Міністерство освіти і науки, молоді та спорту України
Національний університет «Львівська політехніка»
Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

Про виконання лабораторної роботи № 6

«Перевантаження функцій і операцій, дружні функції**»**
з дисципліни «Об'єктно-орієнтоване програмування»

Лектор:
доцент кафедри ПЗ
Коротєєва Т.О.

Виконав:
студ. групи ПЗ-15
Марущак А.С.

Прийняв:
доцент кафедри ПЗ
Яцишин С.І.
«___» _____ 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.

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.

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.

Індивідуальне завдання

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.

Варіант завдання

9. Клас Triangle – трикутник на площині (задаються довжини трьох сторін).

Перевантажити операції, як функції члени:

Збільшення одразу всіх трьох сторін трикутника на константу("+")

Збільшення одразу всіх трьох сторін трикутника у певну кількість разів("*")

Доступ до і-ї сторони трикутника ("[ ]")

Перевантажити оператор приведення трикутника до дійсного типу(повертати його периметр).

Перевантажити операції, як дружні-функції:

Введення трикутника з форми ("<<")

Виведення трикутника на форму (">>")

Більше (">")

Менше ("<")

Рівне ("==") (при порівнянні порівнювати периметри).

Створити статичне поле, в якому б містилась інформація про кількість створених об’єктів, а також статичні функції для роботи з цим полем.

Хід роботи

Виконання роботи розпочнемо зі зміни коду класу трикутника, заданого завданням лабараторної роботи. Опишемо клас наступним чином у заголовочному файлі Triangle.h:

```
class Triangle
{
    double sideA, sideB, sideC;

    static int createdTriangles;

    bool isExist(const double sideA, const double sideB, const double sideC);
    static void increaseCreatedTriangles();

public:
    Triangle();
    Triangle(const double sideA, const double sideB, const double sideC);
    Triangle(const Triangle& other);

    void increaseSidesBy(const double delta);
    void increaseSidesBy(const double deltaA, const double deltaB, const double
deltaC);
    void multiplySidesBy(const double scale);
    void setSides(const double sideA, const double sideB, const double sideC);
    void setSides(const double side);
    void getSides(double &sideA, double &sideB, double &sideC) const;
    void getSides(double &sideA) const;

    double getArea() const;
    void getAngles(double &angleA, double &angleB, double &angleC) const;
    double getPerimeter() const;
    void getMedians(double &medianA, double &medianB, double &medianC) const;
    bool isRectangular() const;

    void applyToForm(TriangleViewer& form);
    void getFromForm(const TriangleViewer& form);

    Triangle operator+(const double delta);
    Triangle operator*(const double scale);
    double& operator[](const char side);
    const double& operator[](const char side) const;
    operator double() const;

    friend const TriangleViewer &operator<<(const TriangleViewer& window, Triangle&
triangle);
    friend TriangleViewer& operator>>(TriangleViewer& window, const Triangle&
triangle);
    friend bool operator>(const Triangle& a, const Triangle& b);
    friend bool operator<(const Triangle& a, const Triangle& b);
    friend bool operator==(const Triangle& a, const Triangle& b);

    static int getCreatedTriangles();


};
```

Як бачимо, тут я задав всі функції, які вимагало від мене завдання. Тепер напишемо реалізацію цих методів у файлі Triangle.cpp:

```cpp
#include "triangle.h"
#include "triangleviewer.h"

int Triangle::createdTriangles = 0;

bool Triangle::isExist(const double sideA, const double sideB, const double sideC)
{
    return (sideA + sideB > sideC) && (sideA + sideC > sideB) && (sideB + sideC > sideA) && (sideA > 0) && (sideB > 0) && (sideC > 0);
}

void Triangle::increaseCreatedTriangles()
{
    Triangle::createdTriangles++;
}

Triangle::Triangle() : sideA(1), sideB(1), sideC(1){increaseCreatedTriangles();}

Triangle::Triangle(const double sideA, const double sideB, const double sideC)
{
    if(isExist(sideA, sideB, sideC))
    {
        this->sideA = sideA;
        this->sideB = sideB;
        this->sideC = sideC;
    }
    else
    {
        this->sideA = this->sideB = this->sideC = 1;
    }
    increaseCreatedTriangles();
}

Triangle::Triangle(const Triangle &other) : sideA(other.sideA), sideB(other.sideB), sideC(other.sideC){increaseCreatedTriangles();}

void Triangle::increaseSidesBy(const double delta)
{
    if(isExist(sideA + delta, sideB + delta, sideC + delta))
    {
        sideA += delta;
        sideB += delta;
        sideC += delta;
    }
}

void Triangle::increaseSidesBy(const double deltaA, const double deltaB, const double deltaC)
{
    if(isExist(sideA + deltaA, sideB + deltaB, sideC + deltaC))
    {
        sideA += deltaA;
        sideB += deltaB;
        sideC += deltaC;
    }
}

void Triangle::multiplySidesBy(const double scale)
{
    if(scale <= 0) return;
    sideA *= scale;
    sideB *= scale;
    sideC *= scale;
}
```

```cpp
void Triangle::setSides(const double sideA, const double sideB, const double sideC)
{
    if(isExist(sideA, sideB, sideC))
    {
        this->sideA = sideA;
        this->sideB = sideB;
        this->sideC = sideC;
    }
}

void Triangle::setSides(const double side)
{
    if(side > 0)
    {
        this->sideA = side;
        this->sideB = side;
        this->sideC = side;
    }
}

void Triangle::getSides(double &sideA, double &sideB, double &sideC) const
{
    sideA = this->sideA;
    sideB = this->sideB;
    sideC = this->sideC;
}

void Triangle::getSides(double &sideA) const
{
    sideA = this->sideA;
}

double Triangle::getArea() const
{
    const double p = (this->sideA + this->sideB + this->sideC)/2;
    return sqrt(p*(p-sideA)*(p-sideB)*(p-sideC));
}

void Triangle::getAngles(double &angleA, double &angleB, double &angleC) const
{
    angleA = acos((sideB * sideB + sideC * sideC - sideA * sideA)/(2*sideB*sideC));
    angleB = acos((sideA * sideA + sideC * sideC - sideB * sideB)/(2*sideA*sideC));
    angleC = acos((sideA * sideA + sideB * sideB - sideC * sideC)/(2*sideA*sideB));
}

double Triangle::getPerimeter() const
{
    return sideA + sideB + sideC;
}

void Triangle::getMedians(double &medianA, double &medianB, double &medianC) const
{
    medianA = sqrt(2*sideB*sideB + 2*sideC*sideC - sideA*sideA)/2;
    medianB = sqrt(2*sideA*sideA + 2*sideC*sideC - sideB*sideB)/2;
    medianC = sqrt(2*sideA*sideA + 2*sideB*sideB - sideC*sideC)/2;
}

bool Triangle::isRectangular() const
{
    const double eps = 1e-6;
    return (fabs(sideA*sideA + sideB*sideB - sideC*sideC) < eps ||
            fabs(sideA*sideA + sideC*sideC - sideB*sideB) < eps ||
            fabs(sideB*sideB + sideC*sideC - sideA*sideA) < eps);
```

```
}

void Triangle::applyToForm(TriangleViewer &form)
{
    form.setTriangle(*this);
}

void Triangle::getFromForm(const TriangleViewer &form)
{
    Triangle triangle = form.getTriangle();
    setSides(triangle.sideA, triangle.sideB, triangle.sideC);
}

Triangle Triangle::operator+(const double delta)
{
    Triangle result(*this);
    result.increaseSidesBy(delta);

    return result;
}

Triangle Triangle::operator*(const double scale)
{
    Triangle result(*this);
    result.multiplySidesBy(scale);

    return result;
}

double& Triangle::operator[](const char side)
{
    switch (tolower(side)) {
    case 'a':
        return sideA;
    case 'b':
        return sideB;
    case 'c':
        return sideC;
    default:
        return sideA;
    }
}

Triangle::operator double() const
{
    return this->getPerimeter();
}

int Triangle::getCreatedTriangles()
{
    return createdTriangles;
}

const double &Triangle::operator[](const char side) const
{
    switch (tolower(side)) {
    case 'a':
        return sideA;
    case 'b':
        return sideB;
    case 'c':
        return sideC;
    default:
        return sideA;
```

```
    }
}

const TriangleViewer &operator<<(const TriangleViewer& window, Triangle& triangle)
{
    triangle = window.getTriangle();
    return window;
}

TriangleViewer& operator>>(TriangleViewer& window, const Triangle& triangle)
{
    window.setTriangle(triangle);
    return window;
}

bool operator>(const Triangle& a, const Triangle& b)
{
    return a.getPerimeter() > b.getPerimeter();
}

bool operator<(const Triangle& a, const Triangle& b)
{
    return a.getPerimeter() < b.getPerimeter();
}

bool operator==(const Triangle& a, const Triangle& b)
{
    return a.getPerimeter() == b.getPerimeter();
}
```

Ну а тепер реалізуємо віконний додаток для демонстрації можливостей програми:

1. Реалізуємо дизайн форми, він має наступний вигляд:

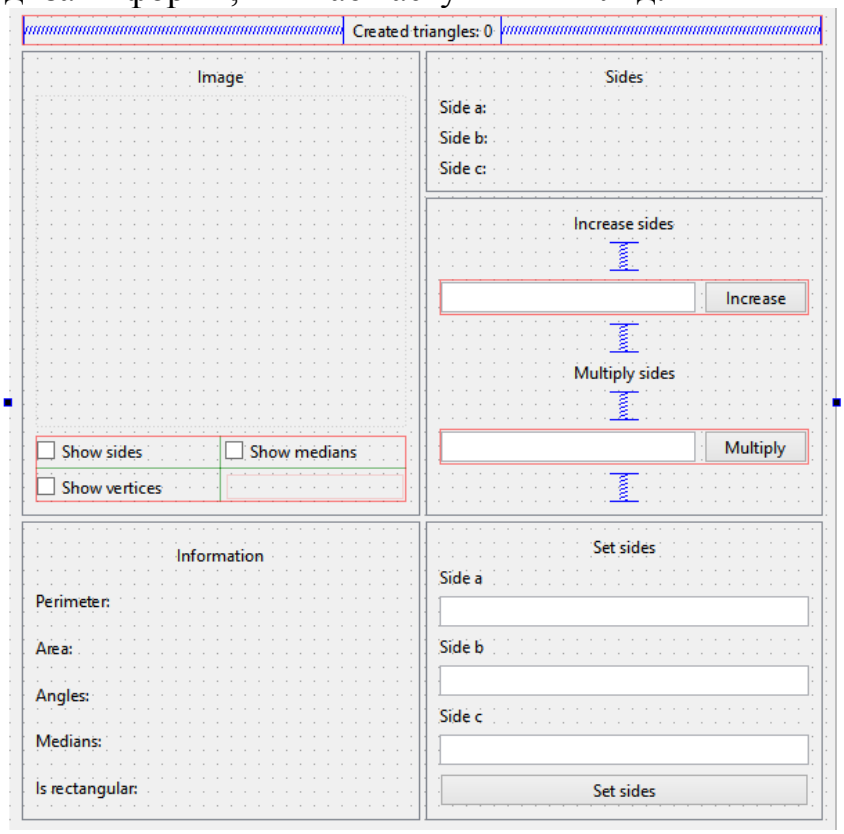

Рис 6.1 Дизайн форми додатку.

2. Створимо дизайн форми діалогу NewTriangleDialog, який згодом будемо використовувати для демонстрації можливості вводу/виводу на форму основного вікна:

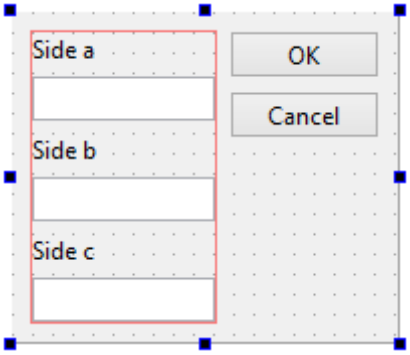

Рис 6.2 Діалог створення трикутника.

3. Створимо дизайн форми вікна ComparasionWindow, який будемо використовувати для демонстрації можлиостей порівняння трикутників:

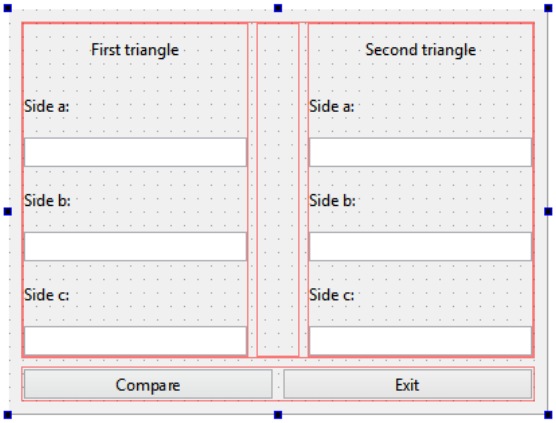

Рис 6.3 Вікно порівняння трикутників.

4. Реалізуємо головне вікно, діалог, вікно порівняння. Нижче я приведу основні фрагменти коду, які безпосередньо взаємодіють з трикутником(в файлі TriangleViewer.h та NewTriangleDialog.h об’явлено поле типу Triangle з назвою triangle) :

triangleviewer.cpp:

```
void TriangleViewer::updateInteraface()
{
    double a = triangle['a'], b = triangle['b'], c = triangle['c'];

    ui->sideALabel->setText(tr("Side a: %1").arg(a));
    ui->sideBLabel->setText(tr("Side b: %1").arg(b));
    ui->sideCLabel->setText(tr("Side c: %1").arg(c));

    ui->perimeterLabel->setText(tr("Perimeter: %1").arg(triangle.getPerimeter()));
    ui->areaLabel->setText(tr("Area: %1").arg(triangle.getArea()));

    double angA, angB, angC;
    triangle.getAngles(angA, angB, angC);
```

```
    ui->anglesLabel->setText(tr("Angles: A: %1°; B: %2°; C: %3°").arg(angA *
(180/M_PI)).arg(angB * (180/M_PI)).arg(angC * (180/M_PI)));

    double medA, medB, medC;
    triangle.getMedians(medA, medB, medC);
    ui->mediansLabel->setText(tr("Medians: Ma: %1; Mb: %2; Mc:
%3").arg(medA).arg(medB).arg(medC));

    ui->rectangularLabel->setText(tr("Is rectangular: %1").arg(triangle.isRectangular()
? "true" : "false"));

    ui->createdTrianglesLabel->setText(tr("Created triangles:
%1").arg(Triangle::getCreatedTriangles()));

    ui->viewWidget->setTriangle(triangle);

    ui->viewWidget->update();
}

void TriangleViewer::on_increaseButton_clicked()
{
    double delta = ui->increaseField->text().toDouble();
    ui->increaseField->setText("");

    triangle = triangle + delta;
    updateInteraface();
}

void TriangleViewer::on_setSidesButton_clicked()
{
    double a = ui->sideAField->text().toDouble();
    double b = ui->sideBField->text().toDouble();
    double c = ui->sideCField->text().toDouble();

    ui->sideAField->setText("");
    ui->sideBField->setText("");
    ui->sideCField->setText("");

    triangle.setSides(a,b,c);
    updateInteraface();

}
void TriangleViewer::on_actionTriangle_triggered()
{
    NewTriangleDialog dialog(this);

    dialog.exec();

}
void TriangleViewer::on_multiplyButton_clicked()
{
    double scale = ui->multiplyField->text().toDouble();
    ui->multiplyField->setText("");

    triangle = triangle * scale;
    updateInteraface();
}

void TriangleViewer::on_actionTriangles_triggered()
{
    delete wind;
    wind = new ComparasionWindow();
```

```
    wind->show();
    updateInteraface();
}
```

newtriangledialog.cpp:

```
#include "newtriangledialog.h"
#include "ui_newtriangledialog.h"
#include "triangleviewer.h"

NewTriangleDialog::NewTriangleDialog(TriangleViewer* viewer, QWidget *parent) :
    QDialog(parent),
    ui(new Ui::NewTriangleDialog)
{
    ui->setupUi(this);
    mainWindow = viewer;

    Triangle triangle;
    *mainWindow << triangle;
    double a, b, c;
    triangle.getSides(a,b,c);

    ui->sideASet->setText(QString::number(a));
    ui->sideBSet->setText(QString::number(b));
    ui->sideCSet->setText(QString::number(c));
}

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

void NewTriangleDialog::on_buttonBox_accepted()
{
    Triangle triangle
            (
                ui->sideASet->text().toDouble(),
                ui->sideBSet->text().toDouble(),
                ui->sideCSet->text().toDouble()
            );

    *mainWindow >> triangle;
}
```

comparasionwindow.cpp

```
#include "comparasionwindow.h"
#include "triangle.h"
#include "ui_comparasionwindow.h"

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

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

void ComparasionWindow::on_compareButton_clicked()
```

```
{
    Triangle a
            (
            ui->sideALabel->text().toDouble(),
            ui->sideBLabel->text().toDouble(),
            ui->sideCLabel->text().toDouble()
            );

    Triangle b
            (
            ui->sideA2Label->text().toDouble(),
            ui->sideB2Label->text().toDouble(),
            ui->sideC2Label->text().toDouble()
            );

    if(a > b) ui->signLabel->setText(">");
    else if(a < b) ui->signLabel->setText("<");
    else if(a == b) ui->signLabel->setText("=");
}


void ComparasionWindow::on_exitButton_clicked()
{
    this->close();
}
```

Також, програма оперує створеним мною віджетом ViewWidget, який малює переданий йому трикутник.

Результат виконання:

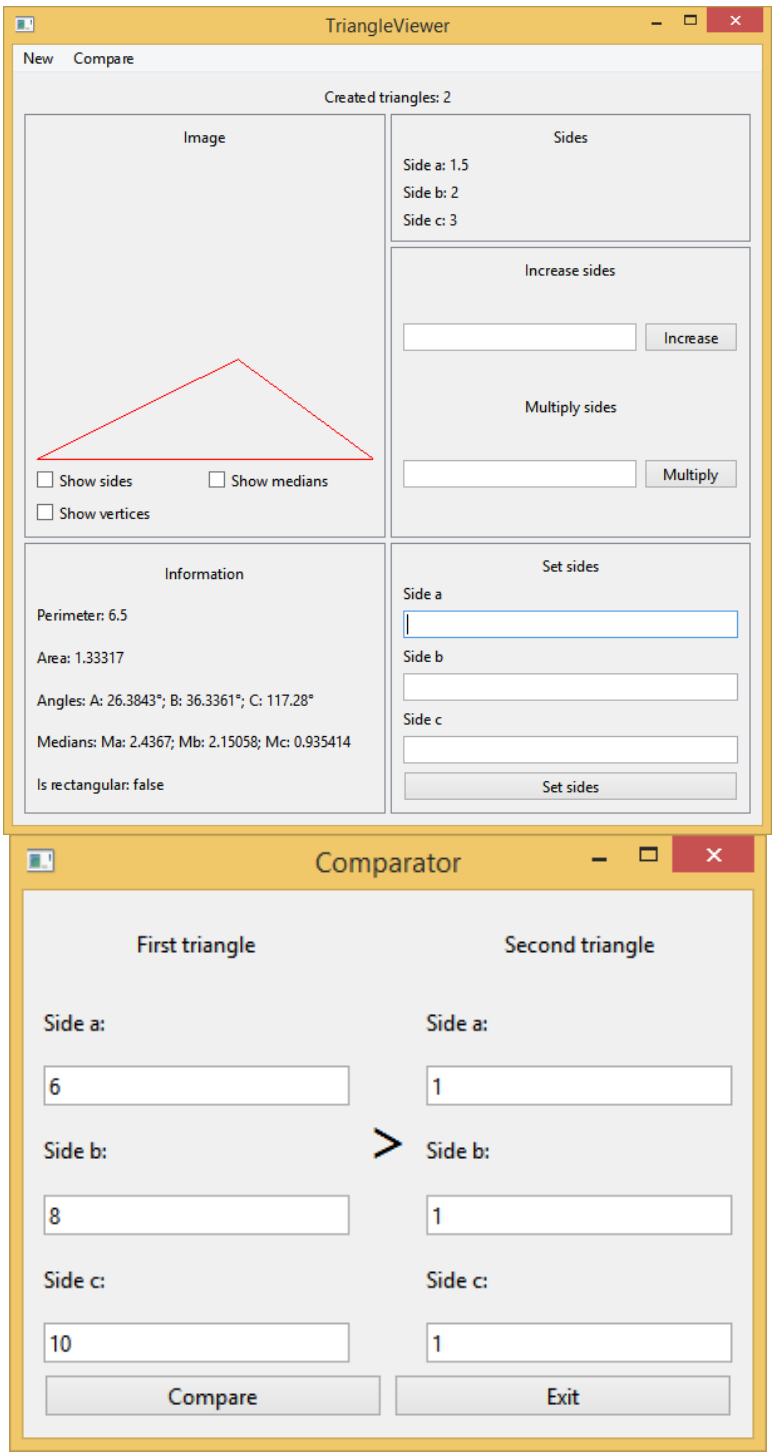

Рис 6.4-5 Результат виконання програми.

Висновок: виконавши лабораторну роботу №6, ми навчилися іперевантажувати функції та оператори. Також, ми навчилися створювати та використовувати дружні функції. При тому, ми ознайомилися зі статичними полями та методами і також навчилися їх використовувати. На основі отриманих знань ми створили додаток для перегляду трикутників. Робота була дуже плідною.