

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №3

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

Студент Морозов Артем Борисович, группа М80-208Б-20

Преподаватель Дорохов Евгений Павлович

Условие

Задание: Вариант 14: 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 расположены в отдельных 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 стандартного 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 C++. 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

```

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;
    }
```

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;

    }

```

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;

}

```

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;
}
```