МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

До лабораторної роботи № 6
На тему: «Перевантаження функцій і операцій, дружні функції»
З дисципліни: «Об'єктно-орієнтоване програмування»

Лектор:
доц. кафедри ПЗ
Коротєєва Т. О.

Виконав:
ст. гр. ПЗ-16
Чаус О. М.

Прийняв:
асист. кафедри ПЗ
Дивак І. В.

« ____ » _____ 2022 р.

∑= _____ .....:

Львів-2022

Тема роботи: Перевантаження функцій і операцій, дружні функції

Мета роботи: 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.

```
class complex

{

 double real, imag;// дійсна та уявна частини

 public:

complex ( double r, double i ) { real = r; imag = i; }// конструктор з ініціалізацією

complex operator+ (complex c1, complex c2)

{return complex (c1.real + c2.real, c1.imag + c2.imag); }// перевизначена функція додавання

}



int main()

{ complex c1(0, 1), c2(1,0), c3;// об'єкти типу клас complex

c3 = c1 + c2;// скорочений запис

c3=operator+(c1,c2); // явний виклик функці.ї Результат однаковий.

return 0;

}
```

Функція **operator+** повертає результат типу **complex** та має параметри С1 та С2 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**. Наприклад,

```
class X

{

  int i;

  friend void F( X *, int);

  public:

void M (int);

};

void F(X *px, int a)

{ px ->i=a;} //доступ до закритого елемента через вказівник об'єкту

void X ::M(int a)

{i=a;} //прямий доступ до закритого елемента


int main( )

{  X mm;

  F(&mm, 6);//виклик фружньої функції

  mm.M(6);//виклик методу класу

}
```

Статичні змінні класу

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

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

```
{
public:
Cat (int age) : itsAge(age) { HowManyCats++;}  // конструктор
virtual ~Cat( ) { HowManyCats--;}  // деструктор
...
static int HowManyCats;


private:
  int itsAge;
};


int Cat : : HowManyCats = 0;
int main( )
{
  const int MaxCats = 5; int I;
   Cat *CatHouse[MaxCats];  // виклик конструктора
   for (i=0; i<MaxCats; i++)
   CatHouse[i] = new Cat(i);
...
return 0;
}
```

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 для лабораторної роботи

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

Клас Polynom – квадратичний тричлен (ax2+bx+c).
Перевантажити операції, як функції члени:
Додавання
Віднімання
Знаходження значення виразу для заданого x ("( )")
Заміна всіх коефіцієнтів полінома на протилежні ("!")
Добуток полінома на скаляр
Перевантажити операції, як дружні-функції:
Введення полінома з форми ("<<")
Виведення полінома на форму (">>")
Доступ до i-го коефіцієнта полінома ("[ ]")
Рівне ("==") (при порівнянні порівнювати значення коефіцієнтів при найстарших степенях x).
Створити статичне поле, в якому б містилась інформація про кількість створених об'єктів, а також статичні функції для роботи з цим полем.

Хід роботи

Файл polynom.h:

```
#ifndef POLYNOM_H
#define POLYNOM_H

#include <QLineEdit>

struct Roots
{
    double x1;
    double x2;
    bool solveable;
};
struct formText
{
    QLineEdit * A;
    QLineEdit * B;
    QLineEdit * C;
};

class Polynom
```

```
{
    private:
    double m_a;
    double m_b;
    double m_c;
    static int ObjectsCreated;
    static int ObjectsExisting;
    public:
    static int GetCreated();
    static int GetExisting();
    Polynom()
    {
        m_a = 0;
        m_b = 0;
        m_c = 0;
        ObjectsCreated++;
        ObjectsExisting++;
    }
    Polynom(double a, double b, double c)
    {
        m_a = a;
        m_b = b;
        m_c = c;
        ObjectsCreated++;
        ObjectsExisting++;
    }
    Polynom(double* array)
    {
        m_a = array[0];
        m_b = array[1];
        m_c = array[2];
        ObjectsCreated++;
        ObjectsExisting++;
    }
    Polynom(Polynom* A)
    {
        m_a = A->m_a;
        m_b = A->m_b;
        m_c = A->m_c;
        ObjectsCreated++;
        ObjectsExisting++;
    }
    ~Polynom() {ObjectsExisting--;};
    Polynom operator+(Polynom &poly);
    Polynom operator-(Polynom &poly);
    void operator!();
    double operator()(double x);
    void operator*(double mult);
    double operator[](int i);
    void friend operator<<(Polynom &poly, formText &form);
    void friend operator>>(Polynom &poly, formText &form);
    bool friend operator==(Polynom &poly, Polynom &poly2);
    double Derivative(double point);
    double Integral(double low, double high);
    Roots findRoots();
    double getA();
    double getB();
    double getC();
    void setA(double a);
    void setA(int a);
    void setB(double b);
    void setB(int b);
    void setC(double c);
    void setC(int c);
};


#endif // POLYNOM_H
```

Файл polynom.cpp:

```
#include "polynom.h"

int Polynom::GetCreated()
{
    return ObjectsCreated;
}
int Polynom::GetExisting()
{
    return ObjectsExisting;
}
Polynom Polynom::operator+(Polynom &poly)
{
    Polynom res;
    res.m_a = m_a + poly.getA();
    res.m_b = m_b + poly.getB();
    res.m_c = m_c + poly.getC();
    return res;
}
Polynom Polynom::operator-(Polynom &poly)
{
    Polynom res;
    res.m_a = m_a - poly[0];
    res.m_b = m_b - poly[1];
    res.m_c = m_c - poly[2];
    return res;
}
void Polynom::operator!()
{
    m_a *= -1;
    m_b *= -1;
    m_c *= -1;
}
double Polynom::operator()(double x)
{
    return m_a * x * x + m_b * x + m_c;
}
void Polynom::operator*(double mult)
{
    m_a *= mult;
    m_b *= mult;
    m_c *= mult;
}
double Polynom::operator[](int i)
{
    switch (i)
    {
    case 0:
        return m_a;
        break;
    case 1:
        return m_b;
        break;
    case 2:
        return m_c;
        break;
    default:
        return NAN;
    }
}
void operator<<(Polynom &poly, formText &form)
{
    poly.m_a = 0;
    poly.m_b = 0;
    poly.m_c = 0;
    if (form.A->text() != "")
    {
```

```
       poly.m_a = form.A->text().toDouble();
    }
    if (form.B->text() != "")
    {
        poly.m_b = form.B->text().toDouble();
    }
    if (form.C->text() != "")
    {
        poly.m_c = form.C->text().toDouble();
    }
}
void operator>>(Polynom &poly, formText &form)
{
    form.A->setText(QString::number(poly.m_a));
    form.B->setText(QString::number(poly.m_b));
    form.C->setText(QString::number(poly.m_c));
}
bool operator==(Polynom &poly, Polynom &poly2)
{
    return (poly.m_a == poly2.m_a);
}
double Polynom::Derivative(double point)
{
    return 2 * m_a * point + m_b;
}
double Polynom::Integral(double low, double high)
{
    return ((m_a * high * high * high) / 3 + (m_b * high * high) / 2 + m_c * high) -
    ((m_a * low * low * low) / 3 + (m_b * low * low) / 2 + m_c * low);
}
Roots Polynom::findRoots()
{
    double D = m_b * m_b - 4 * m_a * m_c;
    if (D < 0)
    {
        return { 0, 0, false };
    }
    else
    {
    double x1 = (-1 * m_b - sqrt(D)) / (2 * m_a);
    double x2 = (-1 * m_b + sqrt(D)) / (2 * m_a);
    return { x1, x2, true };
    }
}
double Polynom::getA()
{
    return m_a;
}
double Polynom::getB()
{
    return m_b;
}
double Polynom::getC()
{
    return m_c;
}
void Polynom::setA(double a)
{
    m_a = a;
}
void Polynom::setA(int a)
{
    m_a = a;
}
void Polynom::setB(double b)
{
    m_b = b;
}
```

```
        void Polynom::setB(int b)
        {
            m_b = b;
        }
        void Polynom::setC(double c)
        {
            m_c = c;
        }
        void Polynom::setC(int c)
        {
            m_c = c;
}
```

Файл mainwindow.cpp:

```
        …
        MainWindow::MainWindow(QWidget *parent)
            : QMainWindow(parent)
            , ui(new Ui::MainWindow)
        {
            ui->setupUi(this);
            QDoubleValidator* validator = new QDoubleValidator(-50, 50, 4);
            QLocale locale(QLocale::English, QLocale::UnitedStates);
            validator->setLocale(locale);
            ui->textBox1->setValidator(validator);
            ui->textBox2->setValidator(validator);
            ui->textBoxA->setValidator(validator);
            ui->textBoxB->setValidator(validator);
            ui->textBoxC->setValidator(validator);
            ui->textBoxX->setValidator(validator);
        }

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


        void MainWindow::on_buttonRes_clicked()
        {
            resClicked = true;
            derivativeClicked = false;
            integralClicked = false;
            plusClicked = false;
            minusClicked = false;
            multiplyClicked = false;
            ui->textBoxX->setReadOnly(false);
            ui->textBox1->setReadOnly(true);
            ui->textBox2->setReadOnly(true);
            ui->buttonEqual->setEnabled(true);
        }


        void MainWindow::on_buttonDer_clicked()
        {
            resClicked = false;
            derivativeClicked = true;
            integralClicked = false;
            plusClicked = false;
            multiplyClicked = false;
            ui->textBoxX->setReadOnly(false);
            ui->textBox1->setReadOnly(true);
            ui->textBox2->setReadOnly(true);
            ui->buttonEqual->setEnabled(true);
        }


        void MainWindow::on_buttonIntegral_clicked()
        {
```

```
    resClicked = false;
    derivativeClicked = false;
    integralClicked = true;
    minusClicked = false;
    plusClicked = false;
    multiplyClicked = false;
    ui->textBoxX->setReadOnly(true);
    ui->textBox1->setReadOnly(false);
    ui->textBox2->setReadOnly(false);
    ui->buttonEqual->setEnabled(true);
}


void MainWindow::on_buttonPlus_clicked()
{
    resClicked = false;
    derivativeClicked = false;
    integralClicked = false;
    plusClicked = true;
    minusClicked = false;
    multiplyClicked = false;
    ui->textBoxX->setReadOnly(true);
    ui->textBox1->setReadOnly(true);
    ui->textBox2->setReadOnly(true);
    ui->buttonEqual->setEnabled(true);
    formText form = {ui->textBoxA, ui->textBoxB, ui->textBoxC};
    poly<<form;
    ui->textBoxA->clear();
    ui->textBoxB->clear();
    ui->textBoxC->clear();
}

void MainWindow::on_buttonMinus_clicked()
{
    resClicked = false;
    derivativeClicked = false;
    integralClicked = false;
    plusClicked = false;
    minusClicked = true;
    multiplyClicked = false;
    ui->textBoxX->setReadOnly(true);
    ui->textBox1->setReadOnly(true);
    ui->textBox2->setReadOnly(true);
    ui->buttonEqual->setEnabled(true);
    formText form = {ui->textBoxA, ui->textBoxB, ui->textBoxC};
    poly<<form;
    ui->textBoxA->clear();
    ui->textBoxB->clear();
    ui->textBoxC->clear();
}

void MainWindow::on_buttonMultiply_clicked()
{
    resClicked = false;
    derivativeClicked = false;
    integralClicked = false;
    plusClicked = false;
    minusClicked = false;
    multiplyClicked = true;
    ui->textBoxX->setReadOnly(false);
    ui->textBox1->setReadOnly(true);
    ui->textBox2->setReadOnly(false);
    ui->buttonEqual->setEnabled(true);
}


void MainWindow::on_buttonRoots_clicked()
{
```

```
    QMessageBox msg;
    formText form = {ui->textBoxA, ui->textBoxB, ui->textBoxC};
    poly<<form;
    Roots roots = poly.findRoots();
    if(roots.solveable)
    {
        if(roots.x1 == roots.x2)
        {
            msg.setText("X = " + QString::number(roots.x1));
            msg.exec();
        }
        else
        {
            msg.setText("X1 = " + QString::number(roots.x1) + ", X2 = " +
QString::number(roots.x2));
            msg.exec();
        }
    }
    else
    {
        msg.setText("No roots");
        msg.exec();
    }
}

void MainWindow::on_buttonInverse_clicked()
{
    formText form = {ui->textBoxA, ui->textBoxB, ui->textBoxC};
    poly<<form;
   !poly;
    poly>>form;
}

void MainWindow::on_buttonEqual_clicked()
{
    if(plusClicked || minusClicked)
    {
        Polynom poly2, poly3;
        formText form = {ui->textBoxA, ui->textBoxB, ui->textBoxC};
        poly2<<form;
        if(poly == poly2)
            ui->label_Equal->setText("Polynoms are equal");
        else ui->label_Equal->setText("Polynoms are not equal");
        if(plusClicked)
            poly3 = poly + poly2;
        else
            poly3 = poly - poly2;
        poly3>>form;
        ui->buttonEqual->setEnabled(false);
    }
    else
    {
        formText form = {ui->textBoxA, ui->textBoxB, ui->textBoxC};
        poly<<form;
    }
    if(multiplyClicked)
    {
        if(ui->textBoxX->text() != "")
        {
            poly * (ui->textBoxX->text().toDouble());
            formText form = {ui->textBoxA, ui->textBoxB, ui->textBoxC};
            poly>>form;
            multiplyClicked = false;
            ui->textBoxX->setText("");
            ui->textBoxX->setReadOnly(true);
            ui->buttonEqual->setEnabled(false);
        }
    }
```

```
        if(resClicked)
        {
            if(ui->textBoxX->text() != "")
            {
                ui->Result->setText(QString::number(poly((ui->textBoxX-
    >text().toDouble()))));
                resClicked = false;
                ui->textBoxX->setText("");
                ui->textBoxX->setReadOnly(true);
                ui->buttonEqual->setEnabled(false);
            }
        }
        if(derivativeClicked)
        {
            if(ui->textBoxX->text() != "")
            {
                ui->Result->setText(QString::number(poly.Derivative(ui->textBoxX-
    >text().toDouble())));
                derivativeClicked = false;
                ui->textBoxX->setText("");
                ui->textBoxX->setReadOnly(true);
                ui->buttonEqual->setEnabled(false);
            }
        }
        if(integralClicked)
        {
            if(ui->textBoxA->text() != "" && ui->textBoxA->text() != "")
            {
                ui->Result->setText(QString::number(poly.Integral(ui->textBox1-
    >text().toDouble(), ui->textBox2->text().toDouble())));
                integralClicked = false;
                ui->textBox1->setText("");
                ui->textBox2->setText("");
                ui->textBox1->setReadOnly(true);
                ui->textBox2->setReadOnly(true);
                ui->buttonEqual->setEnabled(false);
            }
        }
        ui->label_Objects->setText("Objects existing: " +
    QString::number(Polynom::GetExisting()));
        ui->label_Objects1->setText("Objects created: " +
    QString::number(Polynom::GetCreated()));
}
```

Висновок: навчився використовувати механізм перевантаження функцій та операцій. Навчився створювати та використовувати дружні функції. Ознайомився зі статичними полями і методами та навчився їх використовувати.