МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №3

по курсу объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент *Борисов Ян Артурович, группа М80-208Б-20*
Преподаватель *Дорохов Евгений Павлович*

Условие

Задание: Вариант 2: Квадрат, Прямоугольник, 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 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 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. figure.h: описание абстрактного класса фигур
3. point.h: описание класса точки
4. rectangle.h: описание класса прямоугольника, наследующегося от figure.
5. square.h: описание класса квадрата, наследующегося от figure.
6. trapezoid.h: описание класса трапеции, наследующегося от figure.
7. point.cpp: реализация класса точки
8. rectangle.cpp: реализация класса пятиугольника, наследующегося от figure
9. square.cpp: реализация класса шестиугольника, наследующегося от figure.
10. trapezoid.cpp: реализация класса восьмиугольника, наследующегося от figure.

Дневник отладки

Во время выполнения лабораторной работы программа не нуждалась в отладке, все ошибки компиляции были исправлены с первой попытки. После их исправления программа работала так, как было задумано изначально.

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++. Могу сказать, что справился с этой целью весьма успешно: усвоил 3 основных принципа ООП: полиморфизм, наследование, инкапсуляция, освоил базовые понятия ООП, такие как классы, методы, конструкторы, деструкторы… Ознакомился с ключевыми словами virtual, friend, private, public... Повторил тему “директивы условной компиляции”, “перегрузка функций/операторов”, работа со стандартными потоками ввода-вывода. Лабораторная работа №3 успешно выполнена.

Исходный код

Figure.h

```
#include "point.h"

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

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);

        private:
        double x_;
        double y_;
};

Point.cpp
#include "point.h"
#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 <iostream>
#include "square.h"
#include "rectangle.h"
#include "trapezoid.h"

int main() {
        Square a;
        Square sqr(std::cin);
        sqr.Print(std::cout);
        std::cout << "Площадь квадрата: " << sqr.Area() << std::endl;
        std::cout << "Количество вершин в квадрате " << sqr.VertexesNumber()
        << std::endl;
        Rectangle b;
        Rectangle rec(std::cin);
        rec.Print(std::cout);
        std::cout << "Площадь прямоугольника: " << rec.Area() << std::endl;
        std::cout << "Количество вершин в прямоугольнике " <<
        rec.VertexesNumber() << std::endl;
        Trapezoid c;
        Trapezoid tr(std::cin);
        tr.Print(std::cout);
        std::cout << "Площадь трапеции: " << tr.Area() << std::endl;
        std::cout << "Количество вершин в трапеции " << tr.VertexesNumber() <<
std::endl;
        return 0;
}
Rectangle.cpp
#include "rectangle.h"
```

```
Rectangle::Rectangle() : a(), b(), c(), d() {
        std::cout << "Default rectangle was created" << std::endl;
}
Rectangle::Rectangle(std::istream &is) {
        std::cout << "Enter lower left coordinate" << std::endl;
        std::cin >> a;
        std::cout << "Enter upper left coordinate" << std::endl;
        std::cin >> b;
        std::cout << "Enter upper right coordinate" << std::endl;
        std::cin >> c;
        std::cout << "Enter lower right coordinate" << std::endl;
        std::cin >> d;
        std::cout << "Rectangle was created" << std::endl;
}
Rectangle::~Rectangle(){
        std::cout << "Rectangle deleted" << std::endl;
}
void Rectangle::Print(std::ostream& os) {
        std::cout << "Rectangle: " << a << " " << b << " "  << c << " " << d
        << std::endl;
}
double Rectangle::Area() {
        double len_a = a.dist(b);
        double len_b = b.dist(c);
        return len_a * len_b;
}
size_t Rectangle::VertexesNumber() {
         return 4;
}
```

Rectangle.h

```
#include "figure.h"

class Rectangle: public Figure {
        public:
        Rectangle();
        Rectangle(std::istream &is);
        virtual ~Rectangle();
        void Print(std::ostream& os);
        double Area();
        size_t VertexesNumber();
        private:
        Point a, b, c, d;
};
```

Square.h

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

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

```
        Point a, b, c, d;
};
```

Square.cpp

```
#include "square.h"

Square::Square() : a(), b(), c(), d() {
        std::cout << "Default square was created" << std::endl;
}
Square::Square(std::istream &is) {
        std::cout << "Enter lower left coordinate" << std::endl;
        std::cin >> a;
        std::cout << "Enter upper left coordinate" << std::endl;
        std::cin >> b;
        std::cout << "Enter upper right coordinate" << std::endl;
        std::cin >> c;
        std::cout << "Enter lower right coordinate" << std::endl;
        std::cin >> d;
        std::cout << "Square was created" << std::endl;
}
Square::~Square() {
        std::cout << "Square deleted" << std::endl;
}

void Square::Print(std::ostream& os) {
        std::cout << "Square: " << a << " " << b << " "  << c << " " << d <<
        std::endl;
}
double Square::Area() {
        double len_a = a.dist(b);
        double len_b = b.dist(c);
        return len_a * len_b;
}
size_t Square::VertexesNumber() {
        return 4;
}
```

Trapezoid.h

```
#include "figure.h"

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

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

Trapezoid.cpp

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

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

```
        std::cin >> c;
        std::cout << "Enter lower right coordinate" << std::endl;
        std::cin >> d;
        std::cout << "Trapezoid was created" << std::endl;
}
Trapezoid::~Trapezoid() {
        std::cout << "Trapezoid deleted" << std::endl;
}

void Trapezoid::Print(std::ostream& os) {
        std::cout << "Trapezoid: " << a << " " << b << " "  << c << " " << d
        << std::endl;
}
double Trapezoid::Area() {
        double A = a.dist(d);
        double B = b.dist(c);
        double C = a.dist(b);
        double D = c.dist(d);
        double p = (A + B + C + D) / 2;
        double area = ((A + B) / abs(A - B)) * sqrt((p - A) * (p - B) * (p - A
- C) * (p - A- D)) ;
        return area;
}

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