Universitatea Tehnică a Moldovei

Facultatea *Calculatoare, Informatică și Microelectronică*

Specialitatea *Tehnologii Informaționale*

Raport

la lucrarea de laborator nr. 1

Tema:*"Clase si obiecte. Constructorul si destructorul clasei"*

Disciplina: "Programarea orientata pe obiecte"

Varianta 3

A efectuat:	Student grupa TI-231 FR	Apareci Aurica
A verificat:	Asistent universitar	Mititelu Vitalie

Chișinău 2025

Cuprins

1. **Cadru teoretic** 3
2. **Repere teoretice** 3
3. **Listingul programului** 4
4. **Concluzii** 7
5. **Webobrafie** 8

1. Cadru teoretic

Tema lucrării: Clase si obiecte. Constructorii si destructorul clasei

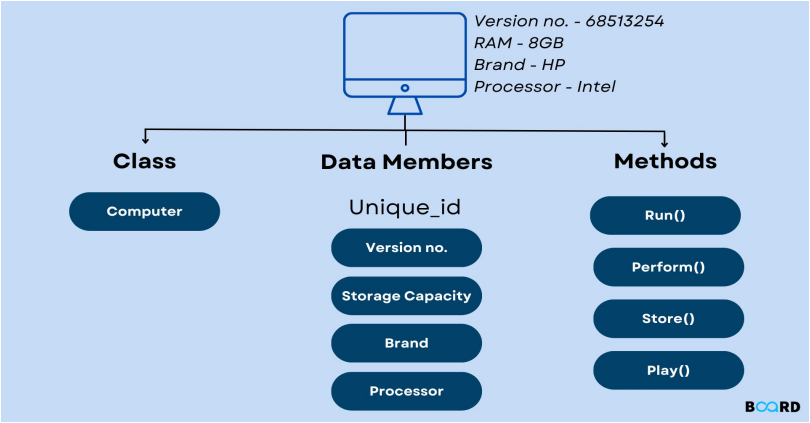
Scopul lucrării:

- Studierea principiilor de definire şi utilizare a constructorilor
- Studierea principiilor de definire şi utilizare a destructorilor
- Studierea tipurilor de constructori

Varianta 3: Să se creeze *clasa Queue* – coadă de tip float. 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.

2. Repere teoretice

Programarea orientată pe obiecte (OOP) constituie un model fundamental în dezvoltarea sistemelor software moderne, având la bază conceptele de **clasă** şi **obiect**. O **clasă** reprezintă o abstracţie care descrie un tip de date compus, incluzând atât atribute, cât şi comportamente (metode). Pe baza unei clase pot fi create **obiecte**, care sunt instanţieri concrete ale acestei descrieri.



Un aspect esenţial în gestionarea ciclului de viaţă al obiectelor îl reprezintă **constructorii** şi **destructorii**. **Constructorul** este o funcţie specială a clasei, invocată automat în momentul creării unui obiect. Rolul său principal este **iniţializarea membrilor clasei**. Acesta poartă acelaşi nume ca şi clasa şi nu are un tip de întoarcere. Există patru tipuri de constructori: implicit, de copiere, de conversie a tipului şi general. Această clasificare este bazată pe regulile de definire şi utilizare a constructorilor.

Constructorul implicit – constructor fără parametri, sau constructor cu toţi parametrii impliciţi.

```
Student() {
        nume = "Necunoscut";
        varsta = 0;
        std::cout << "Constructor implicit apelat.\n";
    }
```

Constructorul de copiere – constructor care are ca parametru referință la obiectul clasei respective. Constructorul de copiere poate avea şi alți parametri care însă trebuie să fie impliciți.

```
Student(const Student& s) {
        nume = s.nume;
        varsta = s.varsta;
        std::cout << "Constructor de copiere apelat.\n";
    }
```

Constructorul de conversie a tipului - constructorul, care transformă un tip de obiect în altul.

```
Student(int v) {
       nume = "Anonim";
       varsta = v;
       std::cout << "Constructor de conversie apelat.\n";
    }
```

Constructorul general – constructorul care nu corespunde categoriilor enumerate mai sus.

```
Student(std::string n, int v) {
      nume = n;
      varsta = v;
      std::cout << "Constructor general apelat.\n";
   }
```

Destructorul este o altă funcție specială a clasei, apelată automat în momentul **distrugerii obiectului**. Scopul său este **eliberarea resurselor alocate dinamic**, precum memorie, fișiere sau conexiuni. În C++, destructorul este precedat de caracterul ~ și nu are parametri sau tip de întoarcere. Spre deosebire de constructori, o clasă poate avea un singur destructor, care nu poate fi suprasolicitat. Gestionarea corectă a constructorilor și destructorului contribuie la asigurarea unei funcționări coerente și eficiente a aplicațiilor, prevenind scurgerile de memorie și comportamentele nedeterminate.

```
~Student() {
     std::cout << "Destructor: " << nume << "\n";
 }
```

3. Listingul programului

main.cpp

```
#include "Queue.h"

int main()
{
	Queue q(5);
	for (size_t i = 0; i < 5; i++)
	{
		q.Enqueue(i);
	}

	Queue q2(5);
	for (size_t i = 0; i < 5; i++)
	{
		cout << "q[" << i << "] = ";
		cin >> q2;
	}
```

```
	cout << "q1: " << q << endl;
	cout << "q2: " << q2 << endl;
	if (q == q2)
	{
		cout << "q1 == q2" << endl;
	}
	else
	{
		cout << "q1 != q2" << endl;
	}
	Queue q3 = q + q2;
	cout << "q3: " << q3 << endl;
}
```

Queue.h

```
#pragma once
#include <iostream>
#include <limits.h>

using namespace std;

class Queue {
private:
    int front, rear, size;
    unsigned capacity;
    int* array;
public:
    Queue(unsigned capacity)
    {
        this->capacity = capacity;
        this->front = this->size = 0;
        this->rear = capacity - 1;
        this->array = new int[capacity];
    }
    bool IsFull()
    {
        return (size == capacity);
    }
    bool IsEmpty()
    {
        return (size == 0);
    }
    void Enqueue(int item)
    {
        if (IsFull())
        {
            cout << "Queue is full\n";
            return;
        }
        this->rear = (this->rear + 1) % this->capacity;
        this->array[this->rear] = item;
        this->size = this->size + 1;
    }
    int Dequeue()
    {
        if (IsEmpty())
        {
            cout << "Queue is empty\n";
            return INT_MIN;
        }
        int item = this->array[this->front];
        this->front = (this->front + 1) % this->capacity;
        this->size = this->size - 1;
        return item;
```

```
    void Display()
    {
        if (IsEmpty())
        {
            cout << "Queue is empty\n";
            return;
        }
        int i = this->front;
        for (int j = 0; j < this->size; j++)
        {
            cout << this->array[i] << " ";
            i = (i + 1) % this->capacity;
        }
        cout << endl;
    }

    int DequeueElement(int position)
    {
        if (IsEmpty())
        {
            cout << "Queue is empty\n";
            return INT_MIN;
        }
        if (position > this->size)
        {
            cout << "Position is out of range\n";
            return INT_MIN;
        }
        int item = this->array[position];
        for (int j = position; j < this->rear; j++)
        {
            this->array[j] = this->array[j + 1];
        }
        this->size = this->size - 1;
        this->front = (this->front) % this->capacity;
        this->rear = (this->rear - 1) % this->capacity;
        return item;
    }
};
```

Rezultatele testării

```
q[0] = 1
q[1] = 2
q[2] = 3
q[3] = 4
q[4] = 5
q1: 0 1 2 3 4
q2: 1 2 3 4 5
q1 != q2
q3: 0 1 2 3 4 1 2 3 4 5
```

```
q[0] = 0
q[1] = 1
q[2] = 2
q[3] = 3
q[4] = 4
q1: 0 1 2 3 4
q2: 0 1 2 3 4
q1 == q2
q3: 0 1 2 3 4 0 1 2 3 4
```

4. Concluzii

În cadrul acestei lucrări de laborator, am realizat implementarea unei structuri de tip coadă (Queue) utilizând principiile programării orientate pe obiect (OOP) în limbajul C++. Deși am mai avut ocazia să aplic conceptele de OOP anterior, în special în cadrul proiectelor dezvoltate în limbajul C#, această sarcină s-a dovedit a fi una mai complexă din perspectiva adaptării la sintaxa și mecanismele limbajului C++.

Spre deosebire de C#, unde managementul memoriei este automatizat prin garbage collector și multe structuri de date sunt deja integrate în biblioteci standard, în C++ a fost necesar să gestionez explicit alocarea și dealocarea memoriei, precum și să construiesc manual funcționalitățile legate de constructori, copiere și eliberare a resurselor. Acest exercițiu m-a ajutat să înțeleg mai profund modul în care funcționează intern o coadă circulară și importanța fiecărui element al unei clase, de la constructori până la metodele auxiliare.

Consider această experiență una utilă și valoroasă în consolidarea cunoștințelor de bază și avansate privind programarea orientată pe obiect în C++, mai ales în contextul unei structuri fundamentale de date precum coada.

5. Webobrafie

- https://else.fcim.utm.md/course/view.php?id=663
- https://www.w3schools.com/cpp/cpp_oop.asp
- https://www.programiz.com/cpp-programming/oop,
- https://code.visualstudio.com/docs/cpp/config-linux
- https://chatgpt.com/