Міністерство освіти і науки України
Національний університет “Львівська політехніка”
Інститут комп’ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

Звіт
Про виконання лабораторних робіт №5

На тему:

« Створення та використання класів»
з дисципліни «Об'єктно-орієнтоване програмування»

Лектор:
доцент каф. ПЗ
Коротєєва Т.О.
Виконав:
ст. гр. ПЗ-11
Морозов О. Р.
Прийняла:
доцент каф. ПЗ
Коротєєва Т.О.

« __ » _____ 2022 р.

∑ = _____ _____ .

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.

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.
Опис класу в першому наближенні виглядає так:
class <ім'я> {
[private:]
<Опис прихованих елементів>
public:
<Опис доступних елементів>
}; //Опис закінчується крапкою з комою.
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).
- Конструктор викликається, якщо в програмі зустрілася будь-яка із синтаксичних конструкцій:

```
імя_класу ім’я_об’єкту [(список параметрів)];
//Список параметрів не повинен бути порожнім
імя_класу (список параметрів);
//Створюється об'єкт без імені (список може бути //порожнім)
```

ім’я_класу ім’я_об’єкту = вираз;
//Створюється об’єкт без імені і копіюється

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

1. Створити клас відповідно до варіанту (див. Додаток).
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)Знаходження модуля комплексного числа 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.

Код програми

complex.h

```
#pragma once

#define _USE_MATH_DEFINES
#include <cmath>
#include "ui_mainwindow.h"


class Complex
{
public:
 Complex();
 Complex(double r, double i);

 double getReal();
 double getImagine();

 void setReal(double a);
 void setImagine(double a);
 void setAll(double a, double b);
```

```
 void readFromForm(QLineEdit *real, QLineEdit *imagine);
 void printToForm(QLabel *resultLine);

 Complex sum(const Complex &b);
 Complex mult(const Complex &b);
 Complex minus(const Complex &b);
 Complex pow(int n);

 double module();
 void trigonometry(QLabel *resultLine);

private:
 double real;
 double imagine;
};
```

complex.cpp

```
#include "complex.h"

Complex::Complex()
{
 real = 0;
 imagine = 0;
}

Complex::Complex(double r, double i){
 real = r;
 imagine = i;
}

void Complex::readFromForm(QLineEdit *real, QLineEdit *imagine){
 if(real->text() != ""){
    this->real = real->text().toDouble();
  }
 if(imagine->text() != "") {
    this->imagine = imagine->text().toDouble();
  }
}

void Complex::printToForm(QLabel *resultLine)
{
 QString str, realSTR, imagineSTR;

 if(QString::number(real) != "0"){
  realSTR = QString::number(real);
```

```
  }
 if(QString::number(imagine) != "0"){
    imagineSTR = QString::number(imagine);
  }

 if (imagine > 0){
 str = realSTR + "+" + imagineSTR + "i";
  } else if(imagine < 0){
 str = realSTR + imagineSTR + "i";
  }else {
    str = realSTR;
  }
 if (real == 0){
    str = imagineSTR + "i";
  }
 resultLine->setText(str);
}

Complex Complex::sum(const Complex &b)
{
 return Complex((real + b.real), (imagine+b.imagine));
}

Complex Complex::mult(const Complex &b)
{
 return Complex(((real * b.real) - (imagine * b.imagine)),
((real*b.imagine)+(imagine*b.real)));

}

Complex Complex::minus(const Complex &b)
{
 return Complex((real - b.real), (imagine - b.imagine));
}

Complex Complex::pow(int n)
{
  if( n == 0){
    return Complex(0, 0);

  } else if(n == 1){
    return *this;

  } else if (n&1){
    return this->mult(this->pow(n - 1));
```

```
  } else {
    return this->pow(n / 2).mult(this->pow(n / 2));
    }
}

double Complex::module() {
 return sqrt( (real * real) + (imagine * imagine) );
}

void Complex::trigonometry(QLabel *resultLine)
{
 double r = this->module();
 double arg;
 double x = this->real;
 double y = this->imagine;
 if(x > 0 && y >= 0){
    arg = atan(y / x);
  } else if(x < 0 && y >= 0){
    arg = M_PI - atan(fabs(y / x));
  } else if(x < 0 && y < 0){
    arg = M_PI + atan(fabs(y / x));
  } else if(x > 0 && y < 0){
    arg = 2 * M_PI - atan(abs(double(y) / double(x)));
  } else if (x == 0 && y > 0) {
    arg = M_PI / 2;
  } else if (x == 0 && y < 0) {
    arg = 3 * M_PI / 2;
  }
 else {
    arg = 0;
    resultLine->setText("Error");
  }
 resultLine->setText(QString::number(r) + "(cos(" + QString::number(arg)
+ ") + i*sin(" + QString::number(arg) + "))" );
}

double Complex::getReal(){
 return real;
}
double Complex::getImagine(){
 return imagine;
}

void Complex::setReal(double a){
```

```
 real = a;
}
void Complex::setImagine(double a){
 imagine = a;
}

void Complex::setAll(double a, double b)
{
 real = a;
 imagine = b;
}
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H


#include <QMainWindow>
#include "complex.h"
#include "ui_mainwindow.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
 Q_OBJECT

public:
 MainWindow(QWidget *parent = nullptr);
 ~MainWindow();

private slots:

 void scan(Complex *Z1, Complex *Z2);
 void scanZ1(Complex *Z1);
 void scanZ2(Complex *Z2);

 void buttonClicked();

 void on_pushButton_mod_Z1_clicked();

 void on_pushButton_mod_Z2_clicked();
```

```
 void onPushButtonZ1PowNClicked();

 void onPushButtonZ2PowNClicked();

 void on_pushButton_Z1_trm_Form_clicked();

 void on_pushButton_Z2_trmForm_clicked();

private:
 Ui::MainWindow *ui;

};
#endif // MAINWINDOW_H
```

mainwindow.cpp

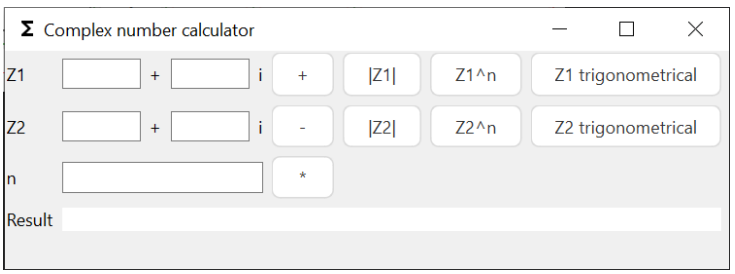
```
#include "mainwindow.h"

Complex Z1, Z2, Z3;

MainWindow::MainWindow(QWidget *parent)
 : QMainWindow(parent)
 , ui(new Ui::MainWindow)
{
 ui->setupUi(this);

 connect(ui->pushButton_plus, SIGNAL(clicked()), this,
SLOT(buttonClicked()));
 connect(ui->pushButton_minus, SIGNAL(clicked()), this,
SLOT(buttonClicked()));
 connect(ui->pushButton_mult, SIGNAL(clicked()), this,
SLOT(buttonClicked()));
}


MainWindow::~MainWindow()
{
 delete ui;
}

void MainWindow::scan(Complex *Z1, Complex *Z2)
{
 Z1->readFromForm(ui->lineEdit_Z1_r, ui->lineEdit_Z1_i);
 Z2->readFromForm(ui->lineEdit_Z2_r, ui->lineEdit_Z2_i);
}
```

```
void MainWindow::scanZ1(Complex *Z1)
{
 Z1->readFromForm(ui->lineEdit_Z1_r, ui->lineEdit_Z1_i);
}

void MainWindow::scanZ2(Complex *Z2)
{
 Z2->readFromForm(ui->lineEdit_Z2_r, ui->lineEdit_Z2_i);
}

void MainWindow::buttonClicked(){
 scan(&Z1, &Z2);

 QString str = ((QPushButton *)sender())->text();

 if(str == '+'){
    Z3 = Z1.sum(Z2);
 } else if(str == '-'){
    Z3 = Z1.minus(Z2);
  } else if(str == '*'){
    Z3 = Z1.mult(Z2);
  }
 Z3.printToForm(ui->label_result);
}

void MainWindow::on_pushButton_mod_Z1_clicked()
{
 scanZ1(&Z1);
 ui->label_result->setText(QString::number(Z1.module()));
}


void MainWindow::on_pushButton_mod_Z2_clicked()
{
 scanZ2(&Z2);
 ui->label_result->setText(QString::number(Z2.module()));
}

void MainWindow::onPushButtonZ1PowNClicked()
{
  int n = (ui->lineEdit_n->text()).toInt();
  scanZ1(&Z1);
  Z3 = Z1.pow(n);
  Z3.printToForm(ui->label_result);
```

```
}

void MainWindow::onPushButtonZ2PowNClicked()
{
 int n = (ui->lineEdit_n->text()).toInt();
 scanZ2(&Z2);
 Z3 = Z2.pow(n);
 Z3.printToForm(ui->label_result);
}


void MainWindow::on_pushButton_Z1_trm_Form_clicked()
{
 scanZ1(&Z1);
 Z1.trigonometry(ui->label_result);
}


void MainWindow::on_pushButton_Z2_trmForm_clicked()
{
   scanZ2(&Z2);
   Z2.trigonometry(ui->label_result);
}
```

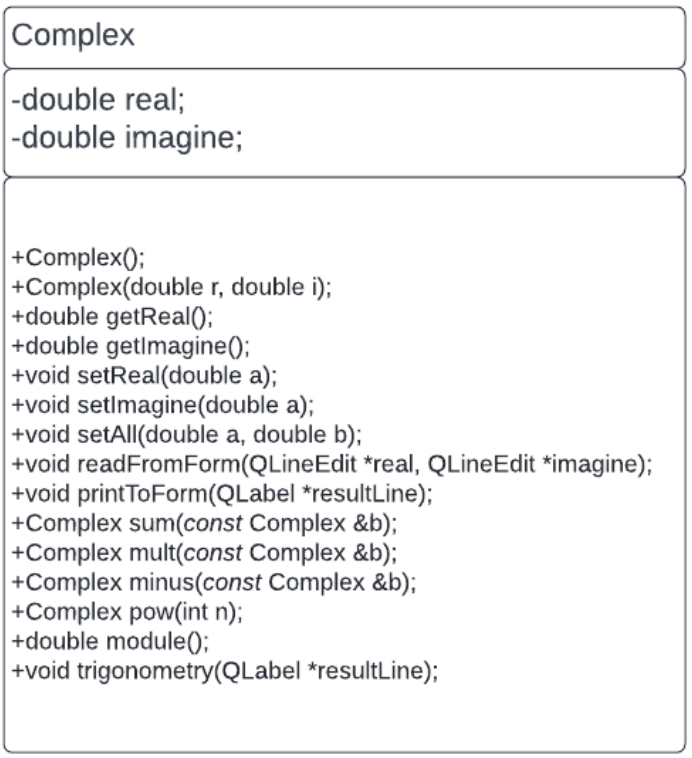
main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
 QApplication a(argc, argv);
 MainWindow w;
 w.setWindowTitle("Complex number calculator");
 w.show();
 return a.exec();
}
```



Вигляд програми

UML-діаграма



ВИСНОВКИ

Виконавши лабораторну роботу №5, я навчився використовувати класи та об’єкти, різні типи полів та методів класу. За допомогою цих знань, я розробив програму для отримання різної інформації, щодо введеного комплексного числа.