Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

**Лабораторной работе № 05**

**по курсу «Объектно-ориентированное программирование»**

**Тема:**

**«Основы работы с коллекциями : итераторы»**

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

Москва

2019

Создать шаблон динамической коллекции, согласно варианту задания:

* Коллекция должна быть реализована с помощью умных указателей
* Реализовать forward\_iterator по коллекции
* Коллекция должна возвращать итераторы begin() и end();
* Коллекция должна содержать метод вставки на позицию итератора insert(iterator)
* Коллекция должна содержать метод удаления из позиции итератора erase(iterator)
* При выполнении недопустимых операций генерировать исключения
* Итератор должен быть совместим со стандартными алгоритмами
* Коллекция должна содержать метод доступа : pop, push, top

**Код программы**

que.h

#ifndef \_QUEUE\_H\_

#define \_QUEUE\_H\_

#include <iterator>

#include <memory>

#include <array>

#include <iostream>

#include<string.h>

template<class T>

struct que{

private:

struct que\_el;

std::shared\_ptr<que\_el> front = nullptr;

std::shared\_ptr<que\_el> back = nullptr;

//que() = default;

//que(const que&) = delete;

que &operator = (const que &) = delete;

size\_t size = 0;

public:

void push(T value);

void pop();

T& top();

struct forward\_it{

using value\_type = T;

using reference = T &;

using pointer = T \*;

using difference\_type = ptrdiff\_t;

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

forward\_it(que\_el \*el) : ptr(el) {};

T& operator\* ();

forward\_it& operator++ ();

forward\_it operator++ (int r);

forward\_it next\_it();

bool operator== (const forward\_it& it) const;

bool operator!= (const forward\_it& it) const;

//private:

que\_el\* ptr;

friend class que;

};

forward\_it begin();

forward\_it end();

void insert(int &ind, const T &val);

void insert\_it(const forward\_it& it, const T& value);

void erase(const forward\_it& it);

private:

struct que\_el{

T value;

std::shared\_ptr<que\_el> tail = nullptr;

forward\_it next\_it();

que\_el(const T& value, std::shared\_ptr<que\_el> next) : value(value), tail(next) {};

};

};

template<class T>

void que<T>::insert(int &ind, const T &val){

int i = ind - 1;

try{

if(i == -1){

if(size == 0){

this -> push(val);

return;

}

std::shared\_ptr<que\_el> p(new que\_el{val, nullptr});

p -> tail = front;

front = p;

return;

} else if(i < -1 || i > this -> size){

throw "Неверный индекс";

} else {

auto it = this -> begin();

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

++it;

}

this -> insert\_it(it, val);

}

}

catch(const char\* str){

std::cout << str << '\n';

return;

}

}

template<class T>

void que<T>::insert\_it(const que<T>::forward\_it &it, const T &value) {

std::shared\_ptr<que\_el> p(new que\_el{value, nullptr});

try{

if(it.ptr == nullptr && size != 0){

throw "индекс неверный";

}

if(it.ptr == back.get()){

this -> push(value);

return;

}

p -> tail = it.ptr -> tail;

it.ptr -> tail = p;

++ size;

} catch(const char\* str){

std::cout << str << '\n';

return;

}

}

template<class T>

void que<T>::erase(const que<T>::forward\_it &it) {

try{

if (it.ptr == nullptr) {

throw "Нельзя удалить";

}

if(it == this -> begin()){

if(front == back){

this -> pop();

return;

}

front = front -> tail;

} else {

auto tmp = this -> begin();

while(tmp.ptr -> next\_it() != it.ptr){

++ tmp;

}

tmp.ptr -> tail = it.ptr -> tail;

}

}

catch(const char\* str){

std::cout << str << "\n";

}

}

template<class T>

typename que<T>::forward\_it que<T>::begin() {

return front.get();

}

template<class T>

typename que<T>::forward\_it que<T>::end() {

return nullptr;

}

template<class T>

typename que<T>::forward\_it que<T>::que\_el::next\_it() {

return tail.get();

}

template<class T>

T &que<T>::forward\_it::operator\*() {

return ptr -> value;

}

template<class T>

typename que<T>::forward\_it &que<T>::forward\_it::operator++() {

\*this = ptr -> next\_it();

return \*this;

}

template<class T>

typename que<T>::forward\_it que<T>::forward\_it::operator++(int) {

forward\_it old = \*this;

++\*this;

return old;

}

template<class T>

bool que<T>::forward\_it::operator==(const forward\_it &it) const{

return ptr == it.ptr;

}

template<class T>

bool que<T>::forward\_it::operator!=(const forward\_it &it) const{

return ptr != it.ptr;

}

template<class T>

void que<T>::push(T value){

std::shared\_ptr<que\_el> p(new que\_el{value, nullptr});

try{

if(p == nullptr){

throw "Новый элемент не вставился";

}

p -> tail = nullptr;

++ size;

if(back == nullptr){

front = p;

back = p;

return;

}

back -> tail = p;

back = p;

}

catch(char\* str){

std::cout << str << "\n";

}

}

template<class T>

void que<T>::pop(){

try{

if(front == nullptr){

throw "Очередь пуста";

}

std::shared\_ptr<que\_el> p = front;

front = front -> tail;

-- size;

if(front == nullptr){

back = nullptr;

return;

}

}

catch(const char\* str){

std::cout << str << "\n";

return;

}

}

template<class T>

T& que<T>::top(){

try{

if(front == nullptr){

throw "Очередь пуста";

}

return front -> value;

}

catch(const char\* str){

std::cout << str << "\n";

}

}

#endif

triangle.h

#ifndef D\_TRIANGLE\_H\_

#define D\_TRIANGLE\_H\_ 1

#include <algorithm>

#include <iostream>

#include "vertex.h"

template<class T>

struct triangle {

private:

vertex<T> verts[3];

public:

triangle(std::istream& is);

triangle(vertex<T>& p1, vertex<T>& p2, vertex<T>& p3);

double area() const;

vertex<double> center() const;

void print(std::ostream& os) const;

};

template<class T>

triangle<T>::triangle(std::istream& is) {

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

is >> verts[i];

}

if((verts[0].x - verts[1].x) \* (verts[2].y - verts[1].y) == (verts[0].y - verts[1].y) \* (verts[2].x - verts[1].x)){

throw std::logic\_error("It is not triangle");

}

}

template<class T>

triangle<T>::triangle(vertex<T>& p1, vertex<T>& p2, vertex<T>& p3){

verts[0].x = p1.x;

verts[0].y = p1.y;

verts[1].x = p2.x;

verts[1].y = p2.y;

verts[2].x = p3.x;

verts[2].y = p3.y;

if((verts[0].x - verts[1].x) \* (verts[2].y - verts[1].y) == (verts[0].y - verts[1].y) \* (verts[2].x - verts[1].x)){

throw std::logic\_error("It is not triangle");

}

}

template<class T>

double triangle<T>::area() const {

const T dx1 = verts[1].x - verts[0].x;

const T dy1 = verts[1].y - verts[0].y;

const T dx2 = verts[2].x - verts[0].x;

const T dy2 = verts[2].y - verts[0].y;

return std::abs(dx1 \* dy2 - dy1 \* dx2) \* 0.5;

}

template<class T>

void triangle<T>::print(std::ostream& os) const {

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

os << '[' << verts[i] << ']';

if(i + 1 != 3){

os << ' ';

}

}

os << '\n';

return;

}

template<class T>

vertex<double> triangle<T>::center() const{

double mid1 = (verts[0].x + verts[1].x + verts[2].x) / 3;

double mid2 = (verts[0].y + verts[1].y + verts[2].y) / 3;

return vertex<double>{mid1, mid2};

}

#endif // D\_TRIANGLE\_H\_

main.cpp

#include <iterator>

#include <memory>

#include <array>

#include <iostream>

#include <string.h>

#include <algorithm>

#include "queue.h"

#include "triangle.h"

#include "vertex.h"

int main(){

que<triangle<double>> q;

for(;;){

char com[40];

std::cin >> com;

if(strcmp(com, "push") == 0){

vertex<double> p1, p2, p3;

std::cin >> p1 >> p2 >> p3;

triangle<double> r\_tr = triangle<double>(p1, p2, p3);

q.push(r\_tr);

} else if(strcmp(com, "print\_top") == 0){

std::cout << q.top().area() << "\n";

} else if(strcmp(com, "pop") == 0){

q.pop();

} else if(strcmp(com, "exit") == 0){

break;

} else if(strcmp(com, "insert") == 0){

int index;

std::cin >> index;

vertex<double> p1, p2, p3;

std::cin >> p1 >> p2 >> p3;

triangle<double> r\_tr = triangle<double>(p1, p2, p3);

auto it = q.begin();

q.insert(index, r\_tr);

} else if(strcmp(com, "erase") == 0){

int index;

std::cin >> index;

auto it = q.begin();

while(index !=0 && it != q.end()){

++ it;

-- index;

}

q.erase(it);

/\*} else if(strcmp(com, "print\_all") == 0){

auto it = q.begin();

while(it != q.end()){

std::cout << it.ptr -> value.area() << " ";

++ it;

}

std::cout << '\n'; \*/

} else if(strcmp(com, "less") == 0){

double val;

std::cin >> val;

int res = std::count\_if(q.begin(), q.end(), [val](triangle<double> fig) { return fig.area() < val; });

std::cout << res << '\n';

} else {

std::cout << "Неверно написана команда" << "\n";

}

}

}

**Ссылка на репозиторий на GitHub**

[https://github.com/AlexPshen/oop\_exercise\_05.git](https://github.com/AlexPshen/oop_exercise_03.git)

**Набор тестов**

test\_01.txt

push 0 0 1 1 0 1

print\_top

push 0 0 2 2 0 2

print\_top

pop

print\_top

pop

pop

exit

test\_02.txt

insert 0

0 0 1 1 0 1

print\_top

insert 0

0 0 2 2 0 2

print\_top

erase 1

insert 1

0 0 4 4 0 4

print\_top

pop

print\_top

erase 0

pop

exit

test\_03.txt

insert 0

0 0 9 9 0 9

less 90

insert 0

0 0 1 1 0 1

less 2

less 80

exit

**Результаты**

test\_01.txt

push 0 0 1 1 0 1

print\_top

0.5

push 0 0 2 2 0 2

print\_top

0.5

pop

print\_top

2

pop

pop

Очередь пуста

exit

test\_02.txt

insert 0

0 0 1 1 0 1

print\_top

0.5

insert 0

0 0 2 2 0 2

print\_top

2

erase 1

insert 1

0 0 4 4 0 4

print\_top

2

pop

print\_top

8

erase 0

pop

Очередь пуста

exit

test\_03.txt

insert 0

0 0 9 9 0 9

less 90

1

insert 0

0 0 1 1 0 1

less 2

1

less 80

2

exit

**Объяснение работы программы**

На ввод подаются команда com1. Если com1 является:

* push, то добавляем треугольник в очредь
* pop, то удаляем треугольник из очереди
* insert, то добавляем по индексу треугольник в очредь
* erase,то удаляем по индексу треугольник из очреди

**Вывод**

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. Проектирование класса итератора обычно тесно связано с соответствующим классом контейнера. Обычно контейнер предоставляет методы создания итераторов.