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.

1. **Адрес репозитория на GitHub** [https://github.com/PowerMasha/oop\_exercise\_0](https://github.com/wAlienUFOx/oop_exercise_01)5
2. **Код программы на С++**

main.cpp

#include <iostream>

#include <algorithm>

#include "square.h"

#include "conteiners/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 в начало списка\n"

<< "2. Добавить фигуру в конец списка\n"

<< "3. Добавить фигуру по индексу\n"

<< "4. Удалить фигуру в начале списка\n"

<< "5. Удалить фигуру в конце списка\n"

<< "6. Удалить фигуру по индексу\n"

<< "7. Вывести первую фигуру в списке\n"

<< "8. Вывести последнюю фигуру в списке\n"

<< "9. Вывести фигуру по индексу\n"

<< "o. Вывести все фигуры\n"

<< "a. Вывести кол-во фигур чья площаль больше чем ...\n";

break;

}

case '1':{

std::cout << "введите вершины квадрата: " << std::endl;

kva = Square<int>(std::cin);

try{

kva.Check();

}catch(std::logic\_error& err){

std::cout << err.what() << std::endl;

break;

}

try{

q.push\_front(kva);}

catch(std::logic\_error& err){

std::cout << err.what()<<std::endl;

break;

}

break;

}

case '2': {

std::cout << "введите вершины квадрата: " << std::endl;

kva = Square<int>(std::cin);

try {

kva.Check();

} catch (std::logic\_error &err) {

std::cout << err.what() << std::endl;

break;

}

try {

q.push\_back(kva);

}

catch (std::logic\_error &err) {

std::cout << err.what() << std::endl;

break;

}

break;

}

case '3': {

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 '4': {

q.pop\_front();

break;

}

case '5': {

q.pop\_back();

break;

}

case '6': {

std::cout << "позиция для удаления: ";

std::cin >> N;

q.delete\_by\_number(N);

break;

}

case '7': {

q.top\_front().Printout(std::cout);

break;

}

case '8': {

q.top\_back().Printout(std::cout);

break;

}

case '9': {

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 <<"количестов 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 push\_back(const T& value);

void push\_front(const T& value);

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) ;

T& top\_front();

T& top\_back();

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>

void list<T>::push\_front(const T& value) {

if (first == nullptr){

first = std::shared\_ptr<element>(new element{value});

} else {

if (first->value.Area() < value.Area()){

throw std::logic\_error("Area is too big");

}

auto tmp = std::shared\_ptr<element>(new element{value});

first->prev\_element = tmp;

tmp->next\_element = first;

first = tmp;

}

size++;

}

template<class T>

void list<T>::push\_back(const T& value) {

if (first == nullptr) {

first = std::shared\_ptr<element>(new element{value});

}else{

if (value.Area() < push\_impl(first) -> value.Area()){

throw std::logic\_error("Area is too low");

}

auto \*tmp = new element{value};

tmp-> prev\_element = push\_impl(first);

push\_impl(first) -> next\_element = std::shared\_ptr<element>(tmp);

}

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;

}

template<class T>

T& list<T>::top\_front() {

if (size == 0) {

throw std::logic\_error("list is empty");

}

return first->value;

}

template<class T>

T& list<T>::top\_back() {

if (size == 0) {

throw std::logic\_error("list is empty");

}

forward\_iterator i = this->begin();

while ( i.it\_ptr->next() != this->end()) {

i++;

}

return \*i;

}

//=================================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();

return;

}

if (d\_it == this->end()) {

if (size == 0) {

throw std::logic\_error("can`t pop from empty list");

}

first = pop\_impl(first);

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+1; ++i) {

if (i == N+1) break;

++it;

}

this->delete\_by\_it(it);

}

template<class T>

void list<T>::insert\_by\_it(containers::list<T>::forward\_iterator ins\_it, T& value) {

auto tmp = std::shared\_ptr<element>(new element{value});

if (ins\_it == this->begin()) {

tmp->next\_element = first;

first->prev\_element = tmp;

first = tmp;

size++;

return;

}

if (ins\_it == this->end()) {

tmp-> prev\_element = push\_impl(first);

push\_impl(first) -> next\_element = std::shared\_ptr<element>(tmp);

size++;

return;

}

tmp->next\_element = std::shared\_ptr<element>(ins\_it.it\_ptr);

tmp->prev\_element = ins\_it.it\_ptr->prev\_element;

tmp->prev\_element->next\_element = tmp;

tmp->next\_element->prev\_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("error");

}

size++;

}

template<class T>

void list<T>::insert\_by\_number(size\_t N, T& value) {

forward\_iterator it = this->begin();

for (size\_t i = 1; i <= N+1; ++i) {

if (i == N+1) break;

++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;

}

}

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() {

**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\_STANDART 14 CXX\_STANDART\_REQUIRED ON)

1. **Набор testcases**

test\_01.txt Ожидаемое действие

1 добавление элемента

0 0 0 1 1 1 1 0

1 добавление элемента

1 1 1 4 4 4 4 1 (ошибка)

o вывод списка

7 печать верхнего элемента

q

test\_02.txt Ожидаемое действие

1 добавление элемента

0 0 0 4 4 4 4 0

1

0 0 0 2 2 2 2 1

o вывод списка

3 вставить по индексу

1

0 0 0 3 3 3 3 0

8 печать последнего элемента

a вывести элементы меньше

2

q

test\_03.txt Ожидаемое действие

1 добавление элемента

0 0 0 1 1 1 1 0

1

0 0 0 0 0 0 0 0

9 печать элемента по индексу

0

4 удалить с начала списка

o

9 печать элемента по индексу

0

q

1. **Результаты выполнения тестов**

/home/masha/2kurs/oop\_exercise\_05/cmake-build-debug/oop\_exercise\_05

выберите опцию (m for man, q to quit)

1

введите вершины квадрата:

0 0 0 1 1 1 1 0

выберите опцию (m for man, q to quit)

1

введите вершины квадрата:

1 1 1 4 4 4 4 1

Area is too big

выберите опцию (m for man, q to quit)

o

(0 0), (0 1), (1 1), (1 0)

выберите опцию (m for man, q to quit)

7

(0 0), (0 1), (1 1), (1 0)

выберите опцию (m for man, q to quit)

q

/home/masha/2kurs/oop\_exercise\_05/cmake-build-debug/oop\_exercise\_05

выберите опцию (m for man, q to quit)

1

введите вершины квадрата:

0 0 0 4 4 4 4 0

выберите опцию (m for man, q to quit)

1

введите вершины квадрата:

0 0 0 2 2 2 2 1

Это не квадрат!

выберите опцию (m for man, q to quit)

1

введите вершины квадрата:

0 0 0 2 2 2 2 0

выберите опцию (m for man, q to quit)

o

(0 0), (0 2), (2 2), (2 0)

(0 0), (0 4), (4 4), (4 0)

выберите опцию (m for man, q to quit)

3

позиция для вставки: 1

введите квадрат:

0 0 0 3 3 3 3 0

выберите опцию (m for man, q to quit)

o

(0 0), (0 2), (2 2), (2 0)

(0 0), (0 3), (3 3), (3 0)

(0 0), (0 4), (4 4), (4 0)

выберите опцию (m for man, q to quit)

8

(0 0), (0 4), (4 4), (4 0)

выберите опцию (m for man, q to quit)

a

площадь для сравнения: 2

количестов элементов с площадью меньше чем 2 :0

/home/masha/2kurs/oop\_exercise\_05/cmake-build-debug/oop\_exercise\_05

выберите опцию (m for man, q to quit)

1

введите вершины квадрата:

0 0 0 1 1 1 1 0

выберите опцию (m for man, q to quit)

1

введите вершины квадрата:

0 0 0 0 0 0 0 0

выберите опцию (m for man, q to quit)

9

введите индекс элемента: 0

(0 0), (0 0), (0 0), (0 0)

выберите опцию (m for man, q to quit)

4

выберите опцию (m for man, q to quit)

o

(0 0), (0 1), (1 1), (1 0)

выберите опцию (m for man, q to quit)

9

введите индекс элемента: 0

(0 0), (0 1), (1 1), (1 0)

1. **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  
top — возвращает значение головного элемента 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 — удаление элемента по итератору  
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.