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: Куценко Борис Дмитриевич

Группа: М8О-207Б-20

Преподаватель: Дорохов Евгений Павлович

Москва, 2021

**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 №13:**

- Фигура 1: Ромб
- Фигура 2: Пятиугольник
- Фигура 3: Шестиугольник
- Стандартный контейнер: очередь

**Описание программы:**

Исходный код разделён на 10 файлов:

- point.h – описание класса точки
- point.cpp – реализация класса точки
- figure.h – описание класса фигуры
- rhombus.h – описание класса ромб (наследуется от фигуры)
- rhombus.cpp – реализация класса ромб
- pentagon.h – описание класса пятиугольник (наследуется от фигуры)
- pentagon.cpp – реализация класса пятиугольник
- hexagon.h – описание класса шестиугольник (наследуется от фигуры)
- hexagon.cpp – реализация класса шестиугольник
- main.cpp – основная программа

**Дневник отладки:**

Все было нормально.

**Вывод:**

В данной лабораторной работе я познакомился с принципами объектно- ориентированного программирования: инкапсуляцией, наследованием и полиморфизмом на примере работы с классами

геометрических фигур. Научился проектировать классы и работать с ними, а также поработал с конструкторами, деструкторами и виртуальными функциями в C++.

**Листинг программ:**

**point.h:**

```
#ifndef POINT_H
#define POINT_H

#include <iostream>

class Point {
public:
    Point();
    Point(std::istream&);
    Point(double x, double y);
    Point(const Point& other);

    double dist(Point& other);
    friend std::istream& operator>>(std::istream& is, Point& p);
    friend std::ostream& operator<<(std::ostream&, const Point& p);
    bool operator==(const Point& other);

public:
    double x_;
    double y_;
};

#endif
```

**point.cpp:**

```
#include "point.h"
#include <iostream>
#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_;
}
Point::Point(const Point& other) : x_(other.x_), y_(other.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;
}
bool Point::operator==(const Point& other) {
    return (x_ == other.x_ && y_ == other.y_);
}
```

**figure.h:**

```
#ifndef FIGURE_H
#define FIGURE_H
#include "point.h"
#include <iostream>
#include <cmath>

class Figure {

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


#endif
```

**rhombus.h:**

```
#ifndef RHOMBUS_H
#define RHOMBUS_H

#include "figure.h"


class Rhombus : public Figure {
public:
    Rhombus();
    virtual ~Rhombus();
    Rhombus(std::istream &in);
    Rhombus(const Rhombus& r);
    Rhombus(Point& x1, Point& x2, Point& x3, Point& x4);
    double Area();
    size_t VertexesNumber();
    bool IsRhombus() ;
    bool operator==(const Rhombus& r);
```

```
    Rhombus& operator=(const Rhombus& r);
    void Print(std::ostream& os);
    friend std::ostream& operator<<(std::ostream &os, const Rhombus& r);
    friend std::istream& operator>> (std::istream &in, Rhombus &r);

protected:
    Point _x1, _x2, _x3, _x4;
};

#endif
```

**trapezoid.cpp:**

```
#include "rhombus.h"
#include <string.h>


Rhombus::Rhombus(): _x1(0, 0), _x2(0, 0), _x3(0, 0), _x4(0, 0) {}
Rhombus::~Rhombus() {}

double Rhombus::Area() {
    return 0.5 *_x1.dist(_x3) * _x2.dist(_x4);
}

bool Rhombus::IsRhombus() {
    if (_x1.dist(_x2) == _x2.dist(_x3) && _x2.dist(_x3) == _x3.dist(_x4) &&
    _x3.dist(_x4) == _x4.dist(_x1) && _x4.dist(_x1) == _x1.dist(_x2))
        return true;
    return false;
}
Rhombus::Rhombus(Point &x1, Point &x2, Point &x3, Point &x4) : _x1(x1), _x2(x2), _x3(x3),
_x4(x4){
    if(!IsRhombus()) {
        std::cout << "ERORR:it isn't rhombus, incorrect input\n";
        exit(-1);
    }
}

size_t Rhombus::VertexesNumber() {
    return 4;
}


void Rhombus::Print(std::ostream& os) {
    os << "Rhombus: (" << _x1.x_ << ", " << _x1.y_ << ") " << '(' << _x2.x_ << ", " <<
_x2.y_ << ") "
    << '(' << _x3.x_ << ", " << _x3.y_ << ") " << '(' << _x4.x_ << ", " << _x4.y_   << ")"
<< std::endl;
}

std::ostream& operator<<(std::ostream &os, const Rhombus& r)
{
```

```
    os << "Rhombus: (" << r._x1.x_ << ", " << r._x1.y_ << ") " << '(' << r._x2.x_ << ", "
<< r._x2.y_ << ") "
       << '(' << r._x3.x_ << ", " << r._x3.y_ << ") " << '(' << r._x4.x_ << ", " <<
r._x4.y_ << ")" << std::endl;
    return os;
}

std::istream &operator>>(std::istream &in, Rhombus &r) {
    in >> r._x1.x_ >> r._x1.y_>> r._x2.x_ >> r._x2.y_ >> r._x3.x_ >> r._x3.y_ >> r._x4.x_
>> r._x4.y_;
    if(!r.IsRhombus()) {
        std::cout << "ERORR:it isn't rhombus, incorrect input\n";
        exit(-1);
    }
    return in;
}

Rhombus::Rhombus(const Rhombus &r) : _x1(r._x1), _x2(r._x2), _x3(r._x3), _x4(r._x4) {}

Rhombus::Rhombus(std::istream &in) {
    in >> _x1.x_ >> _x1.y_ >> _x2.x_ >> _x2.y_ >> _x3.x_ >> _x3.y_ >> _x4.x_ >> _x4.y_;
    if (!IsRhombus())  {
        std::cout << "ERORR:it isn't rhombus, incorrect input\n";
        exit(-1);
    }
}

Rhombus &Rhombus::operator=(const Rhombus &r) {
    if (&r == this)
        return *this;
    _x1.x_ = r._x1.x_;
    _x1.y_ = r._x1.y_;
    _x2.x_ = r._x2.x_;
    _x2.y_ = r._x2.y_;
    _x3.x_ = r._x3.x_;
    _x3.y_ = r._x3.y_;
    _x4.x_ = r._x4.x_;
    _x4.y_ = r._x4.y_;
    return *this;

}

bool Rhombus::operator==(const Rhombus &r)  {
    return _x1 == r._x1 && _x2 == r._x2 && _x3 == r._x3 && _x4 == r._x4;
}
```

**pentagon.h:**

```
#ifndef PENTAGON_H
#define PENTAGON_H
```

```
#include "figure.h"

class Pentagon : public Figure {
public:
    Pentagon();
    virtual ~Pentagon();
    Pentagon(std::istream &in);
    Pentagon(const Pentagon&);
    Pentagon(Point& x1_, Point& x2_, Point& x3_, Point& x4_, Point &x5_);
    double Area();
    size_t VertexesNumber();
    bool operator==(const Pentagon& p);

    Pentagon& operator=(const Pentagon& p);
    void Print(std::ostream& os);
    friend std::ostream& operator<<(std::ostream &out, const Pentagon& r);
    friend std::istream& operator>> (std::istream &in, Pentagon&r);

protected:
    Point x1, x2, x3, x4, x5;
};

#endif
```

**pentagon.cpp:**

```
#include "pentagon.h"

Pentagon::Pentagon() :x1(0, 0),x2(0, 0),x3(0, 0), x4(0, 0), x5(0, 0) {}

size_t Pentagon::VertexesNumber() {
    return 5;
}
Pentagon::Pentagon(const Pentagon& r) : x1(r.x1), x2(r.x2), x3(r.x3), x4(r.x4), x5(r.x5)
{}
Pentagon::Pentagon(Point& x1_, Point& x2_, Point& x3_, Point& x4_, Point &x5_): x1(x1_),
x2(x2_), x3(x3_), x4(x4_), x5(x5_) {}
double triangleArea(double a, double b, double c) {
    double p = (a + b + c) / 2;
    double s = std::sqrt(p * (p - a) * (p - b) * (p - c));
    return s;
}
double Pentagon::Area() {
    return triangleArea(x1.dist(x2), x2.dist(x3), x1.dist(x3)) +
           triangleArea(x1.dist(x3), x3.dist(x4), x1.dist(x4)) +
           triangleArea(x1.dist(x4), x4.dist(x5), x1.dist(x5));
}

std::ostream &operator<<(std::ostream &os, const Pentagon &r) {
    os << "Pentagon: (" << r.x1.x_ << ", " << r.x1.y_ << ") "
       << '(' <<r.x2.x_ << ", " << r.x2.y_ << ") "
       << '(' <<r.x3.x_ << ", " << r.x3.y_ << ") "
```

```
       << '(' <<r.x4.x_ << ", " << r.x4.y_ << ") "
       << '(' <<r.x5.x_ << ", " << r.x5.y_ << ") "
       << std::endl;
    return os;
}

std::istream &operator>>(std::istream &in, Pentagon &r) {
    in >> r.x1.x_ >> r.x1.y_>> r.x2.x_ >> r.x2.y_ >> r.x3.x_ >> r.x3.y_ >> r.x4.x_ >>
r.x4.y_ >> r.x5.x_ >> r.x5.y_;
    return in;
}

void Pentagon::Print(std::ostream& os) {
    os << "Pentagon: ("<< x1.x_ << ", " << x1.y_ << ") "
       << '(' <<x2.x_ << ", " << x2.y_ << ") "
       << '(' <<x3.x_ << ", " << x3.y_ << ") "
       << '(' <<x4.x_ << ", " << x4.y_ << ") "
       << '(' <<x5.x_ << ", " << x5.y_ << ") "
       << std::endl;
}

Pentagon::Pentagon(std::istream &in) {
    in >> x1.x_ >> x1.y_>> x2.x_ >> x2.y_ >> x3.x_ >> x3.y_ >> x4.x_ >> x4.y_ >> x5.x_ >>
x5.y_;
}

Pentagon &Pentagon::operator=(const Pentagon &p) {
    if(&p == this)
        return *this;
    x1 = p.x1;
    x2 = p.x2;
    x3 = p.x3;
    x4 = p.x4;
    x5 = p.x5;
    return *this;
}

bool Pentagon::operator==(const Pentagon &p) {
    return x1 == p.x1 && x2 == p.x2 && x3 == p.x3 && x4 == p.x4 && x5 == p.x5;
}

Pentagon::~Pentagon() {

}
```

**hexagon.h:**

```
#ifndef HEXAGON_H
#define HEXAGON_H

#include "figure.h"


class Hexagon : public Figure {
```

```
public:
    Hexagon();
    virtual ~Hexagon();
    Hexagon(std::istream &in);
    Hexagon(const Hexagon& r);
    Hexagon(Point& x1_, Point& x2_, Point& x3_, Point& x4_, Point &x5_, Point &x6_);
    double Area();
    size_t VertexesNumber();
    bool operator==(const Hexagon& h);

    Hexagon& operator=(const Hexagon& h);
    void Print(std::ostream& os);
    friend std::ostream& operator<<(std::ostream &out, const Hexagon& r);
    friend std::istream& operator>> (std::istream &in, Hexagon& r);

protected:
    Point x1, x2, x3, x4, x5, x6;
};

#endif
```

**Hexagon.cpp:**

```
#include "hexagon.h"

Hexagon::Hexagon() :x1(0, 0),x2(0, 0),x3(0, 0), x4(0, 0), x5(0, 0), x6(0, 0) {}

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

Hexagon::Hexagon(const Hexagon& r) : x1(r.x1), x2(r.x2), x3(r.x3), x4(r.x4), x5(r.x5),
x6(r.x6) {}
Hexagon::Hexagon(Point& x1_, Point& x2_, Point& x3_, Point& x4_, Point &x5_, Point &x6_):
    x1(x1_), x2(x2_), x3(x3_), x4(x4_), x5(x5_), x6(x6_){}

double TriangleArea(double a, double b, double c) {
    double p = (a + b + c) / 2;
    double s = std::sqrt(p * (p - a) * (p - b) * (p - c));
    return s;
}

double Hexagon::Area() {
    return TriangleArea(x1.dist(x2), x2.dist(x3), x1.dist(x3)) +
           TriangleArea(x1.dist(x3), x3.dist(x4), x1.dist(x4)) +
           TriangleArea(x1.dist(x4), x4.dist(x5), x1.dist(x5)) +
           TriangleArea(x1.dist(x5), x5.dist(x6), x1.dist(x6));
}

std::ostream &operator<<(std::ostream &os, const Hexagon &r) {
    os << "Hexagon: (" << r.x1.x_ << ", " << r.x1.y_ << ") "
       << '(' <<r.x2.x_ << ", " << r.x2.y_ << ") "
       << '(' <<r.x3.x_ << ", " << r.x3.y_ << ") "
       << '(' <<r.x4.x_ << ", " << r.x4.y_ << ") "
```

```
       << '(' <<r.x5.x_ << ", " << r.x5.y_ << ") "
       << '(' <<r.x6.x_ << ", " << r.x6.y_ << ") "
       << std::endl;
    return os;
}

std::istream &operator>>(std::istream &in, Hexagon &r) {
    in >> r.x1.x_ >> r.x1.y_>> r.x2.x_ >> r.x2.y_ >> r.x3.x_ >>
    r.x3.y_ >> r.x4.x_ >> r.x4.y_ >> r.x5.x_ >> r.x5.y_ >> r.x6.x_ >> r.x6.y_;
    return in;
}

void Hexagon::Print(std::ostream& os) {
    os << "Hexagon: (" << x1.x_ << ", " << x1.y_ << ") "
       << '(' <<x2.x_ << ", " << x2.y_ << ") "
       << '(' <<x3.x_ << ", " << x3.y_ << ") "
       << '(' <<x4.x_ << ", " << x4.y_ << ") "
       << '(' <<x5.x_ << ", " << x5.y_ << ") "
       << '(' <<x6.x_ << ", " << x6.y_ << ") "
       << std::endl;
}

Hexagon::Hexagon(std::istream &in) {
    in >> x1.x_ >> x1.y_>> x2.x_ >> x2.y_ >> x3.x_ >>
    x3.y_ >> x4.x_ >> x4.y_ >> x5.x_ >> x5.y_ >> x6.x_ >> x6.y_;
}

Hexagon &Hexagon::operator=(const Hexagon &p) {
    if(&p == this)
        return *this;
    x1 = p.x1;
    x2 = p.x2;
    x3 = p.x3;
    x4 = p.x4;
    x5 = p.x5;
    x6 = p.x6;
    return *this;
}

bool Hexagon::operator==(const Hexagon &p) {
    return x1 == p.x1 && x2 == p.x2 && x3 == p.x3 && x4 == p.x4 && x5 == p.x5 && x6 ==
p.x6;
}

Hexagon::~Hexagon() {

}
```

**main.cpp:**

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

```
#include <queue>
int main()
{
    char c;
    std::queue <Rhombus> rhomb;
    std::queue <Pentagon> pent;
    std::queue <Hexagon> hex;
    std::cout << "Press '?' for help:\n";
    while ((c = getchar()) != EOF) {
        if (c == '?') {
            std::cout << "U can:\n";
            std::cout << "press r -- Play with Rhombus\n";
            std::cout << "press p -- Play with Pentagon\n";
            std::cout << "press h -- Play with Hexagon\n";
            std::cout << "press e -- Exit\n";
        }
        else if (c == 'r') {
            std::cout << "Rhombus Mode...\nUse coodinates. Type of points - double\n";
            Rhombus a(std::cin);
            std::cout << "Area = " << a.Area() << std::endl;
            std::cout << "Vertex Number = " << a.VertexesNumber() << std::endl;
            a.Print(std::cout);
            std::cout << "Complete, press next button...\n";
            rhomb.push(a);
        }
        else if (c == 'p') {
            std::cout << "Pentagon Mode...\nUse coodinates. Type of points - double\n";
            Pentagon b(std::cin);
            std::cout << "Area = " << b.Area() << std::endl;
            std::cout << "Vertex Number = " << b.VertexesNumber() << std::endl;
            b.Print(std::cout);
            std::cout << "Complete, press next button...\n";
            pent.push(b);
        }
        else if (c == 'h') {
            std::cout << "Hexagon Mode...\nUse coodinates. Type of points - double\n";
            Hexagon c(std::cin);
            std::cout << "Area = " << c.Area() << std::endl;
            std::cout << "Vertex Number = " << c.VertexesNumber() << std::endl;
            c.Print(std::cout);
            std::cout << "Complete, press next button...\n";
            hex.push(c);
        }
        else if (c == 'e') {
            std:: cout << "\n*****************************************************\n";
            std::cout << "Program LOG\n";
            std::cout << "_____________________________________________________\n";
            Rhombus c;
            std::cout << "Rhombuses:" << std::endl;
            int count = 0;
            while(!rhomb.empty()) {
                ++count;
                std::cout << count << '.';
                c = rhomb.front();
```

```
                c.Print(std::cout);
                std::cout <<"Area = " << c.Area() << std::endl;
                rhomb.pop();
            }
            std::cout << "\nPentagons:" << std::endl;
            Pentagon d;
            count = 0;
            while(!pent.empty()) {
                ++count;
                d = pent.front();
                std::cout << count << '.';
                d.Print(std::cout);
                std::cout <<"Area = " << d.Area() << std::endl;
                pent.pop();
            }
            Hexagon e;
            count = 0;
            std::cout << "\nHexagons:" << std::endl;
            while(!hex.empty()) {
                ++count;
                e = hex.front();
                std::cout << count << '.';
                e.Print(std::cout);
                std::cout << "Area = " << e.Area() << std::endl;
                hex.pop();
            }
            std::cout << "_____________________________________________________\n";
            std::cout << "End session..." << std::endl;
            return 0;
        }
        else if (c != ' ' && c != '\n' && c != '\t') {
            std::cout << "Unexpectable simbol\n";
        }
    }
return 0;
}
```

## Пример работы:

```
Press '?' for help:
?
U can:
press r -- Play with Rhombus
press p -- Play with Pentagon
press h -- Play with Hexagon
press e -- Exit
r
Rhombus Mode...
Use coodinates. Type of points - double
0 3
3 0
0 -3
-3 0
Area = 18
Vertex Number = 4
```

```
Rhombus: (0, 3) (3, 0) (0, -3) (-3, 0)
Complete, press next button...
p
Pentagon Mode...
Use coodinates. Type of points - double
0 0
-1 4
2 6
4 4
4 0
Area = 23
Vertex Number = 5
Pentagon: (0, 0) (-1, 4) (2, 6) (4, 4) (4, 0)
Complete, press next button...
h
Hexagon Mode...
Use coodinates. Type of points - double
0 0
-1 4
2 6
4 4
4 0
3 -4
Area = 31
Vertex Number = 6
Hexagon: (0, 0) (-1, 4) (2, 6) (4, 4) (4, 0) (3, -4)
Complete, press next button...
e

**************************************************
Program LOG
_____________________________________________________
Rhombuses:
1.Rhombus: (0, 3) (3, 0) (0, -3) (-3, 0)
Area = 18

Pentagons:
1.Pentagon: (0, 0) (-1, 4) (2, 6) (4, 4) (4, 0)
Area = 23

Hexagons:
1.Hexagon: (0, 0) (-1, 4) (2, 6) (4, 4) (4, 0) (3, -4)
Area = 31
_____________________________________________________
```