

Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
III семестр
Задание 5: «Основы работы с коллекциями: итераторы»

Группа:	М8О-208Б-18, №2
Студент:	Алексеева Мария Алексеевна
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	09.12.2019

Москва, 2019

1. **Тема:** Основы работы с коллекциями: итераторы

2. **Цель работы:** Изучение основ работы с коллекциями, знакомство с шаблоном проектирования «Итератор»

3. **Задание** (вариант № 2):

Фигура — квадрат. Контейнер — отсортированный по возрастанию двусвязный список.

4. **Адрес репозитория на GitHub**

https://github.com/PowerMasha/oop_exercise_05

5. **Код программы на C++**

main.cpp

```
#include <iostream>
#include <algorithm>
#include "square.h"
#include "containers/list.h"
```

```
int main() {
    size_t N;
    float S;
    char option = '0';
    containers::list<Square<int>> q;
    Square<int> kva{};
    while (option != 'q') {
        std::cout << "выберите опцию (m for man, q to quit)" << std::endl;
        std::cin >> option;
        switch (option) {
            case 'q':
                break;
            case 'm': {
                std::cout << "1. Добавить фигуру \n"
                    << "2. Удалить фигуру \n"
                    << "3. Вывести фигуру по индексу\n"
                    << "o. Вывести все фигуры\n"
                    << "a. Вывести кол-во фигур чья площадь меньше чем ...\n";
                break;
            }

            case '1': {
                std::cout << "позиция для вставки: ";
                std::cin >> N;
                std::cout << "введите квадрат: \n";
                kva = Square<int>(std::cin);
                try {
                    kva.Check();
                }
            }
        }
    }
}
```

```

    } catch (std::logic_error &err) {
        std::cout << err.what() << std::endl;
        break;
    }
    try {
        q.insert_by_number(N, kva);
    } catch (std::logic_error &err) {
        q.delete_by_number(N);
        std::cout << err.what() << std::endl;
        break;
    }

    break;
}
case '2': {
    std::cout << "позиция для удаления: ";
    std::cin >> N;
    if (N==0){
        q.pop_front();

    }else {
        if (N == (q.length() - 1)) {
            q.pop_back();
        } else {
            try {
                q.delete_by_number(N);
            } catch (std::logic_error &err) {
                std::cout << err.what() << std::endl;
                break;
            }
        }
    }
    break;
}
case '3': {
    std::cout << "введите индекс элемента: ";
    std::cin >> N;
    q[N].Printout(std::cout);
    break;
}
case 'o': {
    std::for_each(q.begin(), q.end(), [](Square<int> &X) { X.Printout(std::cout); });
    break;
}
case 'a': {
    std::cout << "площадь для сравнения: ";
    std::cin >> S;
    std::cout << "количеств элементов с площадью меньше чем " << S << " : "
        << std::count_if(q.begin(), q.end(), [=](Square<int> &X) { return X.Area() <
S; })
        << std::endl;
    break;
}

```

```

    }
    default:
        break;
    }
}
return 0;
}

```

list.h

```

#ifndef LIST_H
#define LIST_H

```

```

#include <iterator>
#include <memory>
#include "../square.h"

```

```

namespace containers {

```

```

    template<class T>
    class list {
    private:
        struct element;
        unsigned int size = 0;
    public:
        list() = default;

        class forward_iterator {
        public:
            using value_type = T;
            using reference = T&;
            using pointer = T*;
            using difference_type = std::ptrdiff_t; //для арифметики указателей и индексации
массива
            using iterator_category = std::forward_iterator_tag; //пустой класс для идентификации
прямого итератора
            explicit forward_iterator(element* ptr);
            T& operator*();
            forward_iterator& operator++();
            forward_iterator operator++(int);
            bool operator==(const forward_iterator& other) const;
            bool operator!=(const forward_iterator& other) const;

        private:
            element* it_ptr;
            friend list;

        };
        forward_iterator begin();
        forward_iterator end();
        void pop_back();

```

```

void pop_front();
size_t length();
void delete_by_it(forward_iterator d_it);
void delete_by_number(size_t N);
void insert_by_it(forward_iterator ins_it, T& value);
void insert_by_number(size_t N, T& value);
T& operator[](size_t index) ;
list& operator=(list&& other);

private:
    struct element {
        T value;
        std::shared_ptr<element> next_element = nullptr;
        std::shared_ptr<element> prev_element = nullptr;
        forward_iterator next();
    };
    static std::shared_ptr<element> push_impl(std::shared_ptr<element> cur);
    static std::shared_ptr<element> pop_impl(std::shared_ptr<element> cur);
    std::shared_ptr<element> first = nullptr;
}; //=====end-of-class-
list=====//

template<class T>
typename list<T>::forward_iterator list<T>::begin() {
    return forward_iterator(first.get());
}

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

//=====base-methods-of-
list=====//
template<class T>
size_t list<T>::length() {
    return size;
}

template<class T>
std::shared_ptr<typename list<T>::element> list<T>::push_impl(std::shared_ptr<element>
cur) {
    if (cur -> next_element != nullptr) {
        return push_impl(cur->next_element);
    }
    return cur;
}

template<class T>
void list<T>::pop_front() {
    if (size == 0) {
        throw std::logic_error ("stack is empty");
    }
}

```

```

    }

    first = first->next_element;
    first->prev_element = nullptr;
    size--;
}

template<class T>
void list<T>::pop_back() {
    if (size == 0) {
        throw std::logic_error("can't pop from empty list");
    }
    first = pop_impl(first);
    size--;
}

template<class T>
std::shared_ptr<typename list<T>::element> list<T>::pop_impl(std::shared_ptr<element> cur)
{
    if (cur->next_element != nullptr) {
        cur->next_element = pop_impl(cur->next_element);
        return cur;
    }
    return nullptr;
}

//=====advanced-
methods=====//

template<class T>
void list<T>::delete_by_it(containers::list<T>::forward_iterator d_it) { //удаление по
итератору
    if (d_it == this->begin()) {
        this->pop_front();
        size--;
        return;
    }
    if (d_it == this->end()) {
        this->pop_back();
        size--;
        return;
    }
    if (d_it.it_ptr == nullptr) throw std::logic_error("out of borders");
    d_it.it_ptr->prev_element->next_element = d_it.it_ptr->next_element;
    d_it.it_ptr->next_element->prev_element = d_it.it_ptr->prev_element;

    size--;
}

```

//удаление по номеру

```
template<class T>
void list<T>::delete_by_number(size_t N) {
    forward_iterator it = this->begin();
    for (size_t i = 1; i <= N; ++i) {
        ++it;
    }
    this->delete_by_it(it);
}
```

```
template<class T>
void list<T>::insert_by_it(containers::list<T>::forward_iterator ins_it, T& value) {
    if (first != nullptr) {
        if (ins_it == this->begin()) {
            std::shared_ptr<element> tmp = std::shared_ptr<element>(new element{ value });
            tmp->next_element = first;
            first->prev_element = tmp;
            first = tmp;
            if (tmp->value.Area() > tmp->next_element->value.Area()) {
                throw std::logic_error("Area is too big");
            }
            size++;
            return;
        }
```

```
        if (ins_it.it_ptr == nullptr) {
            std::shared_ptr<element> tmp = std::shared_ptr<element>(new element{ value });
            tmp->prev_element = push_impl(first);
```

```
            push_impl(first)->next_element = std::shared_ptr<element>(tmp);
            if (tmp->value.Area() < tmp->prev_element->value.Area()) {
                throw std::logic_error("Area is too low");
            }
            size++;
            return;
        }
```

```
    } else {
        std::shared_ptr<element> tmp = std::shared_ptr<element>(new element{ value });
        tmp->prev_element = ins_it.it_ptr->prev_element;
        tmp->next_element = ins_it.it_ptr->prev_element->next_element;
        ins_it.it_ptr->prev_element = tmp;
        tmp->prev_element->next_element = tmp;
```

```
        if (tmp->value.Area() > tmp->next_element->value.Area()) {
            throw std::logic_error("Area is too big");
        }
```

```
        if (tmp->value.Area() < tmp->prev_element->value.Area()) {
            throw std::logic_error("Area is too low");
        }
```

```
    }
    } else first = std::shared_ptr<element>(new element{value});
```

```
    size++;
```

```

    }

    template<class T>
    void list<T>::insert_by_number(size_t N, T& value) {
        forward_iterator it = this->begin();
        for (size_t i = 0; i < N; ++i) {
            ++it;
        }
        this->insert_by_it(it, value);
    }
    //=====iterator`s-
stuff=====//
    template<class T>
    typename list<T>::forward_iterator list<T>::element::next() {
        return forward_iterator(this->next_element.get());
    }

    template<class T>
    list<T>::forward_iterator::forward_iterator(containers::list<T>::element *ptr) {
        it_ptr = ptr;
    }

    template<class T>
    T& list<T>::forward_iterator::operator*() {
        return this->it_ptr->value;
    }

    template<class T>
    typename list<T>::forward_iterator& list<T>::forward_iterator::operator++() {
        if (it_ptr == nullptr) throw std::logic_error ("out of list borders");
        *this = it_ptr->next();
        return *this;
    }

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

    template<class T>
    bool list<T>::forward_iterator::operator==(const forward_iterator& other) const {
        return it_ptr == other.it_ptr;
    }
    template<class T>
    list<T>& list<T>::operator=(list<T>&& other){
        size = other.size;
        first = std::move(other.first);
    }

    template<class T>

```



```

bool list<T>::forward_iterator::operator!=(const forward_iterator& other) const {
    return it_ptr != other.it_ptr;
}

template<class T>
T& list<T>::operator[](size_t index) {
    if (index < 0 || index >= size) {
        throw std::out_of_range("out of list's borders");
    }
    forward_iterator it = this->begin();
    for (size_t i = 0; i < index; i++) {
        it++;
    }
    return *it;
}

}

#endif //LIST_H

```

Square.h

```

#ifndef SQUARE
#define SQUARE

#include "vertex.h"

template <class T>
class Square {
public:
    vertex<T> points[4];

    explicit Square<T>(std::istream& is) {
        for (auto & point : points) {
            is >> point;
        }
    }
    Square<T>() = default;

    double Area() const {
        double res = 0;
        for (size_t i = 0; i < 3; i++) {
            res += (points[i].x * points[i+1].y) - (points[i+1].x * points[i].y);
        }
        res = res + (points[3].x * points[0].y) - (points[0].x * points[3].y);
        return std::abs(res)/ 2;
    }

    void Printout(std::ostream& os) {
        for (int i = 0; i < 4; ++i) {
            os << this->points[i];
            if (i != 3) {

```

```

        os << ", ";
    }
}
os << std::endl;
}

void Check() const {
    double a, b, c, d, d1, d2, ABC, BCD, CDA, DAB;
    a = sqrt((points[2].x- points[1].x) * (points[2].x - points[1].x) + (points[2].y - points[1].y) *
(points[2].y - points[1].y));
    b = sqrt((points[3].x- points[2].x) * (points[3].x - points[2].x) + (points[3].y - points[2].y) *
(points[3].y - points[2].y));
    c = sqrt((points[3].x- points[4].x) * (points[3].x - points[4].x) + (points[3].y - points[4].y) *
(points[3].y - points[4].y));
    d = sqrt((points[4].x- points[1].x) * (points[4].x - points[1].x) + (points[4].y - points[1].y) *
(points[4].y - points[1].y));
    d1 = sqrt((points[2].x- points[4].x) * (points[2].x - points[4].x) + (points[2].y - points[4].y)
* (points[2].y - points[4].y));
    d2 = sqrt((points[3].x- points[1].x) * (points[3].x - points[1].x) + (points[3].y - points[1].y)
* (points[3].y - points[1].y));
    ABC = (a * a + b * b - d2 * d2) / 2 * a * b;
    BCD = (b * b + c * c - d1 * d1) / 2 * b * c;
    CDA = (d * d + c * c - d2 * d2) / 2 * d * c;
    DAB = (a * a + d * d - d1 * d1) / 2 * a * d;
    if(ABC != BCD || ABC != CDA || ABC != DAB || a!=b || a!=c || a!=d )
        throw std::logic_error("Это не квадрат");
}

void operator<< (std::ostream& os) {
    for (int i = 0; i < 4; ++i) {
        os << this->points[i];
        if (i != 3) {
            os << ", ";
        }
    }
}
};

#endif

```

CmakeList.txt

```

cmake_minimum_required (VERSION 3.5)
project(lab5)
add_executable(oop_exercise_05
    main.cpp)

```

```

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra")
set_target_properties(oop_exercise_05 PROPERTIES CXX_STANDARD 14
CXX_STANDARD_REQUIRED ON)

```

6. Набор testcases

test_01.txt

1

Ожидаемое действие

```
0
0 0 0 1 1 1 1 0
1
1
0 0 0 2 2 2 2 0
a
5
```

введение элементов

вывод элементов площадью меньше 5

q

test_02.txt

Ожидаемое действие

```
1
0
0 0 0 4 4 4 4 0
1
1
0 0 0 7 7 7 7 0
1
1
0 0 0 8 8 8 8 0
1
1
0 0 0 5 5 5 5 0
2
0
o
q
```

введение элементов

ошибка(площадь слишком большая)

удаление первого элемента

вывод списка

test_03.txt

Ожидаемое действие

```
1
0
0 0 0 3 3 3 3 0
1
0
0 0 0 1 1 1 1 0
1
1
0 0 0 2 2 2 2 0
3
2
q
```

введение элементов

вывод элемента с индексом 2

7. Результаты выполнения тестов

masha@masha-VirtualBox:~/2kurs/oop_exercise_05/tmp\$./oop_exercise_05

<~/2kurs/oop_exercise_05/test_01.txt

выберите опцию (m for man, q to quit)

позиция для вставки: введите квадрат:

выберите опцию (m for man, q to quit)

позиция для вставки: введите квадрат:

выберите опцию (m for man, q to quit)

площадь для сравнения: количеств элементов с площадью меньше чем 5 :2
выберите опцию (m for man, q to quit)
masha@masha-VirtualBox:~/2kurs/oop_exercise_05/tmp\$./oop_exercise_05
<~/2kurs/oop_exercise_05/test_02.txt
выберите опцию (m for man, q to quit)
позиция для вставки: введите квадрат:
выберите опцию (m for man, q to quit)
позиция для вставки: введите квадрат:
выберите опцию (m for man, q to quit)
позиция для вставки: введите квадрат:
Area is too big
выберите опцию (m for man, q to quit)
позиция для вставки: введите квадрат:
выберите опцию (m for man, q to quit)
позиция для удаления: выберите опцию (m for man, q to quit)
(0 0), (0 5), (5 5), (5 0)
(0 0), (0 7), (7 7), (7 0)
выберите опцию (m for man, q to quit)
masha@masha-VirtualBox:~/2kurs/oop_exercise_05/tmp\$./oop_exercise_05
<~/2kurs/oop_exercise_05/test_03.txt
выберите опцию (m for man, q to quit)
позиция для вставки: введите квадрат:
выберите опцию (m for man, q to quit)
позиция для вставки: введите квадрат:
выберите опцию (m for man, q to quit)
позиция для вставки: введите квадрат:
выберите опцию (m for man, q to quit)
введите индекс элемента: (0 0), (0 3), (3 3), (3 0)
выберите опцию (m for man, q to quit)
masha@masha-VirtualBox:~/2kurs/oop_exercise_05/tmp\$

8. Объяснение результатов работы программы - вывод

Методы и члены коллекции:

size — размер коллекции

element — описание элемента коллекции

first — головной элемент коллекции

push — добавление элемента в стек

pop — удаление элемента из стека

delete_by_it — удаление элемента по итератору

delete_by_number — удаление элемента по номеру

insert_by_it — вставка элемента по итератору

insert_by_number — вставка элемента по номеру

forward_iterator — реализация итератора типа forward_iterator

В ходе данной лабораторной работы были получены навыки работы с умными указателями, в частности `unique_ptr`, а так же навыки написания итераторов, совместимыми со стандартными функциями (`std::for_each`, `std::count_if`).

Умные указатели полезны в работе с динамическими структурами, как инструменты более удобного контроля за выделением и освобождением ресурсов, что помогает избежать утечек памяти и висячих ссылок.