**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 6**

Тема: Основы работы с коллекциями: аллокаторы

Студент: Воронов Кирилл Михайлович

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. **Постановка задачи**

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, т.е. равносторонние (кроме трапеции и прямоугольника). 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. Коллекция должна содержать метод доступа:

o Стек – pop, push, top;

o Очередь – pop, push, top;

o Список, 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:

o позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять в коллекцию использующую аллокатор;

o позволяет удалять элемент из коллекции по номеру элемента;

o выводит на экран введенные фигуры c помощью std::for\_each.

*Вариант 27 Прямоугольник, динамический массив, стек*

1. **Описание программы** В файле vector.hpp реализован класс TVector с переменными Max(размером вектора) и умным указателем shared\_ptr<T> Data. Также в нём находится класс Iterator, который тоже имеет умный указатель shared\_ptr<T> ptr, переменные size(размер вектора) и index(текущий индекс вектора, на который указывает итератор). Для итератора и вектора реализованы соответствующие операторы и методы (добавление через итератор insert, удаление erase). В allocator.hpp реализован класс stack\_allocator, который содержит в себе стек указателей типа Т st и указатель типа Т buffer. В нём реализованы соответствующие конструкторы и деструкторы, а также функции allocate и deallocate В main.cpp реализован класс Rectangle с парой t для координат и переменными a,b - длинами сторон прямоугольника. Также добавлена структура deleter для shared\_ptr. Взаимодействие с программой происходит через интерактивное меню, в котором указаны номера команд и их действие. Команды вызывают соответствующие функции.
2. **Набор тестов и результаты их выполнения**

**Тест 1**

Введите 1, чтобы добавить элемент

Введите 2, чтобы удалить элемент

Введите 3, чтобы вывести вектор

Введите 4, чтобы вывести меню

1

Введите координату нижней левой точки прямоугольника и длины сторон, прилегающих к ней, начиная с горизонтальной

1 5 7 8

1

Введите индекс элемента массива, перед которым добавить элемент

0

Введите координату нижней левой точки прямоугольника и длины сторон, прилегающих к ней, начиная с горизонтальной

1.55 2.6 3.8 7.4

3

(1.55,10) (5.35,10)

(1.55,2.6) (5.35,2.6)

-----------------

(1,13) (8,13)

(1,5) (8,5)

-----------------

2

Введите индекс элемента для удаления

1

3

(1.55,10) (5.35,10)

(1.55,2.6) (5.35,2.6)

-----------------

**Тест 2**

Введите 1, чтобы добавить элемент

Введите 2, чтобы удалить элемент

Введите 3, чтобы вывести вектор

Введите 4, чтобы вывести меню

1

Введите координату нижней левой точки прямоугольника и длины сторон, прилегающих к ней, начиная с горизонтальной

1.1 1.1 1.1 1.1

1

Введите индекс элемента массива, перед которым добавить элемент

1

Введите координату нижней левой точки прямоугольника и длины сторон, прилегающих к ней, начиная с горизонтальной

2.2 2.2 2.2 2.2

1

Введите индекс элемента массива, перед которым добавить элемент

2

Введите координату нижней левой точки прямоугольника и длины сторон, прилегающих к ней, начиная с горизонтальной

3.3 3.3 3.3 3.3

3

(1.1,2.2) (2.2,2.2)

(1.1,1.1) (2.2,1.1)

-----------------

(2.2,4.4) (4.4,4.4)

(2.2,2.2) (4.4,2.2)

-----------------

(3.3,6.6) (6.6,6.6)

(3.3,3.3) (6.6,3.3)

-----------------

2

Введите индекс элемента для удаления

2

3

(1.1,2.2) (2.2,2.2)

(1.1,1.1) (2.2,1.1)

-----------------

(2.2,4.4) (4.4,4.4)

(2.2,2.2) (4.4,2.2)

-----------------

2

Введите индекс элемента для удаления

0

2

Введите индекс элемента для удаления

0

3

Вектор пуст

4

Введите 1, чтобы добавить элемент

Введите 2, чтобы удалить элемент

Введите 3, чтобы вывести вектор

Введите 4, чтобы вывести меню

1. **Листинг программы**

allocator.hpp

#ifndef ALLOCATOR\_HPP

#define ALLOCATOR\_HPP

#include <stack>

using namespace std;

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

class stack\_allocator {

private:

stack<T\*> st;

T\* buffer = nullptr;

public:

using allocator\_type = stack\_allocator;

using value\_type = T;

using pointer = T\*;

using reference = T&;

using const\_reference = const T&;

using size\_type = size\_t;

T\* allocate(const size\_t & n) {

if(buffer == nullptr) {

buffer = new T[BLOCK\_SIZE];

for (int i = BLOCK\_SIZE - 1; i >= 0; --i) {

st.push(&buffer[i]);

}

}

if (st.size() < n) {

throw(bad\_alloc());

} else {

T \*p = st.top();

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

st.pop();

}

return p;

}

}

void deallocate(pointer, size\_t n) {

;

}

explicit stack\_allocator(const stack\_allocator<T, BLOCK\_SIZE> & another\_allocator) : stack\_allocator() {

buffer = new T[BLOCK\_SIZE];

for (std::size\_t i = 0; i < BLOCK\_SIZE; ++i) {

buffer[i] = another\_allocator.buffer[i];

st.push(&buffer[i]);

}

}

stack\_allocator() : st(), buffer(nullptr) {

static\_assert(BLOCK\_SIZE > 0);

}

template<class OTHER\_T>

class rebind {

public:

using other = stack\_allocator<OTHER\_T, BLOCK\_SIZE>;

};

~stack\_allocator() {

delete[] buffer;

}

};

#endif

vector.hpp

#ifndef VECTOR\_HPP

#define VECTOR\_HPP

#include<iostream>

#include<memory>

#include"allocator.hpp"

using namespace std;

template <class T, class ALLOCATOR>

class TVector {

public:

struct deleter;

deleter del;

int Max = 0;

shared\_ptr<T[]> Data = shared\_ptr<T[]>(nullptr,del);

using allocator\_type = typename ALLOCATOR::template rebind<T>::other;

static allocator\_type& get\_allocator() {

static allocator\_type allocator;

return allocator;

}

struct deleter {

void operator() (void\* ptr) {

get\_allocator().deallocate((T\*)ptr, 1);

}

};

class Iterator {

public:

typedef Iterator self;

typedef T value\_type;

typedef T &reference;

typedef T \*pointer;

typedef std::random\_access\_iterator\_tag iterator\_category;

typedef ptrdiff\_t difference\_type;

shared\_ptr<T[]> ptr;

int size;

int index = 0;

public:

Iterator(shared\_ptr<T[]> ptr, size\_t const index, int size) : ptr(ptr), index(index), size(size) {};

Iterator(){

}

T operator\*(){

return ptr[index];

}

self &operator++() {

if (index >= size)

throw std::out\_of\_range("Итератор вышел за пределы вектора\n");

index+=1;

return \*this;

}

bool operator<(const Iterator &b) {

return index < b.index;

}

bool operator!=(const Iterator &b) {

return index != b.index;

}

};

Iterator begin(){

return Iterator(Data, 0, Max);

}

Iterator end(){

return Iterator(Data, Max, Max);

}

void insert(T z, Iterator it) {

if(Max == 0) {

T \* u = get\_allocator().allocate(1);

shared\_ptr<T[]> Data2{new(u) T[1],(del)};

Data = Data2;

Data[0] = z;

Max = 1;

it.ptr = Data;

it.size = 1;

} else {

T \* u = get\_allocator().allocate(Max + 1);

shared\_ptr<T[]> Data2{new(u) T[Max + 1],(del)};

for (int i = 0; i < it.index; ++i){

Data2[i] = Data[i];

}

Data2[it.index] = z;

for (int i = it.index + 1; i < Max + 1; ++i) {

Data2[i] = Data[i-1];

}

it.ptr = Data2;

it.size+=1;

Data = Data2;

Max+=1;

}

}

void erase(Iterator it) {

if (Max == 1) {

Data = nullptr;

Max = 0;

it.ptr = nullptr;

it.size = 0;

it.index = 0;

} else {

T \* u = get\_allocator().allocate(Max + 1);

shared\_ptr<T[]> Data2{new(u) T[Max - 1],(del)};

for (int i = 0; i < it.index; ++i) {

Data2[i] = Data[i];

}

for (int i = it.index + 1; i < Max; ++i) {

Data2[i-1] = Data[i];

}

Data = Data2;

Max-=1;

it.ptr = Data;

it.size-=1;

}

}

int size() {

return Max;

}

T& operator[](int i){

return Data[i];

}

};

#endif

main.cpp

/\* Воронов К.М.

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, т.е. равносторонними (кроме трапеции и прямоугольника).

Для хранения координат фигур необходимо использовать шаблон std::pair.

Создать шаблон динамической коллекцию, итератор и аллокатор.

\*/

#include <iostream>

#include <string>

#include <algorithm>

#include"vector.hpp"

const size\_t AR = 100;

template <typename T>

class Rectangle{

public:

using type = T;

pair<T,T> t;

T a;

T b;

Rectangle(){};

Rectangle(T x,T y,T a1, T b1): t(x,y), a(a1), b(b1){};

};

double square(Rectangle<double> p) {

return p.a\*p.b;

}

void print2(Rectangle<double> p) {

pair<double,double> t2((p.t).first + p.a, (p.t).second );

pair<double,double> t3((p.t).first + p.a, (p.t).second + p.b);

pair<double,double> t4((p.t).first, (p.t).second + p.b);

cout << "(" << t4.first << "," << t4.second << ") "

<< "(" << t3.first << "," << t3.second << ")"

<< endl << endl << "(" << (p.t).first << ","

<< (p.t).second << ") " << "(" << t2.first << ","

<< t2.second << ")" << endl << "-----------------" << endl;

}

void Add(TVector<Rectangle<double>, stack\_allocator<double, AR>> &v, TVector<Rectangle<double>, stack\_allocator<double, AR>>::Iterator &begin, Rectangle<double> r){

double x,y,a,b;

if (v.size() < 1) {

printf("Введите координату нижней левой точки прямоугольника и длины сторон, прилегающих к ней, начиная с горизонтальной\n");

cin >> x >> y >> a >> b;

if ((a<0) || (b<0) || !cin){

cout << "Введены неверные данные" << endl;

return;

}

(r.t).first = x;

(r.t).second = y;

r.a = a;

r.b = b;

v.insert(r, begin);

return;

}

int index;

printf("Введите индекс элемента массива, перед которым добавить элемент\n");

cin >> index;

begin = v.begin();

if(index > v.size() || index < 0 || !cin){

cout << "Введены неверные данные" << endl;

return;

}

while(begin.index < index) {

++begin;

}

printf("Введите координату нижней левой точки прямоугольника и длины сторон, прилегающих к ней, начиная с горизонтальной\n");

cin >> x >> y >> a >> b;

if ((a<0) || (b<0) || !cin){

cout << "Введены неверные данные" << endl;

return;

}

(r.t).first = x;

(r.t).second = y;

r.a = a;

r.b = b;

v.insert(r, begin);

return;

}

void Delete(TVector<Rectangle<double>, stack\_allocator<double, AR>> &v, TVector<Rectangle<double>, stack\_allocator<double, AR>>::Iterator &begin){

int index;

if (v.size() == 0){

cout <<"Вектор пуст" << endl;

return;

}

printf("Введите индекс элемента для удаления\n");

cin >> index;

if((index < 0) || (!cin) || (index > v.size())) {

cout << "Введены неверные данные" << endl;

return;

}

begin = v.begin();

while(begin.index < index) {

++begin;

}

v.erase(begin);

return;

}

double s;

bool ifsquare(Rectangle<double> p){

if (square(p) < s)

return true;

else return false;

}

void Print(TVector<Rectangle<double>, stack\_allocator<double, AR>> ::Iterator &it,TVector<Rectangle<double>, stack\_allocator<double, AR>> v ){

if (v.size() == 0){

cout <<"Вектор пуст" << endl;

return;

}

for\_each(v.begin(), v.end(),print2);

return;

}

void Menu(){

printf("Введите 1, чтобы добавить элемент\n");

printf("Введите 2, чтобы удалить элемент\n");

printf("Введите 3, чтобы вывести вектор\n");

printf("Введите 4, чтобы вывести меню\n");

}

int main(){

TVector<Rectangle<double>, stack\_allocator<double, AR>> v;

auto begin = v.begin();

auto end = v.end();

double x,y,a,b;

int input;

Rectangle<double> r;

printf("Введите 1, чтобы добавить элемент\n");

printf("Введите 2, чтобы удалить элемент\n");

printf("Введите 3, чтобы вывести вектор\n");

printf("Введите 4, чтобы вывести меню\n");

while (cin >> input) {

if (!cin){

cout << "Введены неверные данные" << endl;

return -1;

}

switch(input){

case 1:

Add(v,begin,r);

break;

case 2:

Delete(v,begin);

break;

case 3:

Print(begin,v);

break;

case 4:

Menu();

break;

default:

cout<<"Вы ввели неверную команду"<<std::endl;

return -1;

break;

}

}

}

**Список литературы**

# Альтернативные аллокаторы памяти [Электронный ресурс]

URL: <https://habr.com/ru/post/274827/> (дата обращения 29.11.2020).

# Аллокаторы памяти [Электронный ресурс]

URL: https://habr.com/ru/post/505632/ (дата обращения 29.11.2020).