Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Інститут комп’ютерних наук та інформаційних технологій

Кафедра ПЗ

Звіт

до лабораторної роботи №6

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

з дисципліни “Об’єктно-орієнтоване програмування”

Виконав:

студент групи ПЗ-11

Ясногородський Нікіта

Перевірив:

доц. Коротєєва Т.О.

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.

Варіант 10 (поліном)
Завдання:
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 і-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.

Теоретичні відомості:

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 перевизначення для операцій існує і в мові C. 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.

Функції-операції мають бути нестатичними функціями-членами класу або мати мінімум один аргумент типу класу. За виключенням операції присвоєння всі перевизначені оператори наслідуються.

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**.

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. Статичні змінні необхідно обов'язково ініціалізувати.

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.

Результат:

main.cpp

```
#include <QApplication>

#include "widget.h"

int main(int argc, char *argv[]) {
  QApplication a(argc, argv);
  PolynomWidget w;

  w.show();

  return a.exec();
}
```

polynom.cpp

```
#include "polynom.h"
```

```
#include <cmath>
#include <stdexcept>

bool compare_double(double x, double y, double epsilon = 0.01) {
  return fabs(x - y) < epsilon;
}

Polynom::Polynom() {
  a = b = c = 0;
  Polynom::incrementPolynomsCount();
}

Polynom::Polynom(double a, double b, double c) {
  Polynom::incrementPolynomsCount();
  this->a = a;
  this->b = b;
  this->c = c;
}

Polynom::~Polynom() { Polynom::decrementPolynomsCount(); }

Polynom *Polynom::operator*(const double increaseBy) {
  if (increaseBy != 0) {
    this->a *= increaseBy;
    this->b *= increaseBy;
    this->c *= increaseBy;
  }
  return this;
}

Polynom *Polynom::operator-(Polynom substractPolynom) {
  this->a -= substractPolynom.a;
  this->b -= substractPolynom.b;
  this->c -= substractPolynom.c;
  return this;
}

Polynom *Polynom::operator!() {
  this->a = -this->a;
  this->b = -this->b;
  this->c = -this->c;
  return this;
}

Polynom *Polynom::operator+(Polynom addPolynom) {
  this->a += addPolynom.a;
  this->b += addPolynom.b;
  this->c += addPolynom.c;
  return this;
}

double Polynom::operator()(double x) { return a * (pow(x, 2)) + b * x + c; }

void operator<<(QLineEdit *out[POLYNOM_LENGTH], Polynom &polynom) {
  out[0]->setText(QString::number(polynom.a));
  out[1]->setText(QString::number(polynom.b));
  out[2]->setText(QString::number(polynom.c));
};

void operator>>(QLineEdit *in[POLYNOM_LENGTH], Polynom &polynom) {
  polynom.a = in[0]->text().toDouble();
  polynom.b = in[1]->text().toDouble();
```

```
  polynom.c = in[2]->text().toDouble();
};

bool operator==(Polynom &p1, Polynom &p2) {
  return compare_double(p1.a, p2.a) || compare_double(p1.b, p2.b) ||
         compare_double(p1.c, p2.c);
};
```

widget.cpp

```
#include "widget.h"

#include <QGridLayout>
#include <QMessageBox>

#include "polynom.h"

void PolynomWidget::onInputConfirm() {
  QLineEdit *coffInputs[] = {this->coff_a, this->coff_b, this->coff_c};

  coffInputs >> *this->polynom;

  emit valueChanged();
}

void PolynomWidget::onInputIncreaseBy() {
  (*this->polynom) * this->increaseBy->text().toDouble();
  emit valueChanged();
}

void PolynomWidget::onConfirmSubstract() {
  QLineEdit *coffI2nputs[] = {this->coff_a2, this->coff_b2, this->coff_c2};
  Polynom substractPolynom;

  coffI2nputs >> substractPolynom;
  (*this->polynom) - substractPolynom;

  emit valueChanged();
}

void PolynomWidget::onConfirmAdd() {
  QLineEdit *coffI2nputs[] = {this->coff_a2, this->coff_b2, this->coff_c2};
  Polynom addPolynom;

  coffI2nputs >> addPolynom;
  (*this->polynom) + addPolynom;

  emit valueChanged();
}

void PolynomWidget::onConfirmInvertCoff() {
  !(*this->polynom);
  emit valueChanged();
}

void PolynomWidget::onValueChange() {
  QLineEdit *sideInputs[] = {this->coff_a, this->coff_b, this->coff_c};

  sideInputs << *this->polynom;

  this->y_value->setText(
```

```
        QString::number((*this->polynom)(this->x_value->text().toDouble())));
}

PolynomWidget::PolynomWidget(QWidget *parent) : QWidget(parent) {
  QGridLayout *mainLayout = new QGridLayout;

  this->polynom = new Polynom;

  this->coff_a = new QLineEdit("1");
  this->coff_a->setPlaceholderText("Coff A");

  this->coff_b = new QLineEdit("2");
  this->coff_b->setPlaceholderText("Coff B");

  this->coff_c = new QLineEdit("0");
  this->coff_c->setPlaceholderText("Coff C");

  this->coff_a2 = new QLineEdit;
  this->coff_a2->setPlaceholderText("Coff A2");

  this->coff_b2 = new QLineEdit;
  this->coff_b2->setPlaceholderText("Coff B2");

  this->coff_c2 = new QLineEdit;
  this->coff_c2->setPlaceholderText("Coff C2");

  this->x_value = new QLineEdit("1");

  this->increaseBy = new QLineEdit("2");

  this->y_value = new QLineEdit;
  this->y_value->setReadOnly(true);

  this->confirmInput = new QPushButton("Calculate");
  this->confirmIncreaseBy = new QPushButton("Increasy coff by");
  this->confirmInvertCoff = new QPushButton("Invert coff");
  this->confirmAdd = new QPushButton("Add polynoms");
  this->confirmSubstract = new QPushButton("Substract polynoms");

  mainLayout->addWidget(this->coff_a, 0, 0);
  mainLayout->addWidget(this->coff_b, 0, 1);
  mainLayout->addWidget(this->coff_c, 0, 2);
  mainLayout->addWidget(this->confirmInput, 0, 3);
  mainLayout->addWidget(this->confirmInvertCoff, 0, 4);

  mainLayout->addWidget(this->coff_a2, 2, 0);
  mainLayout->addWidget(this->coff_b2, 2, 1);
  mainLayout->addWidget(this->coff_c2, 2, 2);
  mainLayout->addWidget(this->confirmAdd, 2, 3);
  mainLayout->addWidget(this->confirmSubstract, 2, 4);

  mainLayout->addWidget(this->increaseBy, 4, 0);
  mainLayout->addWidget(this->confirmIncreaseBy, 4, 1);

  mainLayout->addWidget(new QLabel("Enter x value"), 5, 0);
  mainLayout->addWidget(this->x_value, 5, 1);
  mainLayout->addWidget(new QLabel("Evaluated y value"), 5, 2);
  mainLayout->addWidget(this->y_value, 5, 3);

  connect(this->confirmInput, &QPushButton::released, this,
          &PolynomWidget::onInputConfirm);

  connect(this->confirmIncreaseBy, &QPushButton::released, this,
```

```
        &PolynomWidget::onInputIncreaseBy);

    connect(this->confirmInvertCoff, &QPushButton::released, this,
        &PolynomWidget::onConfirmInvertCoff);

    connect(this->confirmSubstract, &QPushButton::released, this,
        &PolynomWidget::onConfirmSubstract);

    connect(this->confirmAdd, &QPushButton::released, this,
        &PolynomWidget::onConfirmAdd);

    connect(this, &PolynomWidget::valueChanged, &PolynomWidget::onValueChange);

    setLayout(mainLayout);

    this->onInputConfirm();
}
```

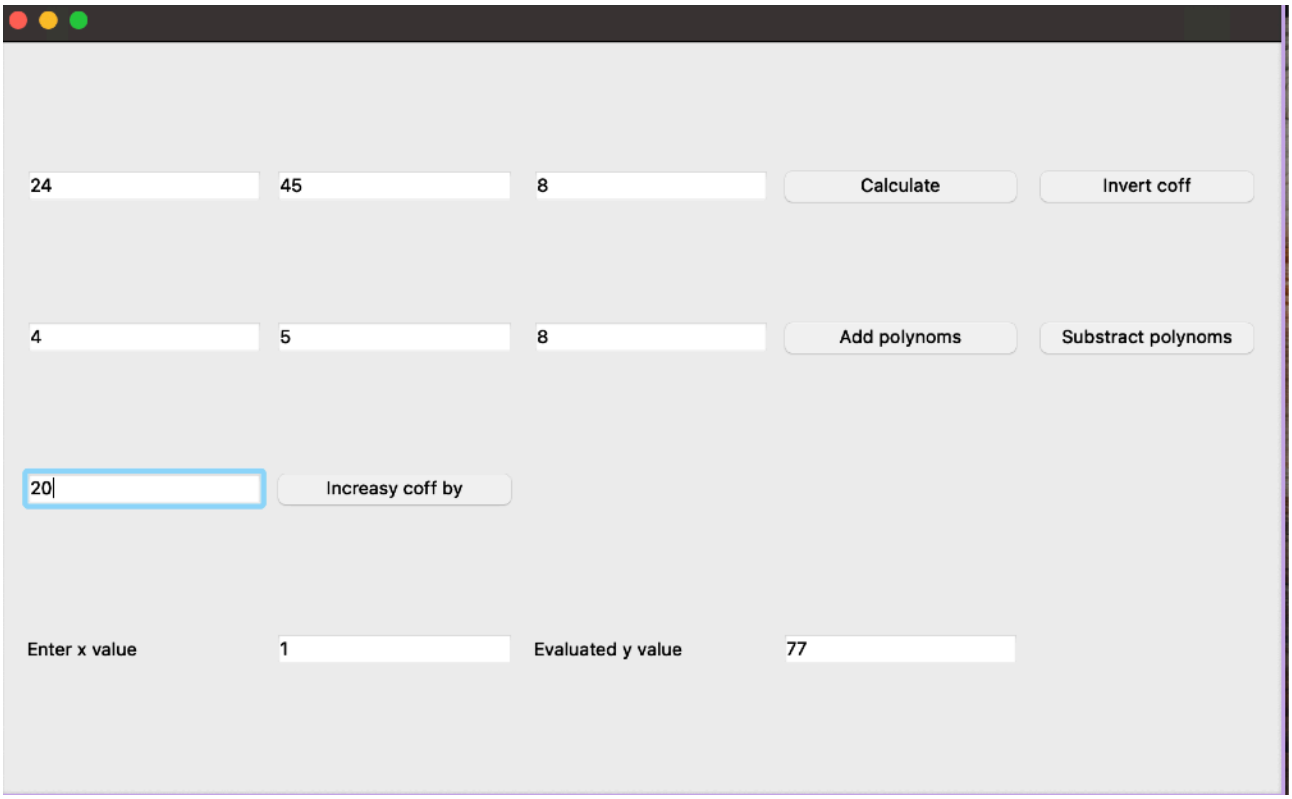

Рис.1. Робота програми

Рис.2. Робота програми

Висновок:

У ході лабораторної роботи №6 я навчився використовувати механізм перевантаження функцій та операцій, створювати та використовувати дружні функції, статичні поля і методи.