

Московский авиационный институт  
(Национальный исследовательский университет)  
Факультет "Информационные технологии и прикладная математика"

Лабораторная работа №3 по курсу "Объектно-ориентированное программирование"

*Студент:* Хисамутдинов Д.С.

*Группа:* М8О-208Б *Преподаватель:*

Журавлев А.А.

*Вариант:* 25

*Оценка: Дата:*

---

Москва

# 1 Исходный код

## Figures.hpp

```

1 #pragma once
2
3 #include <iostream>
4 #include <vector>
5
6     struct Point {
7         double x, y;
8     };
9
10 double calculateDistance(const Point& lhs, const Point& rhs);
11 bool operator<(const Point& lhs, const Point& rhs);
12 std::istream& operator>>(std::istream& is, Point& p);
13 std::ostream& operator<<(std::ostream& os, const Point& p);
14 std::ostream& operator<<(std::ostream& os, const std::vector<Point> v);
15
16 class Figure {
17 public:
18     virtual Point Center() const = 0;
19     virtual double Square() const = 0;
20     virtual void Print(std::ostream& os) const = 0;
21     virtual ~Figure() = default;
22 };
23
24 class Rhombus : public Figure {
25 public:
26     Rhombus(const Point& p1, const Point& p2, const Point& p3, const Point& p4);
27     Point Center() const override;
28     double Square() const override;
29     void Print(std::ostream& os) const override;
30 private:
31     std::vector<Point> points;
32     double smallerDiagonal, biggerDiagonal;
33 };
34
35 class Pentagon : public Figure {
36 public:
37     Pentagon(const Point& p1, const Point& p2, const Point& p3, const Point& p4, const Point& p5);
38     Point Center() const override;
39     double Square() const override;
40     void Print(std::ostream& os) const override;
41 private:
42     std::vector<Point> points;
43 };
44
45 class Hexagon : public Figure {
46 public:
47     Hexagon(const Point& p1, const Point& p2, const Point& p3, const Point& p4, const Point& p5, const Point& p6);
48     Point Center() const override;
49     double Square() const override;

```

```

50 void Print(std::ostream& os) const override; 51 private:
52 std::vector<Point> points;
53 };

```

## Figures.cpp

```

1 #include "Figures.hpp"
2 #include <cmath>
3 #include <algorithm>
4 #include <iomanip>
5
6 double calculateDistance(const Point& lhs, const Point& rhs) {
7     return sqrt(pow(rhs.x - lhs.x, 2) + pow(rhs.y - lhs.y, 2));
8 }
9
10 bool operator<(const Point& lhs, const Point& rhs) {
11     if(lhs.x != rhs.x) {
12         return lhs.x < rhs.x;
13     }
14     return lhs.y < rhs.y;
15 }
16
17 std::istream& operator>>(std::istream& is, Point& p) {
18     is >> p.x >> p.y;
19     return is;
20 }
21
22 std::ostream& operator<<(std::ostream& os, const Point& p) {
23     os << std::fixed << std::setprecision(3) << "[" << p.x << ", "
24         << p.y << "];"
25     return os;
26 }
27
28 std::ostream& operator<<(std::ostream& os, const std::vector
29 <Point> v) {
30     for(unsigned i = 0; i < v.size(); ++i) {
31         os << v[i];
32         if(i != v.size() - 1) {
33             os << ", ";
34         }
35     }
36     return os;
37 }
38
39 double checkIfRhombus(const Point& p1, const Point& p2, const
40 Point& p3, const Point& p4) {
41     double d1 = calculateDistance(p1, p2);
42     double d2 = calculateDistance(p1, p3);
43     double d3 = calculateDistance(p1, p4);
44     if(d1 == d2) { 44 return d3;
45     } else if(d1 == d3) {
46         return d2;

```

```

47     } else if(d2 == d3) {
48         return d1;
49     } else {
50         throw std::invalid_argument("Entered coordinates are not forming Rhombus. Try
entering new coordinates");
51     }
52 }
53
54     Rhombus::Rhombus(const Point& p1, const Point& p2, const Point& p3
, const Point& p4) {
55         try {
56             double d1 = checkIfRhombus(p1, p2, p3, p4);
57             double d2 = checkIfRhombus(p2, p1, p3, p4);
58             double d3 = checkIfRhombus(p3, p1, p2, p4);
59             double d4 = checkIfRhombus(p4, p1, p2, p3);
60             if(d1 == d2 || d1 == d4) {
61                 if(d1 < d3) {
62                     smallerDiagonal = d1;
63                     biggerDiagonal = d3;
64
65                 } else {
66                     smallerDiagonal = d3;
67                     biggerDiagonal = d1;
68                 }
69             } else if(d1 == d3) {
70                 if(d1 < d2) {
71                     smallerDiagonal = d1;
72                     biggerDiagonal = d2;
73                 } else {
74                     smallerDiagonal = d2;
75                     biggerDiagonal = d1;
76                 }
77             }
78         } catch(std::exception& e) {
79             throw std::invalid_argument(e.what());
80             return;
81         }
82         points.push_back(p1);
83         points.push_back(p2);
84         points.push_back(p3);
85         points.push_back(p4);
86     }
87
88     Point Rhombus::Center() const {
89         if(calculateDistance(points[0], points[1]) == smallerDiagonal
||
90         calculateDistance(points[0], points[1]) == biggerDiagonal) {
91             return {((points[0].x + points[1].x) / 2.0), ((points[0].y
+ points[1].y) / 2.0)};
92         } else if(calculateDistance(points[0], points[2]) == smallerDiagonal ||
calculateDistance(points[0], points[2]) == biggerDiagonal) {
93             return {((points[0].x + points[2].x) / 2.0), ((points[0].y
+ points[2].y) / 2.0)};
94         }

```

```

+ points[2].y) / 2.0));
95         } else {
96         return (((points[0].x + points[3].x) / 2.0), ((points[0].y
+ points[3].y) / 2.0));
97         }
98     }
99
100     double Rhombus::Square() const {
101     return smallerDiagonal * biggerDiagonal / 2.0;
102     }
103
104     void Rhombus::Print(std::ostream& os) const {
105     if(points.size()) {
106     os << "Rhombus: " << points << std::endl;
107     }
108     }
109
110     Pentagon::Pentagon(const Point& p1, const Point& p2, const Point& p3,
111     const Point& p4, const Point& p5) {
112     points.push_back(p1);
113     points.push_back(p2);
114     points.push_back(p3);
115     points.push_back(p4);
116     points.push_back(p5);
117     }
118
119     double triangleSquare(const Point& p1, const Point& p2, const
Point& p3) {
120     return 0.5 * fabs((p1.x - p3.x) * (p2.y - p3.y) - (p2.x - p3.x
) * (p1.y - p3.y));
121     }
122
123     Point Pentagon::Center() const {
124     Point insideFigure{0, 0};
125     Point result{0, 0};
126     double square = this->Square();
127     for(unsigned i = 0; i < points.size(); ++i) {
128     insideFigure.x += points[i].x;
129     insideFigure.y += points[i].y;
130     }
131     insideFigure.x /= points.size();
132     insideFigure.y /= points.size();
133     for(unsigned i = 0; i < points.size(); ++i) {
134     double tempSquare = triangleSquare(points[i], points[(i +
1) % points.size()],
135     insideFigure);
136     result.x += tempSquare * (points[i].x + points[(i + 1) %
points.size()].x
137     + insideFigure.x) / 3.0;
138     result.y += tempSquare * (points[i].y + points[(i + 1) %
points.size()].y
139     + insideFigure.y) / 3.0;

```

```

140         }
141         result.x /= square;
142         result.y /= square;
143         return result;
144     }
145
146     double Pentagon::Square() const {
147         double result = 0;
148         for(unsigned i = 0; i < points.size(); ++i) {
149             Point p1 = i ? points[i - 1] : points[points.size() - 1];
150             Point p2 = points[i];
151             result += (p1.x - p2.x) * (p1.y + p2.y);
152         }
153         return fabs(result) / 2.0;
154     }
155
156 void Pentagon::Print(std::ostream& os) const { 157         os <<
158     "Pentagon: " << points << std::endl;
159 }
160
161     Hexagon::Hexagon(const Point& p1, const Point& p2, const Point& p3
162     ,
163     const Point& p4, const Point& p5, const Point& p6) {
164         points.push_back(p1);
165         points.push_back(p2);
166         points.push_back(p3);
167         points.push_back(p4);
168         points.push_back(p5);
169         points.push_back(p6);
170     }
171
172     Point Hexagon::Center() const {
173         Point insideFigure{0, 0};
174         Point result{0, 0};
175         double square = this->Square();
176         for(unsigned i = 0; i < points.size(); ++i) {
177             insideFigure.x += points[i].x;
178             insideFigure.y += points[i].y;
179         }
180         insideFigure.x /= points.size();
181         insideFigure.y /= points.size();
182         for(unsigned i = 0; i < points.size(); ++i) {
183             double tempSquare = triangleSquare(points[i], points[(i +
184             1) % points.size()],
185             insideFigure);
186             result.x += tempSquare * (points[i].x + points[(i + 1) %
187             points.size()].x
188             + insideFigure.x) / 3.0;
189             result.y += tempSquare * (points[i].y + points[(i + 1) %
190             points.size()].y
191             + insideFigure.y) / 3.0;
192         }
193         result.x /= square;

```

```

189         result.y /= square;
190         return result;
191     }
192
193     double Hexagon::Square() const {
194         double result = 0;
195         for(unsigned i = 0; i < points.size(); ++i) {
196             Point p1 = i ? points[i - 1] : points[points.size() - 1];
197             Point p2 = points[i];
198             result += (p1.x - p2.x) * (p1.y + p2.y);
199         }
200         return fabs(result) / 2.0;
201     }
202
203 void Hexagon::Print(std::ostream& os) const { 204 os << "Hexagon: " <<
points << std::endl;
205 }

```

## main.cpp

```

1  #include <iostream>
2  #include "Figures.hpp"
3
4  int get_command() {
5      int command;
6      std::cin >> command;
7      return command;
8  }
9
10 int main() {
11     int command1, command2;
12     std::vector<Figure*> figures;
13     std::cout << "1 - add figure to the vector\n"
14     "2 - delete figure from the vector\n"
15     "3 - call common functions for the whole vector\n"
16     "4 - get total area of figures in vector\n"
17     "0 - exit\n";
18     while((command1 = get_command()) != 0) {
19         if(command1 == 1) {
20             Figure* f;
21             std::cout << "1 - Rhombus, 2 - Pentagon, 3 - Hexagon"
22             << std::endl;
23             std::cin >> command2;
24             if(command2 == 1) {
25                 Point p1, p2, p3, p4;
26                 std::cin >> p1 >> p2 >> p3 >> p4;
27                 try {
28                     f = new Rhombus{p1, p2, p3, p4};
29                     figures.push_back(f);
30                 } catch(std::exception& e) {
31                     std::cerr << e.what() << std::endl;

```

```

31     }
32     } else if(command2 == 2) {
33         Point p1, p2, p3, p4, p5;
34         std::cin >> p1 >> p2 >> p3 >> p4 >> p5;
35         f = new Pentagon{p1, p2, p3, p4, p5};
36         figures.push_back(f);
37     } else if(command2 == 3) {
38         Point p1, p2, p3, p4, p5, p6;
39         std::cin >> p1 >> p2 >> p3 >> p4 >> p5 >> p6;
40         f = new Hexagon{p1, p2, p3, p4, p5, p6};
41         figures.push_back(f);
42     } else {
43         std::cout << "Wrong input" << std::endl;
44     }
45     } else if(command1 == 2) {
46         std::cout << "Enter index" << std::endl;
47         std::cin >> command2;
48         if(command2 < static_cast<int>(figures.size())) {
49             delete figures[command2];
50             figures.erase(figures.begin() + command2);
51         } else {
52             std::cout << "Element with such index doesn't exist" << std::endl;
53         }
54     } else if(command1 == 3) {
55         for(const auto& figure : figures) {
56             figure->Print(std::cout);
57             std::cout << figure->Center() << std::endl;
58             std::cout << figure->Square() << std::endl;
59         }
60     } else if(command1 == 4) {
61         double result = 0;
62         for(const auto& figure : figures) {
63             result += figure->Square();
64         }
65         std::cout << result << std::endl;
66     } else if(command1 == 5) {
67         break;
68     } else {
69         std::cout << "Wrong command" << std::endl;
70     }
71 }
72 return 0;
73 } test.cpp

```

```

1 #include "Figures.hpp"
2
3 #define BOOST_TEST_DYN_LINK
4 #define BOOST_TEST_MODULE testFigures
5
6 #include <boost/test/unit_test.hpp>
7

```



```

8 BOOST_AUTO_TEST_CASE(testRhombusSqaure) {
9     Point p1{-2, 0};
10    Point p2{0, 2};
11    Point p3{2, 0};
12    Point p4{0, -2};
13    Rhombus r{p1,p2,p3,p4};
14    BOOST_CHECK_EQUAL(r.Square(), 8);
15    p2.y = 1;
16    p4.y = -1;
17    Rhombus r1{p1,p2,p3,p4};
18    BOOST_CHECK_EQUAL(r1.Square(), 4);
19 }
20
21 BOOST_AUTO_TEST_CASE(testRhombusCenter) {
22     Point p1{-2, 0};
23     Point p2{0, 2};
24     Point p3{2, 0};
25     Point p4{0, -2};
26     Rhombus r{p1,p2,p3,p4};
27     BOOST_CHECK_EQUAL(r.Center().x, 0);
28     BOOST_CHECK_EQUAL(r.Center().y, 0);
29     p2.y = 1;
30     p4.y = -1;
31     Rhombus r1{p1,p2,p3,p4};
32     BOOST_CHECK_EQUAL(r1.Center().x, 0);
33     BOOST_CHECK_EQUAL(r1.Center().y, 0);
34 }
35
36 BOOST_AUTO_TEST_CASE(testPentagonSquare) {
37     Point p1{-2, 0};
38     Point p2{0, 2};
39     Point p3{2, 0};
40     Point p4{3, 0};
41     Point p5{0, -2};
42     Pentagon p{p1, p2, p3, p4, p5};
43     BOOST_CHECK_EQUAL(p.Square(), 9);
44 }
45
46 BOOST_AUTO_TEST_CASE(testHexagonSquare) {
47     Point p1{-2, 0};
48     Point p2{0, 2};
49     Point p3{2, 0};
50     Point p4{2, -1};
51     Point p5{1, -2};
52     Point p6{0, -2};
53     Hexagon h{p1, p2, p3, p4, p5, p6};
54     BOOST_CHECK_EQUAL(h.Square(), 9.5);
55 }

```

## CMakeLists.txt

```

1 cmake_minimum_required(VERSION 3.1)
2

```

```

3 project(lab3)
4
5 enable_testing()
6
7 set(Figures_source Figures.cpp)
8
9 add_library(figures STATIC ${Figures_source})
10
11 find_package(Boost COMPONENTS unit_test_framework REQUIRED)
12
13 add_executable(testFigures test.cpp)
14 target_link_libraries(testFigures ${Boost_LIBRARIES} figures)
15 add_test(NAME Test1 COMMAND test1)
16
17     add_executable(lab3
18         main.cpp
19         Figures.cpp)
20
21 set_property(TARGET lab3 PROPERTY CXX_STANDARD 17)
22
23 set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra -Werror")

```

## 2 Тестирование

test\_01.txt:

Попробуем добавить в вектор фигуру с координатами (-5, 0), (-4, -1), (3, -1), (-2, 0), которая очевидно не является ромбом, рассчитывая получить сообщение об ошибке. Затем добавим в вектор ромб с координатами (-5, 0), (-3, 1), (-1, 0), (-3, -1), площадь которого равна 4, а центр находится в точке (-3, 0), а также пятиугольник с координатами (-3.000, 0.000), (-2.000, 1.000), (-1.000, 1.000), (0.000, 0.000), (-1.000, -1.000), площадь которого равна 3.5 и шестиугольник с координатами (-3.000, 0.000), (-2.000, 1.000), (-1.000, 1.000), (0.000, 0.000), (-1.000, -1.000), (-2.000, -1.000), (-1.500, -0.000) с площадью равной 4. Попробуем удалить из вектора элемент, находящийся на 3 позиции, надеясь получить сообщение об ошибке. Затем выведем все фигуры, а также найдем общую площадь фигур в массиве, которая должна быть равна 11.5, затем удалим шестиугольник и еще раз выведем все фигуры.

Результат:

```

1 - add figure to the vector
2 - delete figure from the vector
3 - call common functions for the whole vector
4 - get total area of figures in vector
0 - exit
1 - Rhombus, 2 - Pentagon, 3 - Hexagon
Entered coordinates are not forming Rhombus. Try entering new coordinates
1 - Rhombus, 2 - Pentagon, 3 - Hexagon
Enter index
Element with such index doesn't exist
1 - Rhombus, 2 - Pentagon, 3 - Hexagon

```

```

7.5
Rhombus: [-5.000, 0.000], [-3.000, 1.000], [-1.000, 0.000], [-3.000, -1.000]
[-3.000, 0.000]
4.000
Pentagon: [-3.000, 0.000], [-2.000, 1.000], [-1.000, 1.000], [0.000, 0.000], [-1.000,
-1.000]
[-1.429, 0.095]
3.500
1 - Rhombus, 2 - Pentagon, 3 - Hexagon
Rhombus: [-5.000, 0.000], [-3.000, 1.000], [-1.000, 0.000], [-3.000, -1.000]
[-3.000, 0.000]
4.000
Pentagon: [-3.000, 0.000], [-2.000, 1.000], [-1.000, 1.000], [0.000, 0.000], [-1.000,
-1.000]
[-1.429, 0.095]
3.500
Hexagon: [-3.000, 0.000], [-2.000, 1.000], [-1.000, 1.000], [0.000, 0.000], [-1.000, -
1.000], [-2.000, -1.000]
[-1.500, -0.000]
4.000
11.500
Enter index
Rhombus: [-5.000, 0.000], [-3.000, 1.000], [-1.000, 0.000], [-3.000, -1.000]
[-3.000, 0.000]
4.000
Pentagon: [-3.000, 0.000], [-2.000, 1.000], [-1.000, 1.000], [0.000, 0.000], [-1.000,
-1.000]
[-1.429,      0.095]
3.500

```

test\_02.txt

Добавим в вектор ромб с координатами [4.000, 0.000], [8.000, 2.000], [12.000, 0.000], [8.000, -2.000], центром в точке [8, 0] и площадью равной 16, квадрат с координатами [4.000, 2.000], [8.000, 2.000], [8.000, -2.000], [4.000, -2.000] с центром в точке [6, 0] и площадью равной 16, пятиугольник с координатами [4.000, 0.000], [8.000, 2.000], [12.000, 0.000], [8.000, -2.000], [6.000, -2.000] и площадью равной 18. Затем выведем все фигуры и найдем общую площадь, которая должна быть равна 50. Добавим шестиугольник с координатами [4.000, 0.000], [8.000, 2.000], [10.000, 2.000], [12.000, 0.000], [8.000, -2.000], [6.000, -2.000] и площадью равной 20. Еще раз выведем все фигуры и найдем общую площадь, которая должна быть равна 70. Затем удалим пятиугольник и шестиугольник и еще раз выведем все фигуры.

Результат:

- 1 - add figure to the vector
- 2 - delete figure from the vector

```

3 - call common functions for the whole vector
4 - get total area of figures in vector
0 - exit
1 - Rhombus, 2 - Pentagon, 3 - Hexagon
1 - Rhombus, 2 - Pentagon, 3 - Hexagon
1 - Rhombus, 2 - Pentagon, 3 - Hexagon
Rhombus: [4.000, 0.000], [8.000, 2.000], [12.000, 0.000], [8.000, -2.000]
[8.000, 0.000]
16.000
Rhombus: [4.000, 2.000], [8.000, 2.000], [8.000, -2.000], [4.000, -2.000]
[6.000, 0.000]
16.000
Pentagon: [4.000, 0.000], [8.000, 2.000], [12.000, 0.000], [8.000, -2.000], [6.000,
-2.000]
[7.778, -0.148]
18.000
50.000
1 - Rhombus, 2 - Pentagon, 3 - Hexagon
Rhombus: [4.000, 0.000], [8.000, 2.000], [12.000, 0.000], [8.000, -2.000]
[8.000, 0.000]
16.000
Rhombus: [4.000, 2.000], [8.000, 2.000], [8.000, -2.000], [4.000, -2.000]
[6.000, 0.000]
16.000
Pentagon: [4.000, 0.000], [8.000, 2.000], [12.000, 0.000], [8.000, -2.000], [6.000,
-2.000]
[7.778, -0.148]
18.000
Hexagon: [4.000, 0.000], [8.000, 2.000], [10.000, 2.000], [12.000, 0.000], [8.000,
-2.000],
[6.000, -2.000]
[8.000, 0.000]
20.000
70.000
Enter index
Enter index
Rhombus: [4.000, 0.000], [8.000, 2.000], [12.000, 0.000], [8.000, -2.000]
[8.000, 0.000]
16.000
Rhombus: [4.000, 2.000], [8.000, 2.000], [8.000, -2.000], [4.000, -2.000]
[6.000, 0.000]
16.000

```

### 3 Объяснение результатов работы программы

При вводе координат для создания ромба производится проверка этих координат, ведь они могут не образовывать ромб. Для этого реализована функция `checkIfRhombus`, которая вычисляет расстояния от одной точки до трёх остальных, а поскольку фигура является ромбом, то два из них должны быть равны. Третье же значение функция возвращает, ведь оно равно длине одной из диагоналей. Площадь ромба вычисляется как половина произведения диагоналей, центр - точка пересечения диагоналей. Методы вычисления площади и центра для пяти- и шестиугольника совпадают. Чтобы найти площадь необходимо перебрать все ребра и сложить площади трапеций, ограниченных этими ребрами. Чтобы найти центр необходимо разбить фигуры на треугольники (найти одну точку внутри фигуры), для каждого треугольника найти центр и площадь и перемножить их, просуммировать полученные величины и разделить на общую площадь фигуры.

## 4 Выводы

В ходе выполнения лабораторной работы я познакомился с таким понятием как runtime-полиморфизм. Также я познакомился с библиотекой для юнит-тестов из коллекций библиотек boost, которую уже могу сравнить с ранее использованным googletest. На мой взгляд, googletest предоставляет чуть больше возможностей для тестирования кода и имеет более приятную организацию тестов.