МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №3**

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студент: *Тихонов Фёдор Андреевич, группа М8О-207Б-20*

Преподаватель: *Дорохов Евгений Павлович, каф. 806*

**Задание:** Спроектировать и запрограммировать на языке C++ классы трёх фигур. Классы должны удовлетворять следующим правилам:

* + Должны быть названы как в вариантах задания и расположены в раздельных файлах;
  + Иметь общий родительский класс Figure;
  + Содержать конструктор, принимающий координаты вершин фигуры из стандартного потока std::cin, расположенных через пробел (например: 0.0 0.0 1.0 0.0 1.0 1.0 0.0 1.0);
  + Содержать набор общих методов:
    - size\_t VertexesNumber() – метод, возвращающий количество вершин фигуры
    - double Area() – метод расчета площади фигуры

**Вариант №26:**

* + Фигура 1: Квадрат
  + Фигура 2: Прямоугольник
  + Фигура 3: Трапеция

**Описание программы:**

Исходный код разделён на 10 файлов:

* point.h – описание класса точки
* point.cpp – реализация класса точки
* figure.h – описание класса фигуры
* square.h – описание класса квадрат (наследуется от фигуры)
* square.cpp – реализация класса квадрат
* rectangle.h – описание класса прямоугольника (наследуется от фигуры)
* rectangle.cpp – реализация класса прямоугольника
* trapezoid.h – описание класса трапеции (наследуется от фигуры)
* trapezoid.cpp – реализация класса трапеции
* main.cpp – основная программа

**Дневник отладки:**

Была неверная формула для поиска площади трапеции, была заменена на формулу площади Гаусса.

**Вывод:**  
В данной лабораторной работе я познакомился с принципами и концепциями объектно- ориентированного программирования: инкапсуляцией, наследованием и полиморфизмом. Научился проектировать классы и работать с ними, а также поработал с конструкторами, деструкторами и виртуальными функциями в С++.

**Исходный код:**

**point.h:**

#ifndef POINT\_H

#define POINT\_H

#include <iostream>

class Point {

public:

Point();

Point(std::istream &is);

Point(double x, double y);

double fx();

double fy();

double dist(Point& other);

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\_H

**point.cpp:**

#include <iostream>

#include <cmath>

#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::fx(){

return x\_;

};

double Point::fy(){

return 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;

}

**figure.h:**

#ifndef FIGURE\_H

#define FIGURE\_H

#include <iostream>

#include "point.h"

class Figure {

public:

virtual size\_t VertexesNumber() = 0;

virtual double Area() = 0;

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

~Figure() {};

};

#endif //FIGURE\_H

**rectangle.h:**

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <iostream>

#include "point.h"

#include "figure.h"

class Rectangle : Figure {

public:

Rectangle();

Rectangle(Point a, Point b, Point c, Point d);

Rectangle(std::istream& is);

size\_t VertexesNumber();

double Area();

void Print(std::ostream& os);

private:

Point a\_;

Point b\_;

Point c\_;

Point d\_;

};

#endif //RECTANGLE\_H

**rectangle.cpp:**

#include <iostream>

#include "point.h"

#include "rectangle.h"

Rectangle::Rectangle() : a\_(Point()), b\_(Point()), c\_(Point()), d\_(Point()) {}

Rectangle::Rectangle(Point a, Point b, Point c, Point d) : a\_(a), b\_(b), c\_(c), d\_(d) {}

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

is >> a\_ >> b\_ >> c\_ >> d\_;

}

void Rectangle::Print(std::ostream& os) {

os << "Rectangle: " << a\_ << " " << b\_ << " " << c\_ << " " << d\_ << std::endl;

}

size\_t Rectangle::VertexesNumber(){

return 4;

}

double Rectangle::Area(){

return a\_.dist(b\_) \* c\_.dist(d\_);

}

**square.h:**

#ifndef SQUARE\_H

#define SQUARE\_H

#include <iostream>

#include "point.h"

#include "figure.h"

class Square : Figure {

public:

Square();

Square(Point a, Point b, Point c, Point d);

Square(std::istream& is);

size\_t VertexesNumber();

double Area();

void Print(std::ostream& os);

private:

Point a\_;

Point b\_;

Point c\_;

Point d\_;

};

#endif //SQUARE\_H

**square.cpp:**

#include <iostream>

#include "point.h"

#include "square.h"

Square::Square() : a\_(Point()), b\_(Point()), c\_(Point()), d\_(Point()) {}

Square::Square(Point a, Point b, Point c, Point d) : a\_(a), b\_(b), c\_(c), d\_(d) {}

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

is >> a\_ >> b\_ >> c\_ >> d\_;

}

void Square::Print(std::ostream& os) {

os << "Square: " << a\_ << " " << b\_ << " " << c\_ << " " << d\_ << std::endl;

}

size\_t Square::VertexesNumber() {

return 4;

}

double Square::Area() {

return a\_.dist(b\_) \* a\_.dist(b\_);

}

**trapezoid.h:**

#ifndef TRAPEZOID\_H

#define TRAPEZOID\_H

#include <iostream>

#include "point.h"

#include "figure.h"

class Trapezoid : Figure {

public:

Trapezoid();

Trapezoid(Point a, Point b, Point c, Point d);

Trapezoid(std::istream& is);

size\_t VertexesNumber();

double Area();

void Print(std::ostream& os);

private:

Point a\_;

Point b\_;

Point c\_;

Point d\_;

};

#endif //TRAPEZOID\_H

**trapezoid.cpp:**

#include <iostream>

#include <cmath>

#include "point.h"

#include "trapezoid.h"

Trapezoid::Trapezoid() : a\_(Point()), b\_(Point()), c\_(Point()), d\_(Point()) {}

Trapezoid::Trapezoid(Point a, Point b, Point c, Point d) : a\_(a), b\_(b), c\_(c), d\_(d) {}

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

is >> a\_ >> b\_ >> c\_ >> d\_;

}

void Trapezoid::Print(std::ostream& os) {

os << "Trapezoid: " << a\_ << " " << b\_ << " " << c\_ << " " << d\_ << std::endl;

}

size\_t Trapezoid::VertexesNumber() {

return 4;

}

double Trapezoid::Area() {

double det1 = a\_.fx() \* b\_.fy() + b\_.fx() \* c\_.fy() + c\_.fx() \* d\_.fy() + d\_.fx() \* a\_.fy();

double det2 = a\_.fy() \* b\_.fx() + b\_.fy() \* c\_.fx() + c\_.fy() \* d\_.fx() + d\_.fy() \* a\_.fx();

double det = abs(det1 - det2);

return 0.5 \* det;

} // Gauss's Area Calculation Formula (Shoelace Theorem)

**main.cpp:**

#include <iostream>

#include "point.h"

#include "figure.h"

#include "square.h"

#include "rectangle.h"

#include "trapezoid.h"

int main() {

std::cout << "Enter a coordinates of \"Square\"" << std::endl;

Square a(std::cin);

a.Print(std::cout);

std::cout << a.Area() << "\n";

std::cout << "Enter a coordinates of \"Rectangle\"" << std::endl;

Rectangle b(std::cin);

b.Print(std::cout);

std::cout << b.Area() << "\n";

std::cout << "Enter a coordinates of \"Trapezoid\"" << std::endl;

Trapezoid c(std::cin);

c.Print(std::cout);

std::cout << c.Area() << std::endl;

}

**Пример работы:**

Enter a coordinates of "Square"

1 1 2 2 3 3 4 4

Square: (1, 1) (2, 2) (3, 3) (4, 4)

2

Enter a coordinates of "Rectangle"

1 2 3 4 5 6 7 8

Rectangle: (1, 2) (3, 4) (5, 6) (7, 8)

8

Enter a coordinates of "Trapezoid"

8 7 6 5 5 6 7 8

Trapezoid: (8, 7) (6, 5) (5, 6) (7, 8)

4

Process finished with exit code 0