

**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 Прохоров Данила Михайлович, группа М80-208Б-20  
Преподаватель Дорохов Евгений Павлович

## Условие

Задание:

**Вариант 21:** 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 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 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

Так как сама программа не была сложной, то и не нуждалась в отладке. Всё получилось с первого раза.

## 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++. Реализовал на практике идеи полиморфизма, наследования и инкапсуляции в ООП, точно как и классы, конструкторы и деструкторы.

**Лабораторная работа №3 прошла для меня успешно.**  
**Исходный код**

## figure.h

```
#ifndef POINT_H
#define POINT_H
#include <iostream>
#include <cmath>

class Point
{
public:
    Point();
    Point(std::istream& is);
    Point(double x, double y);
    double length(Point& p1, Point& p2);
    friend std::istream& operator>>(std::istream& is, Point& p);
    friend std::ostream& operator<<(std::ostream& os, Point& p);
    friend double dist(Point& p1, Point& p2);
};
```

```
private:
    double x_, y_;
};

#endif
```

## point.h

```
#ifndef POINT_H
#define POINT_H
#include <iostream>
#include <cmath>

class Point
{
public:
    Point();
    Point(std::istream& is);
    Point(double x, double y);
    double length(Point& p1, Point& p2);
    friend std::istream& operator>>(std::istream& is, Point& p);
    friend std::ostream& operator<<(std::ostream& os, Point& p);
    friend double dist(Point& p1, Point& p2);

private:
    double x_, y_;
};

#endif
```

## 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 dist(Point& p1, Point& p2)
{
    double dx = (p1.x_ - p2.x_);
    double dy = (p1.y_ - p2.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 RECTANGLE_H
#define RECTANGLE_H
#include "figure.h"

class Rectangle : public Figure
{
public:
    Rectangle();
    Rectangle(std::istream& is);
    virtual ~Rectangle();
    void Print(std::ostream& os);
    double Square();

```

```

        size_t VertexesNumber();

private:
    Point a, b, c, d;
    double len1, len2;
};

#endif

```

## rectangle.cpp

```

#include "rectangle.h"

Rectangle::Rectangle() : a(0.0, 0.0), b(0.0, 0.0), c(0.0, 0.0), d(0.0, 0.0)
{
    std::cout << "Created default rectangle" << std::endl;
};

Rectangle::Rectangle(std::istream& is)
{
    std::cout << "Enter the values of rectangle's points" << std::endl;
    is >> a >> b >> c >> d;
    len1 = dist(a, b);
    len2 = dist(b, c);
    std::cout << "Created rectangle via istream" << std::endl;
}

void Rectangle::Print(std::ostream& os)
{
    os << "Rectangle: " << a << " " << b << " " << c << " " << d << std::endl;
}

```

```

}

size_t Rectangle::VertexesNumber()
{
    return 4;
}

double Rectangle::Square()
{
    return len1 * len2;
}

Rectangle::~Rectangle()
{
    std::cout << "Deleted rectangle" << std::endl;
}

```

## rhombus.h

```

#ifndef RHOMBUS_H
#define RHOMBUS_H
#include "figure.h"

class Rhombus : public Figure
{
public:
    Rhombus();
    Rhombus(std::istream& is);
    virtual ~Rhombus();
    void Print(std::ostream& os);
    double Square();
    size_t VertexesNumber();
}

```

```
private:
    Point a, b, c, d;
    double diag1, diag2;
};
```

```
#endif
```

## rhombus.cpp

```
#include "rhombus.h"
```

```
Rhombus::Rhombus() : a(0.0, 0.0), b(0.0, 0.0), c(0.0, 0.0), d(0.0, 0.0)
{
    std::cout << "Created default rhombus" << std::endl;
};
```

```
Rhombus::Rhombus(std::istream& is)
{
    std::cout << "Enter the values of rhombus' points" << std::endl;
    is >> a >> b >> c >> d;
    diag1 = dist(a, c);
    diag2 = dist(b, d);
    std::cout << "Created rhombus via istream" << std::endl;
}
```

```
void Rhombus::Print(std::ostream& os)
{
    os << "Rhombus: " << a << " " << b << " " << c << " " << d << std::endl;
}
```

```
size_t Rhombus::VertexesNumber()
{
    return 4;
}
```

```
double Rhombus::Square()
```



```

{
    return (diag1 * diag2) / 2.;
}

Rhombus::~Rhombus()
{
    std::cout << "Deleted rhombus" << std::endl;
}

```

## trapezoid.h

```

#ifndef TRAPEZOID_H
#define TRAPEZOID_H
#include "figure.h"
#include <algorithm>
class Trapezoid : public Figure
{
public:
    Trapezoid();
    Trapezoid(std::istream& is);
    virtual ~Trapezoid();
    void Print(std::ostream& os);
    double Square();
    size_t VertexesNumber();

private:
    Point a, b, c, d;
    double lena, lenb, lenc, lend;
};

#endif

```

# trapezoid.cpp

```
#include "trapezoid.h"

Trapezoid::Trapezoid() : a(0.0, 0.0), b(0.0, 0.0), c(0.0, 0.0), d(0.0, 0.0)
{
    std::cout << "Created default trapezoid" << std::endl;
};

Trapezoid::Trapezoid(std::istream& is)
{
    std::cout << "Enter the values of trapezoid's points" << std::endl;
    is >> a >> b >> c >> d;
    lena = dist(a, b);
    lenb = dist(c, d);
    lenc = dist(b, c);
    lend = dist(a, d);
    if (lena > lenb)
    {
        std::swap(lena, lenb);
        std::swap(lenc, lend);
    }
    std::cout << "Created trapezoid via istream" << std::endl;
}

void Trapezoid::Print(std::ostream& os)
{
    os << "Trapezoid: " << a << " " << b << " " << c << " " << d << std::endl;
}

size_t Trapezoid::VertexesNumber()
{
    return 4;
}

double Trapezoid::Square()
{
    return ((lena + lenb) / 2.) * sqrt(pow(lenc, 2) - pow(((pow(lenb - lena, 2) + pow(lenc, 2) - pow(lend, 2)) / (2. * (lenb - lena))), 2));
}

Trapezoid::~Trapezoid()
{
}
```

```
        std::cout << "Deleted trapezoid" << std::endl;
    }
}
```

## main.cpp

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

int main()
{
    Rectangle rec1(std::cin);
    rec1.Print(std::cout);
    std::cout << rec1.VertexesNumber() << std::endl;
    std::cout << rec1.Square() << std::endl;

    Trapezoid t1(std::cin);
    t1.Print(std::cout);
    std::cout << t1.VertexesNumber() << std::endl;
    std::cout << t1.Square() << std::endl;

    Rhombus r1(std::cin);
    r1.Print(std::cout);
    std::cout << r1.VertexesNumber() << std::endl;
    std::cout << r1.Square() << std::endl;

    Figure* rec2 = new Rectangle(std::cin);
    rec2->Print(std::cout);
    std::cout << rec2->VertexesNumber() << std::endl;
    std::cout << rec2->Square() << std::endl;
    delete rec2;

    Figure* t2 = new Trapezoid(std::cin);
    t2->Print(std::cout);
    std::cout << t2->VertexesNumber() << std::endl;
    std::cout << t2->Square() << std::endl;
    delete t2;
}
```

```
Figure* r2 = new Rhombus(std::cin);
r2->Print(std::cout);
std::cout << r2->VertexesNumber() << std::endl;
std::cout << r2->Square() << std::endl;
delete r2;

system("pause");
return 0;
}
```