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, №22 |
| Студент: | Рыженко Иван Александрович |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: |  |
| Дата: | 11.01.2020 |

Москва, 2020

1. **Задание**

Собрать шаблон динамической коллекции согласно варианту задания.

Вариант 22: Пятиугольник. Очередь.

1. **TestCases**

|  |
| --- |
| **Test 1.//**проверка добавления  1. Add figure in queue  2. Delete element of queue  3. Output element of queue  4. Output all elements of queue  5. Output the number of elements with area more then input  6. Add figure by index  1  Coordinates of 1 vertex:  Coordinate 'x': 21  Coordinate 'y': 12  Coordinates of 2 vertex:  Coordinate 'x': 124  Coordinate 'y': 12  Coordinates of 3 vertex:  Coordinate 'x': 214  Coordinate 'y': 12  Coordinates of 4 vertex:  Coordinate 'x': 214  Coordinate 'y': 12  Coordinates of 5 vertex:  Coordinate 'x': 241  Coordinate 'y': 12  (21 12),(124 12),(214 12),(214 12),(241 12)  1. Add figure in queue  2. Delete element of queue  3. Output element of queue  4. Output all elements of queue  5. Output the number of elements with area more then input  6. Add figure by index  6  Input the index  2  Input coordinates of the pentagon  Coordinates of 1 vertex:  Coordinate 'x': 21  Coordinate 'y': 12  Coordinates of 2 vertex:  Coordinate 'x':  12  Coordinate 'y': 12  Coordinates of 3 vertex:  Coordinate 'x': 21  Coordinate 'y': 214  Coordinates of 4 vertex:  Coordinate 'x': 12  Coordinate 'y': 12  Coordinates of 5 vertex:  Coordinate 'x': 214  Coordinate 'y': 12  1. Add figure in queue  2. Delete element of queue  3. Output element of queue  4. Output all elements of queue  5. Output the number of elements with area more then input  6. Add figure by index  4  Element of queue:  (21 12),(124 12),(214 12),(214 12),(241 12)  Element of queue:  (21 12),(12 12),(21 214),(12 12),(214 12)  1. Add figure in queue  2. Delete element of queue  3. Output element of queue  4. Output all elements of queue  5. Output the number of elements with area more then input  6. Add figure by index  1  Coordinates of 1 vertex:  Coordinate 'x': 123  Coordinate 'y': 12  Coordinates of 2 vertex:  Coordinate 'x': 12  Coordinate 'y': 23  Coordinates of 3 vertex:  Coordinate 'x': 12  Coordinate 'y': 12  Coordinates of 4 vertex:  Coordinate 'x': 12  Coordinate 'y': 123  Coordinates of 5 vertex:  Coordinate 'x': 123  Coordinate 'y': 12  (123 12),(12 23),(12 12),(12 123),(123 12) |
| **Test 2.//**проверка удаления  1. Add figure in queue  2. Delete element of queue  3. Output element of queue  4. Output all elements of queue  5. Output the number of elements with area more then input  6. Add figure by index  2  Empty collection.  1. Add figure in queue  2. Delete element of queue  3. Output element of queue  4. Output all elements of queue  5. Output the number of elements with area more then input  6. Add figure by index  1  Coordinates of 1 vertex:  Coordinate 'x': 12  Coordinate 'y': 12  Coordinates of 2 vertex:  Coordinate 'x': 12  Coordinate 'y': 24  Coordinates of 3 vertex:  Coordinate 'x': 12  Coordinate 'y': 12  Coordinates of 4 vertex:  Coordinate 'x': 412  Coordinate 'y': 124  Coordinates of 5 vertex:  Coordinate 'x': 12  Coordinate 'y': 12  (12 12),(12 24),(12 12),(412 124),(12 12)  1. Add figure in queue  2. Delete element of queue  3. Output element of queue  4. Output all elements of queue  5. Output the number of elements with area more then input  6. Add figure by index  2  1. Delete the top element of queue  2. Delete figure by index  1  1. Add figure in queue  2. Delete element of queue  3. Output element of queue  4. Output all elements of queue  5. Output the number of elements with area more then input  6. Add figure by index  4  Empty collection. |
| **Test 3.//**работа с пустой коллекцией  1. Add figure in queue  2. Delete element of queue  3. Output element of queue  4. Output all elements of queue  5. Output the number of elements with area more then input  6. Add figure by index  2  Empty collection.  1. Add figure in queue  2. Delete element of queue  3. Output element of queue  4. Output all elements of queue  5. Output the number of elements with area more then input  6. Add figure by index  3  Empty collection.  1. Add figure in queue  2. Delete element of queue  3. Output element of queue  4. Output all elements of queue  5. Output the number of elements with area more then input  6. Add figure by index  4  Empty collection.  1. Add figure in queue  2. Delete element of queue  3. Output element of queue  4. Output all elements of queue  5. Output the number of elements with area more then input  6. Add figure by index  5  Input the area:  12  0 |

1. **Адрес репозитория на GitHub**

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

1. **Код программы на С++**

Vertex.h

#pragma once

#include<iostream>

#include<type\_traits>

#include<cmath>

//????? "???????"

template<class T>

struct **Vertex** {

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

vertex coordinates;

Vertex<T>& operator=(Vertex<T> A);

**Vertex**() = default;//??????????? ?? ?????????

};

template<class T>

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

std::cout << "Coordinate 'x': ";

is >> p.coordinates.first;

std::cout << "Coordinate 'y': ";

is >> p.coordinates.second;

return is;

}

template<class T>

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

os << '(' << p.coordinates.first << ' ' << p.coordinates.second << ')';

return os;

}

template<class T>

Vertex<T> operator+(const Vertex<T> A, const Vertex<T> B) {

Vertex<T> res;

res.coordinates.first = A.coordinates.first + B.coordinates.first;

res.coordinates.second = A.coordinates.second + B.coordinates.second;

return res;

}

template<class T>

Vertex<T>& Vertex<T>::operator=(const Vertex<T> A) {

this->x = A.coordinates.first;

this->y = A.coordinates.second;

return (\*this);

}

template<class T>

Vertex<T> operator+=(Vertex<T> A, const Vertex<T> B) {

A.coordinates.first += B.coordinates.first;

A.coordinates.second += B.coordinates.second;

return A;

}

template<class T>

Vertex<T> operator/=(Vertex<T> A, const double B) {

A.coordinates.first /= B;

A.coordinates.second /= B;

}

template<class T>

struct **is\_Vertex** : std::false\_type {};

template<class T>

struct **is\_Vertex**<Vertex<T>> : std::true\_type {};

Pentagon.h

#pragma once

#include"vertex.h"

//???????????? ???? ? ???????? ?????

template<class T>

class **Pentagon** {

public:

Vertex<T> vertices[5];

**Pentagon**() = default;

**Pentagon**(std::istream& in);

void **Read**(std::istream& in);

double **Area**() const;

void **Print**(std::ostream& os) const;

friend std::ostream& operator<< (std::ostream& out, const Pentagon<T>& point);

};

template<class T>

Pentagon<T>::**Pentagon**(std::istream& is)//??????????? ?????? "Pentagon"

{

for (int i = 0; i < 5; i++)

{

std::cout << "Coordinates of " << i+1 << " vertex: \n";

is >> this->vertices[i];

}

}

template<class T>

double Pentagon<T>::**Area**() const {

double Area = 0;

for (int i = 0; i < 5; i++) {

Area += (vertices[i].coordinates.first) \* (vertices[(i + 1) % 5].coordinates.second) - (vertices[(i + 1) % 5].coordinates.first) \* (vertices[i].coordinates.second);

}

Area \*= 0.5;

return abs(Area);

}

template<class T>

void Pentagon<T>::**Print**(std::ostream& os) const {

for (int i = 0; i < 5; i++) {

os << this->vertices[i];

if (i != 4) {

os << ',';

}

}

os << std::endl;

}

template<class T>

void Pentagon<T>::**Read**(std::istream& in) {

for (int i = 0; i < 5; i++)

{

std::cout << "Coordinates of " << i+1 << " vertex: \n";

in >> this->vertices[i];

}

}

template<class T>

std::ostream& operator<<(std::ostream& os, const Pentagon<T>& point) {

for (int i = 0; i < 5; i++) {

os << point.vertices[i];

if (i != 5) {

os << ',';

}

}

}

Queue.h

#include <iterator>

#include <memory>

namespace **Containers** {

//????? ????????? "???????"

template<class T>

class **Queue** {

private:

struct **element**;

size\_t size = 0;//???? ??????, ?????????? ????? ????????? ? ?????????

public:

**Queue**() = default;//??????????? ?? ?????????

class **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;

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 class **Queue**;

};

forward\_iterator **begin**();

forward\_iterator **end**();

void **push**(const T& value);

T& **top**();

T& **bottom**();

void **pop**();

size\_t **length**();

void **delete\_by\_it**(forward\_iterator d\_it);

void **delete\_by\_index**(size\_t N);//???????? ?? ???????

void **insert\_by\_it**(forward\_iterator ins\_it, T& value);

void **insert\_by\_index**(size\_t N, T& value);//?????????? ?? ???????

Queue& operator=(Queue& other);

private:

struct **element**

{

T value;

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

forward\_iterator **next**();

};

void **push\_impl**(std::unique\_ptr<element>& cur, const T& value);

std::unique\_ptr<element> **pop\_impl**(std::unique\_ptr<element> cur);

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

};

template<class T>

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

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

}

template<class T>

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

return forward\_iterator(nullptr);

}

template<class T>

size\_t Queue<T>::**length**() {//?????????? ????? ?????????, ?????????? ? ?????????

return this->size;

}

template<class T>

void Queue<T>::**push**(const T& value)//?????????? ???????? ? ?????????

{

push\_impl(this*->*first, value);

size++;

}

template<class T>

void Queue<T>::**push\_impl**(std::unique\_ptr<element>& cur, const T& value)

{

if (cur == nullptr)

{

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

return;

}

else

push\_impl(*cur->*next\_element, value);

}

template<class T>

void Queue<T>::**pop**()

{

if (size == 0)

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

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

size--;

}

template<class T>

std::unique\_ptr<typename Queue<T>::element> Queue<T>::**pop\_impl**(std::unique\_ptr<element> cur)

{

if (cur->next\_element != nullptr)

{

cur->next\_element = pop\_impl(std::move(cur->next\_element));

return cur;

}

else

return nullptr;

}

template<class T>

T& Queue<T>::**bottom**()

{

if (size == 0)

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

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

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

i++;

}

return \*i;

}

template<class T>

T& Queue<T>::**top**()

{

return first->value;

}

template<class T>

Queue<T>& Queue<T>::operator=(Queue<T>& other)//???????? ???????????? ??? ?????????

{

size = other.size;

first = std::move(other.first);

}

template<class T>

void Queue<T>::**delete\_by\_it**(Containers::Queue<T>::forward\_iterator d\_it)

{

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

if (d\_it == end)

throw std::logic\_error("Out of borders");

if (d\_it == this->begin())

{

std::unique\_ptr<element> tmp;

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

first = std::move(tmp);

return;

}

while ((i.it\_ptr != nullptr) && (i.it\_ptr->next() != d\_it)) {

++i;

}

if (i.it\_ptr == nullptr) throw std::logic\_error("Out of borders");

i.it\_ptr->next\_element = std::move(d\_it.it\_ptr->next\_element);

size--;

}

template<class T>

void Queue<T>::**delete\_by\_index**(size\_t N)

{

if (N >= this->size)

return;

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

for (size\_t i = 0; i < N; ++i)

++it;

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

}

template<class T>

void Queue<T>::**insert\_by\_it**(Containers::Queue<T>::forward\_iterator ins\_it, T& value)

{

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

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

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

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

first = std::move(tmp);

size++;

return;

}

while ((i.it\_ptr != nullptr) && (i.it\_ptr->next() != ins\_it)) {

++i;

}

if (i.it\_ptr == nullptr) throw std::logic\_error("Out of borders");

tmp->next\_element = std::move(i.it\_ptr->next\_element);

i.it\_ptr->next\_element = std::move(tmp);

size++;

}

template<class T>

void Queue<T>::**insert\_by\_index**(size\_t N, T& value) {

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

if (N >= this->length())

it = this->end();

else

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

++it;

}

this->insert\_by\_it(it, 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) {

it\_ptr = ptr;

}

template<class T>

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

return this->it\_ptr->value;

}

template<class T>

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

if (it\_ptr == nullptr) throw std::logic\_error("Out of queue");

\*this = it\_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 it\_ptr == other.it\_ptr;

}

template<class T>

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

return it\_ptr != other.it\_ptr;

}

}

Main.cpp

#include <iostream>

#include <algorithm>

#include <locale.h>

#include "Pentagon.h"

#include "queue.h"

void **Menu1**()

{

std::cout << "1. Add figure in queue\n";

std::cout << "2. Delete element of queue\n";

std::cout << "3. Output element of queue\n";

std::cout << "4. Output all elements of queue\n";

std::cout << "5. Output the number of elements with area more then input\n";

std::cout << "6. Add figure by index\n";

}

void **DeleteMenu**()

{

std::cout << "1. Delete the top element of queue\n";

std::cout << "2. Delete figure by index\n";

}

void **PrintMenu**()

{

std::cout << "1. Output the top element\n";

std::cout << "2. Output the last element\n";

}

int **main**()

{

Containers::Queue<Pentagon<int>> Myqueue;

Pentagon<int> TempPentagon;

while (true) {

Menu1();

int n, m;

size\_t ind;

double s;

std::cin >> n;

switch (n)

{

case 1:

TempPentagon.Read(std*::cin*);

TempPentagon.Print(std*::cout*);

Myqueue.push(TempPentagon);

break;

case 2:

if (Myqueue.length() == 0)

{

std::cout << "Empty collection.\n";

break;

}

DeleteMenu();

std::cin >> m;

switch (m) {

case 1:

Myqueue.pop();

break;

case 2:

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

std::cin >> ind;

if (ind > Myqueue.length())

{

std::cout << "Index is out of bourders.\n";

break;

}

Myqueue.delete\_by\_index(ind);

break;

default:

break;

}

break;

case 3:

if (Myqueue.length() == 0)

{

std::cout << "Empty collection.\n";

break;

}

PrintMenu();

std::cin >> m;

switch (m) {

case 1:

Myqueue.top().Print(std*::cout*);

std::cout << std::endl;

break;

case 2:

Myqueue.bottom().Print(std*::cout*);

std::cout << std::endl;

break;

default:

std::cout << "Incorrect input\n";

break;

}

break;

case 4:

if (Myqueue.length() == 0)

{

std::cout << "Empty collection.\n";

break;

}

std::for\_each(Myqueue.begin(), Myqueue.end(), [](Pentagon<int>& X) { std::cout << "Element of queue:\n"; X.Print(std*::cout*); std::cout << std::endl; });

break;

case 5:

std::cout << "Input the area:\n";

std::cin >> s;

std::cout << std::count\_if(Myqueue.begin(), Myqueue.end(), [=](Pentagon<int>& X) {return X.Area() > s; }) << std::endl;

break;

case 6:

std::cout << "Input the index\n";

std::cin >> ind;

std::cout << "Input coordinates of the pentagon\n";

TempPentagon.Read(std*::cin*);

Myqueue.insert\_by\_index(ind, *TempPentagon*);

break;

default:

return 0;

}

}

return 0;

}

1. **Объяснение результатов работы программы**

При запуске программы в консоль выводится меню:

“Add figure in queue” – функция добавления элемента в очередь.

“Delete element of queue” – удаление верхнего элемента очереди (аналог pop) или удаление элемента по индексу

“Output element of queue” – вывод в консоль первого или последнего элемента коллекции.

“Output all elements of queue” – вывод всех элементов коллекции в консоль. (Если коллекция пуста – выводится “Empty queue”)

“Output the number of elements with area more than input” – вывод элементов коллекции с площадью больше данной.

“Add figure by index” – добавление фигуры по заданному индексу. (Если индекс превосходит размер коллекции – элемент добавляется в конец)

1. **Вывод**

В данной работе была реализована коллекция “Очередь”, а также итератор к ней, позволяющий передвигаться по её элементам.