; col < this->\_cols; col++) {

this->\_matrix[row][col] += (sign \* second.\_matrix[row][col]);

}

}

}

void Matrix::sum(Matrix &second) { sumAndDiv(second, 1); }

void Matrix::substract(Matrix &second) { sumAndDiv(second, -1); }

void Matrix::multiplication(Matrix &second) {

if (this->\_rows == 0 && this->\_rows == 0) {

std::cout << "Matricea principala este nula" << std::endl;

return;

}

if (second.\_rows == 0 && second.\_rows == 0) {

std::cout << "Matricea secundara este nula" << std::endl;

return;

}

if (this->\_cols != second.\_rows) {

this->\_error = 2;

std::cout << "Codul de eroare 2 - marimea matricilor nu corespund";

return;

}

Matrix result(this->\_rows, second.\_cols);

long sum;

for (long mainRow = 0; mainRow < this->\_rows; mainRow++) {

for (long secondCol = 0; secondCol < second.\_cols; secondCol++) {

sum = 0;

for (long i = 0; i < this->\_rows; i++)

sum += this->\_matrix[mainRow][i] \* second.\_matrix[i][secondCol];

result.elAt(mainRow, secondCol) = sum;

}

}

resizeTo(this->\_rows, second.\_cols);

for (long row = 0; row < this->\_rows; row++) {

for (long col = 0; col < this->\_cols; col++) {

this->\_matrix[row][col] = result.elAt(row, col);

}

}

}

void Matrix::multiplication(long element) {

if (this->\_rows == 0 && this->\_rows == 0) {

std::cout << "Matricea este nula" << std::endl;

return;

}

for (long row = 0; row < this->\_rows; row++) {

for (long col = 0; col < this->\_cols; col++) {

this->\_matrix[row][col] \*= element;

}

}

}

void Matrix::resizeTo(long row, long col) {

this->\_rows = row;

this->\_cols = col;

if (this->\_matrix == NULL) {

this->\_matrix = (long\*\*)malloc(sizeof(long\*) \* this->\_rows);

for (long i = 0; i < this->\_rows; i++)

this->\_matrix[i] = (long\*)calloc(sizeof(long), this->\_cols);

}

else {

this->\_matrix = (long\*\*)realloc(this->\_matrix, sizeof(long\*)\*row);

for (int rw = 0; rw < row; rw++)

this->\_matrix[rw] = (long\*)realloc(this->\_matrix[rw], sizeof(long)\*col);

}

}

void Matrix::afiseaza() {

if (this->\_rows == 0 || this->\_cols == 0) {

std::cout << "Matricea e nula" << std::endl;

return;

}

std::cout << std::endl << "Se afiseaza matricea" << std::endl;

for (long r = 0; r < this->\_rows; r++) {

for (long c = 0; c < this->\_cols; c++) {

std::cout << this->\_matrix[r][c] << " ";

}

std::cout << std::endl;

}

}

**Codul :**

#include <iostream>

#include "Matrix.cpp"

int main() {

Matrix prima(3);

Matrix adoua(3);

int ceva[3][3] = {{5, 8, 4}, {6, 6, 6}, {1, 0, 1}};

int alta[3][3] = {{0, 0, 3}, {0, 0, 3}, {1, 1, 7}};

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

prima.elAt(i, j) = ceva[i][j];

}

for (int j = 0; j < 3; j++) {

adoua.elAt(i, j) = alta[i][j];

}

}

std::cout << "S-a creat prima matrice";

prima.afiseaza();

std::cout << std::endl << "S-a creat a doua matrice";

adoua.afiseaza();

prima.sum(adoua);

std::cout << std::endl << "S-a facut suma a doua matrici";

prima.afiseaza();

prima.multiplication(adoua);

std::cout << std::endl << "S-a inmultit prima matrice cu a doua";

prima.afiseaza();

prima.multiplication(5);

std::cout << std::endl << "S-a inmultit prima matrice cu numarul 5";

prima.afiseaza();

prima.substract(adoua);

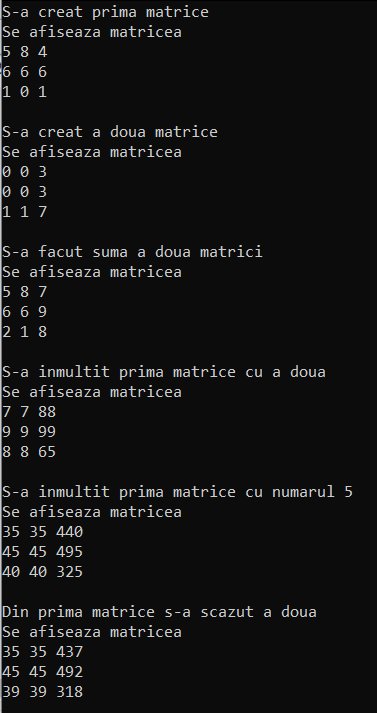
std::cout << std::endl << "Din prima matrice s-a scazut a doua";

prima.afiseaza();

return 0;

}

**Output-ul:** b



**Concluzie:** Efectuand aceasta lucrare de laborator am studiat tipurile de constructori, am studiat principiile de definire și utilizare a constructorilor si destructorilor.