МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №1

по курсу объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент *Васютинский Вадим Александрович М80-208Б-20*
Преподаватель *Дорохов Евгений Павлович*

Условие

Задание: Вариант 3: Прямоугольник Ромб, Трапеция. Необходимо спроектировать и запрограммировать на языке C++ классы трех фигур, согласно варианту задания. Классы должны удовлетворять следующим правилам:

1. Должны быть названы также, как в вариантах задания и расположенны в раздельных файлах: отдельно заголовки (имя_класса_с_маленькой_буквы.h), отдельно описание методов (имя_класса_с_маленькой_буквы.cpp).
2. Иметь общий родительский класс Figure;
3. Содержать конструктор, принимающий координаты вершин фигуры из стандарт- ного потока std::cin, расположенных через пробел. Пример: "0.0 0.0 1.0 0.0 1.0 1.0 0.0 1.0"
4. Содержать набор общих методов:
    - size_t VertexesNumber() - метод, возвращающий количество вершин фигуры;
    - double Area() - метод расчета площади фигуры;
    - void Print(std::ostream os) - метод печати типа фигуры и ее координат вершин в поток вывода os в формате: "Rectangle: (0.0, 0.0) (1.0, 0.0) (1.0, 1.0) (0.0, 1.0)"с переводом строки в конце.

Описание программы

Исходный код лежит в 10 файлах:

1. main.cpp: основная программа, взаимодействие с пользователем посредством команд из меню
2. include/figure.h: описание абстрактного класса фигур
3. include/point.h: описание класса точки
4. include/rectangle.h: описание класса прямоугольника,

наследующегося от figures

5. include/rhombus.h: описание класса ромба, наследующегося от figures
6. include/trapezoid.h: описание класса трапеции, наследующегося от

figures 7. include/point.cpp: реализация класса точки

8. include/rectangle.cpp: реализация класса прямоугольника, наследующегося от figures
9. include/rhombus.cpp: реализация класса ромба, наследующегося от figures

10. include/trapezoid.cpp: реализация класса трапеции, наследующегося от figure

Дневник отладки

Во время выполнения лабораторной работы были выявлены ошибки в рассчёте площади трапеции. После их исправления программа работала так, как было задумано изначально.

Недочеты

Из недочётов невозможно не отметить отсутствие проверки на то, что заданные точки определяют именно необходимую фигуру.

Выводы:

Основная цель лабораторной работы №3 - знакомство с парадигмой объектно-ориентированного программирования на языке C++. Могу сказать, что справился с этой целью весьма успешно: усвоил “3 китов ООП”: полиморфизм, наследование, инкапсуляция, освоил базовые понятия ООП, такие как классы, методы, конструкторы, деструкторы… Ознакомился с ключевыми словами virtual, friend, private, public... Повторил тему “директивы условной компиляции”, “перегрузка функций/операторов”, работа со стандартными потоками ввода-вывода. **Лабораторная работа №3 прошла для меня успешно.**

Исходный код

figure.h

```
#ifndef LAB1_FIGURE_H

#define LAB1_FIGURE_H



#include <cmath>

#include <iostream>

#include "point.h"



class Figure {

public:

    virtual size_t VertexesNumber() = 0;

    virtual double Area() = 0;

    virtual void Print(std::ostream &os) = 0;
```

```
    virtual ~Figure() {};

};

#endif //LAB1_FIGURE_H
```

point.h

```
#ifndef LAB1_POINT_H
#define LAB1_POINT_H

#include <iostream>

class Point {
public:
    Point();
    Point(std::istream &is);
    Point(double x, double y);

    double dist(Point& other);

    friend std::istream& operator>>(std::istream& is, Point& p);
    friend std::ostream& operator<<(std::ostream& os, Point& p);

private:
    double x_;
    double y_;
};

#endif //LAB1_POINT_H
```

point.cpp

```
#include "point.h"

#include <cmath>

Point::Point() : x_(0.0), y_(0.0) {}
```

```
Point::Point(double x, double y) : x_(x), y_(y) {}

Point::Point(std::istream &is) {
   is >> x_ >> y_;
}

double Point::dist(Point& other) {
   double dx = (other.x_ - x_);
   double dy = (other.y_ - y_);
   return std::sqrt(dx*dx + dy*dy);
}

std::istream& operator>>(std::istream& is, Point& p) {
   is >> p.x_ >> p.y_;
   return is;
}

std::ostream& operator<<(std::ostream& os, Point& p) {
   os << "(" << p.x_ << ", " << p.y_ << ")";
   return os;
}
```

rectangle.h

```
#ifndef OOP1_RECTANGLE_H
#define OOP1_RECTANGLE_H


#include "figure.h"
class Rectangle : Figure {
public:
    size_t VertexesNumber() override;
    double Area() override;
    void Print(std::ostream &os) override;


    Rectangle();
    Rectangle(Point a, Point b, Point c, Point d);
    Rectangle(std::istream &is);
```

```
    Rectangle(const Rectangle &other);
    virtual ~Rectangle();
private:
    Point a, b, c, d;
};
#endif //OOP1_RECTANGLE_H
```

rectangle.cpp

#ifndef OOP1_RECTANGLE_H

#define OOP1_RECTANGLE_H

#include "figure.h"

class Rectangle : Figure {

public:

size_t VertexesNumber() override;

double Area() override;

void Print(std::ostream &os) override;

Rectangle();

Rectangle(Point a, Point b, Point c, Point d);

Rectangle(std::istream &is);

Rectangle(const Rectangle &other);

virtual ~Rectangle();

private:

Point a, b, c, d;

};

#endif //OOP1_RECTANGLE_H

trapezoid.h

```
#ifndef OOP1_TRAPEZOID_H

#define OOP1_TRAPEZOID_H



#include "figure.h"



class Trapezoid : Figure {

public:

    size_t VertexesNumber() override;

    double Area() override;

    void Print(std::ostream &os) override;
```

```
    Trapezoid();

    Trapezoid(Point a, Point b, Point c, Point d);

    Trapezoid(std::istream &is);

    Trapezoid(const Trapezoid &other);

    virtual ~Trapezoid();

private:

    Point a, b, c, d;

};




#endif //OOP1_TRAPEZOID_H
```

trapezoid.cpp

```
#include "trapezoid.h"



size_t Trapezoid::VertexesNumber() {

    return 4;

}



double Trapezoid::Area() {

    double A = a.dist(b);

    double B = c.dist(d);

    double C = a.dist(d);

    double D = b.dist(c);
```

```
    double S = (A + B) / 2 * sqrt(C*C - ((B-A)*(B-A) + C*C -
D*D)/(2*(B-A))*((B-A)*(B-A) + C*C - D*D)/(2*(B-A)));

    return S;

}

void Trapezoid::Print(std::ostream &os) {

    os << "Trapezoid: " << a << b << c << d << std::endl;

}

Trapezoid::Trapezoid() {}

Trapezoid::Trapezoid(Point a, Point b, Point c, Point d) : a(a), b(b), c(c),
d(d) {}
```

```
Trapezoid::Trapezoid(std::istream &is) {

    std::cout << "Enter data:" << std::endl;

    is >> a >> b >> c >> d;

    std::cout << "Trapezoid created via istream" << std::endl;

}



Trapezoid::Trapezoid(const Trapezoid &other) : Trapezoid(other.a,
other.b, other.c, other.d) {

    std::cout << "Made copy of Trapezoid" << std::endl;

}



Trapezoid::~Trapezoid() {
```

```
    std::cout << "Trapezoid deleted" << std::endl;

}
```

rhombus.h

```
#ifndef OOP1_RHOMBUS_H

#define OOP1_RHOMBUS_H



#include "figure.h"

class Rhombus : Figure {

public:

    size_t VertexesNumber() override;

    double Area() override;

    void Print(std::ostream &os) override;
```

```
    Rhombus();

    Rhombus(Point a, Point b, Point c, Point d);

    Rhombus(std::istream &is);

    Rhombus(const Rhombus &other);

    virtual ~Rhombus();

private:

    Point a, b, c, d;

};



#endif //OOP1_RHOMBUS_H
```

rhombus.cpp

```
#include "rhombus.h"



size_t Rhombus::VertexesNumber() {

    return 4;

}



double Rhombus::Area() {

    return a.dist(c) * b.dist(d) / 2;

}



void Rhombus::Print(std::ostream &os) {

    os << "Rhombus: " << a << b << c << d << std::endl;
```

```
}



Rhombus::Rhombus() {}



Rhombus::Rhombus(Point a, Point b, Point c, Point d) : a(a), b(b), c(c),
d(d) {}



Rhombus::Rhombus(std::istream &is) {

    std::cout << "Enter data:" << std::endl;

    is >> a >> b >> c >> d;

    std::cout << "Rhombus created via istream" << std::endl;

}
```

Rhombus::Rhombus(const Rhombus &other) : Rhombus(other.a, other.b, other.c, other.d) {

    std::cout << "Made copy of Rhombus" << std::endl;

}

Rhombus::~Rhombus() {

    std::cout << "Rhombus deleted" << std::endl;

}

main.cpp

```
#include "rectangle.h"
#include "trapezoid.h"
#include "rhombus.h"
```

```
int main() {

    Rectangle re(std::cin);

    re.Print(std::cout);

    std::cout << "Number of vertex is " << re.VertexesNumber() <<
std::endl;

    std::cout << "Area is " << re.Area() << std::endl;

    std::cout << "---------------------------" << std::endl;

    Trapezoid t(std::cin);

    t.Print(std::cout);

    std::cout << "Number of vertex is " << t.VertexesNumber() <<
std::endl;

    std::cout << "Area is " << t.Area() << std::endl;

    std::cout << "---------------------------" << std::endl;

    Rhombus rh(std::cin);

    re.Print(std::cout);

    std::cout << "Number of vertex is " << rh.VertexesNumber() <<
std::endl;

    std::cout << "Area is " << rh.Area() << std::endl;

}
```