**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 *Москвин Артём Артурович, группа М80-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

**Условие**

Задание: Вариант 15: Шестиугольник, Восьмиугольник, 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 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/hexagon.h: описание класса шестиугольника, наследующегося от figures 5. include/octagon.h: описание класса восьмиугольника, наследующегося от figures 6. include/triangle.h: описание класса треугольника, наследующегося от figures

7. include/point.cpp: реализация класса точки

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

figures

9. include/octagon.cpp: реализация класса восьмиугольника, 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 <iostream>

class Figure {

public:

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

    virtual double Area() = 0;

    virtual size\_t VertexesNumber() = 0;

    virtual ~Figure() {};

};

#endif // FIGURE\_H

point.h

#ifndef POINT\_H

#define 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 // 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;

}

octagon.h

#ifndef OCTAGON\_H

#define OCTAGON\_H

#include "figure.h"

#include "point.h"

#include <iostream>

class Octagon : public Figure {

public:

    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();

    void Print(std::ostream& os);

    size\_t VertexesNumber();

    double TriangleArea();

    double Area();

private:

    Point A, B, C, D, E, F, G, H;

    size\_t PointCount = 6;

    std::string ClassName = "Octagon";

};

#endif //f OCTAGON\_H

octagon.cpp

#include "octagon.h"

#include "point.h"

#include <string>

#include "triangle.h"

#include <cmath>

Octagon::Octagon()

    : A(0.0, 0.0),

      B(0.0, 0.0),

      C(0.0, 0.0),

      D(0.0, 0.0),

      E(0.0, 0.0),

      F(0.0, 0.0),

      G(0.0, 0.0),

      H(0.0, 0.0) {

  std::cout << "Default octagon created" << std::endl;

}

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) {

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

}

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

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

  is >> A;

  is >> B;

  is >> C;

  is >> D;

  is >> E;

  is >> F;

  is >> G;

  is >> 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 octagon" << std::endl;

}

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

    os << ClassName << A << ", " << B << ", " << C << ", " << D << ", " << E << ", " << F << ", " << G << ", " << H << std::endl;

}

size\_t Octagon::VertexesNumber() {

  return PointCount;

}

double Octagon::Area() {

    return Triangle(A, B, C).Area() + Triangle(A, C, D).Area() + Triangle(A, D, E).Area() + Triangle(A, E, F).Area() + Triangle(A, F, G).Area() + Triangle(A, G, H).Area();

}

Octagon::~Octagon() {

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

}

hexagon.h

#ifndef HEXAGON\_H

#define HEXAGON\_H

#include "figure.h"

#include "point.h"

#include <iostream>

class Hexagon : public Figure {

public:

    Hexagon();

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

    Hexagon(std::istream &is);

    Hexagon(const Hexagon& other);

    virtual ~Hexagon();

    void Print(std::ostream& os);

    size\_t VertexesNumber();

    double TriangleArea();

    double Area();

private:

    Point A, B, C, D, E, F;

    size\_t PointCount = 6;

    std::string ClassName = "Hexagon";

};

#endif //f HEXAGON\_H

hexagon.cpp

#include "hexagon.h"

#include "point.h"

#include <string>

#include "triangle.h"

#include <cmath>

Hexagon::Hexagon()

    : A(0.0, 0.0),

      B(0.0, 0.0),

      C(0.0, 0.0),

      D(0.0, 0.0),

      E(0.0, 0.0),

      F(0.0, 0.0) {

  std::cout << "Default hexagon created" << std::endl;

}

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) {

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

}

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

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

  is >> A;

  is >> B;

  is >> C;

  is >> D;

  is >> E;

  is >> 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;

}

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

    os << ClassName << A << ", " << B << ", " << C << ", " << D << ", " << E << ", " << F << std::endl;

}

size\_t Hexagon::VertexesNumber() {

  return PointCount;

}

double Hexagon::Area() {

    return Triangle(A, B, C).Area() + Triangle(A, C, D).Area() + Triangle(A, D, F).Area() + Triangle(D, F, E).Area();

}

Hexagon::~Hexagon() {

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

}

triangle.h

#ifndef TRIANGLE\_H

#define TRIANGLE\_H

#include "figure.h"

#include "point.h"

#include <iostream>

class Triangle : public Figure {

public:

    Triangle();

    Triangle(Point a, Point b, Point c);

    Triangle(std::istream &is);

    Triangle(const Triangle& other);

    virtual ~Triangle();

    void Print(std::ostream& os);

    size\_t VertexesNumber();

    double Area();

private:

    Point A, B, C;

    size\_t PointCount = 3;

    std::string ClassName = "Triangle";

};

#endif // TRIANGLE\_H

triangle.cpp

#include "triangle.h"

#include "point.h"

#include <string>

#include <cmath>

Triangle::Triangle()

    : A(0.0, 0.0),

      B(0.0, 0.0),

      C(0.0, 0.0) {

  std::cout << "Default triangle created" << std::endl;

}

Triangle::Triangle(Point a, Point b, Point c)

    : A(a),

      B(b),

      C(c) {

  std::cout << "Triangle created" << std::endl;

}

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

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

  is >> A;

  is >> B;

  is >> C;

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

}

Triangle::Triangle(const Triangle& other)

    : Triangle(other.A, other.B, other.C) {

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

}

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

    os << ClassName << A << ", " << B << ", " << C << std::endl;

}

size\_t Triangle::VertexesNumber() {

  return PointCount;

}

double Triangle::Area() {

    double p = (A.dist(B) + B.dist(C) + C.dist(A)) / 2.0;

    return std::sqrt(p \* (p - A.dist(B)) \* (p - B.dist(C)) \* (p - C.dist(A)));

}

Triangle::~Triangle() {

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

}

main.cpp

#include <iostream>

#include "triangle.h"

#include "hexagon.h"

#include "octagon.h"

using namespace std;

int main()

{

    //Triangle

    Triangle TriangleA;

    TriangleA.Print(cout);

    std::cout << TriangleA.Area() << std::endl;

    Triangle TriangleB(Point(0.0, 0.0), Point(1.0, 1.0), Point(0.0, 1.0));

    TriangleB.Print(cout);

    std::cout << TriangleB.Area() << std::endl;

    Triangle TriangleC(cin);

    TriangleC.Print(cout);

    std::cout << TriangleC.Area() << std::endl;

    Triangle TriangleD(TriangleB);

    TriangleD.Print(cout);

    std::cout << TriangleD.Area() << std::endl;

    //Hexagon

    Hexagon HexagonA;

    HexagonA.Print(cout);

    std::cout << HexagonA.Area() << std::endl;

    Hexagon HexagonB(Point(0.0, 1.0), Point(1.0, 0.0), Point(2.0, 1.0), Point(2.0, 2.0), Point(1.0, 3.0), Point(0.0, 2.0));

    HexagonB.Print(cout);

    std::cout << HexagonB.Area() << std::endl;

    Hexagon HexagonC(cin);

    HexagonC.Print(cout);

    std::cout << HexagonC.Area() << std::endl;

    Hexagon HexagonD(HexagonB);

    HexagonD.Print(cout);

    std::cout << HexagonD.Area() << std::endl;

    //Octagon

    Octagon OctagonA;

    OctagonA.Print(cout);

    std::cout << OctagonA.Area() << std::endl;

    Octagon OctagonB(Point(0.0, 1.0), Point(0.0, 2.0), Point(3.0, 1.0), Point(3.0, 2.0), Point(2.0, 3.0), Point(1.0, 3.0), Point(0.0, 2.0), Point(0.0, 1.0));

    OctagonB.Print(cout);

    std::cout << OctagonB.Area() << std::endl;

    Octagon OctagonC(cin);

    OctagonC.Print(cout);

    std::cout << OctagonC.Area() << std::endl;

    Octagon OctagonD(OctagonB);

    OctagonD.Print(cout);

    std::cout << OctagonD.Area() << std::endl;

    return 0;

}