**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 6**

Тема: Основы работы с коллекциями: итераторы

Студент: Савров Никита

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. Постановка задачи

Создать шаблон динамической коллекцию, согласно варианту задания:

1. Коллекция должна быть реализована с помощью умных указателей (std::shared\_ptr, std::weak\_ptr).

Опционально использование std::unique\_ptr;

2. В качестве параметра шаблона коллекция должна принимать тип данных;

3. Коллекция должна содержать метод доступа:

oСтек – pop, push, top;

oОчередь – pop, push, top;

oСписок, Динамический массив – доступ к элементу по оператору [];

4. Реализовать аллокатор, который выделяет фиксированный размер памяти (количество блоков памяти –

является параметром шаблона аллокатора). Внутри аллокатор должен хранить указатель на

используемый блок памяти и динамическую коллекцию указателей на свободные блоки. Динамическая

коллекция должна соответствовать варианту задания (Динамический массив, Список, Стек, Очередь);

5. Коллекция должна использовать аллокатор для выделения и освобождения памяти для своих

элементов.

6. Аллокатор должен быть совместим с контейнерами std::map и std::list (опционально – vector).

7. Реализовать программу, которая:

\*Позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и

добавлять в коллекцию использующую аллокатор;

\*Позволяет удалять элемент из коллекции по номеру элемента;

\*Выводит на экран введенные фигуры c помощью std::for\_each;

1. Описание программы

Для реализации задачи реализуем шаблонные классы Rectangle, Queue и Allocator. В классе очереди реализуем вспомогательные классы: Item и Iterator. Первый будет являться узлом двусвязного списка, второй собственно является реализаций итератора. Для работы итераторов перегрузим такие операторы как \*, ->, ++, --, !=, ==. Для работы очереди реализуем основные методы, такие как Insert и Erase. Класс аллокатор будет принимать тип и количество объектов для которых нужно выделить память.

1. Набор testcases

test\_01

1

0 0

3 4

1

1 1

5 5

1

2 2

12 12

3

4

90

2

3

4

90

test\_02

1

0 3

8 10

1

1 2

13 13

1

0 0

20 20

1

2 4

21 23

3

2

3

4

50

4

300

test\_03

1

0 0

2 2

1

0 0

5 5

1

0 0

10 10

1

0 0

15 15

1

0 0

20 20

1

0 0

30 30

3

2

2

3

4

200

4

700

1. Результаты выполнения тестов.

test\_01

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

1

0 0

3 4

1

1 1

5 5

1

2 2

12 12

3

4

90

2

3

4

90

enter 2 coordinats for 2 point

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

enter 2 coordinats for 2 point

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

enter 2 coordinats for 2 point

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

0 0

0 4

3 4

3 0

1 1

1 5

5 5

5 1

2 2

2 12

12 12

12 2

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

enter test square

2 figures with square less 90

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

First element is delete

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

1 1

1 5

5 5

5 1

2 2

2 12

12 12

12 2

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

enter test square

1 figures with square less 90

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

q

test\_02

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

1

0 3

8 10

1

1 2

13 13

1

0 0

20 20

1

2 4

21 23

3

2

3

4

50

4

300

enter 2 coordinats for 2 point

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

enter 2 coordinats for 2 point

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

enter 2 coordinats for 2 point

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

enter 2 coordinats for 2 point

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

0 3

0 10

8 10

8 3

1 2

1 13

13 13

13 2

0 0

0 20

20 20

20 0

2 4

2 23

21 23

21 4

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

First element is delete

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

1 2

1 13

13 13

13 2

0 0

0 20

20 20

20 0

2 4

2 23

21 23

21 4

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

enter test square

0 figures with square less 50

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

enter test square

1 figures with square less 300

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

q

test\_03.txt

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

1

0 0

2 2

1

0 0

5 5

1

0 0

10 10

1

0 0

15 15

1

0 0

20 20

1

0 0

30 30

3

2

2

3

4

200

4

700

enter 2 coordinats for 2 point

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

enter 2 coordinats for 2 point

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

enter 2 coordinats for 2 point

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

enter 2 coordinats for 2 point

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

enter 2 coordinats for 2 point

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

enter 2 coordinats for 2 point

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

0 0

0 2

2 2

2 0

0 0

0 5

5 5

5 0

0 0

0 10

10 10

10 0

0 0

0 15

15 15

15 0

0 0

0 20

20 20

20 0

0 0

0 30

30 30

30 0

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

First element is delete

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

First element is delete

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

0 0

0 10

10 10

10 0

0 0

0 15

15 15

15 0

0 0

0 20

20 20

20 0

0 0

0 30

30 30

30 0

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

enter test square

1 figures with square less 200

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

enter test square

3 figures with square less 700

1 - Push to queue

2 - Pop to queue

3 - Print queue

4 - Count square lower test

q - exit

q

5.Листинг программы

queue.h

// Реализации класса очередь для ЛР №6.

// Савров Никита. Группа М80-207б-18

#ifndef QUEUE\_H

#define QUEUE\_H

#include <iostream>

#include <memory>

#include <cstdint>

#include "allocator.h"

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

class Queue {

private:

struct Item; //struct for node

class QueueIterator; //class iterator

std::shared\_ptr<Item> F\_elem;

int size;

TAllocator<T,100> all;

public:

using value\_type = T;

Queue() : size(0){ //basic constructor

F\_elem = std::make\_shared<Item>();

}

Queue(const Queue& other){ //copy constructor

size = other.Size();

F\_elem = std::make\_shared<Item>();

std::shared\_ptr<Item> tnew = F\_elem;

std::shared\_ptr<Item> told = other.F\_elem;

for (int i = 0; i < size; i++, tnew = tnew->next, tnew = told->next){

tnew->next = std::make\_shared<Item>(told->next->value);

tnew->next->prev = tnew;

}

if (size >= 2){

tnew->next = F\_elem;

F\_elem->prev = tnew;

}

}

~Queue (){

F\_elem = nullptr;

}

int Size(){

return size;

}

bool Empty(){

return size == 0;

}

void Push(T val){

InsertP(F\_elem, val);

}

void Pop(){

if (this->Empty()){

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

return;

}

Remove(F\_elem->next);

}

T Top(){

if (this->Empty()){

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

}

return F\_elem->prev->value;

}

QueueIterator Begin() { //function for return first iterator

if (this->Empty()){

return QueueIterator(F\_elem, this);

}

return QueueIterator(F\_elem->next, this);

}

QueueIterator End() { //function for return last iterator

return QueueIterator (F\_elem, this);

}

QueueIterator Insert(QueueIterator pos, T val){

InsertP(pos, this);

return QueueIterator (pos->prev, this);

}

QueueIterator Erase(QueueIterator pos){

QueueIterator res = pos;

++res;

Remove(pos);

return res;

}

void Print(){

if (size == 0)

return;

std::shared\_ptr<Item> tmp = F\_elem->next;

for(int i = 0; i < size; i++, tmp = tmp->next){

std::cout << tmp->value;

}

}

private:

struct Item{

T value;

std::shared\_ptr<Item> next;

std::shared\_ptr<Item> prev;

Item () : next(nullptr), prev(nullptr){

}

Item(T val) : value(val), next(nullptr), prev(nullptr){

}

};

class QueueIterator {

private:

std::shared\_ptr<Item> item;

Queue const \*Ths;

friend class Queue;

public:

using difference\_type = uint32\_t;

using value\_type = Queue::value\_type;

using reference = Queue::value\_type&;

using pointer = Queue::value\_type\*;

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

QueueIterator(std::shared\_ptr<Item> node, Queue const \*ths) : item(node), Ths(ths) {

}; //basic constructor

QueueIterator(const QueueIterator& other){ //copy constructor

item = other.item;

Ths = other.Ths;

}

reference operator\*(){

return item->value;

}

pointer operator->(){

return &item->value;

}

QueueIterator& operator++(){

if (Ths->size == 0){

return \*this;

}

if (Ths->size == 1 && item == Ths->F\_elem->next){

item = Ths->F\_elem;

return \*this;

}

item = item->next;

return \*this;

}

const QueueIterator operator++(int){

if (Ths->Size() == 0){

return \*this;

}

if (Ths->Size() == 1 && item == Ths->F\_elem->next){

item = Ths->F\_elem;

return \*this;

}

std::shared\_ptr<Item> res = item;

item = item->next;

return QueueIterator(res);

}

QueueIterator& operator-- (){

if (Ths->Size() == 0)

return \*this;

item = item->prev;

return \*this;

}

const QueueIterator operator-- (int){

if (Ths->Size() == 0){

return \*this;

}

std::shared\_ptr<Item> res = item;

item = item->prev;

return QueueIterator(res);

}

bool operator != (const QueueIterator& other) const noexcept{

return item != other.item;

}

bool operator== (const QueueIterator& other) const noexcept{

return item == other.item;

}

};

void EmptyInsert (T val){

F\_elem->next = std::make\_shared<Item>(val);

F\_elem->next->prev = F\_elem;

F\_elem->prev = F\_elem->next;

}

void InsertP(std::shared\_ptr<Item> node, T val){ //function for insert element on iterator

if (size == 0){

EmptyInsert(val);

size++;

return;

}

std::shared\_ptr<Item> newel = std::make\_shared<Item>(val);

newel->next = node;

newel->prev = node->prev;

node->prev->next = newel;

node->prev = newel;

size++;

}

void Remove (std::shared\_ptr<Item> node){ //function for erase element on iterator

if (size == 1){

F\_elem->next = nullptr;

} else {

if (size == 2){

if (node->next == F\_elem){

F\_elem->next->next = nullptr;

F\_elem->prev = F\_elem->next;

} else {

node->next->prev = F\_elem;

F\_elem->next = node->next;

}

size--;

return;

}

node->next->prev = node->prev;

node->prev->next = node->next;

}

size--;

}

};

#endif

rectangle.h

// Реализации класса прямоугольник для ЛР №6.

// Савров Никита. Группа М80-207б-18

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <algorithm>

#include <utility>

#include <iostream>

#include <vector>

#include <cmath>

template<class T>

class Rectangle{

public:

std::pair <T, T> a,b,c,d;

using type = T;

int number\_vertex = 4;

int s;

Rectangle(){

a.first = 0;

a.second = 0;

b.first = 0;

b.second = 0;

c.first = 0;

c.second = 0;

d.first = 0;

d.second = 0;

}

void Set (std::vector <std::pair<T,T>> cord){

a.first = cord[0].first;

a.second = cord[0].second;

b.first = cord[1].first;

b.second = cord[1].second;

c.first = cord[2].first;

c.second = cord[2].second;

d.first = cord[3].first;

d.second = cord[3].second;

int a = abs(cord[1].second - cord[0].second);

int b = abs(cord[3].first - cord[0].first);

s = a\*b;

}

int Square(){

return s;

}

friend std::istream& operator>>(std::istream& in, Rectangle<T> &obj){

std::vector <std::pair <T,T>> v(4);

in >> v[0].first >> v[0].second >> v[2].first >> v[2].second;

v[1].first = v[0].first;

v[1].second = v[2].second;

v[3].first = v[2].first;

v[3].second = v[0].second;

obj.Set(v);

return in;

}

friend std::ostream& operator<<(std::ostream& out, Rectangle<T> &obj){

out << obj.a.first << " " << obj.a.second << '\n';

out << obj.b.first << " " << obj.b.second << '\n';

out << obj.c.first << " " << obj.c.second << '\n';

out << obj.d.first << " " << obj.d.second << '\n';

return out;

}

};

#endif

allocator.h

// Реализации класса аллокатор для ЛР №6.

// Савров Никита. Группа М80-207б-18

#ifndef ALLOCATOR\_H

#define ALLOCATOR\_H

#include "queue.h"

#include <stack>

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

class TAllocator{

private:

T\* \_buffer;

std::stack<T\*> \_free;

size\_t last\_free;

public:

using value\_type = T;

using pointer = T\*;

using const\_pointer = const T\*;

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

TAllocator() noexcept :\_buffer(nullptr) {

}

~TAllocator(){

free (\_buffer);

}

template <class U>

struct rebind {

using other = TAllocator<U, BLOCK\_SIZE>;

};

T\* Allocate(size\_t n){ //function for return free pointer

if (!\_buffer){

\_buffer = (T\*)malloc(BLOCK\_SIZE \* sizeof(T));

FreeStack();

last\_free = BLOCK\_SIZE - 1;

}

if (n > BLOCK\_SIZE) {

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

return nullptr;

}

if (\_free.size() == 0){

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

return nullptr;

}

\_free.pop();

return \_free.top();

}

void Deallocate(size\_t n){

free(\_free.top());

}

private:

void FreeStack(){

for (int i = 0; i < BLOCK\_SIZE; i++){

\_free.push(\_buffer + i);

}

}

};

#endif

main.cpp

// Лабораторная работа №6 Савров Никита М80-207Б-18

// Разработать шаблоны классов согласно варианту задания.

// Параметром шаблона должен являться скалярный тип данных задающий тип данных для оси координат.

// Классы должны иметь публичные поля. Фигуры являются фигурами вращения. Для хранения координат

// фигур необходимо использовать шаблон std::pair.

// Создать шаблон динамической коллекцию, согласно варианту задания:

// 1.Коллекция должна быть реализована с помощью умных указателей (std::shared\_ptr, std::weak\_ptr). Опционально использование std::unique\_ptr;

// 2.В качестве параметра шаблона коллекция должна принимать тип данных;

// 3.Реализовать forward\_iterator по коллекции;

// 4.Коллекция должны 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 аз 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: o Стек – pop, push, top; o Очередь – pop, push, top;

// Список, Динамический массив – доступ к элементу по оператору [];

// 10. Реализовать программу, которая: o 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; o Выводит на экран введенные фигуры c помощью std::for\_each; o Выводит на экран количество объектов, у которых площадь меньше заданной (с помощью

// std::count\_if);

// Вариант 19:Прямоугольник, Очередь, Стек.

#include "queue.h"

#include "rectangle.h"

#include "allocator.h"

#include <string>

#include <vector>

#include <cmath>

void Insert(Queue<Rectangle<int>> &q){ //function for insert element in queue

Rectangle<int> tmp;

std::cout << "enter 2 coordinats for 2 point\n";

std::cin >> tmp;

q.Push(tmp);

}

void Erase(Queue<Rectangle<int>> &q){ //function for delete element from queue

q.Pop();

std::cout << "First element is delete\n";

}

void PrintRectangle (Rectangle<int> tmp){

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

}

void Print(Queue<Rectangle<int>> &q){ //function for print all figure

std::for\_each(q.Begin(), q.End(), [](Rectangle<int> a){

PrintRectangle(a);

});

}

void Count(Queue<Rectangle<int>> &q){ //function for count figures with square less test square

int test;

std::cout << "enter test square\n";

std::cin >> test;

int res = std::count\_if(q.Begin(), q.End(), [test](Rectangle<int> &a) -> bool{

return a.Square() < test;

});

std::cout << res << " figures with square less " << test << '\n';

}

int main(){

Queue<Rectangle<int>> q;

std::string s;

std::cout << "1 - Push to queue\n2 - Pop to queue\n3 - Print queue\n4 - Count square lower test\nq - exit\n";

while (std::cin >> s){

if (s == "1"){

Insert(q);

} else if (s == "2"){

Erase(q);

} else if (s == "3") {

Print(q);

} else if (s == "4"){

Count(q);

} else if (s == "q"){

break;

}

std::cout << "1 - Push to queue\n2 - Pop to queue\n3 - Print queue\n4 - Count square lower test\nq - exit\n";

}

}

https://github.com/trol53/oop\_exercise\_01/tree/master/oop\_exercise\_06

6.Вывод

Данная программа может быть полезна для работы с геометрическими фигурами. Так-же ввиду того что класс очередь шаблонный, этот код можно использовать в любой программе где требуется этом типа данных .Благодаря парадигме ООП мы можем представлять пользовательские типы данных в виде объектов что значительно упрощает программирование. Так-же эта работа показывает практическое применение шаблонных классов.

Литература

1.Онлайн библиотека c++ cppreference[электронный ресурс] URL:https://ru.cppreference.com/w/cpp/memory/allocator

(дата обращения 06.12.2019)

2.Информационный портал хабр [электронный ресурс] URL: https://habr.com/ru/post/148657/

(дата обращения 07.12.2019)