

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

**ЛАБОРАТОРНАЯ РАБОТА №1**  
по курсу объектно-ориентированное программирование I семестр, 2021/22  
уч. год

Студент Чекменев Вячеслав Алексеевич, группа М8О-207Б-20

Преподаватель Дорохов Евгений Павлович

## Условие

Задание: Вариант 28: Трапеция, Ромб, 5-угольник. 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 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 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. figure.h: описание абстрактного класса фигур
3. point.h: описание класса точки
4. pentagon.h: описание класса 5-угольника, наследующегося от figures
5. rhombus.h: описание класса Ромба, наследующегося от figures
6. trapezoid.h: описание класса Трапеции, наследующегося от figures
7. point.cpp: реализация класса точки
8. pentagon.cpp: реализация класса угольника, наследующегося от figures
9. rhombus.cpp: реализация класса Ромба, наследующегося от figures
10. trapezoid.cpp: реализация класса Трапеции, наследующегося от figures

## Дневник отладки

1. Day 1, 01.10.2021 – программа работает только, когда пользователь вводит с консоли длины сторон фигур,
2. Day 2, 02.10.2021 – нет виртуальных методов в классе Figure,
3. Day 3, 03.10.2021 – добавил виртуальные методы в класс Figure,
4. Day 4, 04.10.2021 – Project structure accepted.

## Недочёты

- пока не выявлены

## Выводы

Вывод: В данной ЛР помогла разобраться с базовыми понятиями ООП в с++. Я познакомился с такими понятиями, как: класс, метод, property, виртуальная функция, абстрактный класс, дружественные функции и тд.

Исходный код

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

#include <iostream>

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

// version 3
#endif // FIGURE_H
```

point.h

```
#ifndef POINT_H  
#define 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);  
  
protected:  
    double x_;  
    double y_;  
};
```

```
#endif // POINT_H
```

## point.cpp

```
#include "point.h"
#include <iostream>
#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;
}
```

## main.cpp

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

int main()
{
    std::cout << "if u wanna use interactive mode press 1, else\n";
    int xui;
    std::cin >> x;
    if (x) {
        char c;
        std::cout << "Press '?' for help:\n";
        while ((c = getchar()) != EOF) {
            if (c == '?') {
                std::cout << "U can:\n";
                std::cout << "press t -- Play with Trapezoid\n";
                std::cout << "press r -- Play with Rhombus\n";
                std::cout << "press p -- Play with Pentagon\n";
            }
            else if (c == 't') {
                std::cout << "Trapezoid Mode...\nUse coodinates\n";
                Trapezoid a(std::cin);
                std::cout << "ab = " << a.len_ab << " bc = " << a.len_bc << "\n";
                std::cout << "Area = " << a.Area() << std::endl;
                std::cout << "Vertex Number = " << a.VerticesNumber << "\n";
                a.Print(std::cout);
                std::cout << "Complete, press next button...\n";
            }
        }
    }
}
```

```

}
else if (c == 'r') {
    std::cout << "Rhombus Mode...\nUse coodinates.
    Rhombus b(std::cin);
    std::cout << "a = " << b.len_a << " d1 (AC) = "
    std::cout << "Area = " << b.Area() << std::endl;
    std::cout << "Vertex Number = " << b.VerticesNu
    b.Print(std::cout);
    std::cout << "Complete, press next button...\n'
}
else if (c == 'p') {
    Pentagon c(std::cin);
    std::cout << "ab = " << c.len_ab
                << " bc = " << c.len_bc
                << " cd = " << c.len_cd
                << " de = " << c.len_de
                << " ea = " << c.len_ea
                << " d1 (AC) = " << c.d_ac
                << " d2 (CE) = " << c.d_ce
                << std::endl;
    std::cout << "Area = " << c.Area() << std::endl;
    std::cout << "Vertex Number = " << c.VerticesNu
    c.Print(std::cout);
    std::cout << "Complete, press next button...\n'
}
else if (!(c == 't' || c == 'r' || c == 'p' || c ==
    std::cout << "Damn bro... wrong button, try aga
}

```



```
        }  
    }  
    return 0;  
}
```

## pentagon.h

```
#ifndef PENTAGON_H
#define PENTAGON_H

#include "figure.h"
#include <iostream>
#include "point.h"

class Pentagon : public Figure {
public:
    Pentagon();
    Pentagon(double ab, double bc, double cd, double de,
            double ea, double d_ac_, double d_ce_);
    Pentagon(std::istream &is);
    Pentagon(const Pentagon& other);

    virtual ~Pentagon();

    size_t VertexesNumber();
    double Area();
    void Print(std::ostream& os);
public:
    double len_ab, len_bc, len_cd, len_de, len_ea, d_ac, d_ce;
};

#endif // PENTAGON_H
```

pentagon.cpp

```
#include "pentagon.h"
```

```
#include <cmath>
```

```
static Point a_o, b_o, c_o, d_o, e_o;
```

```
double triangle_area(double a, double b, double c) {  
    double p = (a + b + c);  
    double s = std::sqrt(p * (p - a) * (p - b) * (p - c));  
    return s;  
}
```

```
Pentagon::Pentagon()  
    : len_ab(0.0),  
      len_bc(0.0),  
      len_cd(0.0),  
      len_de(0.0),  
      len_ea(0.0),  
      d_ac(0.0),  
      d_ce(0.0) {  
    std::cout << "Default Pentagon created" << std::endl;  
}
```

```
Pentagon::Pentagon(double ab, double bc, double cd, double de,  
                   double ea, double d_ac_, double d_ce_)  
    : len_ab(ab),  
      len_bc(bc),
```

```

        len_cd(cd),
        len_de(de),
        len_ea(ea),
        d_ac(d_ac_),
        d_ce(d_ce_) {
std::cout << "Pentagon created" << std::endl;
}

```

```

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

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

    is >> a_o >> b_o >> c_o >> d_o >> e_o;

    len_ab = a_o.dist(b_o);
    len_bc = b_o.dist(c_o);
    len_cd = c_o.dist(d_o);
    len_de = d_o.dist(e_o);
    len_ea = e_o.dist(a_o);
    d_ac = a_o.dist(c_o);
    d_ce = c_o.dist(e_o);

    std::cout << "Pentagon created via istream" << std::endl;
}

```

```

Pentagon::Pentagon(const Pentagon& other)
    : Pentagon(other.len_ab, other.len_bc, other.len_cd,

```

```

        other.len_de, other.len_ea, other.d_ac, other.d_
std::cout << "Made copy of Pentagon" << std::endl;
}

size_t Pentagon::VertexesNumber() {
    return 5;
}

double Pentagon::Area() {
    return triangle_area(len_ab, len_bc, d_ac) +
        triangle_area(d_ac, d_ce, len_ea) +
        triangle_area(len_cd, len_de, d_ce);
}

void Pentagon::Print(std::ostream& os) {
    std::cout << "Pentagon: ";
    os << a_o; std::cout << " ";
    os << b_o; std::cout << " ";
    os << c_o; std::cout << " ";
    os << d_o; std::cout << " ";
    os << e_o; std::cout << std::endl;
}

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

```

## rhombus.h

```
#ifndef RHOMBUS_H
#define RHOMBUS_H

#include "figure.h"
#include <iostream>
#include "point.h"

class Rhombus : public Figure {
public:
    Rhombus();
    Rhombus(double a_, double d_ac_, double d_bd_);
    Rhombus(std::istream &is);
    Rhombus(const Rhombus& other);

    virtual ~Rhombus();

    size_t VertexesNumber();
    double Area();
    void Print(std::ostream& os);
public:
    double len_a, d_ac, d_bd;
};

#endif // RHOMBUS_H
```

## rhombus.cpp

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

```
#include <cmath>
```

```
static Point a_o, b_o, c_o, d_o;
```

```
Rhombus::Rhombus()  
    : len_a(0.0),  
      d_ac(0.0),  
      d_bd(0.0) {  
    std::cout << "Default Rhombus created" << std::endl;  
}
```

```
Rhombus::Rhombus(double a_, double d_ac_, double d_bd_)  
    : len_a(a_),  
      d_ac(d_ac_),  
      d_bd(d_bd_) {  
    std::cout << "Rhombus created" << std::endl;  
}
```

```
Rhombus::Rhombus(std::istream &is) {  
  
    std::cout << "Enter Data:" << std::endl;  
  
    is >> a_o >> b_o >> c_o >> d_o;
```

```

    len_a = a_o.dist(b_o);
    d_ac = a_o.dist(c_o);
    d_bd = b_o.dist(d_o);

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

Rhombus::Rhombus(const Rhombus& other)
    : Rhombus(other.len_a, other.d_ac, other.d_bd) {
    std::cout << "Made copy of Rhombus" << std::endl;
}

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

double Rhombus::Area() {
    return (d_bd * d_ac) / 2;
}

void Rhombus::Print(std::ostream& os) {
    std::cout << "Rhombus: ";
    os << a_o; std::cout << " ";
    os << b_o; std::cout << " ";
    os << c_o; std::cout << " ";
    os << d_o; std::cout << std::endl;
}

```



```
Rhombus::~Rhombus() {  
    std::cout << "Rhombus deleted" << std::endl;  
}
```

## trapezoid.h

```
#ifndef TRAPEZOID_H
#define TRAPEZOID_H

#include "figure.h"
#include <iostream>
#include "point.h"

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

    virtual ~Trapezoid();

    size_t VertexesNumber();
    double Area();
    void Print(std::ostream& os);
public:
    double len_ab, len_bc, len_cd, len_da;
};

#endif // TRAPEZOID_H
```

trapezoid.cpp

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

```
#include <cmath>
```

```
static Point a_o, b_o, c_o, d_o;
```

```
Trapezoid::Trapezoid()  
    : len_ab(0.0),  
      len_bc(0.0),  
      len_cd(0.0),  
      len_da(0.0) {  
    std::cout << "Default Trapezoid created" << std::endl;  
}
```

```
Trapezoid::Trapezoid(double ab, double bc, double cd, double da)  
    : len_ab(ab),  
      len_bc(bc),  
      len_cd(cd),  
      len_da(da) {  
    std::cout << "Trapezoid created" << std::endl;  
}
```

```
Trapezoid::Trapezoid(std::istream &is) {  
  
    std::cout << "Enter Data:" << std::endl;
```

```

is >> a_o >> b_o >> c_o >> d_o;

len_ab = a_o.dist(b_o);
len_bc = b_o.dist(c_o);
len_cd = c_o.dist(d_o);
len_da = d_o.dist(a_o);

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

Trapezoid::Trapezoid(const Trapezoid& other)
    : Trapezoid(other.len_ab, other.len_bc, other.len_cd, other.len_da) {
    std::cout << "Made copy of Trapezoid" << std::endl;
}

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

double Trapezoid::Area() {
    double p = (len_ab + len_bc + len_cd + len_da) / 2;
    return (len_bc + len_da) *
        std::sqrt((p - len_bc) *
                    (p - len_da) *
                    (p - len_da - len_ab) *
                    (p - len_da - len_cd)) /
        std::abs(len_bc - len_da);
}

```

```

void Trapezoid::Print(std::ostream& os) {
    std::cout << "Trapezoid: ";
    os << a_o; std::cout << " ";
    os << b_o; std::cout << " ";
    os << c_o; std::cout << " ";
    os << d_o; std::cout << std::endl;
}

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

```