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: Морозов Артем Борисович, группа М80-208Б-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 №14:**

* + Фигура 1: Пятиугольник (Pentagon)
  + Фигура 2: Шестиугольник (Hexagon)
  + Фигура 3: Восьмиугольник (Octagon)

**Описание программы:**

Исходный код разделён на 10 файлов:

* point.h – описание класса точки
* point.cpp – реализация класса точки
* figure.h – описание класса фигуры
* pentagon.h – описание класса пятиугольника
* pentagon.cpp – реализация класса пятиугольника
* hexagon.h – описание класса шестиугольника
* hexagon.cpp – реализация класса шестиугольника
* octagon.h – описание класса восьмиугольника
* octagon.cpp – реализация класса восьмиугольника
* main.cpp – основная программа

**Дневник отладки:**

Программа в отладке не нуждалась, необходимый функционал был реализован довольно быстро и безошибочно.

**Вывод:**  
Данная лабораторная работа познакомила меня с оставшимися двумя из трех китов ООП: если с инкапсуляцией я уже знаком, то благодаря ЛР №3 я знаю, что такое полиморфизм и наследование. Достичь этого получилось при помощи реализации класса “Figure”. Дело в том, что от этого класса далее наследуются наши пятиугольники, шестиугольники и восьмиугольники. А полиморфизм достигается за счет виртуальных функций (ключевое слово **virtual**). Описав виртуальные методы **Print, Area, VertexesNumber,** мы автоматически позволили сами же себе реализовать эти методы в каждом классе многоугольников по-разному. В этом и заключается принцип полиморфизма в данной ЛР.

**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 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;

}

**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

**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  **pentagon.cpp:**

#include "pentagon.h"

#include <cmath>

Pentagon::Pentagon(std::istream &InputStream)

{

InputStream >> a;

InputStream >> b;

InputStream >> c;

InputStream >> d;

InputStream >> e;

std:: cout << "Pentagon that you wanted to create has been created" << std:: endl;

}

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

OutputStream << "Pentagon: ";

OutputStream << a << " " << b << " " << c << " " << d << " " << e << std:: endl;

}

size\_t Pentagon::VertexesNumber() {

size\_t number = 5;

return number;

}

double Pentagon::Area() {

double q = abs(a.X() \* b.Y() + b.X() \* c.Y() + c.X() \* d.Y() + d.X() \* e.Y() + e.X() \* a.Y() - b.X() \* a.Y() - c.X() \* b.Y() - d.X() \* c.Y() - e.X() \* d.Y() - a.X() \* e.Y());

double s = q / 2;

return s;

}

Pentagon::~Pentagon() {

std:: cout << "My friend, your pentagon has been deleted" << std:: endl;

}  **hexagon.h:**

#ifndef HEXAGON\_H

#define HEXAGON\_H

#include "figure.h"

#include <iostream>

class Hexagon : public Figure {

public:

Hexagon(std::istream &InputStream);

virtual ~Hexagon();

size\_t VertexesNumber();

double Area();

void Print(std::ostream &OutputStream);

private:

Point a;

Point b;

Point c;

Point d;

Point e;

Point f;

};

#endif **hexagon.cpp:**

#include "hexagon.h"

#include <cmath>

Hexagon::Hexagon(std::istream &InputStream)

{

InputStream >> a;

InputStream >> b;

InputStream >> c;

InputStream >> d;

InputStream >> e;

InputStream >> f;

std:: cout << "Hexagon that you wanted to create has been created" << std:: endl;

}

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

OutputStream << "Hexagon: ";

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

}

size\_t Hexagon::VertexesNumber() {

size\_t number = 6;

return number;

}

double Hexagon::Area() {

double q = abs(a.X() \* b.Y() + b.X() \* c.Y() + c.X() \* d.Y() + d.X() \* e.Y() + e.X() \* f.Y() + f.X() \* a.Y() - b.X() \* a.Y() - c.X() \* b.Y() - d.X() \* c.Y() - e.X() \* d.Y() - f.X() \* e.Y() - a.X() \* f.Y());

double s = q / 2;

return s;

}

Hexagon::~Hexagon() {

std:: cout << "My friend, your hexagon has been deleted" << std:: endl;

} **octagon.h:**#ifndef OCTAGON\_H

#define OCTAGON\_H

#include "figure.h"

#include <iostream>

class Octagon : public Figure {

public:

Octagon(std::istream &InputStream);

virtual ~Octagon();

size\_t VertexesNumber();

double Area();

void Print(std::ostream &OutputStream);

private:

Point a;

Point b;

Point c;

Point d;

Point e;

Point f;

Point g;

Point h;

};

#endif

**octagon.cpp:**

#include "octagon.h"

#include <cmath>

Octagon::Octagon(std::istream &InputStream)

{

InputStream >> a;

InputStream >> b;

InputStream >> c;

InputStream >> d;

InputStream >> e;

InputStream >> f;

InputStream >> g;

InputStream >> h;

std:: cout << "Octagon that you wanted to create has been created" << std:: endl;

}

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

OutputStream << "Octagon: ";

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

}

size\_t Octagon::VertexesNumber() {

size\_t number = 8;

return number;

}

double Octagon::Area() {

double q = abs(a.X() \* b.Y() + b.X() \* c.Y() + c.X() \* d.Y() + d.X() \* e.Y() + e.X() \* f.Y() + f.X() \* g.Y() + g.X() \* h.Y() + h.X() \* a.Y() - b.X() \* a.Y() - c.X() \* b.Y() - d.X() \* c.Y() - e.X() \* d.Y() - f.X() \* e.Y() - g.X() \* f.Y() - h.X() \* g.Y() - a.X() \* h.Y());

double s = q / 2;

return s;

}

Octagon::~Octagon() {

std:: cout << "My friend, your octagon has been deleted" << std:: endl;

}

**main.cpp**

#include <iostream>

#include "pentagon.h"

#include "hexagon.h"

#include "octagon.h"

int main () {

Pentagon a (std:: cin);

std:: cout << "The amount of vertices in your figure is : " << a.VertexesNumber() << std:: endl;

a.Print (std::cout);

std:: cout << "The area of your figure is : " << a.Area() << std:: endl;

Hexagon b (std:: cin);

std:: cout << "The amount of vertices in your figure is : " << b.VertexesNumber() << std:: endl;

b.Print (std:: cout);

std:: cout << "The area of your figure is : " << b.Area() << std:: endl;

Octagon c (std:: cin);

std:: cout << "The amount of vertices in your figure is : " << c.VertexesNumber() << std:: endl;

c.Print (std:: cout);

std:: cout << "The area of your figure is : " << c.Area() << std:: endl;

return 0;

}  
  
**Пример работы:|**

 