**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

Інститут **ІКНІ**

Кафедра **ПЗ**

**ЗВІТ**

До лабораторної роботи №5

На тему: «Створення класів»

З дисципліни «Об’єктно-орієнтоване програмування»

**Лектор:** доцент каф. ПЗ

Коротєєва Т.О.

**Виконала:** ст.гр. ПЗ-12

Кохман О.В.

**Прийняла:** доцент каф. ПЗ

Дяконюк Л.М.

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2022р.

∑ \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_.

Львів - 2022

**Тема.** Створення та використання класів.

**Мета.** Навчитися створювати класи, використовувати конструктори для ініціалізації об’єктів, опанувати принципи створення функцій-членів. Навчитися використовувати різні типи доступу до полів та методів класів.

**Індивідуальне завдання**

1.Створити клас відповідно до варіанту.

2.При створенні класу повинен бути дотриманий принцип інкапсуляції.

3.Створити конструктор за замовчуванням та хоча б два інших конструктори для початкової ініціалізації об’єкта.

4.Створити функції-члени згідно з варіантом.

5.Продемонструвати можливості класу завдяки створеному віконному застосуванню.

6.У звіті до лабораторної намалювати UML-діаграму класу, яка відповідає варіанту.

Клас Complex – комплексне число. Клас повинен містити функції-члени, які реалізовують:

а)Додавання.

б)Віднімання.

в)Множення.

г)Піднесення до n-го степеня.

д)Отримання кореня n-го степеня.

е)Задавання значень полів.

є)Зчитування (отримання значень полів).

ж)Представлення в тригонометричній формі.

з)Введення комплексного числа з форми.

и)Виведення комплексного числа на форму.

**Теоретичні відомості**

Ідея класів має на меті дати інструментарій для відображення будови об’єктів реального світу - оскільки кожен предмет або процес має набір характеристик (відмінних рис) іншими словами, володіє певними властивостями і поведінкою. Програми часто призначені для моделювання предметів, процесів і явищ реального світу, тому в мові програмування зручно мати адекватний інструмент для представлення цих моделей.  
Клас є типом даних, який визначається користувачем. У класі задаються властивості і поведінка будь-якого предмету або процесу у вигляді полів даних (аналогічно до того як це є в структурах) і функцій для роботи з ними. Створюваний тип даних володіє практично тими ж властивостями, що і стандартні типи.

Конкретні величини типу даних «клас» називаються екземплярами класу, або об'єктами.

Об’єднання даних і функцій їх обробки з одночасним приховуванням непотрібної для використання цих даних інформації називається інкапсуляцією (encapsulation). Інкапсуляція підвищує ступінь абстракції програми: дані класу і реалізація його функцій знаходяться нижче рівня абстракції, і для написання програми з використанням вже готових класів інформації про них (дані і реалізацію функцій) не потрібно. Крім того, інкапсуляція дозволяє змінити реалізацію класу без модифікації основної частини програми, якщо інтерфейс залишився тим самим (наприклад, при необхідності змінити спосіб зберігання даних з масиву на стек). Простота модифікації, як уже неодноразово зазначалося, є дуже важливим критерієм якості програми.

Специфікатор доступу private і public керують видимістю елементів класу. Елементи, описані після службового слова private, видимі тільки всередині класу. Цей вид доступу прийнятий у класі за замовчуванням. Інтерфейс класу описується після специфікатора public. Дія будь-якого специфікатора поширюється до наступного специфікатора або до кінця класу. Можна задавати кілька секцій private і public, їх порядок значення не має.

Поля класу:

• можуть мати будь-який тип, крім типу цього ж класу (але можуть бути вказівниками або посиланнями на цей клас);

• можуть бути описані з модифікатором const, при цьому вони ініціалізуються тільки один раз (за допомогою конструктора) і не можуть змінюватися;

• можуть бути описані з модифікатором static (розглядається в наступних лабораторних).

Ініціалізація полів при описі не допускається.

Конструктори.  
Конструктор призначений для ініціалізації об’єкту і викликається автоматично при його створенні. Автоматичний виклик конструктора дозволяє уникнути помилок, пов’язаних з використанням неініціалізованих змінних. Нижче наведені основні властивості конструкторів:

• Конструктор не повертає жодного значення, навіть типу void. Неможливо отримати вказівник на конструктор.

• Клас може мати декілька конструкторів з різними параметрами для різних видів ініціалізації (при цьому використовується механізм перевантаження).

• Конструктор без параметрів називається конструктором за замовчуванням.

• Параметри конструктора можуть мати будь-який тип, крім цього ж класу. Можна задавати значення параметрів за замовчуванням. Їх може містити тільки один з конструкторів.

• Якщо програміст не вказав жодного конструктора, компілятор створює його автоматично. Такий конструктор викликає конструктори за замовчуванням для полів класу і конструктори за замовчуванням базових класів. У разі, коли клас містить константи або посилання, при спробі створення об'єкту класу буде видана помилка, оскільки їх необхідно ініціалізувати конкретними значеннями, а конструктор за замовчуванням цього робити не вміє.

• Конструктори не наслідуються.

• Конструктори не можна описувати з модифікаторами const, virtual і static.

• Конструктори глобальних об’єктів викликаються до виклику функції main. Локальні об’єкти створюються, як тільки стає активною область їх дії. Конструктор запускається і при створенні тимчасового об'єкта (наприклад, при передачі об’єкта з функції).

**Код програми**

Назва файлу: Complex.h

#ifndef \_COMPLEX\_H

#define \_COMPLEX\_H

#pragma once

#include <iostream>

#include <cmath>

using namespace std;

const double pi = 3.14159265358979323846;

float findPhi(float real, float imagine);

class Complex {

private:

float real;

float imagine;

public:

Complex();

Complex(float \_imagine);

Complex(float \_real, float \_imagine);

void setReal(float \_real);

void setImagine(float \_imagine);

float getReal();

float getImagine();

Complex addComplex(const Complex Other) const;

Complex substractComplex(const Complex Other) const;

Complex multiplyComplex(const Complex Other) const;

Complex elevateComplexToDegree(const int n) const;

void getComplexRoot(const int n, System::Windows::Forms::TextBox^ textBox) const;

void trigonometricForm(System::Windows::Forms::TextBox^ textBox) const ;

void printToForm(System::Windows::Forms::TextBox^ textBox) const;

};

#endif

#include "MyForm.h"

using namespace CompNum;

int main() {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Application::Run(gcnew MyForm());

return 0;

}

Назва файлу: Complex.cpp

#include "Complex.h"

Complex::Complex() {};

Complex::Complex(float \_imagine) : imagine(\_imagine), real(0) {};

Complex::Complex(float \_real, float \_imagine) : real(\_real), imagine(\_imagine) {};

void Complex::setReal(float \_real) {

real = \_real;

}

void Complex::setImagine(float \_imagine) {

imagine = \_imagine;

}

float Complex::getReal() {

return real;

}

float Complex::getImagine() {

return imagine;

}

void Complex::printToForm(System::Windows::Forms::TextBox^ textBox) const {

System::String^ sign;

if (imagine >= 0) {

sign = "+";

}

else {

sign = "-";

}

textBox->Text = textBox->Text + System::Convert::ToString(real + sign + fabs(imagine) + "i" + Environment::NewLine);

}

Complex Complex::addComplex(const Complex Other) const {

Complex Result;

Result.real = real + Other.real;

Result.imagine = imagine + Other.imagine;

return Complex(Result.real, Result.imagine);

}

Complex Complex::substractComplex(const Complex Other) const {

Complex Result;

Result.real = real - Other.real;

Result.imagine = imagine - Other.imagine;

return Complex(Result.real, Result.imagine);

}

Complex Complex::multiplyComplex(const Complex Other) const {

Complex Result;

Result.real = real \* Other.real - imagine \* Other.imagine;

Result.imagine = real \* Other.imagine + Other.real \* imagine;

return Complex(Result.real, Result.imagine);

}

Complex Complex::elevateComplexToDegree(const int n) const {

Complex Result;

float ro = sqrt(real \* real + imagine \* imagine);

float phi = findPhi(real, imagine);

Result.real = pow(ro, n) \* cos(n \* phi);

Result.imagine = pow(ro, n) \* sin(n \* phi);

return Complex(Result.real, Result.imagine);

}

void Complex::getComplexRoot(const int n, System::Windows::Forms::TextBox^ textBox) const {

Complex Result;

float ro = sqrt(real \* real + imagine \* imagine);

float phi = findPhi(real,imagine);

int k = 0;

while (k < n) {

Result.real = pow(ro, 1 / n) \* cos((phi + 2 \* pi \* k) / n);

Result.imagine = pow(ro, 1 / n) \* sin((phi + 2 \* pi \* k) / n);

Result.printToForm(textBox);

k++;

}

}

void Complex::trigonometricForm(System::Windows::Forms::TextBox^ textBox) const {

float ro = sqrt(this->real \* this->real + this->imagine \* this->imagine);

float phi = findPhi(this->real, this->imagine);

textBox->Text = System::Convert::ToString(ro + " (cos " + phi + "+ isin " + phi + ")");

}

float findPhi(float real, float imagine) {

float phi = 0;

if (real > 0 && imagine >= 0) {

phi = atan(fabs(imagine / real));

}

else if (real < 0 && imagine >= 0) {

phi = pi - atan(fabs(imagine / real));

}

else if (real < 0 && imagine < 0) {

phi = pi + atan(fabs(imagine / real));

}

else if (real > 0 && imagine < 0) {

phi = 2 \* pi - atan(fabs(imagine / real));

}

else if (real == 0 && imagine > 0) {

phi = pi / 2;

}

else if (real == 0 && imagine < 0) {

phi = 3 \* pi / 2;

}

return phi;

}

Назва файлу: MyForm.h

#pragma once

#include "Complex.h"

namespace CompNum {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void)

{

InitializeComponent();

}

protected:

~MyForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::TextBox^ textBox1;

private: System::Windows::Forms::TextBox^ textBox2;

private: System::Windows::Forms::TextBox^ textBox3;

private: System::Windows::Forms::TextBox^ textBox4;

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::Label^ label2;

private: System::Windows::Forms::Label^ label3;

private: System::Windows::Forms::Label^ label4;

private: System::Windows::Forms::Button^ button1;

private: System::Windows::Forms::TextBox^ textBox5;

private: System::Windows::Forms::Button^ button2;

private: System::Windows::Forms::Button^ button3;

private: System::Windows::Forms::Button^ button4;

private: System::Windows::Forms::Button^ button5;

private: System::Windows::Forms::TextBox^ textBox6;

private: System::Windows::Forms::TextBox^ textBox7;

private: System::Windows::Forms::TextBox^ textBox9;

private: System::Windows::Forms::Label^ label5;

private: System::Windows::Forms::Label^ label6;

private: System::Windows::Forms::TextBox^ textBox10;

private: System::Windows::Forms::TextBox^ textBox11;

private: System::Windows::Forms::TextBox^ textBox12;

private: System::Windows::Forms::Button^ button6;

private: System::Windows::Forms::Label^ label7;

private: System::Windows::Forms::Label^ label8;

private: System::Windows::Forms::TextBox^ textBox8;

private: System::Windows::Forms::TextBox^ textBox13;

private: System::Windows::Forms::Label^ label9;

private: System::Windows::Forms::Label^ label10;

private: System::Windows::Forms::Label^ label11;

private: System::Windows::Forms::Label^ label12;

private: System::Windows::Forms::TextBox^ textBox14;

private: System::Windows::Forms::TextBox^ textBox16;

private:

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

void InitializeComponent(void) {

//creating components

}

#pragma endregion

private: System::Void mathOperations(System::Object^ sender, System::EventArgs^ e) {

Complex First(System::Convert::ToDouble(textBox1->Text), System::Convert::ToDouble(textBox2->Text));

Complex Second(System::Convert::ToDouble(textBox3->Text), System::Convert::ToDouble(textBox4->Text));

Complex Result;

System::Windows::Forms::Button^ button = (Button^)sender;

System::String^ textOfButtons = button->Text;

if (textOfButtons == "To Add") {

Result = First.addComplex(Second);

Result.printToForm(textBox5);

}

else if (textOfButtons == "To Substract") {

Result = First.substractComplex(Second);

Result.printToForm(textBox6);

}

else if (textOfButtons == "To Multiply") {

Result = First.multiplyComplex(Second);

Result.printToForm(textBox7);

}

else if (textOfButtons == "To Elevate") {

Result = First.elevateComplexToDegree(System::Convert::ToInt16(textBox10->Text));

Result.printToForm(textBox11);

Result = Second.elevateComplexToDegree(System::Convert::ToInt16(textBox10->Text));

Result.printToForm(textBox12);

}

else if (textOfButtons == "To Get The Root") {

First.getComplexRoot(System::Convert::ToInt16(textBox8->Text), textBox9);

Second.getComplexRoot(System::Convert::ToInt16(textBox8->Text), textBox13);

}

else if (textOfButtons == "Trigonometric Form") {

First.trigonometricForm(textBox16);

Second.trigonometricForm(textBox14);

}

}

};

}

**Протокол роботи**

Табл. 1.1 UML-діаграма

Рис. 1.1 Результат виконання програми

|  |
| --- |
| Complex |
| -float real;  -float imagine; |
| +setReal(float \_real) :void;  +setImagine(float \_imagine) :void;  +getReal():float;  +getImagine():float;  +addComplex(const Complex Other) const: Complex;  +substractComplex(const Complex Other) const: Complex;  +multiplyComplex(const Complex Other) const: Complex;  +elevateComplexToDegree(const int n) const: Complex;  +getComplexRoot(const int n, System::Windows::Forms::TextBox^ textBox) const: void;  +trigonometricForm(System::Windows::Forms::TextBox^ textBox) const: void;  +printToForm(System::Windows::Forms::TextBox^ textBox) const: void; |

Табл. 1.1 UML-діаграма

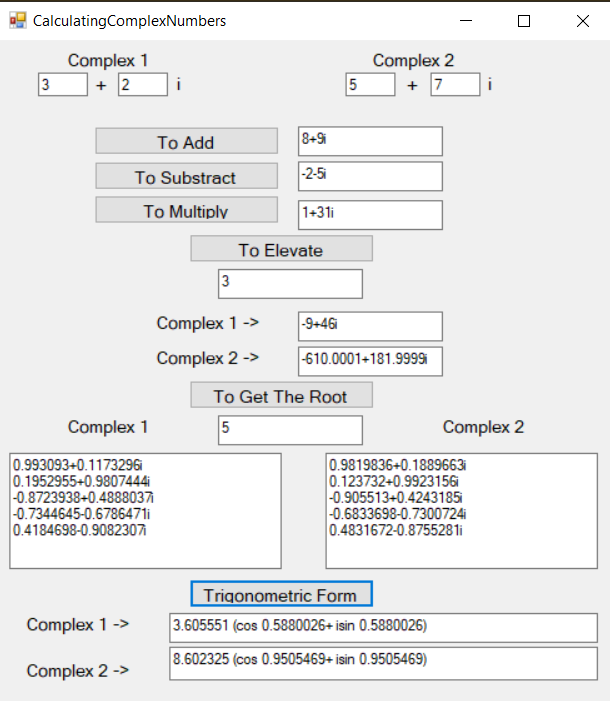


Рис. 1.1 Результат виконання програми

**Висновок**

На цій лабораторній роботі я навчилась створювати класи та продемонструвала можливості класу , використовуючи форму.