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

Лабораторная работа №4 по курсу “Объектно-ориентированное программирование”

Студент: Хисамутдинов Д.С.

Группа: М8О-208Б *Преподаватель:*

Журавлев.А.А.

Вариант: 5

Оценка: Дата:

Москва
2019

1 Исходный код

vertex.hpp

```
#pragma once

#include        <iostream>
#include        <cmath>
#include        <iomanip>

template        <class T>
        struct vertex_t {
        T x;
        T y;
        };

        template<class T>
        std::istream& operator>>(std::istream& is, vertex_t<T>& p) {
        is >> p.x >> p.y;
        return is;
        }

template<class T>
std::ostream& operator<<(std::ostream& os, const vertex_t<T>& p) { os << std::fixed << std::setprecision(3) << "[" << p.x << ",
        " << p.y << "]";
        return os;
        }

        template<class T>
        T calculateDistance(const vertex_t<T>& p1, const vertex_t<T>& p2)
        {
        return sqrt(pow(p2.x - p1.x, 2) + pow(p2.y - p1.y, 2));
        }

                        template<class T>
                        T triangleArea(vertex_t<T> p1, vertex_t<T> p2, vertex_t<T> p3) {
                        return 0.5 * fabs((p1.x - p3.x) * (p2.y - p3.y) - (p2.x - p3.x ) * (p1.y
                        - p3.y));
                        }
```

rhombus.hpp

```
#pragma once

#include <array>

#include "vertex.hpp"

                template<class T>
                double checkIfRhombus(const vertex_t<T> p1, const vertex_t<T>& p2,
                const vertex_t<T>& p3, const vertex_t<T>& p4) {
```

```
            T d1 = calculateDistance(p1, p2);
            T d2 = calculateDistance(p1, p3);
            T d3 = calculateDistance(p1, p4);
            if(d1 == d2) {
            return d3;
            } else if(d1 == d3) {
            return d2;
            } else if(d2 == d3) {
            return d1;
            } else {
    throw std::invalid_argument("Entered coordinates are not forming Rhombus. Try entering new
    coordinates");
     }
     }

template <class T>
struct Rhombus {
std::array<vertex_t<T>, 4> points;     T smallerDiagonal, biggerDiagonal;
                    Rhombus(const vertex_t<T>& p1, const vertex_t<T>& p2, const vertex_t<T>& p3,
                    const vertex_t<T>& p4);
                    double area() const;
                    vertex_t<T> center() const;
                    void print(std::ostream& os) const;
                    };

                        template<class T>
                        Rhombus<T>::Rhombus(const vertex_t<T>& p1, const vertex_t<T>& p2,
                        const vertex_t<T>& p3, const vertex_t<T>& p4) {
                        try {
                        T d1 = checkIfRhombus(p1, p2, p3, p4);
                        T d2 = checkIfRhombus(p2, p1, p3, p4);
                        T d3 = checkIfRhombus(p3, p1, p2, p4);
                        T d4 = checkIfRhombus(p4, p1, p2, p3);
                        if(d1 == d2 || d1 == d4) {
                        if(d1 < d3) {
                        smallerDiagonal = d1;
                        biggerDiagonal = d3;
                        } else {
                        smallerDiagonal = d3;
                        biggerDiagonal = d1;
                        }
                        } else if(d1 == d3) {
                        if(d1 < d2) {
                        smallerDiagonal = d1;
                        biggerDiagonal = d2;
                        } else {
                        smallerDiagonal = d2;
                        biggerDiagonal = d1;
                        }
                        }
                        } catch(std::exception& e) {
                        throw std::invalid_argument(e.what());
```

```
                        return;
                        }
                        points[0] = p1;
                        points[1] = p2;
                        points[2] = p3;
                        points[3] = p4;
                        }

template<class T>
double Rhombus<T>::area() const {
return smallerDiagonal * biggerDiagonal / 2.0;
}

            template<class T>
            vertex_t<T> Rhombus<T>::center() const {
            if(calculateDistance(points[0], points[1]) == smallerDiagonal
||
            calculateDistance(points[0], points[1]) == biggerDiagonal) {
            return {((points[0].x + points[1].x) / 2.0), ((points[0].y
+ points[1].y) / 2.0)};
            } else if(calculateDistance(points[0], points[2]) == smallerDiagonal ||
            calculateDistance(points[0], points[2]) == biggerDiagonal) {
            return {((points[0].x + points[2].x) / 2.0), ((points[0].y
+ points[2].y) / 2.0)};
            } else {
            return {((points[0].x + points[3].x) / 2.0), ((points[0].y
+ points[3].y) / 2.0)};
            }
            }

template<class T>
void Rhombus<T>::print(std::ostream& os) const {
          os << "Rhombus: ";
            for(const auto& p : points) {
            os << p << ' ';
            }
            os << std::endl;
            }
```

pentagon.hpp

```
#pragma once

#include <array>
#include <cmath>

#include "vertex.hpp"

                    template <class T>
                    struct Pentagon {
                    std::array<vertex_t<T>, 5> points;
                    Pentagon(const vertex_t<T>& p1, const vertex_t<T>& p2, const vertex_t<T>& p3,
```

```
                    const vertex_t<T>& p4, const vertex_t<T>& p5);
                    double area() const;
                    vertex_t<T> center() const;
                    void print(std::ostream& os) const;
                    };

 template <class T>
 Pentagon<T>::Pentagon(const vertex_t<T>& p1, const vertex_t<T>& p2,
 const vertex_t<T>& p3, const vertex_t<T>& p4, const vertex_t<T>& p5) {
 points[0] = p1;
 points[1] = p2;
 points[2] = p3;
 points[3] = p4;
 points[4] = p5;
 }

        template <class T>
        double Pentagon<T>::area() const {
        double result = 0;
        for(unsigned i = 0; i < points.size(); ++i) {
        vertex_t p1 = i ? points[i - 1] : points[points.size() - 1];
        vertex_t p2 = points[i];
        result += (p1.x - p2.x) * (p1.y + p2.y);
        }
        return fabs(result) / 2.0;
        }

        template <class T>
        vertex_t<T> Pentagon<T>::center() const {
        T x = 0;
        T y = 0;
        for(const auto& p : points) {
        x += p.x;
        y += p.y;
        }
        x /= points.size();
        y /= points.size();
        return {x, y};
        }

template <class T>
```

```
53              void Pentagon<T>::print(std::ostream& os)            const     {
54              os << "Pentagon: ";
55              for(const auto& p : points) {
56              os << p << ' ';
57              }
58              os << std::endl;
59              }
```

hexagon.hpp

```
1 #pragma once
2
3  #include <array>
4  #include <cmath>
5
6 #include "vertex.hpp"
7
8           template <class T>
9           struct Hexagon {
10          std::array<vertex_t<T>, 6> points;
11 Hexagon(const vertex_t<T>& p1, const vertex_t<T>& p3,          vertex_t<T>& p2,          const
12                                    const vertex_t<T>& p4, constvertex_t<T>& p5,          const
       vertex_t<T>& p6);
13          double area() const;
14          vertex_t<T> center() const;
15          void print(std::ostream& os) const;
16          };
17
18              template<class T>
19              Hexagon<T>::Hexagon(const vertex_t<T>& p1, const vertex_t<T>& p2,
20              const vertex_t<T>& p3, const vertex_t<T>& p4, const vertex_t<T>& p5,
21              const vertex_t<T>& p6) {
22              points[0] = p1;
23              points[1] = p2;
24              points[2] = p3;
25              points[3] = p4;
26              points[4] = p5;
27              points[5] = p6;
28              }
29
30              template<class T>
31              double Hexagon<T>::area() const {
32              double result = 0;
33              for(unsigned i = 0; i < points.size(); ++i) {
34              vertex_t p1 = i ? points[i - 1] : points[points.size() -
       1];
35              vertex_t p2 = points[i];
36              result += (p1.x - p2.x) * (p1.y + p2.y);
37              }
38              return fabs(result) / 2.0;
39              }
40
```

```
template<class T>
vertex_t<T> Hexagon<T>::center() const {
T x = 0;
T y = 0;
for(const auto& p : points) {
x += p.x;
y += p.y;
}
x /= points.size();
y /= points.size();
return {x, y};
}

template<class T>
void Hexagon<T>::print(std::ostream& os) const {
os << "Hexagon: ";
for(const auto& p : points) {
os << p << ' ';
}
os << std::endl;
}
```

templates.hpp

```
#pragma once

#include <iostream>
#include <type_traits>
#include<tuple>

#include "vertex.hpp"

template<class T>
struct is_vertex : std::false_type {};

template<class T>
struct is_vertex<vertex_t<T>> : std::true_type {};

template<class T>
struct is_figurelike_tuple : std::false_type {};

template<class Head, class... Tail>
struct is_figurelike_tuple<std::tuple<Head, Tail...>> :
std::conjunction<is_vertex<Head>,
std::is_same<Head, Tail>...> {};

template<class Type, size_t SIZE>
struct is_figurelike_tuple<std::array<Type, SIZE>> : is_vertex<Type> {};

template<class T>
inline constexpr bool is_figurelike_tuple_v =
is_figurelike_tuple<T>::value;
```

```
template<class T, class = void>
struct has_area_method : std::false_type {};

template<class T>
struct has_area_method<T,
std::void_t<decltype(std::declval<const T>().area())>> : std::true_type {};

        template<class T>
        inline constexpr bool has_area_method_v =
        has_area_method<T>::value;

        template<class T>
        std::enable_if_t<has_area_method_v<T>, double>
        area(const T& figure) {
        return figure.area();
        }

template<class T, class = void>
struct has_print_method : std::false_type {};


        template<class T>
        struct has_print_method<T,
        std::void_t<decltype(std::declval<const T>().print(std::cout))>>
        :
        std::true_type {};

        template<class T>
        inline constexpr bool has_print_method_v =
        has_print_method<T>::value;

        template<class T>
        std::enable_if_t<has_print_method_v<T>, void>
        print(const T& figure, std::ostream& os) {
        figure.print(os);
        }

template<class T, class = void>
struct has_center_method : std::false_type {};

template<class T>
struct has_center_method<T,
std::void_t<decltype(std::declval<const T>().center())>> :          std::true_type {};

        template<class T>
        inline constexpr bool has_center_method_v =
        has_center_method<T>::value;

        template<class T>
        std::enable_if_t<has_center_method_v<T>, vertex_t<decltype(std::
      declval<const T>().center().x)>>
```

```
82          center(const T& figure) {
83          return figure.center();
84          }
85
86  template<size_t ID, class T>
87  double single_area(const T& t) {
88          const      auto& a = std::get<0>(t);
89          const      auto& b = std::get<ID - 1>(t);
90          const      auto& c = std::get<ID>(t);
91          const      double dx1 = b.x - a.x;
92          const      double dy1 = b.y - a.y;
93          const      double dx2 = c.x - a.x;
94          const      double dy2 = c.y - a.y;
95          return std::abs(dx1 * dy2 - dy1 * dx2) * 0.5;
96          }
97
98                  template<size_t ID, class T>
99                  double recursive_area(const T& t) {
100                 if constexpr (ID < std::tuple_size_v<T>){
101                 return single_area<ID>(t) + recursive_area<ID + 1>(t);
102                 } else {
103                 return 0;
104                 }
105                 }
106
107         template<class T>
108         std::enable_if_t<is_figurelike_tuple_v<T>, double>
109         area(const T& fake) {
110         return recursive_area<2>(fake);
111         }
112
113         template<size_t ID, class T>
114         double single_center_x(const T& t) {
115         return std::get<ID>(t).x / std::tuple_size_v<T>;
116         }
117
118         template<size_t ID, class T>
119         double single_center_y(const T& t) {
120         return std::get<ID>(t).y / std::tuple_size_v<T>;
121         }
122
123                 template<size_t ID, class T>
124                 double recursive_center_x(const T& t) {
125                 if constexpr (ID < std::tuple_size_v<T>) {
126                 return single_center_x<ID>(t) + recursive_center_x<ID +
        1>(t);
127                 } else {
128                 return 0;
129                 }
130                 }
131
132                 template<size_t ID, class T>
```

```
133                 double recursive_center_y(const T& t) {
134                 if constexpr (ID < std::tuple_size_v<T>) {
135                 return single_center_y<ID>(t) + recursive_center_y<ID +
        1>(t);
136                 } else {
137                 return 0;
138                 }
139                 }
140
141           template<class T>
142           std::enable_if_t<is_figurelike_tuple_v<T>, vertex_t<double>>
143           center(const T& tup) {
144           return {recursive_center_x<0>(tup), recursive_center_y<0>( tup)};
145           }
146
147   template<size_t ID, class T>
148   void single_print(const T& t, std::ostream& os) { 149         os << std::get<ID>(t)
      << ' ';
150 }
151
152                 template<size_t ID, class T>
153                 void recursive_print(const T& t, std::ostream& os) {
154                 if constexpr (ID < std::tuple_size_v<T>) {
155                 single_print<ID>(t, os);
156                 recursive_print<ID + 1>(t, os);
157                 } else {
158                 return;
159                 }
160                 }
161
162                 template<class T>
163                 std::enable_if_t<is_figurelike_tuple_v<T>, void>
164                 print(const T& tup, std::ostream& os) {
165                 recursive_print<0>(tup, os);
166                 os << std::endl;
167                 }
```

main.cpp

```
1 #include        <iostream>
2
3 #include        "vertex.hpp"
4 #include        "templates.hpp"
5 #include        "rhombus.hpp"
6 #include        "pentagon.hpp"
7 #include        "hexagon.hpp"
8
9                           int main() {
10                          int command;
11                          std::cout << "1 - Rhombus" << std::endl;
12                          std::cout << "2 - Pentagon" << std::endl;
13                          std::cout << "3 - Hexagon" << std::endl;
```

```
14                      std::cout << "0 - Exit" << std::endl;
15                      std::cin >> command;
16                      while(command != 0) {
17                      if(command == 1) {
18                      vertex_t<double> p1, p2, p3, p4;
19                      std::cin >> p1 >> p2 >> p3 >> p4;
20                      try {
21                      Rhombus r{p1, p2, p3, p4};
22                      } catch(std::exception& e) {
23                      std::cout << e.what() << std::endl;
24                      std::cin >> command;
25                      continue;
26                      }
27                      Rhombus r{p1, p2, p3, p4};
28                      print(r, std::cout);
29                      std::cout << area(r) << std::endl;
30                      std::cout << center(r) << std::endl;
31                      std::tuple<vertex_t<double>, vertex_t<double>, vertex_t<double>,
32                      vertex_t<double>> r1{p1, p2, p3, p4};
33                      std::cout << "Rhombus: ";
34                      print(r1, std::cout);
35                      std::cout << area(r1) << std::endl;
36                      std::cout << center(r1) << std::endl;
37                      } else if(command == 2) {
38                      vertex_t<double> p1, p2, p3, p4, p5;
39                      std::cin >> p1 >> p2 >> p3 >> p4 >> p5;
40                      Pentagon r{p1, p2, p3, p4, p5};
41                      print(r, std::cout);
42                      std::cout << area(r) << std::endl;
43                      std::cout << center(r) << std::endl;
44                      std::tuple<vertex_t<double>, vertex_t<double>, vertex_t<double>,
45                      vertex_t<double>, vertex_t<double>> r1{p1, p2, p3,
    p4, p5};
46                      std::cout << "Pentagon: ";
47                      print(r1, std::cout);
48                      std::cout << area(r1) << std::endl;
49                      std::cout << center(r1) << std::endl;
50                      } else if(command == 3) {
51                      vertex_t<double> p1, p2, p3, p4, p5, p6;
52                      std::cin >> p1 >> p2 >> p3 >> p4 >> p5 >> p6;
53                      Hexagon r{p1, p2, p3, p4, p5, p6}; 54          print(r, std::cout);
55                              std::cout << area(r) << std::endl;
56                              std::cout << center(r) << std::endl;
57                              std::tuple<vertex_t<double>, vertex_t<double>, vertex_t<double>,
58                              vertex_t<double>, vertex_t<double>, vertex_t< double>>
59                              r1{p1, p2, p3, p4, p5, p6};
60                              std::cout << "Hexagon: ";
61                              print(r1, std::cout);
62                              std::cout << area(r1) << std::endl;
63                              std::cout << center(r1) << std::endl;
64                              } else {
65                              std::cout << "Wrong command" << std::endl;
66                              }
```

```
                    std::cin >> command;
                    }
                    }
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.1)

project(lab4)









    add_executable(lab4
    main.cpp
    vertex.hpp
    rhombus.hpp
    pentagon.hpp
    hexagon.hpp)

set_property(TARGET lab3 PROPERTY CXX_STANDARD 17)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra -Werror")
```

meson.build

```
project('lab4', 'cpp')

add_project_arguments('-std=c++17', '-Wall', '-Wextra', language : 'cpp')

executable('lab4_meson', 'main.cpp')
```

2 Тестирование

test_01.txt:

Попробуем создать фигуру с координатами (1, 2), (1, 3), (1, 4), (1, 5), которая очевидно не является ромбом, рассчитывая получить сообщение об ошибке. Затем создадим ромб с координатами (-2, 0), (0, 2), (2, 0), (0, -2), площадь которого равна 8, а центр находится в точке (0, 0), а также пятиугольник с координатами (-2.000, 0.000), (-1.000, 1.000), (1.000, 1.000), (2.000, 0.000), (1.000, -1.000), площадь которого равна 5 и шестиугольник с координатами (-2.000, 0.000), (-1.000, 1.000), (1.000, 1.000), (2.000, 0.000), (1.000, -1.000), (-1.000, -1.000) с площадью равной 6. Результат:

1 - Rhombus 2
- Pentagon
3 - Hexagon
0 - Exit
Entered coordinates are not forming Rhombus. Try entering new coordinates
Rhombus: [-2.000, 0.000] [0.000, 2.000] [2.000, 0.000] [0.000, -2.000]
8.000

[0.000, 0.000]
Rhombus: [-2.000, 0.000] [0.000, 2.000] [2.000, 0.000] [0.000, -2.000]
8.000
[0.000, 0.000] Pentagon: [-2.000, 0.000] [-1.000, 1.000] [1.000, 1.000] [2.000, 0.000] [1.000, -1.000]
5.000
[0.200, 0.200]
Pentagon: [-2.000, 0.000] [-1.000, 1.000] [1.000, 1.000] [2.000, 0.000] [1.000, -1.000]
5.000
[0.200, 0.200]
Hexagon: [-2.000, 0.000] [-1.000, 1.000] [1.000, 1.000] [2.000, 0.000] [1.000, -1.000] [-1.000, -1.000]
6.000
[0.000, 0.000]
Hexagon: [-2.000, 0.000] [-1.000, 1.000] [1.000, 1.000] [2.000, 0.000] [1.000, -1.000] [-1.000, -1.000]
6.000
[0.000, 0.000]

test_02.txt

Создадим 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, 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.

Результат:
1 - Rhombus 2
- Pentagon
3 - Hexagon
0 - Exit
Rhombus: [4.000, 0.000] [8.000, 2.000] [12.000, 0.000] [8.000, -2.000]
16.000
[8.000, 0.000]
Rhombus: [4.000, 0.000] [8.000, 2.000] [12.000, 0.000] [8.000, -2.000]
16.000
[8.000, 0.000]
Rhombus: [4.000, 2.000] [8.000, 2.000] [8.000, -2.000] [4.000, -2.000]
16.000
[6.000, 0.000]
Rhombus: [4.000, 2.000] [8.000, 2.000] [8.000, -2.000] [4.000, -2.000]
16.000

[6.000, 0.000]

Pentagon: [4.000, 0.000] [8.000, 2.000] [12.000, 0.000] [8.000, -2.000] [6.000, -2.000]

18.000

[7.600, -0.400]

Pentagon: [4.000, 0.000] [8.000, 2.000] [12.000, 0.000] [8.000, -2.000] [6.000, -2.000]

18.000

[7.600, -0.400]

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]

20.000

[8.000, 0.000]

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]

20.000

[8.000, 0.000]

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 низ 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 центроид и площадь и перемножить их, просуммировать полученные величины и разделить на общую площадь фигуры.

4 Выводы

На мой взгляд, метапрограммирование очень хорошо развито в плюсах. Я на своем примере увидел насколько меньше можно написать кода, если использовать предлагаемые для этого языком механизмы. Также я познакомился с системой для автоматизации сборки meson. По-моему, она имеет более приятный синтаксис, чем cmake, и, в большинстве, из-за этого она мне нравится больше.