

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

Лабораторная работа №5 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

```
1 #pragma once
2
3 #include <iostream>
4 #include <cmath>
5 #include <iomanip>
6
7 template <class T>
8     struct vertex_t {
9         T x;
10        T y;
11    };
12
13    template<class T>
14    std::istream& operator>>(std::istream& is, vertex_t<T>& p) {
15        is >> p.x >> p.y;
16        return is;
17    }
18
19    template<class T>
20    std::ostream& operator<<(std::ostream& os, const vertex_t<T>& p) { 21 os << std::fixed <<
    std::setprecision(3) << "[" << p.x << ",
    " << p.y << "];";
22    return os;
23    }
24
25    template<class T>
26    T calculateDistance(const vertex_t<T>& p1, const vertex_t<T>& p2)
27    {
28        return sqrt(pow(p2.x - p1.x, 2) + pow(p2.y - p1.y, 2));
29    }
30
31    template<class T>
32    T triangleArea(vertex_t<T> p1, vertex_t<T> p2, vertex_t<T> p3) {
33        return 0.5 * fabs((p1.x - p3.x) * (p2.y - p3.y) - (p2.x - p3.x) * (p1.y
34        - p3.y));
35    }
```

rhombus.hpp

```
1 #pragma once
2
3 #include <array>
4
5 #include "vertex.hpp"
6
7    template<class T>
8    double checkIfRhombus(const vertex_t<T> p1, const vertex_t<T>& p2,
9    const vertex_t<T>& p3, const vertex_t<T>& p4) {
10    T d1 = calculateDistance(p1, p2);
```

```

11         T d2 = calculateDistance(p1, p3);
12         T d3 = calculateDistance(p1, p4);
13         if(d1 == d2) {
14             return d3;
15         } else if(d1 == d3) {
16             return d2;
17         } else if(d2 == d3) {
18             return d1;
19         } else {
20             throw std::invalid_argument("Entered coordinates are not forming Rhombus. Try
entering new coordinates");
21         }
22     }
23
24     template <class T>
25     struct Rhombus {
26         std::array<vertex_t<T>, 4> points; 27         T smallerDiagonal, biggerDiagonal;
28         Rhombus(const vertex_t<T>& p1, const vertex_t<T>& p2, const vertex_t<T>& p3,
29             const vertex_t<T>& p4);
30         double area() const;
31         vertex_t<T> center() const;
32         void print(std::ostream& os) const;
33     };
34
35     template<class T>
36     Rhombus<T>::Rhombus(const vertex_t<T>& p1, const vertex_t<T>& p2,
37         const vertex_t<T>& p3, const vertex_t<T>& p4) {
38         try {
39             T d1 = checkIfRhombus(p1, p2, p3, p4);
40             T d2 = checkIfRhombus(p2, p1, p3, p4);
41             T d3 = checkIfRhombus(p3, p1, p2, p4);
42             T d4 = checkIfRhombus(p4, p1, p2, p3);
43             if(d1 == d2 || d1 == d4) {
44                 if(d1 < d3) {
45                     smallerDiagonal = d1;
46                     biggerDiagonal = d3;
47                 } else {
48                     smallerDiagonal = d3;
49                     biggerDiagonal = d1;
50                 }
51             } else if(d1 == d3) {
52                 if(d1 < d2) {
53                     smallerDiagonal = d1;
54                     biggerDiagonal = d2;
55                 } else {
56                     smallerDiagonal = d2;
57                     biggerDiagonal = d1;
58                 }
59             }
60             } catch(std::exception& e) {
61                 throw std::invalid_argument(e.what());
62             return;

```

```

63         }
64         points[0] = p1;
65         points[1] = p2;
66         points[2] = p3;
67         points[3] = p4;
68     }
69
70     template<class T>
71     double Rhombus<T>::area() const {
72         return smallerDiagonal * biggerDiagonal / 2.0;
73     }
74
75     template<class T>
76     vertex_t<T> Rhombus<T>::center() const {
77         if(calculateDistance(points[0], points[1]) == smallerDiagonal
78 ||
79 calculateDistance(points[0], points[1]) == biggerDiagonal) {
80         return {((points[0].x + points[1].x) / 2.0), ((points[0].y
81 + points[1].y) / 2.0)};
82         } else if(calculateDistance(points[0], points[2]) == smallerDiagonal ||
83 calculateDistance(points[0], points[2]) == biggerDiagonal) {
84         return {((points[0].x + points[2].x) / 2.0), ((points[0].y
85 + points[2].y) / 2.0)};
86         } else {
87         return {((points[0].x + points[3].x) / 2.0), ((points[0].y
88 + points[3].y) / 2.0)};
89     }
90     }
91
92     template<class T>
93     void Rhombus<T>::print(std::ostream& os) const {
94         os << "Rhombus: ";
95         for(const auto& p : points) {
96             os << p << ' ';
97         }
98         os << std::endl;
99     }
100 } stack.hpp

```

```

1 #pragma once
2
3 #include <iterator>
4 #include <memory>
5
6 namespace cntrs {
7
8     template<class T>
9     class stack_t {
10 private:
11         struct node_t;
12         public:
13         struct forward_iterator {

```

```

14         using value_type = T;
15         using reference = T&;
16         using pointer = T*;
17         using difference_type = ptrdiff_t;
18         using iterator_category = std::forward_iterator_tag;
19         forward_iterator(node_t* ptr) : ptr_(ptr) {};
20         T& operator*();
21         forward_iterator& operator++();
22         forward_iterator operator++(int);
23         bool operator==(const forward_iterator& it) const;
24         bool operator!=(const forward_iterator& it) const;
25     private:
26         node_t* ptr_;
27         friend stack_t;
28     };
29
30     forward_iterator begin();
31     forward_iterator end();
32     void insert(const forward_iterator& it, const T& value);
33     void insert(const int& pos, const T& value);
34     void erase(const forward_iterator& it);
35     void erase(int pos);
36     void pop();
37     T top();
38     void push(const T& value);
39     private:
40     struct node_t {
41         T value;
42         std::unique_ptr<node_t> nextNode = nullptr;
43         forward_iterator next();
44         node_t(const T& value, std::unique_ptr<node_t> next) :
45         value(value), nextNode(std::move(next)) {};
46
47         };
48         std::unique_ptr<node_t> head = nullptr;
49         node_t* tail = nullptr;
50         void insert_impl(std::unique_ptr<node_t> current, const T& value);
51
52 template<class T>
53 typename stack_t<T>::forward_iterator stack_t<T>::node_t::next() { 53     return nextNode.get();
54 }
55
56 template<class T>
57 T& stack_t<T>::forward_iterator::operator*() { 58     return ptr_ -
58     >value;
59 }
60
61 template<class T>
62 typename stack_t<T>::forward_iterator& stack_t<T>::
63 forward_iterator::operator++() {
64     *this = ptr_ ->next();
65     return *this;
66 }

```

```

66
67     template<class T>
68     typename stack_t<T>::forward_iterator stack_t<T>::forward_iterator::operator++(int) {
69         forward_iterator old = *this;
70         ++*this;
71         return old;
72     }
73
74     template<class T>
75     bool stack_t<T>::forward_iterator::operator!=(const forward_iterator& it) const {
76         return ptr_ != it.ptr_;
77     }
78
79     template<class T>
80     bool stack_t<T>::forward_iterator::operator==(const forward_iterator& it) const {
81         return ptr_ == it.ptr_;
82     }
83
84     template<class T>
85     typename stack_t<T>::forward_iterator stack_t<T>::begin() {
86         return head.get();
87     }
88
89     template<class T>
90     typename stack_t<T>::forward_iterator stack_t<T>::end() {
91         return nullptr;
92     }
93
94         template<class T>
95         void stack_t<T>::insert(const forward_iterator& it, const T& value ) {
96             std::unique_ptr<node_t> newNode(new node_t(value, nullptr));
97             if(head == nullptr) {
98                 head = std::move(newNode);
99             } else if(head->nextNode == nullptr) {
100                 if(it.ptr_) {
101                     tail = head.get();
102                     newNode->nextNode = std::move(head);
103                     head = std::move(newNode);
104                 } else {
105                     tail = newNode.get();
106                     head->nextNode = std::move(newNode);
107                 }
108             } else if(head.get() == it.ptr_) {
109                 newNode->nextNode = std::move(head);
110                 head = std::move(newNode);
111             } else if(it.ptr_ == nullptr) {
112                 tail->nextNode = std::move(newNode);
113                 tail = newNode.get();
114             } else {
115                 auto temp = this->begin();
116                 while(temp.ptr_->next() != it.ptr_) {
117                     ++temp;

```

```

118     }
119
120     newNode->nextNode = std::move(temp.ptr_->nextNode);
121     temp.ptr_->nextNode = std::move(newNode);
122 }
123 }
124
125     template<class T>
126     void stack_t<T>::insert(const int& pos, const T& value) {
127         int i = 0;
128         auto temp = this->begin();
129         if(pos == 0) {
130             insert(temp, value);
131             return;
132         } else if(pos == 1) {
133             if(temp.ptr_ == nullptr) {
134                 throw std::logic_error("2:out of bounds");
135             }
136             ++temp;
137             insert(temp, value);
138             return;
139         }
140         while(i < pos) {
141             if(temp.ptr_ == nullptr) {
142                 break;
143             }
144             ++temp;
145             ++i;
146         }
147         if(i < pos) {
148             throw std::logic_error("2:out of bounds");
149         }
150         this->insert(temp, value);
151     }
152
153     template<class T>
154     void stack_t<T>::erase(const forward_iterator& it) {
155         if(it == nullptr) {
156             throw std::logic_error("Invalid iterator");
157         }
158         if(head == nullptr) {
159             throw std::logic_error("Deleting from empty list");
160         }
161         if(it == this->begin()) {
162             head = std::move(head->nextNode);
163         } else {
164             auto temp = this->begin();
165             while(temp.ptr_->next() != it.ptr_) {
166                 ++temp;
167             }
168             temp.ptr_->nextNode = std::move(it.ptr_->nextNode);
169         }

```

```

170         }
171
172         template<class T>
173         void stack_t<T>::erase(int pos) {
174             auto temp = this->begin();
175             int i = 0;
176             while(i < pos) {
177                 if(temp.ptr_ == nullptr) {
178                     break;
179                 }
180                 ++temp;
181                 ++i;
182             }
183             if(temp.ptr_ == nullptr) {
184                 throw std::logic_error("Out of bounds");
185             }
186             erase(temp);
187         }
188
189     template<class T>
190     void stack_t<T>::pop() {
191         erase(this->begin());
192     }
193
194     template<class T>
195     T stack_t<T>::top() {
196         if(head) {
197             return head->value;
198         } else {
199             throw std::logic_error("Stack is empty");
200         }
201     }
202
203     template<class T>
204     void stack_t<T>::push(const T& value) {
205         insert(this->begin(), value);
206     }
207
208 }

```

main.cpp

```

1 #include <iostream>
2 #include <algorithm>
3
4 #include "stack.hpp"
5 #include "rhombus.hpp"
6
7     int main() {
8         cntrs::stack_t<Rhombus<double>> s;
9         int command, pos;

```



```

10         std::cout << "1 - add element to stack(push/insert by iterator )" <<
        std::endl;
11         std::cout << "2 - delete element from stack(pop/erase by index
/erase by iterator)" << std::endl;
12         std::cout << "3 - range-based for print" << std::endl;
13         std::cout << "4 - count_if example" << std::endl;
14         std::cout << "5 - top element" << std::endl;
15         std::cin >> command;
16         while(true) {
17             if(command == 0) {
18                 break;
19             } else if(command == 1) {
20                 std::cout << "Enter coordinates" << std::endl;
21                 vertex_t<double> v1, v2, v3, v4;
22                 std::cin >> v1 >> v2 >> v3 >> v4;
23                 try {
24                     Rhombus<double> r{v1, v2, v3, v4};
25                 } catch(std::exception& e) {
26                     std::cout << e.what() << std::endl;
27                     std::cin >> command;
28                     continue;
29                 }
30                 Rhombus<double> r{v1, v2, v3, v4};
31                 std::cout << "1 - push to stack" << std::endl;
32                 std::cout << "2 - insert by iterator" << std::endl;
33                 std::cin >> command;
34                 if(command == 1) {
35                     s.push(r);
36                 } else if(command == 2) {
37                     std::cout << "Enter index" << std::endl;
38                     std::cin >> pos;
39                     s.insert(pos, r);
40                 } else {
41                     std::cout << "Wrong command" << std::endl;
42                     std::cin >> command;
43                     continue;
44                 }
45                 } else if(command == 2) {
46                     std::cout << "1 - pop" << std::endl;
47                     std::cout << "2 - erase by index" << std::endl;
48                     std::cout << "3 - erase by iterator" << std::endl;
49                     std::cin >> command;
50                     if(command == 1) {
51                         s.pop();
52                     } else if(command == 2) {
53                         std::cout << "Enter index" << std::endl;
54                         std::cin >> pos;
55                         s.erase(pos);
56                     } else if(command == 3) {
57                         std::cout << "Enter index" << std::endl;
58                         std::cin >> pos;
59                         auto temp = s.begin();

```

```

60     for(int i = 0; i < pos; ++i) {
61         ++temp;
62     }
63     s.erase(temp);
64 } else {
65     std::cout << "Wrong command" << std::endl;
66     std::cin >> command;
67     continue;
68 }
69 } else if(command == 3) {
70     for(const auto& item : s) {
71         item.print(std::cout);
72     }
73 } else if(command == 4) {
74     std::cout << "Enter required square" << std::endl;
75     std::cin >> pos;
76     std::cout << "Number of rhombes with area less than " << pos << "
        equals ";
77     std::cout << std::count_if(s.begin(), s.end(), [pos]( Rhombus<double> r)
        {return r.area() < pos;}) << std::endl;
78 } else if(command == 5) {
79     try {
80         s.top();
81     } catch(std::exception& e) {
82         std::cout << e.what() << std::endl;
83         std::cin >> command;
84         continue;
85     }
86     Rhombus temp = s.top();
87     std::cout << "Top: ";
88     temp.print(std::cout);
89 } else {
90     std::cout << "Wrong command" << std::endl;
91 }
92 std::cin >> command;
93 }
94 return 0;
95 }

```

CMakeLists.txt

```

1 cmake_minimum_required(VERSION 3.1)
2
3 project(lab5)
4
5     add_executable(lab5
6         main.cpp)
7
8 set_property(TARGET lab5 PROPERTY CXX_STANDARD 17)
9
10 set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -g -Wall -Wextra -Werror")

```

2 Тестирование

Набор входных данных для всех тестов одинаковый - ромбы с координатами $([-1, -1], [-1, 1], [1, 1], [1, -1])$, $([-2, -2], [-2, 2], [2, 2], [2, -2])$, $([-3, -3], [-3, 3], [3, 3], [3, -3])$, $([-4, -4], [-4, 4], [4, 4], [4, -4])$. Различия заключаются в методах добавления и удаления этих фигур в стек.

test_01.txt:

Добавим фигуры в стек с помощью метода `push` и напечатаем их. Затем с помощью `count_if` найдем количество ромбов с площадями меньше 4, 16, 36, 64, 81 (0, 1, 2, 3, 4 соответственно). Удалим все фигуры из стека с помощью метода `pop`, перед каждым вызовом которого, выведем элемент на верху стека с помощью функции `top`.

Результат:

1 - add element to stack(push/insert by iterator)

2 - delete element from stack(pop/erase by index/erase by iterator)

3 - range-based for print

4 - count_if example

5 - top element

Enter coordinates

1 - push to stack

2 - insert by iterator

Enter coordinates

1 - push to stack

2 - insert by iterator

Enter coordinates

1 - push to stack

2 - insert by iterator

Enter coordinates

1 - push to stack

2 - insert by iterator

Rhombus: [-4.000, -4.000] [-4.000, 4.000] [4.000, 4.000] [4.000, -4.000]

Rhombus: [-3.000, -3.000] [-3.000, 3.000] [3.000, 3.000] [3.000, -3.000]

Rhombus: [-2.000, -2.000] [-2.000, 2.000] [2.000, 2.000] [2.000, -2.000]

Rhombus: [-1.000, -1.000] [-1.000, 1.000] [1.000, 1.000] [1.000, -1.000]

Enter required square

Number of rhombes with area less than 4 equals 0

Enter required square

Number of rhombes with area less than 16 equals 1 Enter
required square

Number of rhombes with area less than 36 equals 3 Enter
required square

Number of rhombes with area less than 64 equals 3 Enter
required square

Number of rhombes with area less than 81 equals 4

Top: Rhombus: [-4.000, -4.000] [-4.000, 4.000] [4.000, 4.000] [4.000, -4.000]

1 - pop
 2 - erase by index
 3 - erase by iterator
 Top: Rhombus: [-3.000, -3.000] [-3.000, 3.000] [3.000, 3.000] [3.000, -3.000]
 1 - pop
 2 - erase by index
 3 - erase by iterator
 Top: Rhombus: [-2.000, -2.000] [-2.000, 2.000] [2.000, 2.000] [2.000, -2.000]
 1 - pop
 2 - erase by index
 3 - erase by iterator
 Top: Rhombus: [-1.000, -1.000] [-1.000, 1.000] [1.000, 1.000] [1.000, -1.000]
 1 - pop
 2 - erase by index
 3 - erase by iterator Stack is empty

test_02.txt

То же самое, что и предыдущем тесте, кроме того, что фигуры добавляются в стек по итератору на 0,1,1,2 места соответственно.

Результат:

1 - add element to stack(push/insert by iterator)
 2 - delete element from stack(pop/erase by index/erase by iterator)
 3 - range-based for print

4 - count_if example

5 - top element

Enter coordinates

1 - push to stack

2 - insert by iterator

Enter index

Enter coordinates

1 - push to stack

2 - insert by iterator

Enter index

Enter coordinates

1 - push to stack

2 - insert by iterator

Enter index

Enter coordinates

1 - push to stack

2 - insert by iterator

Enter index

Rhombus: [-1.000, -1.000] [-1.000, 1.000] [1.000, 1.000] [1.000, -1.000]

Rhombus: [-3.000, -3.000] [-3.000, 3.000] [3.000, 3.000] [3.000, -3.000]

Rhombus: [-4.000, -4.000] [-4.000, 4.000] [4.000, 4.000] [4.000, -4.000]

Rhombus: [-2.000, -2.000] [-2.000, 2.000] [2.000, 2.000] [2.000, -2.000]

Enter required square
 Number of rhombes with area less than 4 equals 0
 Enter required square
 Number of rhombes with area less than 16 equals 1 Enter
 required square
 Number of rhombes with area less than 36 equals 3
 Enter required square
 Number of rhombes with area less than 64 equals 3 Enter
 required square
 Number of rhombes with area less than 81 equals 4
 Top: Rhombus: [-1.000, -1.000] [-1.000, 1.000] [1.000, 1.000] [1.000, -1.000]
 1 - pop
 2 - erase by index
 3 - erase by iterator
 Top: Rhombus: [-3.000, -3.000] [-3.000, 3.000] [3.000, 3.000] [3.000, -3.000]
 1 - pop
 2 - erase by index
 3 - erase by iterator
 Top: Rhombus: [-4.000, -4.000] [-4.000, 4.000] [4.000, 4.000] [4.000, -4.000]
 1 - pop
 2 - erase by index
 3 - erase by iterator
 Top: Rhombus: [-2.000, -2.000] [-2.000, 2.000] [2.000, 2.000] [2.000, -2.000]
 1 - pop
 2 - erase by index
 3 - erase by iterator Stack is empty

test_03.txt

То же самое, что и предыдущем тесте, кроме того, что фигуры удаляются из стека по индексу в следующем порядке: 3-я, 3-я, 1-я, 1-я. После каждого удаления происходит печать стека.

Результат:

1 - add element to stack(push/insert by iterator)
 2 - delete element from stack(pop/erase by index/erase by iterator)
 3 - range-based for print
 4 - count_if example
 5 - top element
 Enter coordinates
 1 - push to stack
 2 - insert by iterator
 Enter index
 Enter coordinates
 1 - push to stack
 2 - insert by iterator
 Enter index

Enter coordinates
 1 - push to stack
 2 - insert by iterator
 Enter index
 Enter coordinates
 1 - push to stack
 2 - insert by iterator
 Enter index
 Rhombus: [-1.000, -1.000] [-1.000, 1.000] [1.000, 1.000] [1.000, -1.000]
 Rhombus: [-3.000, -3.000] [-3.000, 3.000] [3.000, 3.000] [3.000, -3.000]
 Rhombus: [-4.000, -4.000] [-4.000, 4.000] [4.000, 4.000] [4.000, -4.000]
 Rhombus: [-2.000, -2.000] [-2.000, 2.000] [2.000, 2.000] [2.000, -2.000]
 Enter required square
 Number of rhombes with area less than 4 equals 0
 Enter required square
 Number of rhombes with area less than 16 equals 1 Enter
 required square
 Number of rhombes with area less than 36 equals 3 Enter
 required square
 Number of rhombes with area less than 64 equals 3 Enter
 required square
 Number of rhombes with area less than 81 equals 4
 Top: Rhombus: [-1.000, -1.000] [-1.000, 1.000] [1.000, 1.000] [1.000, -1.000]
 1 - pop
 2 - erase by index
 3 - erase by iterator
 Enter index
 Rhombus: [-1.000, -1.000] [-1.000, 1.000] [1.000, 1.000] [1.000, -1.000]
 Rhombus: [-3.000, -3.000] [-3.000, 3.000] [3.000, 3.000] [3.000, -3.000]
 Rhombus: [-2.000, -2.000] [-2.000, 2.000] [2.000, 2.000] [2.000, -2.000]
 1 - pop
 2 - erase by index
 3 - erase by iterator
 Enter index
 Rhombus: [-1.000, -1.000] [-1.000, 1.000] [1.000, 1.000] [1.000, -1.000]
 Rhombus: [-3.000, -3.000] [-3.000, 3.000] [3.000, 3.000] [3.000, -3.000]
 1 - pop
 2 - erase by index
 3 - erase by iterator
 Enter index
 Rhombus: [-3.000, -3.000] [-3.000, 3.000] [3.000, 3.000] [3.000, -3.000]
 1 - pop
 2 - erase by index
 3 - erase by iterator
 Enter index

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.

4 Выводы

Умные указатели при грамотном использовании позволяют сильно экономить время на выявление утечек памяти и исправления их. Однако при первом их использовании не так просто написать корректно работающую программу, ведь они несколько отличаются от сырых указателей и, соответственно, методов работы с ними.