Universitatea Tehnică a Moldovei

Facultatea *Calculatoare, Informatică și Microelectronică*

Specialitatea *Tehnologii Informaționale*

Raport

la lucrarea de laborator nr. 4

Tema:*"Polimorfism. Funcții virtuale. Clase abstracte"*

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** 4
3. **Listingul programului** 4
4. **Concluzii** 5
5. **Webobrafie** 6

1. Cadru teoretic

Tema lucrării: Polimorfism. Funcții virtuale. Clase abstracte.

Scopul lucrării:

- Studierea polimorfismului;
- Studierea principiilor legăturii întîrziate;
- Studierea funcțiilor virtuale;
- Polimorfismul ad-hoc;
- Realizarea funcțiilor virtuale;
- Studierea claselor abstracte.

Varianta 3: Creați clasa abstractă de bază progresia cu funcția virtuală – suma progresiei. Creați clasele derivate: progresia aritmetică şi progresia geometrică. Fiecare clasă conține două câmpuri de tip double. Primul – primul membrul al progresiei, al doilea (double) – restul permament (pentru aritmetică) şi relațiile permanente (pentru geometrică). Definiți funcția de calculare a sumei, unde ca parametru este cantitatea de elemente a progresiei.

Progresia aritmetică aj=a0+jd, j=0,1,2,…

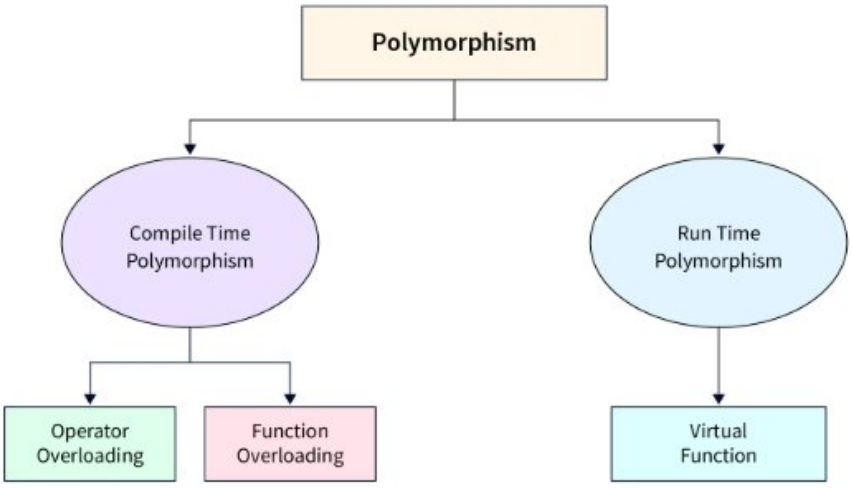
Suma progresiei aritmetice: sn=(n+1)(a0+an)/2

Progresia geometrică: aj=a0r j , j=0,1,2,…

Suma progresiei geometrice: sn=(a0-anr)/(1-r)

2. Repere teoretice

Programarea orientată pe obiecte (OOP) furnizează un set de mecanisme conceptuale ce facilitează dezvoltarea de aplicații flexibile, scalabile şi uşor de întreținut. Printre aceste mecanisme, **polimorfismul** ocupă un loc central, fiind asociat cu capacitatea entităților de a se comporta diferit în funcție de contextul de utilizare. Etimologic derivat din grecescul *"poly"* (multe) şi *"morph"* (forme), polimorfismul exprimă posibilitatea de a trata obiecte de tipuri diferite prin intermediul unei interfețe comune.



Polimorfismul permite utilizarea aceloraşi funcţii sau operatori pentru obiecte din clase diferite, dar care derivă dintr-o clasă de bază comună. Acest comportament este realizat prin legătura întârziată (*late binding*), un mecanism prin care apelul funcţiei este rezolvat în momentul execuţiei şi nu la compilare. Acest principiu permite extinderea flexibilă a funcţionalităţii aplicaţiei fără a modifica codul existent, facilitând respectarea principiului *Open/Closed*.

Funcţiile virtuale sunt o componentă esenţială pentru realizarea polimorfismului la nivel de execuţie. În C++, o funcţie declarată cu cuvântul cheie virtual în clasa de bază permite redefinirea comportamentului său în clasele derivate. Alegerea implementării potrivite a funcţiei se face în funcţie de tipul obiectului instanţiat, nu de tipul pointerului sau referinţei prin care este accesat.

```
class Animal {
public:
    virtual void Vorbeste() {
        std::cout << "Animalul face un sunet generic.\n";
    }
};

class Caine : public Animal {
public:
    void Vorbeste() override {
        std::cout << "Cainele latră.\n";
    }
};
```

O clasă abstractă este o clasă ce conţine cel puţin o funcţie virtuală pură – adică o funcţie declarată cu = 0. Astfel de clase nu pot fi instanţiate direct, fiind destinate să servească drept bază pentru alte clase. Ele definesc o interfaţă comună pentru un set de clase derivate şi asigură implementarea uniformă a comportamentului specific.

```
class Forma {
public:
    virtual void Deseneaza() = 0; // funcţie virtuală pură
};
```

Polimorfismul ad-hoc este o formă de polimorfism care nu implică moştenire sau legătură întârziată. El este realizat prin supraincărcare de funcţii (funcţii cu acelaşi nume, dar parametri diferiţi) sau prin supradefinirea operatorilor. Spre deosebire de polimorfismul de execuţie, acesta este determinat în timpul compilării (early binding).

3. Listingul programului

main.cpp

```
#include "ArithmeticProgression.h"
#include "GeometricProgression.h"

int main()
{
	ArithmeticProgression ap(1, 2);
	GeometricProgression gp(1, 2);
	cout << "Termenul 5 din progresia aritmetica este: " << ap.GetNthTerm(5)
<< endl;
	cout << "Termenul 5 din progresia geometrica este: " << gp.GetNthTerm(5)
<< endl;
	cout << "Suma primelor 10 elemente ale progresiei aritmetice: " <<
ap.Sum(10) << endl;
	cout << "Suma primelor 10 elemente ale progresiei geometrice: " <<
gp.Sum(10) << endl;
}
```

ArithmeticProgression.h

```
#pragma once
#include "Progression.h"
class ArithmeticProgression : public Progression
{
	public:
		ArithmeticProgression(double ft, double st) : Progression(ft, st)
		{

		}
		double GetNthTerm(int n) override
		{
			return GetFirstTerm() + (n * GetSecondTerm());
		}
		double Sum(int n) override
		{
			return (n + 1) * (GetFirstTerm() + GetNthTerm(n)) / 2;
		}
};
```

GeometricProgression.h

```
#pragma once
#include "Progression.h"

class GeometricProgression : public Progression
{
	public:
		GeometricProgression(double ft, double st) : Progression(ft, st)
		{
		}
		double GetNthTerm(int n) override
		{
			return GetFirstTerm() * pow(GetSecondTerm(), n);
		}
		double Sum(int n) override
		{
			return (GetFirstTerm() - GetNthTerm(n)*GetSecondTerm()) / (1
- GetSecondTerm());
		}
};
```

Progression.h

```
#pragma once
#include <iostream>

using namespace std;

class Progression
{
	protected:
		double FirstTerm;
		double SecondTerm;
	public:
		Progression(double ft, double st)
		{
			FirstTerm = ft;
			SecondTerm = st;
		}
		double GetFirstTerm()
		{
			return FirstTerm;
		}
		void SetFirstTerm(double ft)
		{
			FirstTerm = ft;
		}
		double GetSecondTerm()
		{
			return SecondTerm;
		}
		void SetSecondTerm(double st)
		{
			SecondTerm = st;
		}
		virtual double GetNthTerm(int n)
		{
			return 0;
		}
		virtual double Sum(int n)
		{
			return 0;
		}
};
```

Rezultatele testării

```
Termenul 5 din progresia aritmetica este: 11
Termenul 5 din progresia geometrica este: 32
Suma primelor 10 elemente ale progresiei aritmetice: 121
Suma primelor 10 elemente ale progresiei geometrice: 2047
```

4. Concluzii

Lucrarea de laborator a permis aprofundarea unor concepte fundamentale din programarea orientată pe obiecte, cu accent pe polimorfism și mecanismele aferente realizării sale în limbajul C++. Prin intermediul exemplelor implementate, s-a evidențiat modul în care polimorfismul oferă flexibilitate în definirea și utilizarea funcțiilor comune pentru obiecte de tipuri diferite, prin intermediul unei interfețe comune.

Au fost studiate și aplicate în practică funcțiile virtuale, care stau la baza legăturii întârziate (dynamic dispatch), permițând selectarea comportamentului corect al unei metode în funcție de tipul real al obiectului. De asemenea, s-au identificat diferențele dintre polimorfismul ad-hoc, realizat prin suprasarcină și supradefinire, și polimorfismul de execuție, bazat pe moștenire și funcții virtuale.

Prin realizarea clasei abstracte Progresie și a claselor derivate ProgresieAritmetica și ProgresieGeometrica, s-a ilustrat aplicarea principiului de abstracție și utilizarea funcțiilor virtuale pentru implementarea comportamentului specific în clasele derivate. S-a demonstrat modul în care o interfață comună permite apeluri uniforme pentru calculul sumei progresiilor, în funcție de tipul concret de progresie. Această lucrare consolidează înțelegerea conceptelor esențiale ale OOP și pregătește terenul pentru dezvoltarea de aplicații extensibile și bine structurate, în care reutilizarea codului și separarea responsabilităților devin trăsături esențiale ale arhitecturii software.

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/