Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

**Лабораторная работа**

**по курсу «ООП»**

**Тема:**

**Основы работы с коллекциями: итераторы.**

|  |  |
| --- | --- |
| Студент: | Марков А.Н. |
| Группа: | М80-208Б-18 |
| Преподаватель: | Журавлев А.А. |
| Вариант: | 17 |
| Оценка: |  |
| Дата: |  |

Москва

2019

**1. Постановка задачи**

Вариант 17. Фигура: треугольник; контейнер: очередь.

Создать шаблон динамической коллекции, согласно варианту задания:

1. Коллекция должна быть реализована с помощью умных указателей.
2. В качестве параметра шаблона коллекция должна принимать тип данных.
3. Реализовать forward\_iterator по коллекции.
4. Коллекция должна возвращать итераторы begin() и end().
5. Коллекция должна содержать метод вставки на позицию итератора insert(iterator).
6. Коллекция должна содержать метод удаления из позиции итератора erase(iterator).
7. При выполнении недопустимых операций (например выход за границы коллекции или удаление не существующего элемента) необходимо генерировать исключения.
8. Итератор должен быть совместим со стандартными алгоритмами (например, std::count\_if).
9. Коллекция должна содержать метод доступа.
10. Реализовать программу, которая:

* Позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять в коллекцию.
* Позволяет удалять элемент из коллекции по номеру элемента.
* Выводит на экран введенные фигуры с помощью std::for\_each.
* Выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count\_if).

**2. Код программы на языке C++**

**main.cpp:**

#include <iostream>

#include <string>

#include <algorithm>

#include <exception>

#include "queue.h"

#include "triangle.h"

#include "vertex.h"

#include "vector.h"

int main() {

Queue<Triangle<int>> q;

std::string cmd;

std::cout << "Operations: Add/ Remove/ Print/ Front/ Back/ Count\_if/ Menu/ Exit" << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

while (std::cin >> cmd) {

if (cmd == "Add") {

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << "Add an item to the back of the queue[Push] or to the iterator position[Iter]" << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cin >> cmd;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

if (cmd == "Push") {

Triangle<int> t;

std::cout << "Input points: ";

try {

std::cin >> t;

}

catch (std::exception &e) {

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

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

continue;

}

q.Push(t);

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

}

else if (cmd == "Iter") {

Triangle<int> t;

std::cout << "Input points: ";

try {

std::cin >> t;

}

catch (std::exception &e) {

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

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

continue;

}

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << "Input index: ";

int i;

std::cin >> i;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

Queue<Triangle<int>>::ForwardIterator it = q.Begin();

for (int cnt = 0; cnt < i; cnt++) {

it++;

}

q.Insert(it, t);

}

else {

std::cout << "Invalid input" << std::endl;

std::cin.clear();

std::cin.ignore(30000, '\n');

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

continue;

}

}

else if (cmd == "Remove") {

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << "Delete item from front of queue[Pop] or to the iterator position[Iter]" << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cin >> cmd;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

if (cmd == "Pop") {

q.Pop();

}

else if (cmd == "Iter") {

std::cout << "Input index: ";

int i;

std::cin >> i;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

Queue<Triangle<int>>::ForwardIterator it = q.Begin();

for (int cnt = 0; cnt < i; cnt++) {

it++;

}

q.Erase(it);

}

else {

std::cout << "Invalid input" << std::endl;

std::cin.clear();

std::cin.ignore(30000, '\n');

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

continue;

}

}

else if (cmd == "Print") {

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

q.Print();

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

}

else if (cmd == "Front") {

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

Triangle<int> value;

try {

value = q.Front();

}

catch (std::exception &e) {

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

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

continue;

}

std::cout << value << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

}

else if (cmd == "Back") {

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

Triangle<int> value;

try {

value = q.Back();

}

catch (std::exception &e) {

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

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

continue;

};

std::cout << value << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

}

else if (cmd == "Count\_if") {

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << "Input area: ";

double area;

std::cin >> area;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << "The number of figures with an area less than a given " << std::count\_if(q.Begin(), q.End(), [area](Triangle<int> t){

return Area(t) < area;

}) << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

}

else if (cmd == "Menu") {

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << "Operations: Add/ Remove/ Print/ Front/ Back/ Count\_if/ Menu/ Exit" << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

}

else if (cmd == "Exit") {

break;

}

else {

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << "Invalid input" << std::endl;

std::cin.clear();

std::cin.ignore(30000, '\n');

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

}

}

return 0;

}

**queue.h:**

#ifndef QUEUE\_H

#define QUEUE\_H 1

#include <iostream>

#include <memory>

#include <algorithm>

template<typename T>

class Queue {

using value\_type = T;

using size\_type = size\_t;

using reference = value\_type &;

using const\_reference = const value\_type &;

using pointer = value\_type \*;

using const\_pointer = const value\_type \*;

private:

class Node {

public:

Node(T val) : next{nullptr}, prev{next}, value{val} {};

friend class Queue;

private:

std::shared\_ptr<Node> next;

std::weak\_ptr<Node> prev;

T value;

};

public:

class ForwardIterator {

public:

using value\_type = T;

using reference = T&;

using pointer = T\*;

using difference\_type = ptrdiff\_t;

using iterator\_category = std::forward\_iterator\_tag;

friend class Queue;

ForwardIterator(std::shared\_ptr<Node> it = nullptr) : ptr{it} {};

ForwardIterator(const ForwardIterator &other) : ptr{other.ptr} {};

ForwardIterator operator++() {

if (ptr.lock() == nullptr) {

return \*this;

}

ptr = ptr.lock()->next;

return \*this;

}

ForwardIterator operator++(int s) {

if (ptr.lock() == nullptr) {

return \*this;

}

ForwardIterator old{this->ptr.lock()};

++(\*this);

return old;

}

reference operator\*() {

return ptr.lock()->value;

}

const\_reference operator\*() const {

return ptr.lock()->value;

}

std::shared\_ptr<Node> operator->() {

return ptr.lock();

}

std::shared\_ptr<const Node> operator->() const {

return ptr.lock();

}

bool operator==(const ForwardIterator &rhs) const {

return ptr.lock() == rhs.ptr.lock();

}

bool operator!=(const ForwardIterator &rhs) const {

return ptr.lock() != rhs.ptr.lock();

}

ForwardIterator Next() const {

if (ptr.lock() == nullptr)

return ForwardIterator{};

return ptr.lock()->next;

}

private:

std::weak\_ptr<Node> ptr;

};

Queue() : head{nullptr}, tail{head}, size{0} {};

void Push(const T& val) {

if (!head) {

head = std::make\_shared<Node>(val);

tail = head;

}

else {

std::shared\_ptr<Node> newElem = std::make\_shared<Node>(val);

newElem->prev = tail;

tail.lock()->next = newElem;

tail = newElem;

}

size++;

}

ForwardIterator Insert(const ForwardIterator it, const T& val) {

if (it == ForwardIterator{}) {

if (tail.lock() == nullptr) {

Push(val);

return Begin();

}

std::shared\_ptr<Node> newElem = std::make\_shared<Node>(val);

newElem->prev = tail;

tail.lock()->next = newElem;

tail = newElem;

size++;

return newElem;

}

if (it == Begin()) {

std::shared\_ptr<Node> newElem = std::make\_shared<Node>(val);

newElem->next = it.ptr.lock();

it->prev.lock() = newElem;

head = newElem;

size++;

return newElem;

}

std::shared\_ptr<Node> newElem = std::make\_shared<Node>(val);

newElem->next = it.ptr.lock();

it->prev.lock()->next = newElem;

newElem->prev = it->prev;

it->prev.lock() = newElem;

size++;

return newElem;

}

void Pop() {

if (head) {

head = head->next;

size--;

}

}

ForwardIterator Erase(const ForwardIterator it) {

if (it == ForwardIterator{}) { //удаление несуществующего элемента

return End();

}

if (it->prev.lock() == nullptr && head == tail.lock()) { //удаление очереди, состоящей только из одного элемента

head = nullptr;

tail = head;

size = 0;

return End();

}

if (it->prev.lock() == nullptr) { //удаление первого элемента

it->next->prev.lock() = nullptr;

head = it->next;

size--;

return head;

}

ForwardIterator res = it.Next();

if (res == ForwardIterator{}) { //удаление последнего элемента

it->prev.lock()->next = nullptr;

size--;

return End();

}

//удаление элементов в промежутке

it->next->prev = it->prev;

it->prev.lock()->next = it->next;

size--;

return res;

}

reference Front() {

if (head == nullptr)

throw std::out\_of\_range("Empty item");

return this->head->value;

}

const\_reference Front() const {

if (head == nullptr)

throw std::out\_of\_range("Empty item");

return this->head->value;

}

reference Back() {

if (head == nullptr)

throw std::out\_of\_range("Empty item");

return this->tail.lock()->value;

}

const\_reference Back() const {

if (head == nullptr)

throw std::out\_of\_range("Empty item");

return this->tail.lock()->value;

}

ForwardIterator Begin() {

return head;

}

ForwardIterator End() {

return ForwardIterator{};

}

bool Empty() const {

return size == 0;

}

size\_type Size() const {

return size;

}

void Swap(Queue &rhs) {

std::shared\_ptr<Node> temp = head;

head = rhs.head;

rhs.head = temp;

}

void Clear() {

head = nullptr;

tail = head;

size = 0;

}

void Print() {

ForwardIterator it = Begin();

std::for\_each(Begin(), End(), [it, this](auto e)mutable{

std::cout << e;

if (it.Next() != this->End()) {

std::cout << " <- ";

}

it++;

});

std::cout << "\n";

}

private:

std::shared\_ptr<Node> head;

std::weak\_ptr<Node> tail;

size\_t size;

};

#endif //QUEUE\_H

**vertex.h:**

#ifndef VERTEX\_H

#define VERTEX\_H 1

template<typename T>

struct vertex {

using vertex\_t = std::pair<T, T>;

};

template<typename T>

std::istream &operator>>(std::istream &is, std::pair<T, T> &v) {

is >> v.first >> v.second;

return is;

}

template<typename T>

std::ostream &operator<<(std::ostream &os, const std::pair<T,T> &v) {

os << "[" << v.first << ", " << v.second << "]";

return os;

}

#endif // VERTEX\_H

**vector.h:**

#ifndef VECTOR\_H

#define VECTOR\_H 1

#include <utility>

#include <cmath>

#include <iostream>

#include "vertex.h"

template<typename T>

struct Vector {

using vertex\_t = std::pair<T, T>;

T p1, p2;

Vector(T x\_cord, T y\_cord) : p1{x\_cord}, p2{y\_cord} {};

Vector(vertex\_t &p1, vertex\_t &p2) : p1{p2.first - p1.first},

p2{p2.second - p1.second} {};

double operator\*(const Vector<T> &a) const {

return (p1 \* a.p1) + (p2 \* a.p2);

}

Vector<T> &operator=(const Vector<T> &a) {

p1 = a.p1;

p2 = a.p2;

return \*this;

}

};

template<typename T>

double Length(const Vector<T> &vector) {

return sqrt(vector.p1 \* vector.p1 + vector.p2 \* vector.p2);

}

template<typename T>

double Length(const std::pair<T, T> &A,

const std::pair<T, T> &B) {

return sqrt(pow((B.first - A.first), 2) +

pow((B.second - A.second), 2));

}

template<typename T>

bool is\_parallel(const Vector<T> &A, const Vector<T> &B) {

return (A.p1 \* B.p2) - (A.p2 \* B.p1) == 0;

}

#endif //VECTOR\_H

**triangle.h:**

#ifndef TRIANGLE\_H

#define TRIANGLE\_H

#include <utility>

#include <iostream>

#include "vector.h"

#include "vertex.h"

template<typename T>

struct Triangle {

using vertex\_t = std::pair<T,T>;

vertex\_t vertices[3];

};

template<typename T>

typename Triangle<T>::vertex\_t Center(const Triangle<T> &t);

template<typename T>

double Area(const Triangle<T> &t);

template<typename T>

std::ostream &Print(std::ostream &os, const Triangle<T> &t);

template<typename T>

std::istream &Read(std::istream &is, Triangle<T> &t);

template<typename T>

std::istream &operator>>(std::istream &is, Triangle<T> &t);

template<typename T>

std::ostream &operator<<(std::ostream &os, const Triangle<T> &t);

template<typename T>

typename Triangle<T>::vertex\_t Center(const Triangle<T> &t) {

T x, y;

x = (t.vertices[0].first + t.vertices[1].first + t.vertices[2].first) / 3;

y = (t.vertices[0].second + t.vertices[1].second + t.vertices[2].second) / 3;

return std::make\_pair(x, y);

}

template<typename T>

double Area(const Triangle<T> &t) {

double res = 0;

for (int i = 0; i <= 1; i++) {

res += (t.vertices[i].first \* t.vertices[i + 1].second -

t.vertices[i + 1].first \* t.vertices[i].second);

}

res += (t.vertices[2].first \* t.vertices[0].second -

t.vertices[0].first \* t.vertices[2].second);

res = 0.5 \* std::abs(res);

return res;

}

template<typename T>

std::ostream &Print(std::ostream &os, const Triangle<T> &t) {

for (int i = 0; i < 3; i++) {

os << t.vertices[i];

if (i != 2) {

os << " ";

}

}

return os;

}

template<typename T>

std::istream &Read(std::istream &is, Triangle<T> &t) {

for (int i = 0; i < 3; i++) {

is >> t.vertices[i].first >> t.vertices[i].second;

}

double AB = Length(t.vertices[0], t.vertices[1]),

BC = Length(t.vertices[1], t.vertices[2]),

AC = Length(t.vertices[0], t.vertices[2]);

if (AB >= BC + AC || BC >= AB + AC || AC >= AB + BC) {

throw std::logic\_error("Vertices must not be on the same line.");

}

return is;

}

template<typename T>

std::istream &operator>>(std::istream &is, Triangle<T> &t) {

return Read(is, t);

}

template<typename T>

std::ostream &operator<<(std::ostream &os, const Triangle<T> &t) {

return Print(os, t);

}

#endif // TRIANGLE\_H

**3. Ссылка на репозиторий на GitHub.**

<https://github.com/Markov-A-N/oop_exercise_05.git>

**4. Набор testcases.**

**test\_01.txt:**

Add

Push

0 0 3 0 3 7

Add

Push

1 1 4 1 4 8

Add

Iter

-1 -1 3 -1 3 7

1

Print

Remove

Pop

Front

Back

Count\_if

22test

Remove

Iter

2

Print

Remove

Pop

Remove

Pop

Print

Exit

**test\_02.txt:**

Add

Push

0 0 5 0 5 9

Add

Iter

10 10 12 10 12 15

3

Add

Iter

0 0 1 1 -1 -1

Print

Front

Back

Exit

**test\_03.txt:**

Back

Front

Add

Push

0 0 1 0 1 3

Add

Push

0 0 1 0 1 1

Remove

Pop

Print

Exit

**5. Результаты выполнения тестов.**

**test\_01.txt:**

Operations: Add/ Remove/ Print/ Front/ Back/ Count\_if/ Menu/ Exit

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add an item to the back of the queue[Push] or to the iterator position[Iter]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Push

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input points: 0 0 3 0 3 7

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add an item to the back of the queue[Push] or to the iterator position[Iter]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Push

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input points: 1 1 4 1 4 8

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add an item to the back of the queue[Push] or to the iterator position[Iter]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Iter

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input points: -1 -1 3 -1 3 7

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input index: 1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Print

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[0, 0] [3, 0] [3, 7] <- [-1, -1] [3, -1] [3, 7] <- [1, 1] [4, 1] [4, 8]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Remove

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Delete item from front of queue[Pop] or to the iterator position[Iter]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pop

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Front

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[-1, -1] [3, -1] [3, 7]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Back

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[1, 1] [4, 1] [4, 8]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Count\_if

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input area: 22

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The number of figures with an area less than a given 2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Remove

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Delete item from front of queue[Pop] or to the iterator position[Iter]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Iter

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input index: 2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Print

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[-1, -1] [3, -1] [3, 7] <- [1, 1] [4, 1] [4, 8]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Remove

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Delete item from front of queue[Pop] or to the iterator position[Iter]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pop

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Remove

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Delete item from front of queue[Pop] or to the iterator position[Iter]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pop

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Print

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Exit

**test\_02.txt:**

Operations: Add/ Remove/ Print/ Front/ Back/ Count\_if/ Menu/ Exit

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add an item to the back of the queue[Push] or to the iterator position[Iter]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Push

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input points: 0 0 5 0 5 9

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add an item to the back of the queue[Push] or to the iterator position[Iter]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Iter

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input points: 10 10 12 10 12 15

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input index: 3

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add an item to the back of the queue[Push] or to the iterator position[Iter]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Iter

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input points: 0 0 1 1 -1 -1

Vertices must not be on the same line.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Print

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[0, 0] [5, 0] [5, 9] <- [10, 10] [12, 10] [12, 15]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Front

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[0, 0] [5, 0] [5, 9]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Back

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[10, 10] [12, 10] [12, 15]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Exit

**test\_03.txt:**

Operations: Add/ Remove/ Print/ Front/ Back/ Count\_if/ Menu/ Exit

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Back

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Empty item

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Front

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Empty item

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add an item to the back of the queue[Push] or to the iterator position[Iter]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Push

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input points: 0 0 1 0 1 3

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add an item to the back of the queue[Push] or to the iterator position[Iter]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Push

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input points: 0 0 1 0 1 1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Remove

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Delete item from front of queue[Pop] or to the iterator position[Iter]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pop

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Print

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[0, 0] [1, 0] [1, 1]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Exit

**6. Объяснение результатов работы программы.**

Контейнер «очередь» реализован с помощью умных указателей. Сам класс Queue содержит умный указатель std::shared\_ptr на первый элемент очереди, умный указатель std::weak\_\_ptr на последний элемент и размер очереди size\_t size.

Элемент очереди реализован с помощью class Node, который содержит в себе умный указатель std::shared\_ptr на следующий элемент очереди, умный указатель std::weak\_ptr на предыдущий элемент и значение.

Указатель на предыдущий элемент очереди сделан с помощью weak\_ptr, чтобы можно было легко удалять элемент из очереди, изменяя только указатели next, тем самым то, на что указывали раньше shared\_ptr`ы, удаляется, т. к. на него указывают только weak\_ptr`ы, а weak\_ptr содержит слабую ссылку и не учитывается при подсчете количества указателей на какой-то объект.

**7. Вывод.**

Выполняя данную лабораторную, я получил опыт работы с умными указателями и итераторами.

Умные указатели — хорошая вещь, т. к. они при выходе из области видимости сами удаляют то, на что указывали, и поэтому они позволяют избежать утечек памяти.

Главное предназначение итераторов заключается в предоставлении возможности пользователю обращаться к любому элементу контейнера при сокрытии внутренней структуры контейнера от пользователя. Это позволяет контейнеру хранить элементы любым способом при допустимости работы пользователя с ним как с простой последовательностью или [списком](https://ru.wikipedia.org/wiki/Список_(информатика)).