**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 *Зубко Дмитрий Валерьевич М80-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

**Условие**

Задание: Вариант 6: Пятиугольник, Шестиугольник, Восьмиугольник. 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. include/figure.h: описание абстрактного класса фигур

3. include/point.h: описание класса точки

4. include/pentagon.h: описание класса пятиугольника, наследующегося от figures 5. include/hexagon.h: описание класса шестиугольника, наследующегося от figures 6. include/octagon.h: описание класса восьмиугольника, наследующегося от figures 7. include/point.cpp: реализация класса точки

8. include/pentagon.cpp: реализация класса пятиугольника, наследующегося от

figures

9. include/hexagon.cpp: реализация класса шестиугольника, наследующегося от figures

10. include/octagon.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 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 LAB1\_FIGURE\_H

#define LAB1\_FIGURE\_H

#include <cmath>

#include <iostream>

#include "point.h"

class Figure {

public:

virtual size\_t VertexesNumber() = 0;

virtual double Area() = 0;

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

virtual ~Figure() {};

};

#endif //LAB1\_FIGURE\_H

point.h

#ifndef LAB1\_POINT\_H

#define LAB1\_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\_;

};

#endif //LAB1\_POINT\_H

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;

}

Pentagon.h

#ifndef LAB1\_PENTAGON\_H

#define LAB1\_PENTAGON\_H

#include "figure.h"

class Pentagon : Figure{

public:

size\_t VertexesNumber() override;

double Area() override;

void Print(std::ostream &os) override;

Pentagon();

Pentagon(Point a\_, Point b\_, Point c\_, Point d\_, Point e\_);

Pentagon(std::istream &is);

Pentagon(const Pentagon &other);

virtual ~Pentagon();

private:

Point a, b, c, d, e;

double SquareTriangle(Point a, Point b, Point c);

};

#endif //LAB1\_PENTAGON\_H

pentagon.cpp

#include "pentagon.h"

size\_t Pentagon::VertexesNumber() {

return 5;

}

double Pentagon::SquareTriangle(Point a, Point b, Point c){

double p = (a.dist(b) + b.dist(c) + c.dist(a)) / 2;

return sqrt(p \* (p - a.dist(b)) \* (p - b.dist(c)) \* (p - c.dist(a)));

}

double Pentagon::Area() {

return SquareTriangle(a, b, c) + SquareTriangle(a, c, d) + SquareTriangle(a, d, e);

}

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

os << "Pentagon: " << a << b << c << d << e << std::endl;

}

Pentagon::Pentagon(){}

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

Pentagon::Pentagon(const Pentagon &other) : Pentagon(other.a, other.b, other.c, other.d, other.e) {

std::cout << "Made copy of pentagon" << std::endl;

}

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

std::cout << "Enter data:" << std::endl;

is >> a >> b >> c >> d >> e;

std::cout << "Pentagon created via istream" << std::endl;

}

Pentagon::~Pentagon() {

std::cout << "Pentagon deleted" << std::endl;

}

octagon.h

#ifndef LAB1\_OCTAGON\_H

#define LAB1\_OCTAGON\_H

#include "figure.h"

class Octagon : Figure{

public:

size\_t VertexesNumber() override;

double Area() override;

void Print(std::ostream &os) override;

Octagon();

Octagon(Point a\_, Point b\_, Point c\_, Point d\_, Point e\_, Point f\_, Point g\_, Point h\_);

Octagon(std::istream &is);

Octagon(const Octagon &other);

virtual ~Octagon();

private:

Point a, b, c, d, e, f, g, h;

double SquareTriangle(Point a, Point b, Point c);

};

#endif //LAB1\_OCTAGON\_H

octagon.cpp

#include "octagon.h"

size\_t Octagon::VertexesNumber() {

return 8;

}

double Octagon::Area() {

return SquareTriangle(a, b, c) + SquareTriangle(a, c, d) + SquareTriangle(a, d, e)

+ SquareTriangle(a, e, f) + SquareTriangle(a, f, g) + SquareTriangle(a, g, h);

}

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

os << "Octagon: " << a << b << c << d << e << f << g << h << std::endl;

}

Octagon::Octagon() {}

Octagon::Octagon(Point a\_, Point b\_, Point c\_, Point d\_, Point e\_, Point f\_, Point g\_, Point h\_) : a(a\_), b(b\_),

c(c\_), d(d\_), e(e\_), f(f\_), g(g\_), h(h\_) {}

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

std::cout << "Enter data:" << std::endl;

is >> a >> b >> c >> d >> e >> f >> g >> h;

std::cout << "Octagon created via istream" << std::endl;

}

Octagon::Octagon(const Octagon &other) : Octagon(other.a, other.b, other.c, other.d,

other.e, other.f, other.g, other.h){

std::cout << "Made copy of hexagon" << std::endl;

}

Octagon::~Octagon() {

std::cout << "Octagon deleted" << std::endl;

}

double Octagon::SquareTriangle(Point a, Point b, Point c){

double p = (a.dist(b) + b.dist(c) + c.dist(a)) / 2;

return sqrt(p \* (p - a.dist(b)) \* (p - b.dist(c)) \* (p - c.dist(a)));

}

hexagon.h

#ifndef LAB1\_HEXAGON\_H

#define LAB1\_HEXAGON\_H

#include "figure.h"

class Hexagon : Figure{

public:

size\_t VertexesNumber() override;

double Area() override;

void Print(std::ostream &os) override;

Hexagon();

Hexagon(Point a\_, Point b\_, Point c\_, Point d\_, Point e\_, Point f\_);

Hexagon(std::istream &is);

Hexagon(const Hexagon &other);

virtual ~Hexagon();

private:

Point a, b, c, d, e, f;

double SquareTriangle(Point a, Point b, Point c);

};

#endif //LAB1\_HEXAGON\_H

hexagon.cpp

#include "hexagon.h"

size\_t Hexagon::VertexesNumber() {

return 6;

}

double Hexagon::Area() {

return SquareTriangle(a, b, c) + SquareTriangle(a, c, d) + SquareTriangle(a, d, e)

+ SquareTriangle(a, e, f);

}

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

os << "Hexagon: " << a << b << c << d << e << f << std::endl;

}

Hexagon::Hexagon() {}

Hexagon::Hexagon(Point a\_, Point b\_, Point c\_, Point d\_, Point e\_, Point f\_) : a(a\_), b(b\_),

c(c\_), d(d\_), e(e\_), f(f\_) {}

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

std::cout << "Enter data:" << std::endl;

is >> a >> b >> c >> d >> e >> f;

std::cout << "Hexagon created via istream" << std::endl;

}

Hexagon::Hexagon(const Hexagon &other) : Hexagon(other.a, other.b, other.c, other.d, other.e, other.f){

std::cout << "Made copy of hexagon" << std::endl;

}

Hexagon::~Hexagon() {

std::cout << "Hexagon deleted" << std::endl;

}

double Hexagon::SquareTriangle(Point a, Point b, Point c){

double p = (a.dist(b) + b.dist(c) + c.dist(a)) / 2;

return sqrt(p \* (p - a.dist(b)) \* (p - b.dist(c)) \* (p - c.dist(a)));

}

main.cpp

#include "pentagon.h"

#include "hexagon.h"

#include "octagon.h"

int main() {

//Pentagon

Pentagon p(std::cin);

p.Print(std::cout);

std::cout << "Number of vertex is " << p.VertexesNumber() << std::endl;

std::cout << "Area is " << p.Area() << std::endl;

std::cout << "---------------------------" << std::endl;

//Hexagon

Hexagon h(std::cin);

h.Print(std::cout);

std::cout << "Number of vertex is " << h.VertexesNumber() << std::endl;

std::cout << "Area is " << h.Area() << std::endl;

std::cout << "---------------------------" << std::endl;

//Octagon

Octagon o(std::cin);

o.Print(std::cout);

std::cout << "Number of vertex is " << o.VertexesNumber() << std::endl;

std::cout << "Area is " << o.Area() << std::endl;

std::cout << "---------------------------" << std::endl;

}