

**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.6**

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

Verificat:

Kulev Mihail dr., conf. univ.

Departamentul Informatică şi IS,

Facultatea FCIM, UTM

**Chișinău 2024**

**Tema:** Polimorfism. Funcţii virtuale. Clase abstracte

**Scopul lucrării:**

• Studierea polimorfismului;

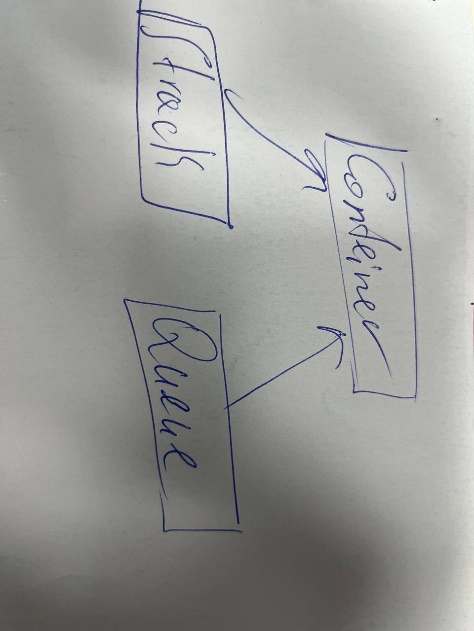
• Studierea principiilor legăturii întîrziate;

• Studierea funcţiilor virtuale;

• Polimorfismul ad-hoc;

• Realizarea funcţiilor virtuale;

• Studierea claselor abstracte.



**Figura 1:**Schema pentru [a]

|  |  |
| --- | --- |
| **Codul:** | **Output:** |
| **#include <iostream>**  **#include <queue>**  **#include <stack>**  **// Clasa abstractă de bază**  **//functie virtuala pura este ca este egal cu 0 i nu are cod sieste de compilator generat**  **class Container {**  **public:**  **virtual void inserare(int element) = 0;**  **virtual int extragere() = 0;**  **};**  **// Clasa derivată Stack**  **class Stack : public Container {**  **private:**  **std::stack<int> stiva;**  **public:**  **void inserare(int element) override {**  **stiva.push(element);**  **}**  **int extragere() override {**  **if (stiva.empty()) {**  **std::cerr << "Stiva este goală!\n";**  **return -1; // Valoare specială pentru a semnala eroarea**  **}**  **int element = stiva.top();**  **stiva.pop();**  **return element;**  **}**  **};**  **// Clasa derivată Queue**  **class Queue : public Container {**  **private:**  **std::queue<int> coada;**  **public:**  **void inserare(int element) override {**  **coada.push(element);**  **}**  **int extragere() override {**  **if (coada.empty()) {**  **std::cerr << "Coada este goală!\n";**  **return -1; // Valoare specială pentru a semnala eroarea**  **}**  **int element = coada.front();**  **coada.pop();**  **return element;**  **}**  **};**  **int main() {**  **// Masiv de pointeri la clasa abstractă**  **Container\* containere[2];**  **// Obiecte ale claselor derivate**  **Stack stiva;**  **Queue coada;**  **// Atribuire adrese în masiv**  **containere[0] = &stiva;**  **containere[1] = &coada;**  **// Inserare elemente în stivă și coadă**  **//stiva este ca ultimul introdus primul extras**  **//coada primul indorus primul extras**  **containere[0]->inserare(10);**  **containere[0]->inserare(20);**  **containere[0]->inserare(30);**  **containere[1]->inserare(21);**  **containere[1]->inserare(54);**  **containere[1]->inserare(65);**  **// Extragere elemente din stivă și coadă**  **std::cout << "Extragere din stivă: ";**  **std::cout << containere[0]->extragere() << " ";**  **std::cout << containere[0]->extragere() << " ";**  **std::cout << containere[0]->extragere() << "\n";**  **std::cout << "Extragere din coadă: ";**  **std::cout << containere[1]->extragere() << " ";**  **std::cout << containere[1]->extragere() << " ";**  **std::cout << containere[1]->extragere() << "\n";**  **return 0;**  **}** |  |

# **Concluzii:**

În concluzie, codul pe care l-ai trimis respectă pe deplin principiile polimorfismului, funcțiilor virtuale și ale claselor abstracte, așa cum sunt definite în C++. Prin utilizarea funcțiilor virtuale, se obține polimorfismul, ceea ce permite apelarea diferitelor implementări ale aceleași funcții (cum ar fi Say()) în funcție de tipul dinamic al obiectului, chiar dacă acesta este tratat printr-un pointer sau o referință la clasa de bază. De asemenea, aplicarea principiului de substituție permite utilizarea obiectelor de tipuri derivate în locul celor de tipul clasei de bază fără a compromite corectitudinea programului.

Codul demonstrează cum se poate implementa un comportament specific pentru fiecare clasă derivată, respectând astfel conceptul de polimorfism. Folosirea unui masiv de pointeri la clasa abstractă Container pentru a manipula obiecte de tip Stack și Queue este un alt exemplu de polimorfism și respectă cerințele temei. În general, acest exemplu este bine structurat și ilustrează corect utilizarea funcțiilor virtuale și a conceptului de clase abstracte în C++.

# **Bibliografie:**

1. Lucrare de laborator 6: <https://else.fcim.utm.md/pluginfile.php/37341/mod_resource/content/1/lab6Rom.pdf> Accesat pe 12.11.2024
2. POO:<https://www.youtube.com/watch?v=rZcTaRU7AAw&list=PLQOaTSbfxUtBm7DxblJZShqBQnBAVzlXX> Acesat pe 12.11.2024
3. Intelegență artificială: <https://chatgpt.com/c/6749b566-c1e8-8001-b9e4-027fa5295f07Accesat> pe 12.11.2024