**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 *Зинин Владислав Владимирович, группа М80-208Б-20*
Преподаватель *Дорохов Евгений Павлович*

**Условие**

Задание: Вариант 5: Rhombus, Hexagon, Pentagon. 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/rhombus.h: описание класса ромба, 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/rhombus.cpp: реализация класса ромба, 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 <iostream>
#include "point.h"

class Figure
{
public:
virtual ~Figure(){};
virtual double Area() = 0;
virtual void Print(std::ostream &os) = 0;
virtual size_t VertexesNumber() = 0;
};

                    #endif //FIGURE_H
```

### point.h

```
#ifndef POINT_H
#define POINT_H

#include <iostream>

class Point
{
public:
Point();
Point(double x, double y);
Point(std::istream &is);
double dist(Point &other);
friend double get_x(Point &other);
friend double get_y(Point &other);

friend std::istream& operator>>(std::istream& is, Point& p);
```

```
    friend std::ostream& operator<<(std::ostream& os, Point& p);

    private:
    double x_, y_;
};
;

#endif //POINT_H
```

**point.cpp**

```
#include <iostream>
#include "point.h"

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_;
}

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;
}

double get_x(Point &other)
{
return other.x_;
}

double get_y(Point &other)
{
return
other.y_;
```

}**pentagon.h**

```
#ifndef PENTAGON_H
#define PENTAGON_H
#include <iostream>
#include "figure.h"

class Pentagon : public Figure
{
public:
Pentagon(std::istream &is);
double Area();
void Print(std::ostream &os);
size_t VertexesNumber();

virtual ~Pentagon();
private:
Point a, b, c, d, e;
};
```

```
#endif // PENTAGON_H
```

**pentagon.cpp**

```
#include "pentagon.h"
#include <cmath>
#include <iostream>

Pentagon::Pentagon(std::istream &is)
{
is >> a;
is >> b;
is >> c;
is >> d;
```

```
is >> e;
}

void Pentagon::Print(std::ostream &os)
{
os << "Pentagon" << std::endl;
os << a << "," << b << "," << c << "," << d << "," << e << std::endl;
}

double Pentagon::Area()
{
return 0.5 * fabs(get_x(a)*get_y(b) + get_x(b)*get_y(c) + get_x(c)*get_y(d) + get_x(d)*get_y(e) + get_x(e)*get_y(a) - get_x(b)*get_y(a) - get_x(c)*get_y(b) - get_x(d)*get_y(c) - get_x(e)*get_y(d) - get_x(a)*get_y(e));
}

size_t Pentagon::VertexesNumber()
{
return 5;
}

Pentagon::~Pentagon()
{
std::cout << "Pentagon deleted" << std::endl;
}
```

## rhombus.h

```
#ifndef RHOMBUX_H
#define RHOMBUX_H
#include <iostream>
#include "point.h"
#include "figure.h"

class Rhombus : public Figure
{
```

```
public:
Rhombus(std::istream &is);
double Area();
void Print(std::ostream &os);
size_t VertexesNumber();
virtual ~Rhombus();

private:
Point a, b, c, d;
};

#endif //RHOMBUX_H
```

## rhombus.cpp

```
#include <iostream>
#include "rhombus.h"
#include <math.h>

Rhombus::Rhombus(std::istream &is)
{
is >> a;
is >> b;
is >> c;
is >> d;
}

void Rhombus::Print(std::ostream &os)
{
os << "Rhombus" << std::endl;
os << a << ',' << b << ',' << c << ',' << d << std::endl;
}

double Rhombus::Area()
```

```
{
return 0.5 * fabs(get_x(a)*get_y(b) + get_x(b)*get_y(c) + get_x(c)*get_y(d) + get_x(d)*get_y(a) - get_x(b)*get_y(a) - get_x(c)*get_y(b) - get_x(d)*get_y(c) - get_x(a)*get_y(d));
}

Rhombus::~Rhombus()
{
std::cout << "Rhombus deleted" << std::endl;
}

size_t Rhombus::VertexesNumber()
{
return 4;



}
```

## Hexagon.h

```
#ifndef HEXAGON_H
#define HEXAGON_H
#include <iostream>
#include "figure.h"

class Hexagon : public Figure
{
public:
Hexagon(std::istream &is);
double Area();
void Print(std::ostream &os);
size_t VertexesNumber();

virtual ~Hexagon();
private:
Point a, b, c, d, e, f;
};
```

```
#endif // HEXAGON_H
```

## hexagon.cpp

```
#include "hexagon.h"
#include <cmath>
#include <iostream>

Hexagon::Hexagon(std::istream &is)
{
is >> a;
is >> b;
is >> c;
is >> d;
is >> e;
is >> f;
}

void Hexagon::Print(std::ostream &os)
{
os << "Hexagon" << std::endl;
os << a << "," << b << "," << c << "," << d << "," << e << "," << f << std::endl;
}

size_t Hexagon::VertexesNumber()
{
return 6;
}

double Hexagon::Area()
{
return 0.5 * fabs(get_x(a)*get_y(b) + get_x(b)*get_y(c) + get_x(c)*get_y(d) + get_x(d)*get_y(e) + get_x(e)*get_y(f) + get_x(f)*get_y(a)
- get_x(b)*get_y(a) - get_x(c)*get_y(b) - get_x(d)*get_y(c) - get_x(e)*get_y(d) - get_x(f)*get_y(e) - get_x(a)*get_y(f));
}

Hexagon::~Hexagon()
```

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



        }
```

## main.cpp

```
#include <iostream>
#include "rhombus.h"
#include "pentagon.h"
#include "hexagon.h"

int main()
{
    Rhombus a(std::cin);
    std::cout << "Square = " << a.Area() << std::endl;
    a.Print(std::cout);

    Pentagon b(std::cin);
    std::cout << "Square = " << b.Area() << std::endl;
    b.Print(std::cout);

    Hexagon c(std::cin);
    std::cout << "Square = " << c.Area() << std::endl;
    c.Print(std::cout);

    return 0;
}
```