

Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

**Лабораторная работа
по курсу «ООП»**

**Тема:
Простые классы.**

Студент:	Коротков Д.П.
Группа:	М80-208Б-18
Преподаватель:	Журавлев А.А.
Вариант:	9
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

Polygon.hpp:

```
#ifndef __POLYGON__
#define __POLYGON__

#include <iostream>
#include <array>

struct Point {
    double x;
    double y;
    Point() {x = 0; y = 0;}
    Point(double a, double b) {x = a; y = b;}
};

struct Polygon {
public:
    virtual ~Polygon() {}
    virtual void print (std::ostream& os) const = 0;
    virtual void scan (std::istream& is) = 0;
    virtual Point center() const = 0;
    virtual double area() const = 0;
};

struct Triangle : public Polygon {
public:
    Triangle ();
    Triangle (std::istream& is);
    void print (std::ostream& os) const override;
    void scan (std::istream& is) override;
    Point center() const override;
    double area() const override;
    ~Triangle() {}
private:
    std::array<Point, 3> vertexes;
};

struct Square : public Polygon {
public:
    Square ();
    Square (std::istream& is);
```

```

    void print (std::ostream& os) const override;
    void scan (std::istream& is) override;
    Point center() const override;
    double area() const override;
    ~Square() {}
private:
    std::array<Point, 4> vertexes;
};

```

```

struct Rectangle : public Polygon {
public:
    Rectangle ();
    Rectangle (std::istream& is);
    void print (std::ostream& os) const override;
    void scan (std::istream& is) override;
    Point center() const override;
    double area() const override;
    ~Rectangle() {}
private:
    std::array<Point, 4> vertexes;
};

```

```

std::istream& operator>> (std::istream& is, Polygon& m);
std::ostream& operator<< (std::ostream& os, const Polygon& m);

```

```

#endif

```

Polygon.cpp:

```

#include <iostream>
#include <vector>
#include <array>
#include <math.h>
#include <exception>
#include "Polygon.hpp"

```

```

void Triangle::print (std::ostream& os) const {
    os << "Triangle" << ':';
    for (int i = 0; i < vertexes.size(); ++i) {
        os << vertexes[i].x << ' ' << vertexes[i].y << ' ';
    }
}

```

```

    os << '\n';
}

```

```

Point Triangle::center() const {
    Point ans = {0, 0};
    for (int i = 0; i < 3; ++i) {
        ans.x += vertexes[i].x;
        ans.y += vertexes[i].y;
    }
    ans = {ans.x / 3, ans.y / 3};
    return ans;
}

```

```

double Triangle::area() const {
    Point vec1 = {vertexes[1].x - vertexes[0].x, vertexes[1].y - vertexes[0].y};
    Point vec2 = {vertexes[2].x - vertexes[0].x, vertexes[2].y - vertexes[0].y};
    double ans = fabs(vec1.x * vec2.y - vec2.x * vec1.y);
    return ans / 2;
}

```

```

Triangle::Triangle (std::istream& is) {
    for (int i = 0; i < vertexes.size(); ++i) {
        is >> vertexes[i].x >> vertexes[i].y;
    }
    if (area() == 0) {
        throw std::logic_error("non pravilni");
    }
}

```

```

void Triangle::scan (std::istream& is) {
    for (int i = 0; i < vertexes.size(); ++i) {
        is >> vertexes[i].x >> vertexes[i].y;
    }
    if (area() == 0) {
        throw std::logic_error("non pravilni");
    }
}

```

```

void Square::print (std::ostream& os) const {
    os << "Square" << ' ';
}

```

```

for (int i = 0; i < vertexes.size(); ++i) {
    os << vertexes[i].x << ' ' << vertexes[i].y << ' ';
}
os << '\n';
}

```

```

Point Square::center() const {
    Point ans = {0, 0};
    for (int i = 0; i < vertexes.size(); ++i) {
        ans.x += vertexes[i].x;
        ans.y += vertexes[i].y;
    }
    ans = {ans.x / 4, ans.y / 4};
    return ans;
}

```

```

double Square::area() const {
    Point vec1 = {vertexes[1].x - vertexes[0].x, vertexes[1].y - vertexes[0].y};
    Point vec2 = {vertexes[2].x - vertexes[0].x, vertexes[2].y - vertexes[0].y};
    double ans = fabs(vec1.x * vec2.y - vec2.x * vec1.y);
    std::cout << ans << '\n';
    return ans;
}

```

```

Square::Square (std::istream& is) {
    for (int i = 0; i < 4; ++i) {
        is >> vertexes[i].x >> vertexes[i].y;
    }
    //for (int i = 0; i < 4; ++i) {
        //std::cout << vertexes[i].x << ' ' << vertexes[i].y << '\n';
    //}
    Point vec1 {vertexes[1].x - vertexes[0].x, vertexes[1].y - vertexes[0].y};
    Point vec2 {vertexes[2].x - vertexes[1].x, vertexes[2].y - vertexes[1].y};
    Point vec3 {vertexes[3].x - vertexes[2].x, vertexes[3].y - vertexes[2].y};
    Point vec4 {vertexes[0].x - vertexes[3].x, vertexes[0].y - vertexes[3].y};
    double l1 = vec1.x * vec1.x + vec1.y * vec1.y;
    double l2 = vec2.x * vec2.x + vec2.y * vec2.y;
    double l3 = vec3.x * vec3.x + vec3.y * vec3.y;
    double l4 = vec4.x * vec4.x + vec4.y * vec4.y;
    //std::cout << vec1.x << ' ' << vec1.y << '\n';
    //std::cout << l1 << ' ' << l2 << ' ' << l3 << ' ' << l4;
}

```

```

    if (!(l1 == l2 && l2 == l3 && l3 == l4) || !(vec1.x * vec2.x + vec1.y * vec2.y ==
0)) {
        throw std::logic_error("non pravilni");
    }
}

```

```

void Square::scan (std::istream& is) {
    for (int i = 0; i < 4; ++i) {
        is >> vertexes[i].x >> vertexes[i].y;
    }
    Point vec1 {vertexes[1].x - vertexes[0].x, vertexes[1].y - vertexes[0].y};
    Point vec2 {vertexes[2].x - vertexes[1].x, vertexes[2].y - vertexes[1].y};
    Point vec3 {vertexes[3].x - vertexes[2].x, vertexes[3].y - vertexes[2].y};
    Point vec4 {vertexes[0].x - vertexes[3].x, vertexes[0].y - vertexes[3].y};
    double l1 = vec1.x * vec1.x + vec1.y * vec1.y;
    double l2 = vec2.x * vec2.x + vec2.y * vec2.y;
    double l3 = vec3.x * vec3.x + vec3.y * vec3.y;
    double l4 = vec4.x * vec4.x + vec4.y * vec4.y;
    if (!(l1 == l2 && l2 == l3 && l3 == l4) || !(vec1.x * vec2.x + vec1.y * vec2.y ==
0)) {
        throw std::logic_error("non pravilni");
    }
}

```

```

void Rectangle::print (std::ostream& os) const {
    os << "Rectangle" << ':';
    for (int i = 0; i < vertexes.size(); ++i) {
        os << vertexes[i].x << ' ' << vertexes[i].y << ' ';
    }
    os << '\n';
}

```

```

Point Rectangle::center() const {
    Point ans = {0, 0};
    for (int i = 0; i < vertexes.size(); ++i) {
        ans.x += vertexes[i].x;
        ans.y += vertexes[i].y;
    }
    ans = {ans.x / 4, ans.y / 4};
    return ans;
}

```

```

double Rectangle::area() const {
    Point vec1 = {vertexes[1].x - vertexes[0].x, vertexes[1].y - vertexes[0].y};
    Point vec2 = {vertexes[2].x - vertexes[0].x, vertexes[2].y - vertexes[0].y};
    double ans = fabs(vec1.x * vec2.y - vec2.x * vec1.y);
    return ans;
}

```

```

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

    for (int i = 0; i < vertexes.size(); ++i) {
        is >> vertexes[i].x >> vertexes[i].y;
    }
    Point vec1 = {vertexes[1].x - vertexes[0].x, vertexes[1].y - vertexes[0].y};
    Point vec2 = {vertexes[2].x - vertexes[1].x, vertexes[2].y - vertexes[1].y};
    Point vec3 = {vertexes[3].x - vertexes[2].x, vertexes[3].y - vertexes[2].y};
    Point vec4 = {vertexes[0].x - vertexes[3].x, vertexes[0].y - vertexes[3].y};
    double l1 = vec1.x * vec1.x + vec1.y * vec1.y;
    double l2 = vec2.x * vec2.x + vec2.y * vec2.y;
    double l3 = vec3.x * vec3.x + vec3.y * vec3.y;
    double l4 = vec4.x * vec4.x + vec4.y * vec4.y;
    if (!(l1 == l3 && l2 == l4) || !(vec1.x * vec2.x + vec1.y * vec2.y == 0)) {
        throw std::logic_error("non pravilni");
    }
}

```

```

void Rectangle::scan (std::istream& is) {
    for (int i = 0; i < vertexes.size(); ++i) {
        is >> vertexes[i].x >> vertexes[i].y;
    }
    Point vec1 = {vertexes[1].x - vertexes[0].x, vertexes[1].y - vertexes[0].y};
    Point vec2 = {vertexes[2].x - vertexes[1].x, vertexes[2].y - vertexes[1].y};
    Point vec3 = {vertexes[3].x - vertexes[2].x, vertexes[3].y - vertexes[2].y};
    Point vec4 = {vertexes[0].x - vertexes[3].x, vertexes[0].y - vertexes[3].y};
    double l1 = vec1.x * vec1.x + vec1.y * vec1.y;
    double l2 = vec2.x * vec2.x + vec2.y * vec2.y;
    double l3 = vec3.x * vec3.x + vec3.y * vec3.y;
    double l4 = vec4.x * vec4.x + vec4.y * vec4.y;
    if (!(l1 == l3 && l2 == l4) || !(vec1.x * vec2.x + vec1.y * vec2.y == 0)) {
        throw std::logic_error("non pravilni");
    }
}

```

```
Triangle::Triangle () {
    Point a = {0, 0};
    for (int i = 0; i < 3; ++i) {
        vertexes[i] = a;
    }
}
```

```
Square::Square () {
    Point a = {0, 0};
    for (int i = 0; i < 4; ++i) {
        vertexes[i] = a;
    }
}
```

```
Rectangle::Rectangle () {
    Point a = {0, 0};
    for (int i = 0; i < 4; ++i) {
        vertexes[i] = a;
    }
}
```

```
std::istream& operator>> (std::istream& is, Polygon& m) {
    m.scan(is);
    return is;
}
```

```
std::ostream& operator<< (std::ostream& os, const Polygon& m) {
    m.print(os);
    return os;
}
```

main.cpp:

```
#include <iostream>
#include <vector>
#include <array>
#include <math.h>
#include "Polygon.hpp"
```



```

int main() {
    std::vector<Polygon*> data;
    char st;
    char figure;
    int ind;
    Polygon* f;
    std::cout.precision(3);
    while (1) {
        std::cin >> st;
        if (st == 'q') {
            break;
        } else if (st == 'a') {
            std::cin >> figure;
            if (figure == 't') {
                f = new Triangle(std::cin);
                data.push_back(f);
            } else if (figure == 's') {
                f = new Square(std::cin);
                data.push_back(f);
            } else if (figure == 'r') {
                f = new Rectangle(std::cin);
                data.push_back(f);
            } else {
                std::cout << "invalid command" << '\n';
            }
        } else if (st == 'd') {
            std::cin >> ind;
            delete(data[ind]);
            data.erase(data.begin() + ind);
        } else if (st == 'p') {
            std::cin >> ind;
            data[ind]->print(std::cout);
        } else if (st == 's') {
            std::cin >> ind;
            std::cout << data[ind]->area() << '\n';
        } else if (st == 'c') {
            std::cin >> ind;
            Point cntr = data[ind]->center();
            std::cout << cntr.x << ' ' << cntr.y << '\n';
        } else {
            std::cout << "invalid command" << '\n';
        }
    }
    for (int i = 0; i < data.size(); ++i) {

```

```
        delete data[i];
    }
}
```

CmakeLists.txt:

```
cmake_minimum_required(VERSION 3.2)
```

```
project(lab3)
```

```
add_executable(lab3
    main.cpp
    Polygon.cpp
)
```

```
set_property(TARGET lab3 PROPERTY CXX_STANDARD 11)
```

test.sh:

```
#!/usr/bin/env bash
```

```
executable=$1
```

```
for file in test_?.test
do
    $executable < $file > tmp
    if cmp tmp ${file%%.test}.ans
    then
        echo Test "$file": SUCCESS
    else
        echo Test "$file": FAIL
    fi
    rm tmp
done
```

2. Ссылка на репозиторий на GitHub.

https://github.com/KorotkovDenis/oop_exercise_02

3. Набор testcases.

test_01.test:

```
a r
0 0 2 0 2 1 0 1
s 0
c 0
```

```
q
test_02.test:
a t
0 0 0 1 1 1
s 0
c 0
d 0
a s
0 0 1 0 1 1 0 1
s 0
c 0
```

4. Результаты выполнения тестов.

```
test_01.ans:
2
1 0.5
test_02.ans:
0.5
0.333 0.667
1
0.5 0.5
```

5. Вывод.

Выполняя данную лабораторную я получил опыт работы с наследованием и полиморфизмом классов в C++. Создал класс, соответствующий варианту моего задания, реализовал для него операции площади и средней точки.