

**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII**

**AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**gr. IA-231, Chistol Maxim**

**Raport**

**pentru lucrarea de laborator Nr.2**

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

Verificat:

Kulev Mihail dr., conf. univ.

Departamentul Informatică şi IS,

Facultatea FCIM, UTM

**Chișinău 2024**

**Tema:** Clase și obiecte. Constructori și destructor pentru clasa

**Scopul lucrării:**

• Studierea principiilor de definire şi utilizare a constructorilor

• Studierea principiilor de definire şi utilizare a destructorilor

• Studierea tipurilor de constructori

# **Sarcina Individual:**

**Varianta 7**

а) Să se creeze clasa Queue – coadă. Cîmpurile – numărul de elemente şi un pointer pentru alocarea dinamică a memoriei. Să se definească constructorii: implicit, de copiere şi cu un parametru – numărul necesar de elemente; funcţiile add şi get pentru punerea unui element în coadă şi pentru scoaterea unui element din coadă respectiv; funcţiile: isEmpty, care returnează valoarea 1 dacă coada este vidă şi zero în caz contrar, şi isFull care returnează valoarea 1 dacă coada este plină şi zero în caz contrar.

b) Să se creeze clasa Matrix – matrice. Clasa conţine pointer spre int, 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.

# **Listingul programului:**

**a)**

#include <iostream>

using namespace std;

class Queue {

private:

int\* data; // Pointer pentru alocarea dinamică a memoriei

int maxSize; // Numărul maxim de elemente

int front; // Indexul elementului din față

int rear; // Indexul ultimului element

int count; // Numărul actual de elemente în coadă

public:

// Constructor implicit

Queue() : maxSize(10), front(0), rear(-1), count(0) {

data = new int[maxSize];

}

// Constructor cu parametru - numărul necesar de elemente

Queue(int size) : maxSize(size), front(0), rear(-1), count(0) {

data = new int[maxSize];

}

// Constructor de copiere

Queue(const Queue& other) : maxSize(other.maxSize), front(other.front), rear(other.rear), count(other.count) {

data = new int[maxSize];

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

data[i] = other.data[i];

}

}

// Destructor

~Queue() {

delete[] data;

}

// Funcția de adăugare a unui element în coadă

bool add(int element) {

if (isFull()) {

cout << "Coada este Full\n";

return false;

}

rear = (rear + 1) % maxSize;

data[rear] = element;

count++;

return true;

}

// Funcția de scoatere a unui element din coadă

int get() {

if (isEmpty()) {

cout << "Coada este goală\n";

return -1; // Valoare de eroare

}

int element = data[front];

front = (front + 1) % maxSize;

count--;

return element;

}

// Funcția care verifică dacă coada este goală

bool isEmpty() const {

return count == 0;

}

// Funcția care verifică dacă coada este plină

bool isFull() const {

return count == maxSize;

}

// Funcție pentru afișarea elementelor din coadă (opțional, pentru testare)

void display() const {

if (isEmpty()) {

cout << "Coada este goală\n";

return;

}

std::cout << "Elementul Cozii: ";

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

cout << data[(front + i) % maxSize] << " ";

}

cout << "\n";

}

};

int main() {

int n;

cout << "Introduceti MaxSize a cozii: ";

cin >> n;

Queue q(n);

int a, b, c;

while (true) {

int sel;

cout << "Selectati:\n1. Introduceti elemente\n2. Run\n3. Display\n";

cin >> sel;

if (sel == 1) {

// Testarea funcționalităților cozii

cout << "Introduceti elemente: ";

cin >> a >> b >> c;

}

else if (sel == 2) {

// Adăugarea elementelor în coadă

q.add(a);

q.add(b);

q.add(c);

}

else if (sel == 3) {

// Afișarea cozii și alte informații

q.display();

cout << "Get element: " << q.get() << "\n";

q.display();

cout << "Coada este goală? " << (q.isEmpty() ? "Da" : "Nu") << "\n";

cout << "Coada este Full? " << (q.isFull() ? "Da" : "Nu") << "\n";

}

else {

cout << "Selectie invalida. Incercati din nou.\n";

}

}

/\* Queue q(n); // Creează o coadă cu maxim 5 elemente

// Testarea funcționalităților cozii

cout<<"introduceti elemente";

cin>>a;

cin>>b;

cin>>c;

q.add(a);

q.add(b);

q.add(c);

q.display();

cout << "Get element: " << q.get() << "\n";

q.display();

cout << "Coada este goală? " << (q.isEmpty() ? "Da" : "Nu") << "\n";

cout << "Coada este Full? " << (q.isFull() ? "Da" : "Nu") << "\n";\*/

return 0;

}

**b)**

#include <iostream>

using namespace std;

class Matrice {

private:

int\* data;

int rows;

int cols;

int codEroare; // 0 = fără eroare, 1 = eroare de memorie, 2 = eroare de dimensiuni, 3 = index în afara limitelor

public:

// constructorul implicit

Matrice() : rows(0), cols(0), data(nullptr), codEroare(0) {}

// constructor pentru matrice pătratică

Matrice(int size) : rows(size), cols(size), codEroare(0) {

data = new int[rows \* cols]();

if (data == nullptr) {

codEroare = 1;

}

}

// constructor pentru matrice dreptunghiulară

Matrice(int r, int c) : rows(r), cols(c), codEroare(0) {

data = new int[rows \* cols]();

if (data == nullptr) {

codEroare = 1;

}

}

// Destructor

~Matrice() {

delete[] data;

}

int getCodEroare() const { return codEroare; }

int obtineElement(int i, int j) const {

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

return 0;

}

return data[i \* cols + j];

}

void seteazaElement(int i, int j, int value) {

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

codEroare = 3;

return;

}

data[i \* cols + j] = value;

}

Matrice aduna(const Matrice& other) const {

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

return Matrice();

}

Matrice rezultat(rows, cols);

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

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

rezultat.seteazaElement(i, j, obtineElement(i, j) + other.obtineElement(i, j));

}

}

return rezultat;

}

Matrice scade(const Matrice& other) const {

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

return Matrice();

}

Matrice rezultat(rows, cols);

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

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

rezultat.seteazaElement(i, j, obtineElement(i, j) - other.obtineElement(i, j));

}

}

return rezultat;

}

Matrice inmulteste(const Matrice& other) const {

if (cols != other.rows) {

return Matrice();

}

Matrice rezultat(rows, other.cols);

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

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

int suma = 0;

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

suma += obtineElement(i, k) \* other.obtineElement(k, j);

}

rezultat.seteazaElement(i, j, suma);

}

}

return rezultat;

}

Matrice inmulteste(int scalar) const {

Matrice rezultat(rows, cols);

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

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

rezultat.seteazaElement(i, j, obtineElement(i, j) \* scalar);

}

}

return rezultat;

}

void inputMatrice() {

cout << "Introduceți elementele matricei:\n";

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

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

int value;

cin >> value;

seteazaElement(i, j, value);

}

}

}

void afiseazaMatrice() const {

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

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

cout << obtineElement(i, j) << " ";

}

cout << "\n";

}

}

};

void meniu() {

int alegere;

Matrice A, B, rezultat;

int size, rows, cols, scalar;

while (true) {

cout << "\nMeniu:\n";

cout << "1. Crearea matricei pătratice\n";

cout << "2. Crearea matricei dreptunghiulare\n";

cout << "3. Adunare matrice\n";

cout << "4. Scădere matrice\n";

cout << "5. Înmulțire matrice\n";

cout << "6. Înmulțire matrice cu scalar\n";

cout << "7. Afișare matrice\n";

cout << "8. Ieșire\n";

cout << "Introduceți alegerea dumneavoastră: ";

cin >> alegere;

switch (alegere) {

case 1:

cout << "Introduceți dimensiunea pentru matricea pătratică: ";

cin >> size;

A = Matrice(size);

A.inputMatrice();

break;

case 2:

cout << "Introduceți numărul de rânduri și coloane pentru matricea dreptunghiulară: ";

cin >> rows >> cols;

A = Matrice(rows, cols);

A.inputMatrice();

break;

case 3: // Adunare matrice

cout << "Introduceți elementele pentru a doua matrice:\n";

B = Matrice(A);

B.inputMatrice();

rezultat = A.aduna(B);

if (rezultat.getCodEroare() == 2) {

cout << "Eroare: Neconcordanță de dimensiuni.\n";

} else {

rezultat.afiseazaMatrice();

}

break;

case 4: // Scădere matrice

cout << "Introduceți elementele pentru a doua matrice:\n";

B = Matrice(A);

B.inputMatrice();

rezultat = A.scade(B);

if (rezultat.getCodEroare() == 2) {

cout << "Eroare: Neconcordanță de dimensiuni.\n";

} else {

rezultat.afiseazaMatrice();

}

break;

case 5: // Înmulțire matrice

cout << "Introduceți elementele pentru a doua matrice:\n";

cin >> rows >> cols;

B = Matrice(rows, cols);

B.inputMatrice();

rezultat = A.inmulteste(B);

if (rezultat.getCodEroare() == 2) {

cout << "Eroare: Neconcordanță de dimensiuni.\n";

} else {

rezultat.afiseazaMatrice();

}

break;

case 6: // Înmulțire matrice cu scalar

cout << "Introduceți valoarea scalară: ";

cin >> scalar;

rezultat = A.inmulteste(scalar);

rezultat.afiseazaMatrice();

break;

case 7:

A.afiseazaMatrice();

break;

case 8:

return;

default:

cout << "Alegere invalidă, încercați din nou.\n";

}

}

}

int main() {

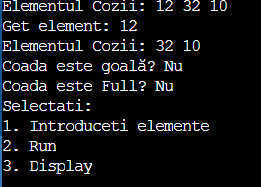
meniu();

return 0;

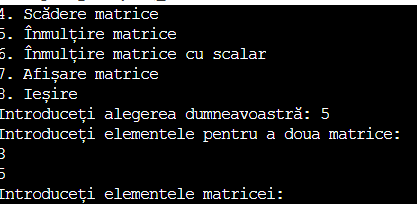
}

# **Testarea și verificarea programelor cu rezultatele obținute:**

**a)**

****

**b)**

****

# **Concluzii:**

În urma acestei lucrări, am învățat să definim și să utilizăm constructori și destructori în cadrul unei clase, acestea fiind componente fundamentale ale programării orientate pe obiecte. Constructorii sunt folosiți pentru a inițializa obiectele la momentul creării lor, asigurând alocarea corectă a resurselor necesare funcționării obiectului. De asemenea, am învățat despre diferitele tipuri de constructori, precum constructorul implicit, constructorul cu parametri și constructorul de copiere, fiecare având un rol specific în controlul inițializării.

Destructorul este esențial pentru eliberarea resurselor (de exemplu, memoria alocată dinamic), prevenind scurgerile de memorie și alte erori legate de gestionarea resurselor.

Prin definirea și utilizarea corectă a constructorilor și destructorilor, am dobândit o înțelegere mai profundă a principiilor de gestionare a ciclului de viață al obiectelor și a resurselor în C++, ceea ce este esențial pentru dezvoltarea de aplicații eficiente și robuste.

# **Bibliografie:**

1. Lucrare de laborator 2:<https://ocw.cs.pub.ro/courses/sd-ca/2019/laboratoare/lab-02> Acesat pe 29.10.2024
2. POO:<https://www.youtube.com/watch?v=rZcTaRU7AAw&list=PLQOaTSbfxUtBm7DxblJZShqBQnBAVzlXX> Acesat pe 29.10.2024