****

**MINISTERUL EDUCAȚIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**Pascaru Ioana**

**Gr. MI-221**

**Raport**

**pentru lucrarea de laborator Nr.2**

***la cursul “Programarea Orientată pe Obiecte”***

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

Verificat:

**Buldumac Oleg,** *lect. univ.*

Facultatea FCIM, UTM

**Chișinău – 2023**

**Cuprins**

[**Introducere** 3](#_Toc148557298)

[**Sarcina I** 5](#_Toc148557299)

[**Codul programului cu comentarii relevante:** 5](#_Toc148557300)

[**Output-ul programului:** 8](#_Toc148557301)

[**Sarcina II** 9](#_Toc148557302)

[**Codul programului cu comentarii relevante:** 9](#_Toc148557303)

[**Output-ul programului:** 14](#_Toc148557304)

[**Concluzie** 15](#_Toc148557305)

# **Introducere**

**Scopul lucrării:**

• Studierea principiilor de definire şi utilizare a constructorilor

• Studierea principiilor de definire şi utilizare a destructorilor

• Studierea tipurilor de constructori

Constructorii reprezintă un concept fundamental în limbajul de programare C++. Aceștia sunt funcții speciale utilizate pentru a inițializa obiectele unei clase. Constructorii definesc modul în care obiectele unei clase sunt create și configurate în momentul instantierii.

În limbajul C++, fiecare clasă poate avea unul sau mai mulți constructori. Constructorii sunt invocați automat atunci când se creează obiecte ale respectivei clase și se ocupă de alocarea resurselor necesare, de inițializarea datelor și de efectuarea altor operații importante pentru a face obiectul utilizabil.

Un constructor are următoarele caracteristici:

1. **Numele Constructorului**: Constructorul are același nume ca și clasa în care este definit, dar fără tipul de returnare. De exemplu, pentru o clasă numită **Carte**, constructorul va fi denumit **Carte()**.
2. **Nu Are Tip de Returnare**: Constructorii nu returnează niciun tip de date, nici măcar **void**.
3. **Se Apelează Automat**: Constructorii sunt apelați automat atunci când un obiect al clasei este creat. Nu este necesară o apelare explicită din codul client.
4. **Pot Avea Parametri**: Constructorii pot accepta parametri pentru a inițializa obiectele cu valori specifice.

Constructorii pot fi supraincarcati, ceea ce înseamnă că o clasă poate avea mai mulți constructori cu semnături diferite. Aceasta oferă flexibilitate programatorului pentru a inițializa obiectele în mai multe moduri, în funcție de necesități.

# **Sarcina I**

а) Să se creeze clasa String – şir, utilizînd memoria dinamică. Să se definească constructorii: implicit, de copiere şi cu un parametru – pointer spre un şir de tip char. Să se definească funcţiile de atribuire a unui şir la altul, de comparaţie, de căutare a unui subşir, de numărare a simbolurilor ş. a.

## **Codul programului cu comentarii relevante:**

#include <iostream>

#include <cstring>

class String {

private:

    char\* data;

    int length;

public:

    String() : data(nullptr), length(0) {}

    String(const String& other) : data(nullptr), length(other.length) {

        if (length > 0) {

            data = new char[length + 1];

            strcpy(data, other.data);

        }

    }

    String(const char\* str) : data(nullptr), length(0) {

        if (str) {

            length = strlen(str);

            data = new char[length + 1];

            strcpy(data, str);

        }

    }

    ~String() {

        delete[] data;

    }

    String& operator=(const String& other) {

        if (this != &other) {

            delete[] data;

            length = other.length;

            if (length > 0) {

                data = new char[length + 1];

                strcpy(data, other.data);

            } else {

                data = nullptr;

            }

        }

        return \*this;

    }

    bool operator==(const String& other) const {

        return (length == other.length && strcmp(data, other.data) == 0);

    }

    bool operator!=(const String& other) const {

        return !(\*this == other);

    }

    int find(const String& substring) const {

        const char\* result = strstr(data, substring.data);

        if (result) {

            return result - data;

        }

        return -1;

    }

    int count(char symbol) const {

        int count = 0;

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

            if (data[i] == symbol) {

                count++;

            }

        }

        return count;

    }

    int getLength() const {

        return length;

    }

    const char\* c\_str() const {

        return data;

    }

};

int main() {

    String str1, str2, str3;

    // Citirea de la tastatură

    std::cout << "Introduceti sirul 1: ";

    char input[100];

    std::cin.getline(input, sizeof(input));

    str1 = input;

    std::cout << "Introduceti sirul 2: ";

    std::cin.getline(input, sizeof(input));

    str2 = input;

    std::cout << "Introduceti sirul 3: ";

    std::cin.getline(input, sizeof(input));

    str3 = input;

    if (str1 == str2) {

        std::cout << "str1 este egal cu str2." << std::endl;

    } else {

        std::cout << "str1 nu este egal cu str2." << std::endl;

    }

    int index = str3.find("is");

    if (index != -1) {

        std::cout << "Substring gasit la indexul " << index << std::endl;

    } else {

        std::cout << "Substring-ul nu a fost gasit." << std::endl;

    }

    char symbol;

    std::cout << "Introduceti un caracter cautat in str3: ";

    std::cin >> symbol;

    int count = str3.count(symbol);

    std::cout << "Caracterul '" << symbol << "' apare de" << count << " ori." << std::endl;

    std::cout << "str1: " << str1.c\_str() << std::endl;

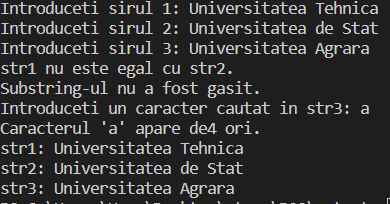
    std::cout << "str2: " << str2.c\_str() << std::endl;

    std::cout << "str3: " << str3.c\_str() << std::endl;

    return 0;

}

## **Output-ul programului:**



# **Sarcina II**

b) Să se creeze clasa Matrix – matrice. Clasa conţine pointer spre double, 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 programului cu comentarii relevante:**

#include <iostream>

#include <cstdlib>

#include <stdexcept>

class Matrix {

private:

    double\* data;

    int rows;

    int cols;

    int errorCode;

public:

    // Constructorul implicit

    Matrix() : data(nullptr), rows(0), cols(0), errorCode(0) {}

    // Constructor pentru o matrice pătrată

    Matrix(int n) : data(nullptr), rows(n), cols(n), errorCode(0) {

        if (n <= 0) {

            errorCode = -1; // Cod de eroare pentru dimensiuni incorecte

            return;

        }

        data = new double[n \* n];

    }

    // Constructor pentru o matrice dreptunghiulară

    Matrix(int rows, int cols) : data(nullptr), rows(rows), cols(cols), errorCode(0) {

        if (rows <= 0 || cols <= 0) {

            errorCode = -1; // Cod de eroare pentru dimensiuni incorecte

            return;

        }

        data = new double[rows \* cols];

    }

    // Destructor

    ~Matrix() {

        delete[] data;

    }

    // Funcții pentru accesarea și modificarea elementelor matricei

    double get(int i, int j) const {

        if (i < 0 || i >= rows || j < 0 || j >= cols) {

            throw std::out\_of\_range("Indicii specificati nu sunt in limitele matricei.");

        }

        return data[i \* cols + j];

    }

    void set(int i, int j, double value) {

        if (i < 0 || i >= rows || j < 0 || j >= cols) {

            throw std::out\_of\_range("Indicii specificati nu sunt in limitele matricei.");

        }

        data[i \* cols + j] = value;

    }

    // Funcții pentru adunare și scădere de matrici

    Matrix add(const Matrix& other) const {

        if (rows != other.rows || cols != other.cols) {

            throw std::invalid\_argument("Dimensiunile matricelor nu corespund.");

        }

        Matrix result(rows, cols);

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

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

                result.set(i, j, get(i, j) + other.get(i, j));

            }

        }

        return result;

    }

    Matrix subtract(const Matrix& other) const {

        if (rows != other.rows || cols != other.cols) {

            throw std::invalid\_argument("Dimensiunile matricelor nu corespund.");

        }

        Matrix result(rows, cols);

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

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

                result.set(i, j, get(i, j) - other.get(i, j));

            }

        }

        return result;

    }

    // Funcții pentru înmulțire de matrici și cu un număr

    Matrix multiply(const Matrix& other) const {

        if (cols != other.rows) {

            throw std::invalid\_argument("Numarul de coloane din prima matrice nu corespunde numarului de randuri din a doua matrice.");

        }

        Matrix result(rows, other.cols);

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

            for (int j = 0; j < other.cols; j++) {

                double sum = 0;

                for (int k = 0; k < cols; k++) {

                    sum += get(i, k) \* other.get(k, j);

                }

                result.set(i, j, sum);

            }

        }

        return result;

    }

    Matrix multiply(double scalar) const {

        Matrix result(rows, cols);

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

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

                result.set(i, j, get(i, j) \* scalar);

            }

        }

        return result;

    }

    // Funcție pentru afișarea matricei

    void print() const {

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

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

                std::cout << get(i, j) << ' ';

            }

            std::cout << std::endl;

        }

    }

    // Funcție pentru a verifica codul de eroare

    int getErrorCode() const {

        return errorCode;

    }

};

int main() {

    // Exemplu de utilizare a clasei Matrix

    Matrix mat1(2, 2);

    mat1.set(0, 0, 1);

    mat1.set(0, 1, 2);

    mat1.set(1, 0, 3);

    mat1.set(1, 1, 4);

    Matrix mat2(2, 2);

    mat2.set(0, 0, 5);

    mat2.set(0, 1, 6);

    mat2.set(1, 0, 9);

    mat2.set(1, 1, 8);

    // Adunare

    Matrix resultAdd = mat1.add(mat2);

    std::cout << "Matricea rezultat dupa adunare:\n";

    resultAdd.print();

    // Scădere

    Matrix resultSubtract = mat1.subtract(mat2);

    std::cout << "Matricea rezultat dupa scadere:\n";

    resultSubtract.print();

    // Înmultire

    Matrix mat3(2, 3);

    mat3.set(0, 0, 1);

    mat3.set(0, 1, 2);

    mat3.set(0, 2, 3);

    mat3.set(1, 0, 9);

    mat3.set(1, 1, 5);

    mat3.set(1, 2, 6);

    Matrix resultMultiply = mat1.multiply(mat3);

    std::cout << "Matricea rezultat dupa inmultire:\n";

    resultMultiply.print();

    // Înmulțire cu scalar

    Matrix resultScalar = mat1.multiply(2.0);

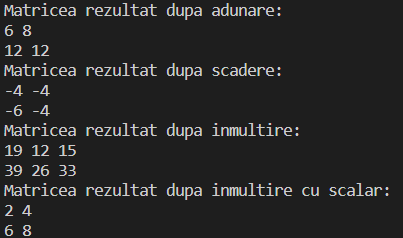
    std::cout << "Matricea rezultat dupa inmultire cu scalar:\n";

    resultScalar.print();

    return 0;

}

## **Output-ul programului:**



# **Concluzie**

În această lucrare, am explorat conceptul de constructori în limbajul de programare C++ în contextul codului care conținea structuri de date pentru matrice și șiruri de caractere (string-uri). Constructorii reprezintă elemente esențiale în programarea orientată pe obiect și sunt folosiți pentru inițializarea obiectelor claselor.

În contextul matricilor, constructorii au fost utilizați pentru a crea matrici pătrate și dreptunghiulare. Am definit constructori pentru inițializarea acestor matrici, fie fără parametri, pentru a crea matrice goale, fie cu un număr specific de rânduri și coloane. Acest lucru a permis crearea și configurarea obiectelor matrice în mod eficient, asigurând că acestea sunt gata de utilizare.

În ceea ce privește șirurile de caractere (string-urile), constructorii au fost folosiți pentru a inițializa șiruri de caractere, inclusiv constructori pentru șiruri goale și constructori care primeau un șir de caractere ca parametru. Acești constructori au facilitat gestionarea și manipularea eficientă a datelor de tip șir de caractere în cadrul codului.

Astfel, constructorii au jucat un rol crucial în crearea și inițializarea structurilor de date definite în acest cod. Ei ne-au permis să creăm obiecte personalizate, să le configurăm și să le folosim în mod eficient în cadrul aplicației noastre.

În concluzie, constructorii sunt un concept fundamental în C++ și reprezintă o componentă-cheie în programarea orientată pe obiect. Ei facilitează crearea și inițializarea obiectelor, sporind astfel eficiența și structura codului nostru.

Top of Form