

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ
ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №3
по курсу “Объектно-ориентированное программирование”
III семестр, 2021/22 учебный год

Выполнила студентка группы М8О-208Б-20
Шатунова Юлия Викторовна
Преподаватель: Дорохов Евгений Павлович

Москва, 2021

Постановка задачи

Вариант 26: квадрат, прямоугольник, трапеция. Необходимо спроектировать и запрограммировать на языке C++ классы трех фигур согласно варианту задания. Классы должны удовлетворять следующим правилам:

1. должны быть названы так же, как в вариантах задания, и расположены в отдельных 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 стандартного потока std::cin, расположенные через пробел (пример: "0.0 0.0 1.0 0.0 1.0 1.0 0.0 1.0");
4. содержать набор общих методов:
 - 4.1. size_t VertexesNumber() - метод, возвращающий количество вершин фигуры;
 - 4.2. double Area() - метод расчета площади фигуры;
 - 4.3. void Print(std::ostream& os) - метод печати типа фигуры и ее координат вершин в поток вывода os в формате: "Square: (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. figure.h: описание абстрактного класса фигур
2. point.h: описание класса точки
3. square.h: описание класса квадрата, наследующегося от Figure
4. rectangle.h: описание класса прямоугольника, наследующегося от Figure
5. trapezoid.h: описание класса трапеции, наследующегося от Figure
6. point.cpp: реализация класса точки
7. square.cpp: реализация класса квадрата, наследующегося от Figure
8. rectangle.cpp: реализация класса прямоугольника, наследующегося от Figure
9. trapezoid.cpp: реализация класса трапеции, наследующегося от Figure
10. main.cpp: основная программа

Дневник отладки

Во время выполнения лабораторной работы программа не нуждалась в отладке, все ошибки компиляции были исправлены с первой попытки. После их исправления программа работала так, как было задумано изначально.

Недочеты

Во время выполнения лабораторной работы недочетов в программе обнаружено не было.

Выводы

Основная цель лабораторной работы №3 - знакомство с парадигмой объектно-ориентированного программирования (ООП) на языке C++. В ходе выполнения данной лабораторной работы я познакомилась с такими фундаментальными понятиями, как полиморфизм, наследование, инкапсуляция. Были освоены такие базовые понятия, как классы, методы, конструкторы, деструкторы. Ознакомилась с ключевыми словами virtual, friend, private, public.

Исходный код

figure.h

```
#ifndef FIGURE_H
```

```
#define FIGURE_H
```

```
#include <ostream>
```

```
#include "point.h"
```

```
class Figure {
```

```
public:
```

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

```
    virtual double Area() = 0;
```

```
    virtual ~Figure() {};
```

```
};
```

```
#endif // FIGURE_H
```

main.cpp

```
#include "square.h"
```

```
#include "rectangle.h"
```

```
#include "trapezoid.h"
```

```
int main(int argc, char** argv) {
```

```
    std::cout << "DEFAULT FIGURES" << std::endl;
```

```
    std::cout << std::endl;
```

```
    Square sq_default;
```

```
    sq_default.Print(std::cout);
```

```
    Rectangle re_default;
```

```
    re_default.Print(std::cout);
```

```
    Trapezoid tr_default;
```

```
    tr_default.Print(std::cout);
```

```
    std::cout << std::endl;
```

```
    std::cout << "SET VALUES FIGURES" << std::endl;
```

```
    std::cout << std::endl;
```

```
    Point a(2.0, 1.0);
```

```
    Point b(2.0, 3.0);
```

```
    Point c(4.0, 3.0);
```

```
    Point d(4.0, 1.0);
```

```
    Point e(5.0, 3.0);
```

```
    Point f(5.0, 1.0);
```

```
    Square sq_values(a, b, c, d);
```

```
    std::cout << "Number of square's vertexes: " << sq_values.VertexesNumber() << std::endl;
```

```

std::cout << "Square's area: " << sq_values.Area() << std::endl;
sq_values.Print(std::cout);

Rectangle re_values(a, b, e, f);
std::cout << "Number of rectangle's vertexes: " << re_values.VertexesNumber() << std::endl;
std::cout << "Rectangle's area: " << re_values.Area() << std::endl;
re_values.Print(std::cout);

Trapezoid tr_values(a, c, e, f);
std::cout << "Number of trapezoid's vertexes: " << tr_values.VertexesNumber() << std::endl;
std::cout << "Trapezoid's square: " << tr_values.Area() << std::endl;
tr_values.Print(std::cout);

std::cout << std::endl;
std::cout << "CIN FIGURES" << std::endl;
std::cout << std::endl;

Square sq_cin(std::cin);
std::cout << "Number of square's vertexes: " << sq_cin.VertexesNumber() << std::endl;
std::cout << "Square's area: " << sq_cin.Area() << std::endl;
sq_cin.Print(std::cout);

Rectangle re_cin(std::cin);
std::cout << "Number of rectangle's vertexes: " << re_cin.VertexesNumber() << std::endl;
std::cout << "Rectangle's area: " << re_cin.Area() << std::endl;
re_cin.Print(std::cout);

Trapezoid tr_cin(std::cin);
std::cout << "Number of trapezoid's vertexes: " << tr_cin.VertexesNumber() << std::endl;
std::cout << "Trapezoid's square: " << tr_cin.Area() << std::endl;
tr_cin.Print(std::cout);

std::cout << std::endl;
std::cout << "COPIED FIGURES" << std::endl;
std::cout << std::endl;

Square sq_copy = sq_values;
sq_copy.Print(std::cout);

Rectangle re_copy = re_values;
re_copy.Print(std::cout);

Trapezoid tr_copy = tr_values;
tr_copy.Print(std::cout);

std::cout << std::endl;
std::cout << "POINTER FIGURES" << std::endl;
std::cout << std::endl;

Figure* sq_ptr = new Square(sq_cin);
sq_ptr->Print(std::cout);
std::cout << "Square's area: " << sq_ptr->Area() << std::endl;

Figure* re_ptr = new Rectangle(re_cin);
re_ptr->Print(std::cout);
std::cout << "Rectangle's area: " << re_ptr->Area() << std::endl;

Figure* tr_ptr = new Trapezoid(tr_cin);
tr_ptr->Print(std::cout);
std::cout << "Trapezoid's area: " << tr_ptr->Area() << std::endl;

delete sq_ptr;

```

```

        delete re_ptr;
        delete tr_ptr;

        return 0;
}

point.cpp
#include "point.h"

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;
}

point.h
#ifndef POINT_H
#define POINT_H

#include <iostream>
#include <vector>
#include <cmath>

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);

    friend class Square;
    friend class Rectangle;
    friend class Trapezoid;

private:
    double x_;
    double y_;
};

#endif // POINT_H#pragma once

```

```

rectangle.cpp
#include "rectangle.h"

Rectangle::Rectangle() : point_a(0.0, 0.0), point_b(0.0, 0.0), point_c(0.0, 0.0), point_d(0.0, 0.0) {
    std::cout << "Default rectangle is created" << std::endl;
}

Rectangle::Rectangle(Point a, Point b, Point c, Point d) : point_a(a), point_b(b), point_c(c), point_d(d) {
    std::cout << "Rectangle is created with vertices: ";
    std::cout << point_a << ", ";
    std::cout << point_b << ", ";
    std::cout << point_c << ", ";
    std::cout << point_d << std::endl;
}

Rectangle::Rectangle(std::istream& is) {
    is >> point_a >> point_b >> point_c >> point_d;
}

Rectangle::Rectangle(const Rectangle& other) : Rectangle(other.point_a, other.point_b, other.point_c,
other.point_d) {
    std::cout << "Rectangle's copy is created" << std::endl;
}

size_t Rectangle::VertexesNumber() {
    int vert_num = 4;
    return vert_num;
}

double Rectangle::Area() {
    double side_a = 0.0;
    double side_b = 0.0;
    double fig_square = 0.0;
    side_a = point_b.dist(point_a);
    side_b = point_c.dist(point_b);
    fig_square = side_a * side_b;
    return fig_square;
}

void Rectangle::Print(std::ostream& os) {
    os << "Rectangle: ";
    os << point_a << " ";
    os << point_b << " ";
    os << point_c << " ";
    os << point_d << std::endl;
}

Rectangle::~~Rectangle() {
    std::cout << "Rectangle is deleted" << std::endl;
}

rectangle.h
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include <iostream>

#include "figure.h"

class Rectangle : public Figure {
public:

```

```

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

    size_t VertexesNumber();
    double Area();
    void Print(std::ostream& os);

    virtual ~Rectangle();

private:
    Point point_a; // lower left corner, then clockwise
    Point point_b;
    Point point_c;
    Point point_d;
};

#endif // RECTANGLE_H

square.cpp
#include "square.h"

Square::Square() : point_a(0.0, 0.0), point_b(0.0, 0.0), point_c(0.0, 0.0), point_d(0.0, 0.0) {
    std::cout << "Default square is created" << std::endl;
}

Square::Square(Point a, Point b, Point c, Point d) : point_a(a), point_b(b), point_c(c), point_d(d) {
    std::cout << "Square is created with vertices: ";
    std::cout << point_a << ", ";
    std::cout << point_b << ", ";
    std::cout << point_c << ", ";
    std::cout << point_d << std::endl;
}

Square::Square(std::istream& is) {
    is >> point_a >> point_b >> point_c >> point_d;
}

Square::Square(const Square& other) : Square(other.point_a, other.point_b, other.point_c, other.point_d) {
    std::cout << "Square's copy is created" << std::endl;
}

size_t Square::VertexesNumber() {
    int vert_num = 4;
    return vert_num;
}

double Square::Area() {
    double side = 0.0;
    double fig_square = 0.0;
    side = point_b.dist(point_a);
    fig_square = side * side;
    return fig_square;
}

void Square::Print(std::ostream& os) {
    os << "Square: ";
    os << point_a << " ";
    os << point_b << " ";
    os << point_c << " ";
    os << point_d << std::endl;
}

```

```

}

Square::~~Square() {
    std::cout << "Square is deleted" << std::endl;
}

square.h
#ifndef SQUARE_H
#define SQUARE_H

#include <iostream>

#include "figure.h"

class Square : public Figure {
public:
    Square();
    Square(Point a, Point b, Point c, Point d);
    Square(std::istream& is);
    Square(const Square& other);

    size_t VertexesNumber();
    double Area(); // площадь
    void Print(std::ostream& os);

    virtual ~Square();

private:
    Point point_a; // lower left corner, then clockwise
    Point point_b;
    Point point_c;
    Point point_d;
};

#endif // SQUARE_H

trapezoid.cpp
#include "trapezoid.h"

Trapezoid::Trapezoid() : point_a(0.0, 0.0), point_b(0.0, 0.0), point_c(0.0, 0.0), point_d(0.0, 0.0) {
    std::cout << "Default trapezoid is created" << std::endl;
}

Trapezoid::Trapezoid(Point a, Point b, Point c, Point d) : point_a(a), point_b(b), point_c(c), point_d(d) {
    std::cout << "Trapezoid is created with vertices: ";
    std::cout << point_a << ", ";
    std::cout << point_b << ", ";
    std::cout << point_c << ", ";
    std::cout << point_d << std::endl;
}

Trapezoid::Trapezoid(std::istream& is) {
    is >> point_a >> point_b >> point_c >> point_d;
}

Trapezoid::Trapezoid(const Trapezoid& other) : Trapezoid(other.point_a, other.point_b, other.point_c,
other.point_d) {
    std::cout << "Trapezoid's copy is created" << std::endl;
}

size_t Trapezoid::VertexesNumber() {
    int vert_num = 4;

```



```

        return vert_num;
    }

double Trapezoid::Area() {
    double fig_square = 0.0;
    double side_a = 0.0;
    double side_b = 0.0;
    double height = 0.0;
    if (point_a.y_ == point_d.y_ && point_b.y_ == point_c.y_) {
        side_a = point_d.dist(point_a);
        side_b = point_c.dist(point_b);
        height = point_b.y_ - point_a.y_;
    }
    else {
        side_a = point_b.dist(point_a);
        side_b = point_c.dist(point_d);
        height = point_c.x_ - point_b.x_;
    }
    fig_square = 0.5 * (side_a + side_b) * height;
    return fig_square;
}

```

```

void Trapezoid::Print(std::ostream& os) {
    os << "Trapezoid: ";
    os << point_a << " ";
    os << point_b << " ";
    os << point_c << " ";
    os << point_d << std::endl;
}

```

```

Trapezoid::~Trapezoid() {
    std::cout << "Trapezoid is deleted" << std::endl;
}

```

```

trapezoid.h
#ifndef TRAPEZOID_H
#define TRAPEZOID_H

```

```

#include <iostream>

```

```

#include "figure.h"

```

```

class Trapezoid : public Figure {
public:
    Trapezoid();
    Trapezoid(Point a, Point b, Point c, Point d);
    Trapezoid(std::istream& is);
    Trapezoid(const Trapezoid& other);

    size_t VertexesNumber();
    double Area(); // площадь
    void Print(std::ostream& os);

    virtual ~Trapezoid();

private:
    Point point_a; // lower left corner, then clockwise
    Point point_b;
    Point point_c;
    Point point_d;
};

```

```
#endif // TRAPEZOID_H
```