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

Лабораторная работа №6 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_06

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;
}
```

stack.hpp

```
#pragma once

#include <iterator>
#include <memory>
#include <iostream>

namespace cntrs {

template<class T, class Allocator = std::allocator<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;
```

```cpp
        bool operator!=(const forward_iterator& it) const;
        private:
            node_t* ptr_;
            friend stack_t;
    };

    forward_iterator begin();
    forward_iterator end();
    void insert(const forward_iterator& it, const T& value);
    void insert(const int& pos, const T& value);
    void erase(const forward_iterator& it);
    void erase(int pos);
    void pop();
    T top();
    void push(const T& value);
    stack_t() = default;
    stack_t(const stack_t&) = delete;
private:
    using allocator_type = typename Allocator::template rebind<node_t>::other;

    struct deleter {
        deleter(allocator_type* allocator) : allocator(allocator) {};

        void operator()(node_t* ptr) {
            if(ptr != nullptr) {
                std::allocator_traits<allocator_type>::destroy(*allocator, ptr);
                allocator->deallocate(ptr, 1);
            }
        }

        private:
            allocator_type* allocator;
    };

    struct node_t {
        T value;
        std::unique_ptr<node_t, deleter> nextNode{nullptr, deleter{&this->allocator}};
        forward_iterator next();
        node_t(const T& value, std::unique_ptr<node_t, deleter> next) : value(value), nextNode(std::move(next)) {};
    };
    std::unique_ptr<node_t, deleter> head{nullptr, deleter{&this->allocator}};
    node_t* tail = nullptr;
    stack_t& operator=(const stack_t&);
    allocator_type allocator {};
};

template<class T, class Allocator>
typename stack_t<T, Allocator>::forward_iterator stack_t<T, Allocator>::node_t::next() {
    return nextNode.get();
}

template<class T, class Allocator>
```

```cpp
T& stack_t<T, Allocator>::forward_iterator::operator*() {
    return ptr_->value;
}

template<class T, class Allocator>
typename stack_t<T, Allocator>::forward_iterator& stack_t<T, Allocator>::forward_iterator::operator++() {
    *this = ptr_->next();
    return *this;
}

template<class T, class Allocator>
typename stack_t<T, Allocator>::forward_iterator stack_t<T, Allocator>::forward_iterator::operator++(int) {
    forward_iterator old = *this;
    ++*this;
    return old;
}

template<class T, class Allocator>
bool stack_t<T, Allocator>::forward_iterator::operator!=(const forward_iterator& it) const {
    return ptr_ != it.ptr_;
}

template<class T, class Allocator>
bool stack_t<T, Allocator>::forward_iterator::operator==(const forward_iterator& it) const {
    return ptr_ == it.ptr_;
}

template<class T, class Allocator>
typename stack_t<T, Allocator>::forward_iterator stack_t<T, Allocator>::begin() {
    return head.get();
}

template<class T, class Allocator>
typename stack_t<T, Allocator>::forward_iterator stack_t<T, Allocator>::end() {
    return nullptr;
}

template<class T, class Allocator>
void stack_t<T, Allocator>::insert(const forward_iterator& it, const T& value) {
    node_t* ptr = this->allocator.allocate(1);
    std::allocator_traits<allocator_type>::construct(this->allocator, ptr, value, std::unique_ptr<node_t,
            deleter>(nullptr, deleter{&this->allocator}));
    std::unique_ptr<node_t, deleter> newNode(ptr, deleter{&this->allocator});
    if(head == nullptr) {
        head = std::move(newNode);
    } else if(head->nextNode == nullptr) {
        if(it.ptr_) {
            tail = head.get();
            newNode->nextNode = std::move(head);
            head = std::move(newNode);
        } else {
            tail = newNode.get();
            head->nextNode = std::move(newNode);
        }
    } else if(head.get() == it.ptr_) {
        newNode->nextNode = std::move(head);
        head = std::move(newNode);
    } else if(it.ptr_ == nullptr) {
        tail->nextNode = std::move(newNode);
        tail = newNode.get();
    } else {
        auto temp = this->begin();
        while(temp.ptr_->next() != it.ptr_) {
            ++temp;
        }

        newNode->nextNode = std::move(temp.ptr_->nextNode);
        temp.ptr_->nextNode = std::move(newNode);
    }
}

template<class T, class Allocator>
void stack_t<T, Allocator>::insert(const int& pos, const T& value) {
    int i = 0;
    auto temp = this->begin();
    if(pos == 0) {
        insert(temp, value);
        return;
    }
    while(i < pos) {
        if(temp.ptr_ == nullptr) {
            break;
        }
        ++temp;
        ++i;
    }
    if(i < pos) {
        throw std::logic_error("Out of bounds");
    }
    this->insert(temp, value);
}

template<class T, class Allocator>
void stack_t<T, Allocator>::erase(const forward_iterator& it) {
    if(it == nullptr) {
        throw std::logic_error("Invalid iterator");
    }
    if(head == nullptr) {
        throw std::logic_error("Deleting from empty list");
    }
    if(it == this->begin()) {
        head = std::move(head->nextNode);
    } else {
        auto temp = this->begin();
        while(temp.ptr_->next() != it.ptr_) {
            ++temp;
        }
        temp.ptr_->nextNode = std::move(it.ptr_->nextNode);
    }
}

template<class T, class Allocator>
void stack_t<T, Allocator>::erase(int pos) {
    auto temp = this->begin();
    int i = 0;
    while(i < pos) {
        if(temp.ptr_ == nullptr) {
            break;
        }
        ++temp;
        ++i;
    }
    if(temp.ptr_ == nullptr) {
        throw std::logic_error("Out of bounds");
    }
    erase(temp);
}

template<class T, class Allocator>
void stack_t<T, Allocator>::pop() {
    erase(this->begin());
}

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

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

}
```

allocator.hpp

```
#pragma once

#include <iostream>
#include <type_traits>

#include "tvector.hpp"
#include "stack.hpp"

namespace allctr {

template<class T, size_t ALLOC_SIZE>
struct allocator_t {
    using value_type = T;
    using size_type = size_t;
    using difference_type = std::ptrdiff_t;
    using is_always_equal = std::false_type;

    template<class U>
    struct rebind {
        using other = allocator_t<U, ALLOC_SIZE>;
    };

    allocator_t() : memory_pool_begin(new char[ALLOC_SIZE]),
   memory_pool_end(memory_pool_begin + ALLOC_SIZE),
        memory_pool_tail(memory_pool_begin) {};

    allocator_t(const allocator_t&) = delete;
    allocator_t(allocator_t&&) = delete;


    ~allocator_t() {
        delete[] memory_pool_begin;
    }
    T* allocate(size_t n);
    void deallocate(T* ptr, size_t n);
private:
    char* memory_pool_begin;
    char* memory_pool_end;
    char* memory_pool_tail;
    cntrs::vector_t<char*> free_blocks;
};

template<class T, size_t ALLOC_SIZE>
T* allocator_t<T, ALLOC_SIZE>::allocate(size_t n) {
    if(n != 1) {
        throw std::logic_error("Can't allocate arrays");
    }
    if(size_t(memory_pool_end - memory_pool_tail) < sizeof(T)) {
        if(free_blocks.getSize()) {
            auto it = free_blocks.begin();
            char* ptr = *it;
            free_blocks.erase(it);
            return reinterpret_cast<T*>(ptr);

        }
        throw std::bad_alloc();
    }
    T* result = reinterpret_cast<T*>(memory_pool_tail);
    memory_pool_tail += sizeof(T);
    return result;
}

template<class T, size_t ALLOC_SIZE>
void allocator_t<T, ALLOC_SIZE>::deallocate(T* ptr, size_t n) {
    if(n != 1) {
        throw std::logic_error("Can't allocate arrays");
    }
    if(ptr == nullptr) {
        return;
    }
    free_blocks.push_back(reinterpret_cast<char*>(ptr));
}
}
```

tvector.hpp

```cpp
#pragma once

#include <memory>

const int GROWTH = 2;

namespace cntrs {

template <class T>
class vector_t {
public:
    using value_type = T;
    using iterator = T*;
    vector_t() : data(std::move(std::unique_ptr<T[]>(new T[GROWTH
   ]))), size(0), allocated(GROWTH) {};
    vector_t(size_t size) : data(std::move(std::unique_ptr<T[]>(
   new T[size]))), size(0), allocated(size) {};
    void push_back(const T& item);
    void resize(size_t size);
    void erase(iterator pos);
    size_t getSize() const;
    T& operator[](size_t pos);
    iterator begin() const;
    iterator end() const;
    ~vector_t() {};
private:
    std::unique_ptr<T[]> data;
    size_t size;
    size_t allocated;
};

template <class T>
void vector_t<T>::push_back(const T& item) {
    if(size == allocated) {
        this->resize(size * GROWTH);
    }
    data[size++] = item;
}

template <class T>
void vector_t<T>::resize(size_t size) {
    std::unique_ptr<T[]> newData(new T[size]);
    int n = std::min(size, this->size);
    for(int i = 0; i < n; ++i) {
        newData[i] = data[i];
    }
    data = std::move(newData);
    this->size = n;
    allocated = size;
}

template<class T>
void vector_t<T>::erase(typename vector_t<T>::iterator pos) {
    auto end = this->end();
    while(pos != end) {
        *pos = *(pos + 1);
        ++pos;
    }
    this->size--;
}

template<class T>
size_t vector_t<T>::getSize() const {
    return size;
}

template<class T>
T& vector_t<T>::operator[](size_t pos) {
    if(pos >= this->size) {
        throw std::out_of_range("out of range");
    }
    return data[pos];
}

template<class T>
typename vector_t<T>::iterator vector_t<T>::begin() const {
    return data.get();
}

template<class T>
typename vector_t<T>::iterator vector_t<T>::end() const {
    if(data) {
        return data.get() + size;
    }

    return nullptr;
}


}
```

main.cpp

```cpp
#include <iostream>
#include <algorithm>
#include <map>

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

int main() {
    std::map<int, int, std::less<int>,
        allctr::allocator_t<std::pair<const int, int>, 1000>> m;
    for(int i = 0; i < 10; ++i) {
        m[i] = i;
    }
    m.erase(1);
    cntrs::stack_t<Rhombus<double>, allctr::allocator_t<Rhombus<double>, 1000>> 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<double> 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(lab6)

add_executable(lab6
    main.cpp)

set_property(TARGET lab6 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.