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 C++.

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
```