**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: Трофимов М.А.

Группа: 80-201

Преподаватель: Чернышов Л.Н.

Дата:\_\_\_\_\_\_\_\_\_

Оценка:\_\_\_\_\_\_\_\_\_

Подпись:\_\_\_\_\_\_\_\_\_

Москва, 2019

1. **Постановка задачи**

Реализовать список, который хранит квадраты, на умных указателях с использованием итераторов и собственного аллокатора.

1. **Репозиторий github**

<https://github.com/student31415/oop_exercise_06>

1. **Описание программы**

Программа принимает на вход команды на вставку и удаление квадратов. Квадраты определяются по координатам двух противоположных вершин. Также можно произвести распечатку всех характеристик хранящихся квадратов, одного квадрата и подсчитать количество квадратов с площадью строго меньше заданной.

4. **Набор testcases**

**test\_01.txt**

1

0 0 5 5

0

1

5 5 0 0

1

1

0 3 7 4

2

1

1 2 3 4

0

1

5 5 5 6

1

1

0 3 7 5

2

1

1 0 2 5

0

1

0 5 0 40

1

1

0 3 7 4

2

3

4 50

2 1

2 3

5

3

3

0

Действия теста:

наполнение списка квадратами

распечатка характеристик этих квадратов

нахождение количества квадратов с площадью меньше 50

удаление квадратов с индексами 1 3

печать третьего квадрата и печать всех квадратов

5. **Результаты выполнения тестов**

schizo@VivoBook:~/labs/2nd\_kurs/OOP/oop\_exercise\_05$ ./oop\_exercise\_05

Program for working with square in list.

1. insert square at number

2. remove square at number

3. print information about all square

4. print number of elemets with square less then

5. print information of one square at number

0. exit

>>1

input 2 square vertexes

>>0 0 5 5

input index

>>0

>>1

input 2 square vertexes

>>5 5 0 0

input index

>>1

>>1

input 2 square vertexes

>>0 3 7 4

input index

>>2

>>1

input 2 square vertexes

>>1 2 3 4

input index

>>0

>>

1

input 2 square vertexes

>>5 5 5 6

input index

>>1

>>1

input 2 square vertexes

>>0 3 7 5

input index

>>2

>>1

input 2 square vertexes

>>1 0 2 5

input index

>>0

>>1

input 2 square vertexes

>>0 5 0 40

input index

>>1

>>1

input 2 square vertexes

>>0 3 7 4

input index

>>2

>>3

0: ( 1 , 0 );( -1 , 3 );( 2 , 5 );( 4 , 2 )

square 13

center ( 1 , 2 )

1: ( 0 , 5 );( -18 , 22 );( 0 , 40 );( 18 , 22 )

square 630.257

center ( 0 , 22 )

2: ( 0 , 3 );( 3 , 7 );( 7 , 4 );( 4 , 0 )

square 25

center ( 3 , 3 )

3: ( 1 , 2 );( 1 , 4 );( 3 , 4 );( 3 , 2 )

square 4

center ( 2 , 3 )

4: ( 5 , 5 );( 4 , 5 );( 5 , 6 );( 6 , 5 )

square 1.41421

center ( 5 , 5 )

5: ( 0 , 3 );( 2 , 7 );( 7 , 5 );( 5 , 0 )

square 24.0832

center ( 3 , 4 )

6: ( 0 , 0 );( 0 , 5 );( 5 , 5 );( 5 , 0 )

square 25

center ( 2 , 2 )

7: ( 5 , 5 );( 5 , 0 );( 0 , 0 );( 0 , 5 )

square 25

center ( 2 , 2 )

8: ( 0 , 3 );( 3 , 7 );( 7 , 4 );( 4 , 0 )

square 25

center ( 3 , 3 )

>>4

input maximum square

>>50

number of squares with square less then 50 is 8

>>2

input index

>>1

>>2

input index

>>3

>>5

input index

>>3

3: ( 0 , 3 );( 2 , 7 );( 7 , 5 );( 5 , 0 )

square 24.0832

center ( 3 , 4 )

>>3

0: ( 1 , 0 );( -1 , 3 );( 2 , 5 );( 4 , 2 )

square 13

center ( 1 , 2 )

1: ( 0 , 3 );( 3 , 7 );( 7 , 4 );( 4 , 0 )

square 25

center ( 3 , 3 )

2: ( 1 , 2 );( 1 , 4 );( 3 , 4 );( 3 , 2 )

square 4

center ( 2 , 3 )

3: ( 0 , 3 );( 2 , 7 );( 7 , 5 );( 5 , 0 )

square 24.0832

center ( 3 , 4 )

4: ( 0 , 0 );( 0 , 5 );( 5 , 5 );( 5 , 0 )

square 25

center ( 2 , 2 )

5: ( 5 , 5 );( 5 , 0 );( 0 , 0 );( 0 , 5 )

square 25

center ( 2 , 2 )

6: ( 0 , 3 );( 3 , 7 );( 7 , 4 );( 4 , 0 )

square 25

center ( 3 , 3 )

>>0

exit

1. **Листинг программы**

**main.cpp**

#include<iostream>

#include<algorithm>

#include<string>

#include "list.h"

#include "square\_templ"

#include"allocator"

/\*

Трофимов М.А

вариант 10

список квадратов с алокатором

\*/

void print\_menu(){

std::cout

<<"1. insert square at number"<<std::endl

<<"2. remove square at number"<<std::endl

<<"3. print information about all square"<<std::endl

<<"4. print number of elemets with square less then"<<std::endl

<<"5. print information of one square at number"<<std::endl

<<"6. save list to file"<<std::endl

<<"7. load list from file"<<std::endl

<<"0. exit"<<std::endl;

}

bool is\_number(std::string& str){

for(int i=0; i<str.size(); ++i)

if( !( (str[i] <= '9') && (str[i] >= '0') || str[i]=='.') )

return false;

return true;

}

int main(){

List<Square<int>, Allocator<Square<int>, 500>> list;

std::cout<<"Program for working with square in list."<<std::endl<<std::endl;

print\_menu();

while(true){

try{

std::string buf;

std::cout<<">>";

std::cin>>buf;

if(!is\_number(buf)){

std::cout<<"wrong input!"<<std::endl;

continue;

}

int cmd=atoi(buf.c\_str());

switch(cmd){

case 0:

std::cout<<"exit"<<std::endl;

return 0;

case 1 :{

std::cout<<"input 2 square vertexes"<<std::endl<<">>";

std::string b2, b3, b4;

std::cin>>buf>>b2>>b3>>b4;

if(!is\_number(buf) || !is\_number(b2) || !is\_number(b3) || !is\_number(b4)){

std::cout<<"wrong input!"<<std::endl;

continue;

}

std::cout<<"input index"<<std::endl<<">>";

std::string ind;

std::cin>>ind;

if(!is\_number(ind)){

std::cout<<"wrong input!"<<std::endl;

continue;

}

list.insert(

std::next( list.begin() , atoi( ind.c\_str() ) ),

Square( std::pair{ atoi( buf.c\_str() ) , atoi( b2.c\_str() ) } ,

std::pair{ atoi( b3.c\_str() ) , atoi( b4.c\_str() ) } )

);

break;

}

case 2 :{

std::cout<<"input index"<<std::endl<<">>";

std::cin>>buf;

if(!is\_number(buf)){

std::cout<<"wrong input!"<<std::endl;

continue;

}

list.erase( std::next( list.begin() , atoi( buf.c\_str() ) ) );

break;

}

case 3 :{

int i=0;

std::for\_each(list.begin(), list.end(),

[&i](const Square<int>& t){

std::cout<<i++<<": "<<t<<std::endl

<<"square "<<t.square()<<std::endl

<<"center "<<t.center()<<std::endl<<std::endl;

});

break;

}

case 4 :{

std::cout<<"input maximum square"<<std::endl<<">>";

std::cin>>buf;

if(!is\_number(buf)){

std::cout<<"wrong input!"<<std::endl;

continue;

}

double sq=atoi(buf.c\_str());

std::cout<<"number of squares with square less then " << sq <<" is "<<std::count\_if(list.begin(), list.end(), [sq](const Square<int>& t){ return sq>=t.square(); })<<std::endl;

break;

}

case 5 :{

std::cout<<"input index"<<std::endl<<">>";

std::cin>>buf;

if(!is\_number(buf)){

std::cout<<"wrong input!"<<std::endl;

continue;

}

int ind=atoi(buf.c\_str());

std::cout<<ind<<": "<<list[ind]<<std::endl

<<"square "<<list[ind].square()<<std::endl

<<"center "<<list[ind].center()<<std::endl

<<std::endl;

break;

}

case 6 : {

std::cout<<"input file name"<<std::endl<<">>";

std::cin>>buf;

list.save(buf);

break;

}

case 7 : {

std::cout<<"input file name"<<std::endl<<">>";

std::cin>>buf;

list.load(buf);

break;

}

default:

std::cout<<"Wrong Comand"<<std::endl;

}

}catch(std::exception& e){

std::cerr<<"ERROR "<<e.what()<<std::endl;

}

}

return 0;

}

**list.h**

#pragma once

#include <memory>

#include<fstream>

template<class T, class A = std::allocator<T> > class List{

private:

struct Element{

T data;

std::shared\_ptr<Element> next;

std::weak\_ptr<Element> prev;

Element() {}

Element(T d) : data(d) {}

Element(T d, std::shared\_ptr<Element> n) : data(d), next(n) {}

Element(T d, std::shared\_ptr<Element> n, std::shared\_ptr<Element> p) : data(d), next(n), prev(p) {}

Element(T d, std::shared\_ptr<Element> n, std::weak\_ptr<Element> p) : data(d), next(n), prev(p) {}

};

A allocator;

std::shared\_ptr<Element> head, tail;

public:

class iterator{

private:

friend class List;

std::weak\_ptr<List::Element> ukazatel;

public:

using difference\_type = int;

using value\_type = T;

using reference = T&;

using pointer = T\*;

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

reference operator\*(){

if (!ukazatel.lock()->next)

throw std::out\_of\_range("attempt to access unexisting element value");

return ukazatel.lock()->data;

}

pointer operator->(){

if (!ukazatel.lock()->next)

throw std::out\_of\_range("attempt to access unexisting element");

return &(ukazatel.lock()->data);

}

iterator& operator++(){

if (!ukazatel.lock()->next)

throw std::out\_of\_range("attempt to move further than end()");

ukazatel = (\*ukazatel.lock()).next;

return \*this;

}

bool operator!=(const iterator& other){

return ukazatel.lock() != other.ukazatel.lock();

}

};

friend class iterator;

List(){

tail = std::make\_shared<Element>(Element());

head = tail;

}

~List(){}

iterator begin(){

iterator i;

i.ukazatel = head;

return i;

}

iterator end(){

iterator i;

i.ukazatel = tail;

return i;

}

bool empty()const{

return head==tail;

}

size\_t size()const{

size\_t size = 0;

for(std::weak\_ptr<Element> W(head); W.lock() !=tail; W=W.lock()->next, ++size);

return size;

}

void clear(){

head = tail;

return;

}

void insert(iterator i, const T &val){

if (!(i.ukazatel.lock()->prev.lock())){

head = std::allocate\_shared<Element>(allocator, Element(val, head));

head->next->prev = head;

}else{

std::shared\_ptr<Element> el = std::allocate\_shared<Element>(allocator, Element(val, i.ukazatel.lock(), i.ukazatel.lock()->prev));

el->prev.lock()->next = el;

el->next->prev = el;

}

}

iterator erase(iterator i){

if (i.ukazatel.lock() == tail)

throw std::out\_of\_range("attempt to remove end()");

iterator ret = i;

++ret;

if(!(i.ukazatel.lock()->prev.lock())){

head->next->prev = std::weak\_ptr<Element>();

head = head->next;

}else{

std::shared\_ptr<List<T, A>::Element> el = i.ukazatel.lock();

el->next->prev = el->prev;

el->prev.lock()->next = el->next;

}

return ret;

}

T& operator[](int i){

return \*std::next(begin(), i);

}

void save(const std::string& fname){

std::ofstream file(fname, std::ios::binary);

if(!file.good())

throw std::logic\_error("can't open file "+fname);

size\_t \_size=size();

file.write((char\*)&\_size, sizeof(\_size));

std::for\_each(begin(), end(), [&file](const T& elem){ file.write((char\*)&elem, sizeof(elem)); });

file.close();

}

void load(const std::string& fname){

std::ifstream file(fname, std::ios::binary);

if(!file.good())

throw std::logic\_error("can't open file "+fname);

clear();

size\_t \_size=0;

file.read((char\*)&\_size, sizeof(\_size));

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

T buf;

file.read((char\*)&buf, sizeof(buf));

insert(std::next(begin(), i), buf);

}

file.close();

}

};

**allocator**

#pragma once

#include <memory>

#include <list>

template<class T, size\_t N = 100> //type of values and size

class Allocator

{

private:

using store\_t = std::list<std::pair<char\*,char\*>>; //pointers to begin and end of Bucket

std::shared\_ptr<store\_t> freeBuckets; //list of free Buckets

std::shared\_ptr<char[]> pool; //array of pointers

public:

using value\_type = T;

using pointer = T\* ;

using const\_pointer = const T\*;

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

template<typename U>

struct rebind {

using other = Allocator<U, N>;

};

std::shared\_ptr<char[]> GetPool()const{

return pool;

}

std::shared\_ptr<store\_t> GetList()const{

return freeBuckets;

}

Allocator()noexcept{

static\_assert(N > 0);

pool = std::shared\_ptr<char[]>(new char[N], [](char \*p){ delete[] p; });

freeBuckets = std::make\_shared<store\_t>(store\_t());

store\_t& l = \*freeBuckets.get();

l.push\_back(std::pair(pool.get(), pool.get() + N));

}

template<class U>

Allocator(const Allocator<U,N>& a)noexcept{

pool = a.GetPool();

freeBuckets = a.GetList();

}

~Allocator(){}

T\* allocate(size\_t size){

size \*= sizeof(T);

store\_t& l = \*freeBuckets.get();

for(auto i = l.begin(); i != l.end(); ++i){

auto &chunk = \*i;

if(chunk.second - chunk.first < size)

continue;

size\_t diff = chunk.second - chunk.first - size;

if(diff > 0){

char\* ptr = chunk.first + size;

l.insert(i, std::pair(ptr, ptr + diff));

}

l.erase(i);

return (T\*)chunk.first;

}

throw std::bad\_alloc();

}

void deallocate(T\* ptr1, size\_t n){

n \*= sizeof(T);

char \*ptr = (char\*)ptr1;

store\_t& l = \*freeBuckets.get();

auto i = l.begin();

for (; i != l.end(); ++i)

if((\*i).first > ptr)

break;

l.insert(i, std::pair(ptr, ptr + n));

if(i != l.end()){

auto &chunk = \*i;

--i;

if (ptr + n == chunk.first){

(\*i).second += chunk.second - chunk.first;

n += chunk.second - chunk.first;

auto j = i; ++j;

l.erase(j);

}

}

else

--i;

if (i != l.begin())

{

auto j = i; --j;

auto &chunk = \*j;

if(chunk.second == ptr) {

chunk.second += n;

l.erase(i);

}

}

}

template<typename U, typename ...Args>

void construct(U \*p, Args &&...args){

new (p) U(std::forward<Args>(args)...);

}

void destroy(pointer p){

p->~T();

}

};

template<class T, size\_t N1, class U, size\_t N2>

bool operator==(Allocator<T,N1>& a, Allocator<U,N2>& b)noexcept{

return (sizeof(T)==sizeof(U)) && (N1==N2) && (a.pool.get()==b.pool.get());

}

template<class T, size\_t N1, class U, size\_t N2>

bool operator!=(Allocator<T,N1>& a, Allocator<U,N2>& b)noexcept{

return !(a==b);

}

**square\_templ**

#pragma once

#include<iostream>

#include<cmath>

template <class T>

class Square{

public:

using type = T;

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

vertex p1={}, p2={};

Square(){}

Square(vertex a, vertex b){

if(a.first!=b.first || a.second!=b.second){

p1 = a;

p2 = b;

}

}

double square()const {

std::pair<T, T> c {

-p2.second + (p2.second + p1.second + p2.first + p1.first)/2,

p2.first + (p2.second + p1.second - p2.first - p1.first)/2

};

return (double) length( p1, c )

\* length( c, p2 );

}

vertex center()const {

return std::pair{

( p1.first + p2.first ) / 2,

( p1.second + p2.second ) / 2

};

}

template<class A> friend std::ostream& operator <<(std::ostream &o, const Square<A> a);

private:

double length(std::pair<T, T> p1, std::pair<T, T> p2)const {

return sqrt( (p2.first-p1.first)\*(p2.first-p1.first)

+ (p2.second-p1.second)\*(p2.second-p1.second) );

}

double cos(std::pair<T, T> p1, std::pair<T, T> p2, std::pair<T, T> p3, std::pair<T, T> p4)const {

return ( (p2.first-p1.first) \* (p4.first-p3.first)

+ (p2.second-p1.second) \* (p4.second-p3.second) )

/ ( length(p1, p2) \* length( p3, p4) );

}

};

template<typename T>

std::ostream& operator <<(std::ostream &o, const Square<T> a){

o<<a.p1<<';'<<std::pair<T, T>{

-a.p2.second + (a.p2.second + a.p1.second + a.p2.first + a.p1.first)/2,

a.p2.first + (a.p2.second + a.p1.second - a.p2.first - a.p1.first)/2

}<<';'<<a.p2<<';'<<std::pair<T, T>{

a.p2.second + (a.p2.first + a.p1.first - a.p2.second - a.p1.second)/2,

-a.p2.first + (a.p2.second + a.p1.second + a.p2.first + a.p1.first)/2

};

return o;

}

template<typename T>

std::ostream& operator <<(std::ostream &o, const std::pair<T, T>& a){

o<<"( "<<a.first<<" , "<<a.second<<" )";

return o;

}

7. **Вывод**

Использование умных указателей в некоторых случаях позволяет избежать утечек памяти, но также усложняет написание кода.

Программа позволяет работать со стеком из правильных пятиугольников, а реализованные итераторы - со стандартными функциями std::count, std::count\_if и т.п.

**Список литературы**

1. Справочник по языку С++ [Электронный ресурс]. URL:

<https://en.cppreference.com/w/> (дата обращения: 23.12.2019).