**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 *Клитная Анастасия Виктороровна, группа М80-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

**Условие**

Задание: Вариант 9: Восьмиугольник, Квадрат, Треугольник. 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

**Описание программы**

Исходный код лежит в 10 файлах:

1. main.cpp: основная программа, взаимодействие с пользователем посредством команд из меню

2. include/figure.h: описание абстрактного класса фигур

3. include/point.h: описание класса точки

4. include/octagon.h: описание класса восьмиугольника, наследующегося от figures 5. include/squere.h: описание класса квадрата, наследующегося от figures 6. include/triangle.h: описание класса треугольника, наследующегося от figures 7. include/point.cpp: реализация класса точки

8. include/octagon.cpp: реализация класса восьмиугольника, наследующегося от

figures

9. include/square.cpp: реализация класса квадрата, наследующегося от figures

10. include/triangle.cpp: реализация класса треугольника, 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 С++. Могу сказать, что справился с этой целью весьма успешно: усвоил “3 китов 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 прошла для меня успешно.**

**Исходный код**

figure.h

#ifndef FIGURE\_H

#define FIGURE\_H

#include "point.h"

class Figure {

public:

virtual double Area() = 0;

virtual void Print(std::ostream &os) = 0; virtual size\_t VertexesNumber() = 0; virtual ~Figure() {};

};

#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 X();

double Y();

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.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::X() {

return x\_;

};

double Point::Y() {

return y\_;

};

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;

}

pentagon.h

#ifndef PENTAGON\_H

#define PENTAGON\_H

#include "figure.h"

#include <iostream>

class Pentagon : public Figure {

public:

Pentagon(std::istream& InputStream); virtual ~Pentagon();

size\_t VertexesNumber();

double Area();

void Print(std::ostream &OutputStream);

private:

Point a;

Point b;

Point c;

Point d;

Point e;

};

#endif

Octagon.cpp

#include "octagon.h"

#include <cmath>

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

std::cout << "Enter the octagon's vertexes:" << std::endl;

is >> point\_a;

is >> point\_b;

is >> point\_c;

is >> point\_d;

is >> point\_e;

is >> point\_f;

is >> point\_g;

is >> point\_h;

std::cout << "The octagon is created" << std::endl;

}

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

std::cout << "Octagon: ";

std::cout << point\_a << ", ";

std::cout << point\_b << ", ";

std::cout << point\_c << ", ";

std::cout << point\_d << ", ";

std::cout << point\_e << ", ";

std::cout << point\_f << ", ";

std::cout << point\_g << ", ";

std::cout << point\_h << std::endl;

}

size\_t Octagon::VertexesNumber() {

size\_t number = 8;

return number;

}

double Octagon::Area() {

double q = abs(point\_a.X() \* point\_b.Y() + point\_b.X() \* point\_c.Y() + poiny\_c.X() \* point\_d.Y() + point\_d.X() \* point\_e.Y() + point\_e.X() \* point\_f.Y() + point\_f.X() \* point\_g.Y() + point\_g.X() \* point\_h.Y() + point\_h.X() \* point\_a.Y() - point\_b.X() \* point\_a.Y() - point\_c.X() \* point\_b.Y() - point\_d.X() \* point\_c.Y() - point\_e.X() \* point\_d.Y() - point\_f.X() \* point\_e.Y() - point\_g.X() \* point\_f.Y() - point\_h.X() \* point\_g.Y() - point\_a.X() \* point\_h.Y());

double s = q / 2;

return s;

}

Octagon::~Octagon() {

std::cout << "The octagon is deleted" << std::endl;

}

Octagon.h

#ifndef OCTAGON\_H

#define OCTAGON\_H

#include "figure.h"

class Octagon : public Figure{

public:

Octagon(std::istream& is);

size\_t VertexesNumber();

double Area();

void Print(std::ostream& os);

virtual ~Octagon();

private:

Point point\_a, point\_b, point\_c, point\_d, point\_e, point\_f, point\_g, point\_h, ;

};

#endif // OCTAGON\_H

Square.cpp

#include "square.h"

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

std::cout << "Enter the square's vertexes:" << std::endl;

is >> point\_a;

is >> point\_b;

is >> point\_c;

is >> point\_d;

std::cout << "The square is created" << std::endl;

}

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

std::cout << "Square: ";

std::cout << point\_a << ", ";

std::cout << point\_b << ", ";

std::cout << point\_c << ", ";

std::cout << point\_d << std::endl;

}

size\_t Square::VertexesNumber() {

size\_t number = 4;

return number;

}

double Square::Area() {

try {

double len\_a = point\_b.dist(point\_a);

if (len\_a < 0) {

throw "The length of the side is < 0";

}

return pow(len\_a, 2);

}

catch (const char\* exception) {

std::cerr << "Error: " << exception << std::endl;

}

}

Square::~Square() {

std::cout << "The square is deleted" << std::endl;

}

Square.h

#ifndef SQUARE\_H

#define SQUARE\_H

#include "figure.h"

class Square : public Figure {

public:

Square(std::istream& is);

size\_t VertexesNumber();

double Area();

void Print(std::ostream& os);

virtual ~Square();

private:

Point point\_a, point\_b, point\_c, point\_d; //ïåðâàÿ âåðøèíà - ëåâàÿ íèæíÿÿ, äàëåå - ïî ÷àñîâîé ñòðåëêå

};

#endif // SQUARE\_H

Triangle.cpp

#include "triangle.h"

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

std::cout << "Enter the triangle's vertexes:" << std::endl;

is >> point\_a;

is >> point\_b;

is >> point\_c;

std::cout << "The triangle is created" << std::endl;

}

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

std::cout << "Triangle: ";

std::cout << point\_a << ", ";

std::cout << point\_b << ", ";

std::cout << point\_c << std::endl;

}

size\_t Triangle::VertexesNumber() {

size\_t number = 3;

return number;

}

double Trapezoid::Area() {

double p = 0.5\*abs((point\_b.X()-point\_a.X())\*(point\_c.Y()-point\_a.Y())-(point\_c.X()-point\_a.X()\*(point\_b.Y()-point\_a.Y())));

return p;

}

Triangle::~Triangle() {

std::cout << "The triangle is deleted" << std::endl;

}

Triangle.h

#ifndef TRIANGLE\_H

#define TRIANGLE\_H

#include "figure.h"

class Triangle : public Figure{

public:

Triangle(std::istream& is);

size\_t VertexesNumber();

double Area();

void Print(std::ostream& os);

virtual ~Triangle();

private:

Point point\_a, point\_b, point\_c;

};

#endif // TRIANGLE\_H