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: | Симонов С.Я. |
| Группа: | М8О-206Б-18 |
| Преподаватель: | Журавлев А.А. |
| Вариант: | 21 |
| Оценка: |  |
| Дата: |  |

Москва

2019

**1. Код на C++:**

vertex.hpp

#ifndef D\_VERTEX\_HPP\_

#define D\_VERTEX\_HPP\_ 1

#include <iostream>

template<class T>

struct vertex {

T x;

T y;

};

template <class T>

std::istream& operator>> (std::istream& is, vertex<T>& p) {

is >> p.x >> p.y;

return is;

}

template<class T>

std::ostream& operator<< (std::ostream& os, const vertex<T>& p) {

os << '[' << ' ' << p.x << ' ' << p.y << ' ' << ']';

return os;

}

template <class T>

vertex<T> operator+ (vertex<T> p1, vertex<T> p2) {

vertex<T> p;

p.x = p1.x + p2.x;

p.y = p1.y + p2.y;

return p;

}

template <class T>

vertex<T>& operator/ (vertex<T>& p, int num) {

p.x = p.x / num;

p.y = p.y / num;

return p;

}

#endif // D\_VERTEX\_HPP\_

rhombus.hpp

#ifndef D\_RHOMBUS\_HPP\_

#define D\_RHOMBUS\_HPP\_ 1

#include <algorithm>

#include <iostream>

#include <assert.h>

#include <cmath>

#include "vertex.hpp"

template<class T>

struct rhombus {

public:

rhombus (std::istream& is);

bool correct() const;

vertex<double> center() const;

double area() const;

double perimeter() const;

void print() const;

private:

vertex<T> a1, a2, a3, a4;

};

template <class T>

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

is >> a1 >> a2 >> a3 >> a4;

assert(correct());

}

template <class T>

bool rhombus<T>::correct() const {

T str1, str2, str3, str4;

str1 = sqrt((a2.x - a1.x) \* (a2.x - a1.x) + (a2.y - a1.y) \* (a2.y - a1.y));

str2 = sqrt((a3.x - a2.x) \* (a3.x - a2.x) + (a3.y - a2.y) \* (a3.y - a2.y));

str3 = sqrt((a4.x - a3.x) \* (a4.x - a3.x) + (a4.y - a3.y) \* (a4.y - a3.y));

str4 = sqrt((a1.x - a4.x) \* (a1.x - a4.x) + (a1.y - a4.y) \* (a1.y - a4.y));

if (str1 == str2 && str2 == str3 && str3 == str4) {

return true;

}

return false;

}

template <class T>

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

vertex<double> p;

p.x = (a1.x + a2.x + a3.x + a4.x) / 4;

p.y = (a1.y + a2.y + a3.y + a4.y) / 4;

return p;

}

template <class T>

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

const T s1 = 0.5 \* abs((a2.x - a1.x) \* (a3.y - a1.y) - (a3.x - a1.x) \* (a2.y - a1.y));

const T s2 = 0.5 \* abs((a3.x - a1.x) \* (a4.y - a1.y) - (a4.x - a1.x) \* (a3.y - a1.y));

return s1 + s2;

}

template <class T>

double rhombus<T>::perimeter() const {

const T str1 = sqrt((a2.x - a1.x) \* (a2.x - a1.x) + (a2.y - a1.y) \* (a2.y - a1.y));

const T str2 = sqrt((a3.x - a2.x) \* (a3.x - a2.x) + (a3.y - a2.y) \* (a3.y - a2.y));

const T str3 = sqrt((a4.x - a3.x) \* (a4.x - a3.x) + (a4.y - a3.y) \* (a4.y - a3.y));

const T str4 = sqrt((a1.x - a4.x) \* (a1.x - a4.x) + (a1.y - a4.y) \* (a1.y - a4.y));

return str1 + str2 + str3 + str4;

}

template <class T>

void rhombus<T>::print() const {

std::cout << a1 << ' ' << a2 << ' ' << a3 << ' ' << a4 << '\n';

}

#endif

queue.hpp

#ifndef D\_QUEUE\_HPP\_

#define D\_QUEUE\_HPP\_ 1

#include <iterator>

#include <memory>

#include <utility>

namespace containers {

template <class T>

struct queue {

private:

struct element;

public:

queue() = default;

struct forward\_iterator {

public:

using value\_type = T;

using reference = T&;

using pointer = T\*;

using difference\_type = std::ptrdiff\_t;

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

forward\_iterator(element \*ptr);

T& operator\* ();

forward\_iterator& operator++ ();

forward\_iterator operator++ (int);

bool operator== (const forward\_iterator& o) const;

bool operator!= (const forward\_iterator& o) const;

private:

element\* ptr\_ = nullptr;

friend queue;

};

forward\_iterator begin();

forward\_iterator end();

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

void insert\_to\_num(int pos, const T& value);

void erase(forward\_iterator& it);

void erase\_to\_num(int pos);

void pop();

void push(const T& value);

T& top();

private:

struct element {

T value;

std::unique\_ptr<element> next\_element = nullptr;

forward\_iterator next();

};

std::unique\_ptr<element> first = nullptr;

};

template <class T>

typename queue<T>::forward\_iterator queue<T>::begin() {

if (first == nullptr) {

return nullptr;

}

return forward\_iterator(first.get());

}

template <class T>

typename queue<T>::forward\_iterator queue<T>::end() {

return forward\_iterator(nullptr);

}

template <class T>

void queue<T>::insert\_to\_num(int pos, const T& value) {

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

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

if (i == pos) {

break;

}

++iter;

}

this->insert(iter, value);

}

template <class T>

void queue<T>::insert(containers::queue<T>::forward\_iterator& ptr, const T& value) {

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

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

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

val->next\_element = std::move(first);

first = std::move(val);

return;

}

while ((it.ptr\_ != nullptr) && (it.ptr\_->next() != ptr)) {

++it;

}

if (it.ptr\_ == nullptr) {

throw std::logic\_error ("ERROR");

}

val->next\_element = std::move(it.ptr\_->next\_element);

it.ptr\_->next\_element = std::move(val);

}

template <class T>

void queue<T>::erase\_to\_num(int pos) {

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

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

if (i == pos) {

break;

}

++iter;

}

this->erase(iter);

}

template <class T>

void queue<T>::erase(containers::queue<T>::forward\_iterator& ptr) {

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

if (ptr == end) {

throw std::logic\_error("ERROR");

}

if (ptr == it) {

this->pop();

return;

}

while ((it.ptr\_ != nullptr) && (it.ptr\_->next() != ptr)) {

++it;

}

if (it.ptr\_ == nullptr) {

throw std::logic\_error("ERROR");

}

it.ptr\_->next\_element = std::move(ptr.ptr\_->next\_element);

}

template <class T>

void queue<T>::pop() {

if (first == nullptr) {

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

}

first = std::move(first->next\_element);

}

template <class T>

void queue<T>::push(const T& value) {

forward\_iterator it(nullptr);

insert(it, value);

}

template <class T>

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

if (first == nullptr) {

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

}

return first->value;

}

template<class T>

typename queue<T>::forward\_iterator queue<T>::element::next() {

return forward\_iterator(this->next\_element.get());

}

template<class T>

queue<T>::forward\_iterator::forward\_iterator(containers::queue<T>::element \*ptr) {

ptr\_ = ptr;

}

template<class T>

T& queue<T>::forward\_iterator::operator\*() {

return this->ptr\_->value;

}

template<class T>

typename queue<T>::forward\_iterator& queue<T>::forward\_iterator::operator++() {

if (ptr\_ == nullptr) throw std::logic\_error ("out of queue borders");

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

return \*this;

}

template<class T>

typename queue<T>::forward\_iterator queue<T>::forward\_iterator::operator++(int) {

forward\_iterator old = \*this;

++\*this;

return old;

}

template<class T>

bool queue<T>::forward\_iterator::operator==(const forward\_iterator& other) const {

return ptr\_ == other.ptr\_;

}

template<class T>

bool queue<T>::forward\_iterator::operator!=(const forward\_iterator& other) const {

return ptr\_ != other.ptr\_;

}

}

#endif // D\_QUEUE\_HPP\_

main.cpp

#include <algorithm>

#include <iostream>

#include "queue.hpp"

#include "rhombus.hpp"

int main()

{

int posision;

containers::queue<rhombus<int>> q;

std::cout << "1 - push\n"

<< "2 - top\n"

<< "3 - pop\n"

<< "4 - erase\_to\_num\n"

<< "5 - insert\_to\_num\n"

<< "6 - for\_each\n"

<< "7 - count\_if\n"

<< "0 - exit\n";

for (;;) {

int command;

std::cin >> command;

if (command == 1) {

rhombus<int> rhomb(std::cin);

q.push(rhomb);

std::cout << std::endl;

} else if (command == 2) {

q.top().print();

} else if (command == 4) {

std::cin >> posision;

q.erase\_to\_num(posision);

} else if (command == 0) {

break;

} else if (command == 5) {

std::cin >> posision;

rhombus<int> f(std::cin);

q.insert\_to\_num(posision, f);

} else if (command == 3) {

q.pop();

} else if (command == 6) {

std::for\_each(q.begin(), q.end(), [] (rhombus<int> rhomb) {return rhomb.print();});

} else if (command == 7) {

int are;

std::cin >> are;

std::cout << std::count\_if(q.begin(), q.end(), [are](rhombus<int> r){return r.area() < are;}) << std::endl;

} else {

std::cout << "ERROR" << std::endl;

break;

}

}

return 0;

}

**2. Ссылка на репозиторий в GitHub:**

https://github.com/keoni02032/oop\_exercise\_05

**3. Набор testcases:**

test\_01.test

1

-1 0

0 1

1 0

0 -1

1

-2 0

0 2

2 0

0 -2

5

1

-3 0

0 3

3 0

0 -3

2

3

6

4

1

2

0

test\_02.test

5

1

-1 0

0 1

1 0

0 -1

1

-3 0

0 3

3 0

0 -3

5

2

-2 0

0 2

2 0

0 -2

6

7

10

7

20

1

-20 0

0 1

1 0

0 -1

**4.Результаты выполнения программы:**

**test\_01.result**

**1 - push   
2 - top   
3 - pop   
4 - erase\_to\_num   
5 - insert\_to\_num   
6 - for\_each   
7 - count\_if   
0 - exit   
1   
-1 0   
0 1   
1 0   
0 -1   
  
1   
-2 0   
0 2   
2 0   
0 -2   
  
5   
1   
-3 0   
0 3   
3 0   
0 -3   
7   
10   
2   
2   
[ -3 0 ] [ 0 3 ] [ 3 0 ] [ 0 -3 ]   
3   
6   
[ -1 0 ] [ 0 1 ] [ 1 0 ] [ 0 -1 ]   
[ -2 0 ] [ 0 2 ] [ 2 0 ] [ 0 -2 ]   
4   
1   
2   
[ -2 0 ] [ 0 2 ] [ 2 0 ] [ 0 -2 ]   
0**

**test\_02.result**

**1 - push   
2 - top   
3 - pop   
4 - erase\_to\_num   
5 - insert\_to\_num   
6 - for\_each   
7 - count\_if   
0 - exit   
5   
1   
-1 0   
0 1   
1 0   
0 -1   
1   
-3 0   
0 3   
3 0   
0 -3   
  
5   
2   
-2 0   
0 2   
2 0   
0 -2   
6   
[ -1 0 ] [ 0 1 ] [ 1 0 ] [ 0 -1 ]   
[ -2 0 ] [ 0 2 ] [ 2 0 ] [ 0 -2 ]   
[ -3 0 ] [ 0 3 ] [ 3 0 ] [ 0 -3 ]   
7   
10   
2   
7   
20   
3   
1   
-20 0   
0 1   
1 0   
0 -1   
lab5: /home/sergey/labs/OOP/lab5/example/rhombus.hpp:30: rhombus<T>::rhombus(std::istream&) [with T = int; std::istream = std::basic\_istream<char>]: Assertion `correct()' failed.   
Аварийный останов (стек памяти сброшен на диск)**

**5. Объяснение результатов работы программы:**

При вводе команды «1» происходит вставка элемента в очередь, при вводе «2» выводится первый элемент из нашей очереди, «3» удаляется первый элемент очереди, «4» удаление элемента очереди по итератору, «5» удаление из очереди по номеру итератора, «6» выводятся все фигуры, «7» выводится количество фигур площадь которых меньше заданной.

**6. Вывод:**

В данной лабораторной работе я освоил основы работы с коллекциями и итераторами. Создал свой STL контейнер основанный на умных указателях.