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

Звіт
Про виконання лабораторних робіт №6

На тему:

« Перевантаження функцій і операцій, дружні функції, статичні члени класу»

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: Перевантаження функцій і операцій, дружні функції, статичні члени класу.
Мета:Навчитися використовувати механізм перевантаження функцій та операцій. Навчитися створювати та використовувати дружні функції. Ознайомитися зі статичними полями і методами та навчитися їх використовувати.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Перевизначення операцій.

C++ підтримує спеціальні засоби, які дозволяють перевизначити вже існуючі операції. Наприклад, для операції + можна ввести своє власне визначення, яке реалізує операцію додавання для об'єктів певного класу. Фактично пере визначення для операцій існує і в мові С. Так, операція + може використовувати як об'єкти типу int, так і об'єкти типу float. C++ розширює цю ідею.

Для визначення операції використовується функція, що вводиться користувачем. Тип функції визначається іменем класу, далі записується ключове слово operator, за яким слідує символ операції, в круглих дужках дається перелік параметрів, серед яких хоча б один типу клас.

Функція operator+ повертає результат типу complex та має параметри C1 та C2 типу complex. Функції-операції мають бути нестатичними функціями-членами класу або мати мінімум один аргумент типу класу. За виключенням операції присвоєння всі перевизначені оператори наслідуються.

Дружні функції

Дружньою функцією класу називається функція, яка сама не є членом класу, але має повні права на доступ до закритих та захищених елементів класу. Оскільки така функція не є членом класу, то вона не може бути вибрана з допомогою операторів (.) та (->), Дружня функція класу викликається звичайним способом. Для опису дружньої функції використовується ключове слово friend

Статичні змінні класу

До тепер рахувалось, що дані в кожному об'єкті є власністю саме цього об'єкту та не використовуються іншими об'єктами цього класу. Та іноді приходиться слідкувати за накопиченням даних. Наприклад, необхідно з'ясувати скільки об'єктів даного класу було створено на даний момент та скільки з них існує. Статичні змінні-члени досяжні для всіх екземплярів класу. Це є компроміс між глобальними даними, які досяжні всім елементам програми, та даними, що досяжні тільки об'єктам даного класу.

Статична змінні створюється в одному екземплярі для всіх об'єктів даного класу. Розглянемо приклад. Об'єкт класу Dog включає статичну змінну-член HowManyDogs (скільки котів). Ця змінна нараховує кількість об'єктів класу Dog, що створені під час виконання програми. Для цього статична змінна

HowManyDogs збільшується на 1 при кожному виклику конструктора класу Dog , а при виклику деструктора зменшується на 1.

Статичні змінні необхідно обов'язково ініціалізувати. Якщо необхідно обмежити доступ до статичних змінних, то оголошуйте їх закритими або захищеними.

Статичні функції класу

Статичні функції класу подібні до статичних змінних: вони не належать одному об'єкту, а знаходяться в області дії всього класу. Статичні функції-члени не мають вказівника this. Відповідно їх не можна оголосити як const. Статичні функції-члени не можуть звертатись до нестатичних змінних. До статичних функцій-членів можна звертатись з об'єкту їх класу, або вказавши повне ім'я, включаючи ім'я об'єкту.

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

На основі класу з попередньої лабораторної:
Перевантажити як мінімум три функції-члени з попереднього завдання.
Перевантажити операції згідно з варіантом (див. Додаток). Для операцій, для яких не вказані символи, вибрати символи самостійно.
Створити дружні функції згідно з варіантом.
Створити статичні поля та статичні методи згідно з варіантом.
Продемонструвати розроблені можливості класу завдяки створеному віконному застосуванню.
Оформити звіт до лабораторної роботи.

Завдання згідно варіанту

7. 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.

Перевантажити операції, як функції члени:
Додавання
Віднімання
Множення
Знаходження модуля.
Перевантажити операції, як дружні-функції:
Введення комплексного числа з форми ("<<")
Виведення комплексного числа на форму(">>")
Більше (">")

Менше ("<")
Рівне ("==") (при порівнянні порівнювати модулі комплексних чисел).
Створити статичне поле, в якому б містилась інформація про кількість створених об'єктів, а також статичні функції для роботи з цим полем.

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);
 ~Complex();

 friend void operator<<(QLabel *resultLine, Complex &a);
 friend void operator>>(QLineEdit *array[], Complex &a);

 friend bool operator<(const Complex &a, const Complex &b);
 friend bool operator>(const Complex &a, const Complex &b);
 friend bool operator==(const Complex &a, const Complex &b);

 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 operator+(const Complex &b);
 Complex operator-(const Complex &b);
 Complex operator*(const Complex &b);

 void powComplex(int n, QLabel *resultLine);
 double module() const;
 double operator!();
 void trigonometry(QLabel *resultLine);

 static int getAmount();

private:
 double real;
 double imagine;
 static int amount;
};
```

complex.cpp

```
#include "complex.h"

int Complex::amount = 0;

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

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

Complex::~Complex()
{
 amount--;
}

int Complex::getAmount(){
 return amount;
```

```
}

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));
}

void Complex::powComplex(int n, 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");
   }
  double rr = pow(r, n);
  double rarg = pow(arg, n);
  resultLine->setText(QString::number(rr) + "(cos(" +
QString::number(rarg) + ") + i*sin(" + QString::number(rarg) + "))" );
}

double Complex::module() const{
 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;
}

void operator>>(QLabel *resultLine, Complex &a){
 a.printToForm(resultLine);
}
void operator<<(QLineEdit *array[], Complex &a){
```

```
 if(array[0]->text().length() != 0){
    a.setReal((array[0]->text()).toDouble());
  }
 if (array[1]->text().length() != 0) {
    a.setImagine((array[1]->text()).toDouble());
  }
}

bool operator<(const Complex &a, const Complex &b){
 return a.module() < b.module();
}
bool operator>(const Complex &a, const Complex &b){
 return a.module() > b.module();
}
bool operator==(const Complex &a, const Complex &b){
 return a.module() == b.module();
}

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

double Complex::operator!(){
 return sqrt( (real * real) + (imagine * imagine) );
}
```

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();

 void onAmountClicked();

 void boolButtonClicked();

 void operatorButtonClicked();

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()));

 connect(ui->pushButton_Z1_pow_n, SIGNAL(clicked()), this,
SLOT(onPushButtonZ1PowNClicked()));
 connect(ui->pushButton_Z2_pow_n, SIGNAL(clicked()), this,
SLOT(onPushButtonZ2PowNClicked()));

 connect(ui->pushButton_l, SIGNAL(clicked()), this,
SLOT(boolButtonClicked()));
 connect(ui->pushButton_m, SIGNAL(clicked()), this,
SLOT(boolButtonClicked()));
 connect(ui->pushButton_equal, SIGNAL(clicked()), this,
SLOT(boolButtonClicked()));

 connect(ui->pushButton_amount, SIGNAL(clicked()), this,
SLOT(onAmountClicked()));

 connect(ui->pushButton_plus_2, SIGNAL(clicked()), this,
SLOT(operatorButtonClicked()));
 connect(ui->pushButton_minus_2, SIGNAL(clicked()), this,
SLOT(operatorButtonClicked()));
 connect(ui->pushButton_mult_2, SIGNAL(clicked()), this,
SLOT(operatorButtonClicked()));
}


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));
}

void MainWindow::onPushButtonZ1PowNClicked()
{
```

```
 int n = (ui->lineEdit_n->text()).toInt();
 scanZ1(&Z1);
 Z1.powComplex(n, ui->label_result);
}

void MainWindow::onPushButtonZ2PowNClicked()
{
 int n = (ui->lineEdit_n->text()).toInt();
 scanZ2(&Z2);
 Z2.powComplex(n, 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);
}

void MainWindow::onAmountClicked()
{
 QString str = QString::number(Complex::getAmount());
 ui->label_result->setText(str);
}

void MainWindow::boolButtonClicked()
{
 QString str = ((QPushButton*)sender())->text();
 scan(&Z1, &Z2);

 if(str == "<"){
    (Z1 < Z2) ? str = "true" : str = "false";
    ui->label_result->setText(str);
  } else if(str == ">"){
    (Z1 > Z2) ? str = "true" : str = "false";
    ui->label_result->setText(str);
  } else {
    (Z1 == Z2) ? str = "true" : str = "false";
    ui->label_result->setText(str);
```

```
  }
}

void MainWindow::operatorButtonClicked()
{
 QString str = ((QPushButton*)sender())->text();
 scan(&Z1, &Z2);

 if(str == "(+)"){
    Z3 = Z1+Z2;
 } else if(str == "(-)"){
    Z3 = Z1-Z2;
  } else if(str == "(*)"){
    Z3 = Z1*Z2;
  }
 Z3.printToForm(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

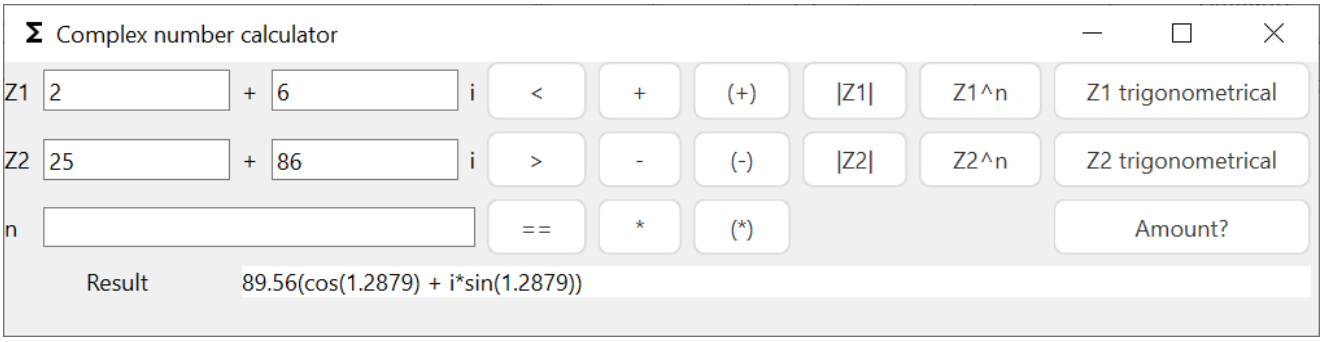

рис 1. тригонометрична форма другого числа

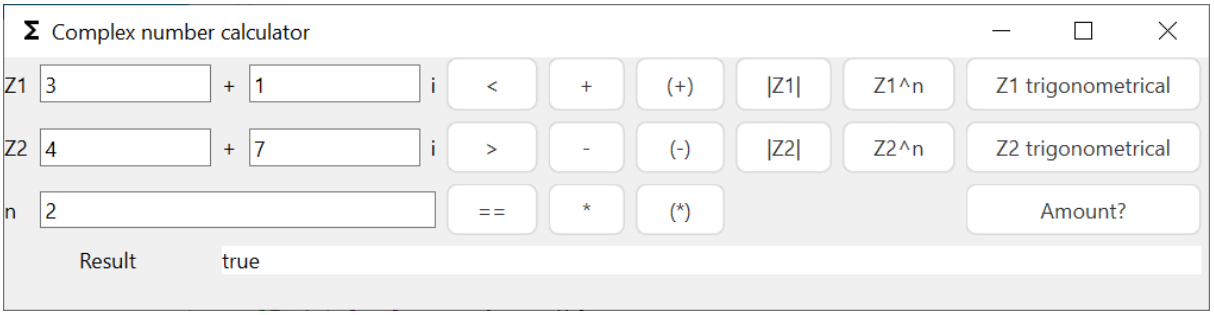

рис 2. порівняння двох чисел(перше число менше другого)

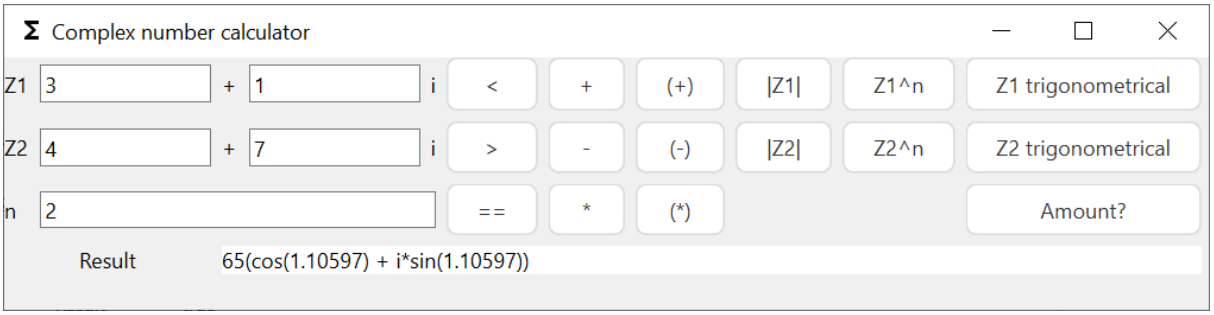

рис 3. (друге число в степені n)

ВИСНОВКИ

Виконавши лабораторну роботу №6, я навчився використовувати механізм перевантаження функцій та операцій. Навчився створювати та використовувати дружні функції. Ознайомився зі статичними полями і методами та навчився їх використовувати.