# **Звіт про виконання лабораторної роботи № 2.1**

## «Конструктори та перевантаження операцій для класів з двома полями» з дисципліни «Об’єктно-орієнтоване програмування»

## Студента групи «**ІТ-12**» - **Степанчука Сергія**

## Мета роботи

Освоїти використання конструкторів та перевантаження операцій.

## Умова завдання

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому

вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

 конструктор ініціалізації (один або декілька),

 конструктор без аргументів і

 конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих

операцій.

У всіх завданнях обов’язково мають бути реалізовані відповідні операції:

 присвоєння,

 введення з клавіатури,

 виводу на екран, 79

 приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

 інкременту в обох формах (префіксній та постфіксній) і

 декременту в обох формах (префіксній та постфіксній).

При цьому префіксні операції інкременту, декременту модифікують поле first, а

постфіксні – поле second.

Перевантаження операцій виконується таким чином: підходящі операції реалізуються

як методи класу, а інші – як зовнішні дружні функції.

Для демонстрації роботи з об’єктами нового типу у всіх завданнях потрібно написати

головну функцію. У програмі обов’язково мають бути продемонстровані різні способи

створення об’єктів і масивів об’єктів. Програма має демонструвати використання всіх

функцій і методів. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без

нього.

Визначення класу та реалізацію його методів слід розмістити в окремих модулях.

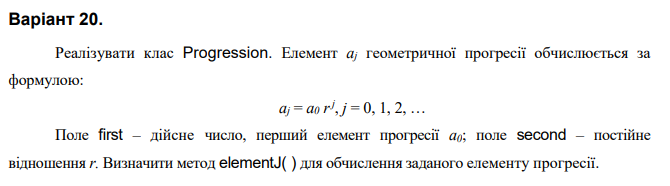
Завдання наступне:

Виконати завдання свого варіанту Лабораторної роботи № 1.1 (Класи з двома

полями), реалізувавши для кожного класу вказані конструктори та операції. Функції

введення / виведення оформити як дружні.

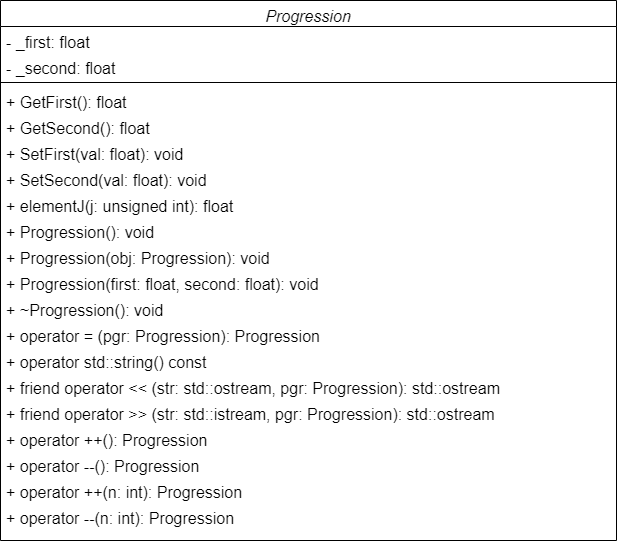
Метод Init() стане конструкторами, методи Read() та Display() – операціями

вводу / виводу.

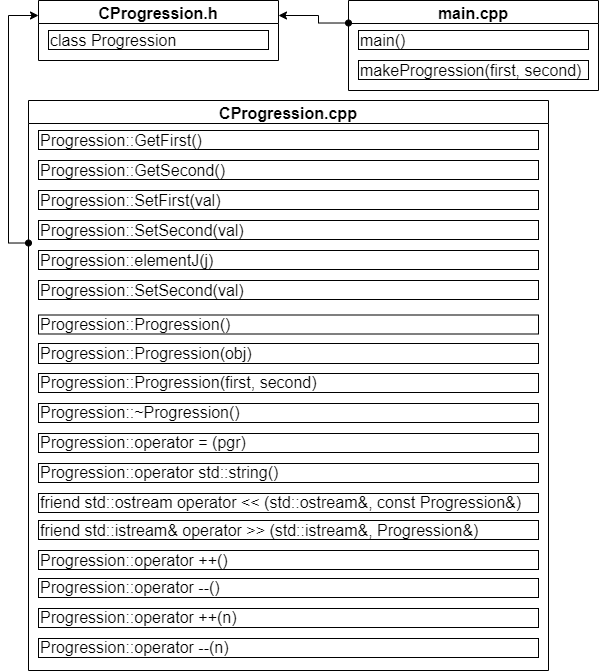
## Посилання на git-репозиторій з проектом:

<https://github.com/SergiyStepanchuk/OOP_Lab_2.1>

## UML-діаграма класів



## Структурна схема



## Текст програми

### // CProgression.h

#pragma once

#include <iostream>

#include <cmath>

#include <string>

class Progression

{

float \_first = 0;

float \_second = 0;

public:

Progression() {};

Progression(const Progression& obj) :

\_first(obj.\_first), \_second(obj.\_second) {}

Progression(const float &first, const float &second);

~Progression() {};

Progression& operator = (const Progression&);

operator std::string() const;

friend std::ostream& operator << (std::ostream&, const Progression&);

friend std::istream& operator >> (std::istream&, Progression&);

// prefix

Progression& operator ++();

Progression& operator --();

// postfix

Progression operator ++(int);

Progression operator --(int);

float GetFirst() const;

float GetSecond() const;

void SetFirst(const float& val);

void SetSecond(const float& val);

float elementJ(const unsigned int& j) const;

};

### // CProgression.cpp

#include "CProgression.h"

#include <sstream>

using namespace std;

Progression::Progression(const float &first, const float &second)

{

\_first = first;

\_second = second;

}

Progression& Progression::operator = (const Progression& pr)

{

\_first = pr.\_first;

\_second = pr.\_second;

return \*this;

}

Progression::operator std::string() const

{

stringstream ss;

ss << \_first << ", " << \_second;

return ss.str();

}

ostream & operator << (ostream & out, const Progression& r)

{

out << string(r);

return out;

}

istream& operator >> (istream& in, Progression& r)

{

cout << "first = "; in >> r.\_first;

cout << "second = "; in >> r.\_second;

return in;

return in;

}

Progression& Progression::operator ++()

{

\_first++;

return \*this;

}

Progression& Progression::operator --()

{

\_first--;

return \*this;

}

Progression Progression::operator ++(int)

{

Progression t(\*this);

\_second++;

return t;

}

Progression Progression::operator --(int)

{

Progression t(\*this);

\_second--;

return t;

}

float Progression::GetFirst() const

{

return \_first;

}

float Progression::GetSecond() const

{

return \_second;

}

void Progression::SetFirst(const float& val)

{

\_first = val;

}

void Progression::SetSecond(const float& val)

{

\_second = val;

}

float Progression::elementJ(const unsigned int& j) const

{

return \_first \* powf(\_second, j);

}

### // main.cpp

#include <iostream>

#include "CProgression.h"

using namespace std;

Progression makeProgression(const float &first, const float &second)

{

return Progression(first, second);

}

int main()

{

Progression pgr(13, 3.4);

cout << pgr << endl;

cout << "elementJ(5) = " << pgr.elementJ(5) << endl << endl;

cout << "++pgr: " << ++pgr << endl;

cout << "--pgr: " << --pgr << endl;

cout << "pgr++: " << pgr++ << endl;

cout << "pgr--: " << pgr-- << endl

<< endl;

pgr = Progression(4, 1.1);

cout << string(pgr) << endl

<< endl;

pgr = Progression(4, 1.1);

cout << "first = " << pgr.GetFirst() << endl;

cout << "second = " << pgr.GetSecond() << endl

<< endl;

cout << pgr << endl;

pgr = makeProgression(1, 2);

cout << "first = " << pgr.GetFirst() << endl;

cout << "second = " << pgr.GetSecond() << endl

<< endl;

cin >> pgr;

cout << pgr

<< endl;

return 0;

}

## UnitTest

### Код

#include "pch.h"

#include "CppUnitTest.h"

#include "../Program/CProgression.h"

#include "../Program/CProgression.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest

{

TEST\_CLASS(TestProgression)

{

public:

TEST\_METHOD(Init)

{

Progression pgr(0, 2);

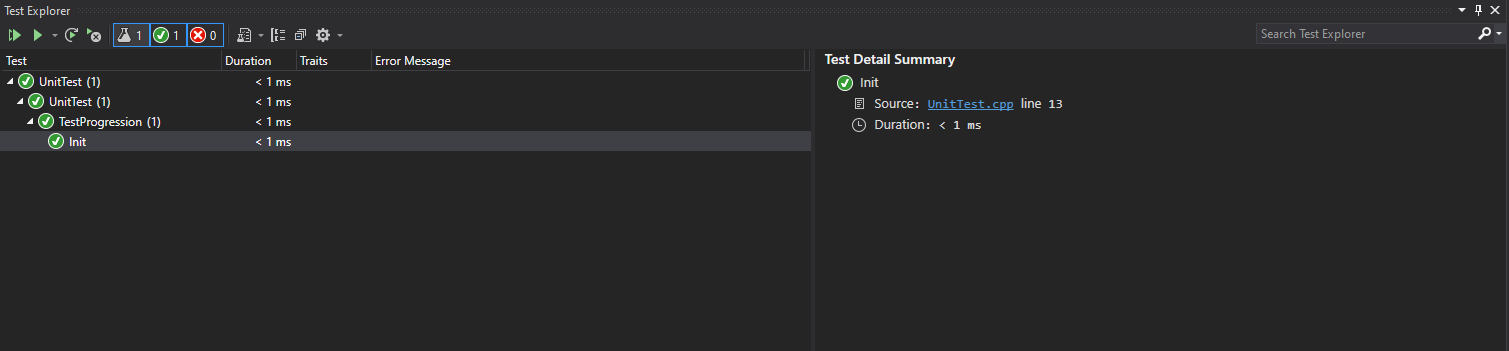
Assert::AreEqual(pgr.elementJ(1), 0.f);

}

};

}

### Результат



## Висновок

Освоїв використання конструкторів та перевантаження операцій.