**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 *Ядров Артем Леонидович, группа М8О-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

## Условие

Задание: Вариант 27: Прямоугольник, трапеция, ромб. 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 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 11 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. trapezoid.h: описание класса трапеции, наследующегося от figures
5. rectangle.h: описание класса прямоугольника, наследующегося от figures
6. rhombus.h: описание класса ромба, наследующегося от figures
7. point.cpp: реализация класса точки
8. trapezoid.cpp: реализация класса трапеции, наследующегося от figures
9. rectangle.cpp: реализация класса прямоугольника, наследующегося от figures
10. rhombus.cpp: реализация класса ромба, наследующегося от figures

## Дневник отладки

## Недочёты

## Выводы

Я научился создавать и реализовывать классы в C++, а также познакомился с дружественными функциями, перегрузкой операторов и изучил основные понятия ООП.

## Исходный код

# figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

#include "point.h"
#include <cstddef>
#include <iostream>
#include <vector>

class Figure {
public:
  virtual ~Figure(){};

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

  virtual double Area() = 0;

  virtual size_t VertexesNumber() = 0;
};

#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 dist(Point &other);

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

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

  friend class Rectangle; // Дружественные классы, чтобы были доступны
                          // координаты точки

  friend class Trapezoid;

  friend class Rhombus;

private:
  double x_;
  double y_;
};

#endif // POINT_H
```

## point.cpp

```cpp
#include "point.h"

#include <cmath>

Point::Point() : x_(0.0), y_(0.0) {} // стандартный конструктор

Point::Point(double x, double y) : x_(x), y_(y) {} // конструктор через значения

Point::Point(std::istream &is) { // конструктор через поток
  is >> x_ >> y_;
}

double Point::dist(Point &other) { // расстояние между двумя точками
  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) { // перегруженный оператор >>
  is >> p.x_ >> p.y_;
  return is;
}

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

# rectangle.h

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include "figure.h"

using namespace std;

class Rectangle : public Figure {
public:
  Rectangle();

  Rectangle(vector<Point> v);

  Rectangle(istream &is);

  Rectangle(const Rectangle &other);

  virtual ~Rectangle();

  void Print(ostream &os);

  size_t VertexesNumber();

  double Area() override;

private:
  Point a, b, c, d; // a - левая нижняя вершина, b - левая верхняя, c - правая
                    // верхняя, d - правая нижняя
};

#endif
```

# rectangle.cpp

```cpp
#include "rectangle.h"

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

double Rectangle::Area() { return (abs(a.x_ - b.x_) * abs(a.y_ - b.y_)); }

Rectangle::Rectangle() : a(), b() {
  cout << "Default rectangle is created" << endl;
}

Rectangle::Rectangle(vector<Point> v) : a(v[0]), b(v[1]), c(v[2]), d(v[3]) {
  cout << "Rectangle with vertices " << a << ", " << b << ", " << c << ", " << d
       << " was created" << endl; // создание с помощью вектора координат вершин
}

Rectangle::Rectangle(istream &is) {
  cout << "Enter lower left coordinate" << endl;
  cin >> a;
  cout << "Enter upper left coordinate" << endl;
  cin >> b;
  cout << "Enter upper right coordinate" << endl;
  cin >> c;
  cout << "Enter lower right coordinate" << endl;
  cin >> d;
  cout << "Rectangle was created via stream" << endl;
}

Rectangle::Rectangle(const Rectangle &other)
    : a(other.a), b(other.b), c(other.c), d(other.d) {
  cout << "Made copy of rectangle" << endl;
}

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

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

# trapezoid.h

```
#ifndef INC_1_LAB_TRAPEZOID_H
#define INC_1_LAB_TRAPEZOID_H

#include "figure.h"

using namespace std;

class Trapezoid : public Figure {
public:
  Trapezoid();

  Trapezoid(istream &is);

  Trapezoid(const Trapezoid &other);

  Trapezoid(vector<Point>);

  virtual ~Trapezoid();

  void Print(ostream &os);

  size_t VertexesNumber();

  double Area();

private:
  Point a, b, c, d; // a - левая нижняя, b - левая верхняя, c - правая верхняя,
                    // d - правая нижняя
};

#endif
```

# trapezoid.cpp

```cpp
#include "trapezoid.h"
#include <cmath>

void Trapezoid::Print(ostream &os) { // вывод координат
  cout << "Trapezoid: " << a << ", " << b << ", " << c << ", " << d << endl;
}

double Trapezoid::Area() {
  double k = (a.y_ - d.y_) / (a.x_ - d.x_); // вычисляем прямую ad
  double m = a.y_ - k * a.x_;
  double h =
      abs(b.y_ - k * b.x_ - m) / sqrt(1 + k * k); // находим высоту от bc до ad
  return 0.5 * (a.dist(d) + b.dist(c)) * h;
}

Trapezoid::Trapezoid() : a(), b(), c(), d() { // конструкторы
  cout << "Default trapezoid was created" << endl;
}

Trapezoid::Trapezoid(vector<Point> v) : a(v[0]), b(v[1]), c(v[2]), d(v[3]) {
  cout << "Trapezoid with vertices " << a << ", " << b << ", " << c << ", " << d
       << " was created" << endl;
}

Trapezoid::Trapezoid(istream &is) {
  cout << "Enter lower left coordinate" << endl;
  cin >> a;
  cout << "Enter upper left coordinate" << endl;
  cin >> b;
  cout << "Enter upper right coordinate" << endl;
  cin >> c;
  cout << "Enter lower right coordinate" << endl;
  cin >> d;
  cout << "Trapezoid was created" << endl;
}

Trapezoid::Trapezoid(const Trapezoid &other)
    : a(other.a), b(other.b), c(other.c), d(other.d) {
  cout << "Made copy of trapezoid" << endl;
}
```

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

size_t Trapezoid::VertexesNumber() { // количество вершин
  return 4;
}
```

# rhombus.h

```cpp
#ifndef INC_1_LAB_RHOMBUS_H
#define INC_1_LAB_RHOMBUS_H

#include "figure.h"

using namespace std;

class Rhombus : public Figure {
public:
  Rhombus();

  Rhombus(vector<Point> v);

  Rhombus(istream &is);

  Rhombus(const Rhombus &other);

  double Area();

  virtual ~Rhombus();

  size_t VertexesNumber();

  void Print(ostream &os);

private:
  Point a, b, c, d;
};

#endif // INC_1_LAB_RHOMBUS_H
```

# rhombus.cpp

```cpp
#include "rhombus.h"

Rhombus::Rhombus() : a(), b(), c(), d() {
  cout << "Default Rhombus was created" << endl;
}

Rhombus::Rhombus(vector<Point> v) : a(v[0]), b(v[1]), c(v[2]), d(v[3]) {
  cout << "Rhombus with vertices " << a << ", " << b << ", " << c << ", " << d
       << " was created" << endl;
}

Rhombus::Rhombus(istream &is) {
  cout << "Enter lower coordinate" << endl;
  cin >> a;
  cout << "Enter left coordinate" << endl;
  cin >> b;
  cout << "Enter upper coordinate" << endl;
  cin >> c;
  cout << "Enter right coordinate" << endl;
  cin >> d;
  cout << "Rhombus was created via stream" << endl;
}

Rhombus::Rhombus(const Rhombus &other)
    : a(other.a), b(other.b), c(other.c), d(other.d) {
  cout << "Made copy of rhombus" << endl;
}

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

double Rhombus::Area() { return 0.5 * a.dist(c) * b.dist(d); }

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

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

# main.cpp

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

using namespace std;

int main() {
  Rectangle Rec1; // создаем дефолтный прямоугольник
  Point a(0, 0); // создаем вершины для следующего прямоугольника, а затем
                 // вектор из вершин
  Point b(0, 1);
  Point c(1, 1);
  Point d(1, 0);
  vector<Point> v{a, b, c, d}; //
  Rectangle Rec2(v); // создаем прямоугольник через вектор вершин
  Rectangle Rec3(cin); // создаем прямоугольник через стандартный ввод
  Rectangle Rec4(Rec3); // создаем копию
  Rec1.Print(cout); // выводим первый прямоугольник
  cout << Rec1.VertexesNumber() << endl; // количество вершин прямоугольника
  cout << "Rectangle area is " << Rec2.Area()
       << endl; // вывод площади прямоугольника
  Figure *f = new Rectangle(Rec2); // указатель класса фигуры на прямоугольник
  delete f; // освобождаем память указателя
  Rhombus Rh1;
  Rhombus Rh2(v);
  Rhombus Rh3(cin);
  Rhombus Rh4(Rh1);
  Rh2.Print(cout);
  cout << "Rhombus area is " << Rh2.Area() << endl;
  cout << Rh3.VertexesNumber() << endl;
  f = new Rhombus(Rh3);
  delete f;
  Trapezoid Tr1;
  Point tr1 = a, tr2 = b, tr3(2, 1), tr4(3, 0);
  vector<Point> tr{tr1, tr2, tr3, tr4};
  Trapezoid Tr2(tr);
  Trapezoid Tr3(cin);
  Trapezoid Tr4(Tr1);
  cout << "Trapezoid area is " << Tr2.Area() << endl;
  Tr3.Print(cout);
  cout << Tr4.VertexesNumber() << endl;
```

```
  f = new Trapezoid(Tr4);
  delete f;
  return 0;
}
```