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

Лабораторная работа №5 по курсу “Объектно-ориентированное программирование”

Студент: Живалев Е.А.

Группа: М8О-206Б

Преподаватель: Журавлев А.А.

Вариант: 5

Оценка: _____

Дата: _____

Москва
2019

1 Исходный код

Ссылка на github : https://github.com/QElderDelta/oop_exercise_05

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>
```

```cpp
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;
}
```

stack.hpp

```cpp
#pragma once

#include <iterator>
#include <memory>

namespace cntrs {

template<class T>
class stack_t {
private:
    struct node_t;
public:
    struct forward_iterator {
        using value_type = T;
        using reference = T&;
        using pointer = T*;
        using difference_type = ptrdiff_t;
        using iterator_category = std::forward_iterator_tag;
        forward_iterator(node_t* ptr) : ptr_(ptr) {};
        T& operator*();
        forward_iterator& operator++();
        forward_iterator operator++(int);
        bool operator==(const forward_iterator& it) const;
        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) :
        value(value), nextNode(std::move(next)) {};
45      };
46      std::unique_ptr<node_t> head = nullptr;
47      node_t* tail = nullptr;
48      void insert_impl(std::unique_ptr<node_t> current, const T&
        value);
49  };
50
51  template<class T>
52  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_->value;
59  }
60
61  template<class T>
62  typename stack_t<T>::forward_iterator& stack_t<T>::
        forward_iterator::operator++() {
63      *this = ptr_->next();
64      return *this;
65  }
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>
```

```
T stack_t<T>::top() {
    if(head) {
        return head->value;
    } else {
        throw std::logic_error("Stack is empty");
    }
}

template<class T>
void stack_t<T>::push(const T& value) {
    insert(this->begin(), value);
}

}
```

main.cpp

```
#include <iostream>
#include <algorithm>

#include "stack.hpp"
#include "rhombus.hpp"

int main() {
    cntrs::stack_t<Rhombus<double>> s;
    int command, pos;
    std::cout << "1 - add element to stack(push/insert by iterator)" << std::endl;
    std::cout << "2 - delete element from stack(pop/erase by index/erase by iterator)" << std::endl;
    std::cout << "3 - range-based for print" << std::endl;
    std::cout << "4 - count_if example" << std::endl;
    std::cout << "5 - top element" << std::endl;
    std::cin >> command;
    while(true) {
        if(command == 0) {
            break;
        } else if(command == 1) {
            std::cout << "Enter coordinates" << std::endl;
            vertex_t<double> v1, v2, v3, v4;
            std::cin >> v1 >> v2 >> v3 >> v4;
            try {
                Rhombus<double> r{v1, v2, v3, v4};
            } catch(std::exception& e) {
                std::cout << e.what() << std::endl;
                std::cin >> command;
                continue;
            }
            Rhombus<double> r{v1, v2, v3, v4};
            std::cout << "1 - push to stack" << std::endl;
            std::cout << "2 - insert by iterator" << std::endl;
            std::cin >> command;
            if(command == 1) {
                s.push(r);
            } else if(command == 2) {
                std::cout << "Enter index" << std::endl;
                std::cin >> pos;
                s.insert(pos, r);
            } else {
```

```
                std::cout << "Wrong command" << std::endl;
                std::cin >> command;
                continue;
            }
        } else if(command == 2) {
            std::cout << "1 - pop" << std::endl;
            std::cout << "2 - erase by index" << std::endl;
            std::cout << "3 - erase by iterator" << std::endl;
            std::cin >> command;
            if(command == 1) {
                s.pop();
            } else if(command == 2) {
                std::cout << "Enter index" << std::endl;
                std::cin >> pos;
                s.erase(pos);
            } else if(command == 3) {
                std::cout << "Enter index" << std::endl;
                std::cin >> pos;
                auto temp = s.begin();
                for(int i = 0; i < pos; ++i) {
                    ++temp;
                }
                s.erase(temp);
            } else {
                std::cout << "Wrong command" << std::endl;
                std::cin >> command;
                continue;
            }
        } else if(command == 3) {
            for(const auto& item : s) {
                item.print(std::cout);
            }
        } else if(command == 4) {
            std::cout << "Enter required square" << std::endl;
            std::cin >> pos;
            std::cout << "Number of rhombes with area less than "
    << pos << " equals ";
            std::cout << std::count_if(s.begin(), s.end(), [pos](
    Rhombus<double> r) {return r.area() < pos;}) << std::endl;
        } else if(command == 5) {
            try {
                s.top();
            } catch(std::exception& e) {
                std::cout << e.what() << std::endl;
                std::cin >> command;
                continue;
            }
            Rhombus temp = s.top();
            std::cout << "Top: ";
            temp.print(std::cout);
        } else {
            std::cout << "Wrong command" << std::endl;
        }
        std::cin >> command;
    }
    return 0;
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.1)

project(lab5)

add_executable(lab5
    main.cpp)

set_property(TARGET lab5 PROPERTY CXX_STANDARD 17)

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 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.

4 Выводы

Умные указатели при грамотном использовании позволяют сильно сэкономить 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.