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 6: «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: Фигура: Пятиугольник. Коллекция: Очередь. Аллокатор: Список.

1. **TextCases**

|  |
| --- |
| Test 1: Проверка функции добавления и вывода элементов  1. Add figure in queue  2. Delete figure from queue  3. Output figure  4. Output all figures  5. Add figure by index  Input the number of function: 1  Coordinates of 1 vertex:  Coordinate 'x': 124  Coordinate 'y': 12  Coordinates of 2 vertex:  Coordinate 'x': 124  Coordinate 'y': 12  Coordinates of 3 vertex:  Coordinate 'x': 12  Coordinate 'y': 24  Coordinates of 4 vertex:  Coordinate 'x': 12  Coordinate 'y': 21  Coordinates of 5 vertex:  Coordinate 'x': 12  Coordinate 'y': 21  Coordinates of pentagon's vertexes: (124 12),(124 12),(12 24),(12 21),(12 21)  1. Add figure in queue  2. Delete figure from queue  3. Output figure  4. Output all figures  5. Add figure by index  Input the number of function: 4  Coordinates of pentagon's vertexes: (124 12),(124 12),(12 24),(12 21),(12 21)  1. Add figure in queue  2. Delete figure from queue  3. Output figure  4. Output all figures  5. Add figure by index  Input the number of function: 1  Coordinates of 1 vertex:  Coordinate 'x': 12  Coordinate 'y': 214  Coordinates of 2 vertex:  Coordinate 'x': 1  Coordinate 'y': 12  Coordinates of 3 vertex:  Coordinate 'x': 125  Coordinate 'y': 12  Coordinates of 4 vertex:  Coordinate 'x': 125  Coordinate 'y': 12  Coordinates of 5 vertex:  Coordinate 'x': 215  Coordinate 'y': 12  Coordinates of pentagon's vertexes: (12 214),(1 12),(125 12),(125 12),(215 12)  1. Add figure in queue  2. Delete figure from queue  3. Output figure  4. Output all figures  5. Add figure by index  Input the number of function: 4  Coordinates of pentagon's vertexes: (124 12),(124 12),(12 24),(12 21),(12 21)  Coordinates of pentagon's vertexes: (12 214),(1 12),(125 12),(125 12),(215 12) |
| Test 2. Проверка методов удаления из коллекции и вывода первого и последнего элемента  1. Add figure in queue  2. Delete figure from queue  3. Output figure  4. Output all figures  5. Add figure by index  Input the number of function: 1  Coordinates of 1 vertex:  Coordinate 'x': 123  Coordinate 'y': 12  Coordinates of 2 vertex:  Coordinate 'x': 214  Coordinate 'y': 21  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  Coordinates of pentagon's vertexes: (123 12),(214 21),(21 214),(12 12),(214 12)  1. Add figure in queue  2. Delete figure from queue  3. Output figure  4. Output all figures  5. Add figure by index  Input the number of function: 1  Coordinates of 1 vertex:  Coordinate 'x': 214  Coordinate 'y': 12  Coordinates of 2 vertex:  Coordinate 'x': 12  Coordinate 'y': 12  Coordinates of 3 vertex:  Coordinate 'x': 214  Coordinate 'y': 12  Coordinates of 4 vertex:  Coordinate 'x': 21  Coordinate 'y': 241  Coordinates of 5 vertex:  Coordinate 'x': 12  Coordinate 'y': 21  Coordinates of pentagon's vertexes: (214 12),(12 12),(214 12),(21 241),(12 21)  1. Add figure in queue  2. Delete figure from queue  3. Output figure  4. Output all figures  5. Add figure by index  Input the number of function: 3  1. Output the top element  2. Output the last element  Input the number of function: 1  Coordinates of pentagon's vertexes: (123 12),(214 21),(21 214),(12 12),(214 12)  1. Add figure in queue  2. Delete figure from queue  3. Output figure  4. Output all figures  5. Add figure by index  Input the number of function: 3  1. Output the top element  2. Output the last element  Input the number of function: 2  Coordinates of pentagon's vertexes: (214 12),(12 12),(214 12),(21 241),(12 21)  1. Add figure in queue  2. Delete figure from queue  3. Output figure  4. Output all figures  5. Add figure by index  Input the number of function: 2  1. Delete the top element  2. Delete figure by index  Input the number of function: 1  1. Add figure in queue  2. Delete figure from queue  3. Output figure  4. Output all figures  5. Add figure by index  Input the number of function: 4  Coordinates of pentagon's vertexes: (214 12),(12 12),(214 12),(21 241),(12 21)  1. Add figure in queue  2. Delete figure from queue  3. Output figure  4. Output all figures  5. Add figure by index  Input the number of function: 2  1. Delete the top element  2. Delete figure by index  Input the number of function: 2  Input the index for deleating: 0  1. Add figure in queue  2. Delete figure from queue  3. Output figure  4. Output all figures  5. Add figure by index  Input the number of function: 4  Empty queue |
| Test 3. Работа с пустой коллекцией  1. Add figure in queue  2. Delete figure from queue  3. Output figure  4. Output all figures  5. Add figure by index  Input the number of function: 3  1. Output the top element  2. Output the last element  Input the number of function: 1  Empty queue.  1. Add figure in queue  2. Delete figure from queue  3. Output figure  4. Output all figures  5. Add figure by index  Input the number of function: 3  1. Output the top element  2. Output the last element  Input the number of function: 2  Empty queue.  1. Add figure in queue  2. Delete figure from queue  3. Output figure  4. Output all figures  5. Add figure by index  Input the number of function: 2  1. Delete the top element  2. Delete figure by index  Input the number of function: 1  Empty queue.  1. Add figure in queue  2. Delete figure from queue  3. Output figure  4. Output all figures  5. Add figure by index  Input the number of function: 2  1. Delete the top element  2. Delete figure by index  Input the number of function: 2  Empty queue. |

1. **Адрес репозитория на GitHub**

https://github.com/THEproVANO/oop\_exercise\_06

1. **Код программы на С++**

Vertex.h

#pragma once

#include <iostream>

#include <type\_traits>

#include <cmath>

template<class T>

struct **vertex** {

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

coordinates coord;

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

};

template<class T>

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

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

is >> p.coord.first;

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

is >> p.coord.second;

return is;

}

template<class T>

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

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

return os;

}

template<class T>

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

vertex<T> res;

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

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

return res;

}

template<class T>

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

this->x = A.coord.first;

this->y = A.coord.second;

return \*this;

}

template<class T>

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

A.coord.first += B.coord.first;

A.coord.second += B.coord.second;

return A;

}

template<class T>

double **vector** (vertex<T>& A, vertex<T>& B) {

double res = sqrt(pow(B.coord.first - A.coord.first, 2) + pow(B.coord.second - A.coord.second, 2));

return res;

}

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<math.h>

#include<stdio.h>

#include<iostream>

#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)

{

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

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

Area \*= 0.5;

return abs(Area);

}

template<class T>

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

{

std::cout << "Coordinates of pentagon's vertexes: ";

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

#pragma once

#include <iterator>

#include <memory>

namespace **containers** {

template<class T, class Allocator = std::allocator<T>>

class **Queue**

{

private:

struct **element**;

size\_t size = 0;

public:

**Queue**() = default;//Конструктор по умолчанию

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 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 deleted\_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:

using allocator\_type = typename Allocator::template rebind<element>::other;

struct **deleter**

{

**deleter**(allocator\_type\* allocator) : allocator\_(allocator) {}

void operator() (element\* ptr) {

if (ptr != nullptr) {

std::allocator\_traits<allocator\_type>::destroy(\*allocator\_, ptr);

allocator\_->deallocate(ptr, 1);

}

}

private:

allocator\_type\* allocator\_;

};

using unique\_ptr = std::unique\_ptr<element, deleter>;

void **push\_impl**(unique\_ptr &cur, const T& value);

struct **element**

{

T value;

unique\_ptr next\_element{ nullptr, deleter{nullptr} };

**element**(const T& value\_) : value(value\_) {}

forward\_iterator **next**();

};

allocator\_type allocator\_{};

unique\_ptr first{ nullptr, deleter{nullptr} };

element\* tail = nullptr;

};

/\*Методы класса\*/

template<class T, class Allocator>

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

{

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

}

template<class T, class Allocator>

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

{

return forward\_iterator(nullptr);

}

template<class T, class Allocator>

size\_t Queue<T, Allocator>::**length**()

{

return size;

}

template<class T, class Allocator>

void Queue<T, Allocator>::**push**(const T& value)

{

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

size++;

}

template<class T, class Allocator>

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

{

if (cur == nullptr)

{

element\* result = this->allocator\_.allocate(1);//выделение памяти под элемент

std::allocator\_traits<allocator\_type>::construct(this->allocator\_, result, value);//вызов конструктора по адресу result

cur = unique\_ptr(result, deleter{&this->allocator\_});

return;

}

else

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

}

template<class T, class Allocator>

void Queue<T, Allocator>::**pop**()//метод удаления элемента из очереди

{

if (size == 0)

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

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

size--;

}

template<class T, class Allocator>

T& Queue<T, Allocator>::**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, class Allocator>

T& Queue<T, Allocator>::**top**() {

return first->value;

}

template<class T, class Allocator>

Queue<T, Allocator>& Queue<T, Allocator>::operator=(Queue<T, Allocator>& other) {

size = other.size;

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

}

template<class T, class Allocator>

void Queue<T, Allocator>::**delete\_by\_index**(size\_t N)//метод удаления по индексу

{

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

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

++it;

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

}

template<class T, class Allocator>

void Queue<T, Allocator>::**delete\_by\_it**(containers::Queue<T, Allocator>::forward\_iterator deleted\_it) {

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

if (deleted\_it == end)

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

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

{

size--;

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

first = std::move(tmp);

return;

}

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

++i;

if (i.it\_ptr == nullptr)

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

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

size--;

}

template<class T, class Allocator>

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

{

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

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

{

this->push(value);

return;

}

element\* tmp = this->allocator\_.allocate(1);//освобождение памяти под элемент

std::allocator\_traits<allocator\_type>::construct(this->allocator\_, tmp, value);//вызов конструктора элемнта по данному адресу

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

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

first = unique\_ptr(tmp, deleter{ &this->allocator\_ });

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 = unique\_ptr(tmp, deleter{ &this->allocator\_ });

size++;

}

template<class T, class Allocator>

void Queue<T, Allocator>::**insert\_by\_index**(size\_t N, T& value)//метод вставки по индексу

{

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, class Allocator>

typename Queue<T, Allocator>::forward\_iterator Queue<T, Allocator>::element::**next**() {

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

}

template<class T, class Allocator>

Queue<T, Allocator>::forward\_iterator::**forward\_iterator**(containers::Queue<T, Allocator>::element\* ptr) {

it\_ptr = ptr;

}

template<class T, class Allocator>

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

return this->it\_ptr->value;

}

template<class T, class Allocator>

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

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

\*this = it\_ptr->next();

return \*this;

}

template<class T, class Allocator>

typename Queue<T, Allocator>::forward\_iterator Queue<T, Allocator>::forward\_iterator::operator++(int) {

forward\_iterator old = \*this;

++\* this;

return old;

}

template<class T, class Allocator>

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

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

}

template<class T, class Allocator>

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

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

}

}

Allocator.h

#include <cstdlib>

#include <iostream>

#include <type\_traits>

#include <list>

#include "Queue.h"

//Класс аллокатора//

namespace **allocators**

{

////Реализованный класс аллокатора

template<class T, size\_t ALLOC\_SIZE>

struct **my\_allocator** {

using value\_type = T;

using size\_type = std::size\_t;

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

using is\_always\_equal = std::false\_type;

template<class U>

struct **rebind** {

using other = my\_allocator<U, ALLOC\_SIZE>;

};

**my\_allocator**() :

pool\_begin(new char[ALLOC\_SIZE]),

pool\_end(pool\_begin + ALLOC\_SIZE),

pool\_tail(pool\_begin)

{}

**my\_allocator**(const my\_allocator&) = delete;

**my\_allocator**(my\_allocator&&) = delete;

~**my\_allocator**()

{

delete[] pool\_begin;

}

T\* **allocate**(std::size\_t n);

void **deallocate**(T\* ptr, std::size\_t n);

private:

char\* pool\_begin;

char\* pool\_end;

char\* pool\_tail;

std::list<char\*> free\_blocks;

};

template<class T, size\_t ALLOC\_SIZE>

T\* my\_allocator<T, ALLOC\_SIZE>::**allocate**(std::size\_t n) {

if (n != 1) {

throw std::logic\_error("Error: Сan`t allocate arrays");

}

if (std::size\_t(pool\_end - pool\_tail) < sizeof(T)) {

if (free\_blocks.size())

{

auto it = free\_blocks.begin();

char\* ptr = \*it;

free\_blocks.pop\_front();

return reinterpret\_cast<T\*>(ptr);

}

throw std::bad\_alloc();

}

T\* result = reinterpret\_cast<T\*>(pool\_tail);

pool\_tail += sizeof(T);

return result;

}

template<class T, size\_t ALLOC\_SIZE>

void my\_allocator<T, ALLOC\_SIZE>::**deallocate**(T\* ptr, std::size\_t n) {

if (n != 1) {

throw std::logic\_error("Error: Can`t allocate arrays, consequently can`t deallocate them too");

}

if (ptr == nullptr)

return;

free\_blocks.push\_back(reinterpret\_cast<char\*>(ptr));

}

};

main.cpp

#include<iostream>

#include<algorithm>

#include<locale.h>

#include"Pentagon.h"

#include"Queue.h"

#include"Allocator.h"

void **Menu1**() {

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

std::cout << "2. Delete figure from queue\n";

std::cout << "3. Output figure\n";

std::cout << "4. Output all figures\n";

std::cout << "5. Add figure by index\n";

std::cout << "Input the number of function: ";

}

void **DeleteMenu**() {

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

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

std::cout << "Input the number of function: ";

}

void **PrintMenu**() {

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

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

std::cout << "Input the number of function: ";

}

int **main**() {

containers::Queue<Pentagon<int>, allocators::my\_allocator<Pentagon<int>, 1000>> MyQueue;

Pentagon<int> TempPentagon;

while (true) {

Menu1();

int n, m;

size\_t ind;

std::cin >> n;

switch (n)

{

case 1:

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

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

MyQueue.push(TempPentagon);

break;

case 2:

DeleteMenu();

std::cin >> m;

switch (m) {

case 1:

if (MyQueue.length() == 0)

{

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

break;

}

MyQueue.pop();

break;

case 2:

if (MyQueue.length() == 0)

{

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

break;

}

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

std::cin >> ind;

MyQueue.delete\_by\_index(ind);

break;

default:

break;

}

break;

case 3:

PrintMenu();

std::cin >> m;

switch (m) {

case 1:

if (MyQueue.length() == 0)

{

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

break;

}

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

std::cout << std::endl;

break;

case 2:

if (MyQueue.length() == 0)

{

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

break;

}

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

std::cout << std::endl;

break;

default:

break;

}

break;

case 4:

if (MyQueue.length() == 0)

{

std::cout << "Empty queue\n";

break;

}

std::for\_each(MyQueue.begin(), MyQueue.end(), [](Pentagon<int>& X) { X.Print(std*::cout*); std::cout << std::endl; });

break;

case 5:

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

std::cin >> ind;

if (ind >= MyQueue.length())

{

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

break;

}

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

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

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

break;

default:

return 0;

}

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

}

return 0;

}

1. **Объяснение результатов работы программы**

При запуске программы в консоль выводится меню, которое позволяет работать с созданной коллекцией. Функция “Add figure in queue” добавляет элемент в коллекцию, “Delete figure from queue” удаляет самый первый элемент коллекции (аналог метода pop) или удаляет элемент по индексу, заданному пользователем. “Output figure” – выводит первый элемент коллекции или элемент с заданным пользователем индексом. “Output all figures” – выводит все фигуры, хранящиеся в коллекции. “Add figure by index” – добавляет фигуру по заданному индексу.

1. **Вывод**

В данной работе был реализован класс аллокатора, позволяющий контролировать выделение памяти, изучен примерный алгоритм его работы.

Кроме этого были получены навыки работы с функцией стандартной библиотеки std::allocator\_traits, который позволяет вызывать конструктор и деструктор нужного типа на выделенной памяти.