ЗВІТ

про виконання лабораторної роботи № 2.3

« Конструктори та перевантаження операцій для класів »

з дисципліни

«Об’єктно-орієнтоване програмування»

студента групи ІТ-12

М’ягких Владислава Юрійовича

**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,

• 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), – зміст цих операцій визначити самостійно.

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 № 1.3. «Об’єкти – параметри методів (дії над кількома об’єктами)» як незалежні класи з конструкторами і перевантаженням операцій.

Метод Init() стане конструкторами, методи Read() та Display() – операціями вводу / виводу.

**Лабораторна робота № 1.3:**

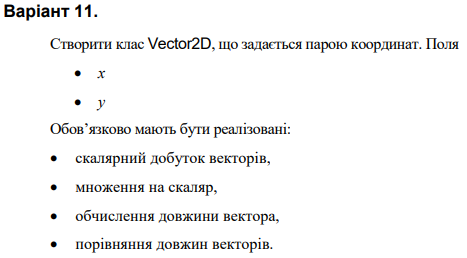
У всіх завданнях, крім вказаних в завданні операцій, обов’язково мають бути реалізовані наступні методи:

* методи доступу (константні методи зчитування та методи запису) значення кожного поля;
* метод ініціалізації Init( );
* метод введення з клавіатури Read( );
* метод виведення на екран Displау( );
* метод перетворення до літерного рядку toString( ).

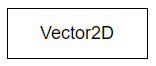
Всі завдання мають бути реалізовані як клас із закритими полями, де операції реалізуються як методи класу.

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

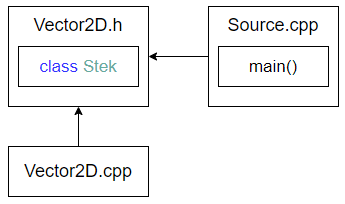
Для демонстрації роботи з об’єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов’язково мають бути продемонстровані різні способи створення об’єктів і масивів об’єктів – різними конструкторами. Програма має демонструвати використання всіх функцій і методів.



**UML-діаграма класів:**



**Структурна схема програми:**



**Текст програми:**

///////////////

// Source.cpp

///////////////

#include <iostream>

#include "Vector2D.h"

using namespace std;

int main()

{

Vector2D A(1, 1);

cout << "A " << endl; cin >> A;

cout << "A = " << A << endl;

Vector2D B;

cout << "B " << endl; cin >> B;

cout << "B = " << B << endl;

double scal = A \* B;

cout << "ScalVect = " << scal << endl;

Vector2D C = A \* scal;

cout << "C = " << C;

Vector2D D = B \* scal;

cout << "D = " << D;

cout << "|A| = " << A.Modul() << endl;

cout << "|B| = " << B.Modul() << endl;

cout << "|C| = " << C.Modul() << endl;

cout << "|D| = " << D.Modul() << endl << endl;

A < B;

A <= B;

A > B;

A >= B;

A == B;

A != B;

++A;

--A;

B++;

B--;

cout << "A = " << A;

cout << "B = " << B;

return 0;

}

///////////////

// Vector2D.h

///////////////

#pragma once

#include<iostream>

#include <string>

using namespace std;

class Vector2D

{

private:

double x, y;

public:

Vector2D();

Vector2D(double x, double y);

Vector2D(const Vector2D&);

void setX(double value) { x = value; }

void setY(double value) { y = value; }

double getX() const { return x; }

double getY() const { return y; }

Vector2D& operator = (const Vector2D&);

operator string () const;

friend double operator \*(Vector2D g, Vector2D f);

friend Vector2D operator \*(Vector2D g, double l);

double Modul();

friend bool operator <(Vector2D g, Vector2D f);

friend bool operator > (Vector2D g, Vector2D f);

friend bool operator ==(Vector2D g, Vector2D f);

friend bool operator >=(Vector2D g, Vector2D f);

friend bool operator <=(Vector2D g, Vector2D f);

friend bool operator !=(Vector2D g, Vector2D f);

friend istream& operator >> (istream& in, Vector2D& a);

friend ostream& operator << (ostream& in, Vector2D& a);

Vector2D& operator ++();

Vector2D& operator --();

Vector2D operator ++(int);

Vector2D operator --(int);

};

///////////////

// Vector2D.cpp

///////////////

#include "Vector2D.h"

#include<iostream>

#include <string>

#include <sstream>

using namespace std;

Vector2D::Vector2D()

: x(0), y(0)

{}

Vector2D::Vector2D(double a, double b)

{

setX(a), setY(b);

}

Vector2D::Vector2D(const Vector2D& t)

{

\*this = t;

}

Vector2D::operator string() const

{

stringstream sout;

sout << "(" << getX() << "; " << getY() << ")" << endl;

return sout.str();

}

Vector2D& Vector2D::operator = (const Vector2D& r)

{

x = r.x;

y = r.y;

return \*this;

}

double operator \* (Vector2D g, Vector2D f)

{

double t = g.x \* f.x + g.y \* f.y;

return t;

}

Vector2D operator \* (Vector2D g, double l)

{

Vector2D tmp;

tmp.x = g.x \* l;

tmp.y = g.y \* l;

return tmp;

}

double Vector2D::Modul()

{

double mod = sqrt((x \* x) + (y \* y));

return mod;

}

bool operator > (Vector2D a, Vector2D b)

{

if (a.Modul() > b.Modul())

{

return true;

}

else

return false;

}

bool operator < (Vector2D a, Vector2D b)

{

if (a.Modul() < b.Modul())

{

return true;

}

else

return false;

}

bool operator ==(Vector2D a, Vector2D b)

{

if (a.Modul() == b.Modul())

{

return true;

}

else

return false;

}

bool operator <= (Vector2D a, Vector2D b)

{

if (a.Modul() <= b.Modul())

{

return true;

}

else

return false;

}

bool operator >= (Vector2D a, Vector2D b)

{

if (a.Modul() >= b.Modul())

{

return true;

}

else

return false;

}

bool operator != (Vector2D a, Vector2D b)

{

if (a.Modul() != b.Modul())

{

return true;

}

else

return false;

}

istream& operator >> (istream& in, Vector2D& a)

{

cout << "x = "; in >> a.x;

cout << "y = "; in >> a.y;

return in;

}

ostream& operator << (ostream& out, Vector2D& a)

{

out << string(a) << endl;

return out;

}

Vector2D& Vector2D::operator ++()

{

++x;

return \*this;

}

Vector2D& Vector2D::operator --()

{

--y;

return \*this;

}

Vector2D Vector2D::operator ++(int)

{

Vector2D tmp = \*this;

++x;

return tmp;

}

Vector2D Vector2D::operator --(int)

{

Vector2D tmp = \*this;

--y;

return tmp;

}

**Посилання на git-репозиторій з проектом:**

**Висновки:**

На даній лабораторній роботі, я навчився використовувати конструктори та перевантаження операцій.