

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

ЛАБОРАТОРНАЯ РАБОТА №3

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

Студент Клитная Анастасия Викторововна, группа М80-208Б-20

Преподаватель Дорохов Евгений Павлович

Условие

Задание: Вариант 9: Восьмиугольник, Квадрат, Треугольник. Необходимо спроектировать и запрограммировать на языке С++ классы трех фигур, согласно варианту задания. Классы должны удовлетворять следующим правилам:

1. Должны быть названы также, 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`, 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`

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/octagon.h`: описание класса восьмиугольника, наследующегося от `figures`
5. `include/square.h`: описание класса квадрата, наследующегося от `figures`
6. `include/triangle.h`: описание класса треугольника, наследующегося от `figures`
7. `include/point.cpp`: реализация класса точки
8. `include/octagon.cpp`: реализация класса восьмиугольника, наследующегося от `figures`
9. `include/square.cpp`: реализация класса квадрата, наследующегося от `figures`
10. `include/triangle.cpp`: реализация класса треугольника, наследующегося от `figure`

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 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 FIGURE_H
#define FIGURE_H

#include "point.h"

class Figure {
public:
    virtual double Area() = 0;
    virtual void Print(std::ostream &os) = 0;
    virtual size_t VertexesNumber() = 0;
    virtual ~Figure() {};
};
```

```
#endif
```

point.h

```
#ifndef POINT_H
```

```
#define POINT_H
```

```
#include <iostream>
```

```
class Point {
```

```
public:
```

```
    Point();
```

```
    Point(std::istream &is);
```

```
    Point(double x, double y);
```

```
    double X();
```

```
    double Y();
```

```
    friend std::istream& operator>>(std::istream& is, Point& p);
```

```
    friend std::ostream& operator<<(std::ostream& os, Point& p);
```

```
private:
```

```
    double x_;
```

```
    double y_;
```

```
};
```

```
#endif
```

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::X() {  
    return x_;  
};
```

```
double Point::Y() {  
    return y_;  
};
```

```
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;  
}
```

pentagon.h

```
#ifndef PENTAGON_H  
#define PENTAGON_H  
  
#include "figure.h"  
#include <iostream>
```

```
class Pentagon : public Figure {
```

```

public:
    Pentagon(std::istream& InputStream);

    virtual ~Pentagon();

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

private:
    Point a;

    Point b;

    Point c;
    Point d;

    Point e;
};
#endif

```

Octagon.cpp

```

#include "octagon.h"
#include <cmath>

Octagon::Octagon(std::istream& is) {
    std::cout << "Enter the octagon's vertexes:" << std::endl;
    is >> point_a;
    is >> point_b;
    is >> point_c;
    is >> point_d;
    is >> point_e;
    is >> point_f;
    is >> point_g;
    is >> point_h;

    std::cout << "The octagon is created" << std::endl;
}

```

```

void Octagon::Print(std::ostream& os) {
    std::cout << "Octagon: ";
    std::cout << point_a << ", ";
    std::cout << point_b << ", ";
    std::cout << point_c << ", ";
    std::cout << point_d << ", ";
    std::cout << point_e << ", ";
    std::cout << point_f << ", ";
    std::cout << point_g << ", ";
    std::cout << point_h << std::endl;
}

size_t Octagon::VertexesNumber() {
    size_t number = 8;
    return number;
}

double Octagon::Area() {
    double q = abs(point_a.X() * point_b.Y() + point_b.X() * point_c.Y() +
point_c.X() * point_d.Y() + point_d.X() * point_e.Y() + point_e.X() *
point_f.Y() + point_f.X() * point_g.Y() + point_g.X() * point_h.Y() +
point_h.X() * point_a.Y() - point_b.X() * point_a.Y() - point_c.X() *
point_b.Y() - point_d.X() * point_c.Y() - point_e.X() * point_d.Y() -
point_f.X() * point_e.Y() - point_g.X() * point_f.Y() - point_h.X() *
point_g.Y() - point_a.X() * point_h.Y());

    double s = q / 2;

    return s;
}

Octagon::~~Octagon() {
    std::cout << "The octagon is deleted" << std::endl;
}

```

Octagon.h

```

#ifndef OCTAGON_H
#define OCTAGON_H

#include "figure.h"

class Octagon : public Figure{
public:
    Octagon(std::istream& is);

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

private:
    Point point_a, point_b, point_c, point_d, point_e,
    point_f, point_g, point_h, ;
};

#endif // OCTAGON_H

```

Square.cpp

```

#include "square.h"

Square::Square(std::istream& is) {
    std::cout << "Enter the square's vertexes:" <<
    std::endl;

    is >> point_a;
    is >> point_b;

```



```

        is >> point_c;
        is >> point_d;
        std::cout << "The square is created" << std::endl;
    }

```

```

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

```

```

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

```

```

double Square::Area() {
    try {
        double len_a = point_b.dist(point_a);
        if (len_a < 0) {
            throw "The length of the side is < 0";
        }
        return pow(len_a, 2);
    }
    catch (const char* exception) {
        std::cerr << "Error: " << exception <<

```

```

std::endl;
    }
}

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

Square.h
#ifndef SQUARE_H
#define SQUARE_H

#include "figure.h"

class Square : public Figure {
public:
    Square(std::istream& is);

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

private:
    Point point_a, point_b, point_c, point_d; //ïâðâàÿ
    ââðøèà - ëääâÿ íèæíÿÿ, äèëää - ïî ÷àñîâîé ñòðåëå
};

#endif // SQUARE_H

```

Triangle.cpp

```
#include "triangle.h"
```

```
Triangle::Triangle(std::istream& is) {  
    std::cout << "Enter the triangle's vertexes:" <<  
std::endl;  
    is >> point_a;  
    is >> point_b;  
    is >> point_c;  
    std::cout << "The triangle is created" <<  
std::endl;  
}
```

```
void Triangle::Print(std::ostream& os) {  
    std::cout << "Triangle: ";  
    std::cout << point_a << ", ";  
    std::cout << point_b << ", ";  
    std::cout << point_c << std::endl;  
}
```

```
size_t Triangle::VertexesNumber() {  
    size_t number = 3;  
    return number;  
}
```

```
double Trapezoid::Area() {  
    double p = 0.5*abs((point_b.X()-  
point_a.X())*(point_c.Y()-point_a.Y())-(point_c.X()-  
point_a.X())*(point_b.Y()-point_a.Y())));  
}
```

```

        return p;

    }

Triangle::~~Triangle() {
    std::cout << "The triangle is deleted" <<
std::endl;
}

Triangle.h
#ifndef TRIANGLE_H
#define TRIANGLE_H

#include "figure.h"

class Triangle : public Figure{
public:
    Triangle(std::istream& is);

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

private:
    Point point_a, point_b, point_c;
};

#endif // TRIANGLE_H

```

