**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: Хренникова Ангелина Сергеевна

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. **Постановка задачи**

Цель: Изучить основы работы с контейнерами, познакомиться концепцией аллокаторов памяти.

Задача: Разработать шаблоны классов фигура: шестиугольник, контейнер: стек, аллокатор: стек. 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. 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: стек – pop, push, top;
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;
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 с помощью std::for\_each.

**2. Описание программы**

Пользователю доступны действия: push (добавить элемент), pop (удалить элемент), print (распечатать стек), insert (вставить шестиугольник на конкретную позицию), erase (удалить шестиугольник с конкретной позиции), count (подсчитать количество шестиугольников, чья площадь меньше заданной). В программе есть специальное меню, позволяющее выбрать определенное действие. В случае некорректно введенных данных генерируются исключения.

Шестиугольник задается координатами центральной точкой и одной из точек самой фигуры, остальные точки вычисляются программой. Для каждой фигуры печатаются координаты и площадь. Площадь вычисляется по формуле: , где R - радиус описанной окружности(разность координат центра и одной из точек шестиугольника).

В программе реализован класс шестиугольника, для работы с фигурой(полями являются пары координат и площадь), класс стека с помощью умных указателей, класс итераторов, класс аллокатора. В реализации стека учтены все основные функции коллекции, такие как pop, push, top и insert, erase. Позиции в стеке начинаются с 0.

**3. Набор тестов**

Тест 1: 1 0 0 0 6 1 0 0 0 5 4 1 0 0 0 3 4 0 0 0 0 1 3 2 6 25 0

1. В стек добавляется шестиугольник с центром в точке (0;0) и точкой (0;6);
2. В стек добавляется шестиугольник с центром в точке (0;0) и точкой (0;5);
3. В стек добавляется шестиугольник с центром в точке (0;0) и точкой (0;3) на позицию 1;
4. В стек добавляется шестиугольник с центром в точке (0;0) и точкой (0;1) на позицию 0;
5. Печатается весь стек;
6. Удаляется элемент из стека;
7. Считается количество элементов в стеке, площадь которых меньше 25;
8. Программа завершается.

Тест 2: 1 0 0 1 1 1 0 0 2 2 4 0 0 0 3 3 1 0 0 4 4 5 2 3 6 50 0

1. В стек добавляется шестиугольник с центром в точке (0;0) и точкой (1;1);
2. В стек добавляется шестиугольник с центром в точке (0;0) и точкой (2;2);
3. В стек добавляется шестиугольник с центром в точке (0;0) и точкой (3;3) на позицию 0;
4. В стек добавляется шестиугольник с центром в точке (0;0) и точкой (4;4);
5. Удаляется элемент на 2 позиции стека;
6. Печатается весь стек;
7. Считается количество элементов в стеке, площадь которых меньше 50;
8. Программа завершается.

**4. Результат выполнения тестов**

Тест1:

Menu:

1. Push figure to stack

2. Pop figure from stack

3. Print stack

4. Insert item to pos N

5. Delete item from pos N

6. Count figures with area less then area

0. Exit

Select menu item: 1

Enter figure cords: 0 0 0 6

Select menu item: 1

Enter figure cords: 0 0 0 5

Select menu item: 4

Enter position: 1

Enter figure cords: 0 0 0 3

Select menu item: 4

Enter position: 0

Enter figure cords: 0 0 0 1

Select menu item: 3

[(0, 1), (-0.866025, 0.5), (-0.866025, -0.5), (-1.11022e-16, -1), (0.866025, -0.5), (0.866025, 0.5)]

Area: 2.59808

[(0, 5), (-4.33013, 2.5), (-4.33013, -2.5), (-4.44089e-16, -5), (4.33013, -2.5), (4.33013, 2.5)]

Area: 64.9519

[(0, 3), (-2.59808, 1.5), (-2.59808, -1.5), (0, -3), (2.59808, -1.5), (2.59808, 1.5)]

Area: 23.3827

[(0, 6), (-5.19615, 3), (-5.19615, -3), (0, -6), (5.19615, -3), (5.19615, 3)]

Area: 93.5307

Select menu item: 2

Select menu item: 6

Enter area: 25

Result = 1

Select menu item: 0

Тест2:

Menu:

1. Push figure to stack

2. Pop figure from stack

3. Print stack

4. Insert item to pos N

5. Delete item from pos N

6. Count figures with area less then area

0. Exit

Select menu item: 1

Enter figure cords: 0 0 1 1

Select menu item: 1

Enter figure cords: 0 0 2 2

Select menu item: 4

Enter position: 0

Enter figure cords: 0 0 3 3

Select menu item: 1

Enter figure cords: 0 0 4 4

Select menu item: 5

Enter position: 2

Select menu item: 3

[(4, 4), (-1.4641, 5.4641), (-5.4641, 1.4641), (-4, -4), (1.4641, -5.4641), (5.4641, -1.4641)]

Area: 83.1384

[(3, 3), (-1.09808, 4.09808), (-4.09808, 1.09808), (-3, -3), (1.09808, -4.09808), (4.09808, -1.09808)]

Area: 46.7654

[(1, 1), (-0.366025, 1.36603), (-1.36603, 0.366025), (-1, -1), (0.366025, -1.36603), (1.36603, -0.366025)]

Area: 5.19615

Select menu item: 6

Enter area: 50

Result = 2

Select menu item: 0

**5. Листинг программы**

#include <iostream>

#include <string>

#include <cmath>

#include <ctime>

#include <algorithm>

#include <memory>

#include <exception>

#include <stdexcept>

#include <map>

#include <list>

template <class T>

struct node { //Элемент

using type = T;

node() {};

node(const T& key) : value(key) {}; //Значение

std::shared\_ptr<node<T>> next; //Указатель на следующий

std::weak\_ptr<node<T>> prev; //Указатель на предыдущий

T value;

};

template <class T, size\_t block\_count>

class myAllocator { //Аллокатор

public:

using value\_type = T;

using pointer = T\*;

using double\_pointer = T\*\*;

using const\_pointer = const T\*;

using const\_double\_pointer = const T\*\*;

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

myAllocator() { //Конструктор

used\_blocks = (pointer)malloc(block\_count \* sizeof(T));

free\_blocks = (double\_pointer)malloc(block\_count \* sizeof(pointer));

for (size\_type i = 0; i < block\_count; i++) {

free\_blocks[i] = &used\_blocks[i];

}

free\_count = block\_count;

}

~myAllocator() { //Деструктор

if (free\_count < block\_count) {

std::cout << "~myAllocator(): memory leak\n";

}

free(used\_blocks);

free(free\_blocks);

}

pointer allocate(size\_type junk) { //Выделение блока

pointer result = nullptr;

if (free\_count > 0) {

result = free\_blocks[--free\_count];

} else {

std::cout << "allocate(): not enough memory\n";

throw std::bad\_alloc();

}

return result;

}

void deallocate(pointer p, size\_type junk) { //Освобождение памяти

free\_blocks[free\_count++] = p;

}

void destroy(pointer p) { //Удаление указателя

p->~T();

}

template <class U, class ...Args>

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

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

}

template <class U>

struct rebind {

using other = myAllocator<U, block\_count>;

};

struct rebind\_to\_node {

using other = myAllocator<node<T>, block\_count>;

};

private:

pointer used\_blocks; //Указатель на используемые блоки

double\_pointer free\_blocks; //Указатель на свободные блоки

size\_type free\_count; //Размер памяти/количество блоков

};

template <class T>

std::istream& operator>>(std::istream& is, std::pair<T, T>& p) { //Считывание координат

is >> p.first >> p.second;

return is;

}

template <class T>

class hexagon { //Шестиугольник

public:

friend std::ostream& operator<<(std::ostream& os, hexagon<T>& p) { //Вывод шестиугольника

os << "[";

for (int i = 1; i < 6; i++) {

os << "(" << p.cords[i].first << ", " << p.cords[i].second << "), ";

}

os << "(" << p.cords[6].first << ", " << p.cords[6].second << ")";

os << "]\n";

os << "Area: " << p.area\_;

return os;

}

friend std::istream& operator>>(std::istream& is, hexagon<T>& p) { //Ввод шестиугольника

for (int i = 0; i < 2; i++) {

is >> p.cords[i];

}

double c = 0.5;

double s = sqrt(3) / 2;

p.cords[2].first = p.cords[0].first + (p.cords[1].first - p.cords[0].first) \* c - (p.cords[1].second - p.cords[0].second) \* s;

p.cords[2].second = p.cords[0].second + (p.cords[1].first - p.cords[0].first) \* s + (p.cords[1].second - p.cords[0].second) \* c;

p.cords[3].first = p.cords[0].first + (p.cords[2].first - p.cords[0].first) \* c - (p.cords[2].second - p.cords[0].second) \* s;

p.cords[3].second = p.cords[0].second + (p.cords[2].first - p.cords[0].first) \* s + (p.cords[2].second - p.cords[0].second) \* c;

p.cords[4].first = p.cords[0].first + (p.cords[3].first - p.cords[0].first) \* c - (p.cords[3].second - p.cords[0].second) \* s;

p.cords[4].second = p.cords[0].second + (p.cords[3].first - p.cords[0].first) \* s + (p.cords[3].second - p.cords[0].second) \* c;

p.cords[5].first = p.cords[0].first + (p.cords[4].first - p.cords[0].first) \* c - (p.cords[4].second - p.cords[0].second) \* s;

p.cords[5].second = p.cords[0].second + (p.cords[4].first - p.cords[0].first) \* s + (p.cords[4].second - p.cords[0].second) \* c;

p.cords[6].first = p.cords[0].first + (p.cords[5].first - p.cords[0].first) \* c - (p.cords[5].second - p.cords[0].second) \* s;

p.cords[6].second = p.cords[0].second + (p.cords[5].first - p.cords[0].first) \* s + (p.cords[5].second - p.cords[0].second) \* c;

double r = sqrt(pow(p.cords[1].first - p.cords[0].first, 2) + pow(p.cords[1].second - p.cords[0].second, 2));

p.area\_ = 3 \* r \* r \* s;

return is;

}

hexagon() {}; //Контструктор

hexagon(std::pair<T, T> p0, std::pair<T, T> p1, std::pair<T, T> p2, std::pair<T, T> p3,

std::pair<T, T> p4, std::pair<T, T> p5, std::pair<T, T> p6) { //Конструктор

cords[0] = p0;

cords[1] = p1;

cords[2] = p2;

cords[3] = p3;

cords[4] = p4;

cords[5] = p5;

cords[6] = p6;

area\_ = 0;

double s = sqrt(3) / 2;

double r = sqrt(pow(cords[1].first - cords[0].first, 2) + pow(cords[1].second - cords[0].second, 2));

area\_ = 3 \* r \* r \* s;

}

double area() { //Возвращение площади

return area\_;

}

~hexagon() {}; //Деструктор

private:

std::pair<T, T> cords[7]; //Координаты

double area\_ = 0; //Площадь

};

template <class T>

class iterator { //Итератор

protected:

iterator() {};

public:

typedef std::bidirectional\_iterator\_tag iterator\_category;

typedef T value\_type;

typedef int difference\_type;

typedef T\* pointer;

typedef T& reference;

iterator(std::shared\_ptr<T> pointer) : ptr(pointer) {}; //Конструктор

iterator operator++() { //Следующий итератор

ptr = (ptr.lock())->next;

return \*this;

}

iterator operator++(int junk) { //Следующий итератор

iterator i = \*this;

ptr = (ptr.lock())->next;

return i;

}

iterator operator--() { //Предыдущий итератор

ptr = (ptr.lock())->prev;

return \*this;

}

iterator operator--(int junk) { //Предыдущий итератор

iterator i = \*this;

ptr = (ptr.lock())->prev;

return i;

}

typename T::type& operator\*() { //Значение

return (ptr.lock())->value;

}

std::shared\_ptr<T> operator->() { //Указатель

return ptr.lock();

}

bool operator==(const iterator& rhs) { //Равенство

return ptr.lock() == rhs.ptr.lock();

}

bool operator!=(const iterator& rhs) { //Отсутствие равенства

return ptr.lock() != rhs.ptr.lock();

}

private:

std::weak\_ptr<T> ptr; //Указатель на элемент

};

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

class stack : public iterator<node<T>> { //Стек

public:

stack() { //Конструктор

fake = nodeAllocate();

fake->next = fake;

fake->prev = fake;

size\_ = 0;

}

~stack() { //Деструктор

while (fake->next != fake) {

pop();

}

fake->next = nullptr;

}

void push(const T& item) { //Добавление в стек

auto temp = nodeAllocate(item);

if (fake->next == fake) {

fake->next = temp;

fake->prev = temp;

temp->next = fake;

temp->prev = fake;

} else {

auto first = fake->next;

temp->next = first;

temp->prev = fake;

first->prev = temp;

fake->next = temp;

}

size\_++;

}

void pop() { //Удаление из стека

if (is\_empty()) {

throw std::logic\_error("Empty stack pop");

}

if (fake->next == (fake->prev).lock()) {

fake->next = fake;

} else {

fake->next = fake->next->next;

fake->next->prev = fake;

}

size\_--;

}

T top() const { //Верхний элемент

if (is\_empty()) {

throw std::logic\_error("Empty stack top");

}

return fake->next->value;

}

void insert(iterator<node<T>> it, const T& key) { //Вставка на позицию

if (is\_empty()) {

push(key);

} else {

auto temp = std::make\_shared<node<T>>(key);

temp->next = (it->next->prev).lock();

temp->prev = it->prev;

((it->prev).lock())->next = temp;

it->prev = temp;

size\_++;

}

}

void erase(iterator<node<T>> it) { //Удаление с позиции

if (is\_empty()) {

throw std::logic\_error("Empty stack erase");

}

if (it == end()) {

throw std::logic\_error("Removing fake item is permitted");

}

it->next->prev = it->prev;

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

}

bool is\_empty() const { //Проверка на пустоту

return size\_ == 0;

}

size\_t size() const { //Возвращение размера

return size\_;

}

iterator<node<T>> begin() { //Начало стека

return iterator<node<T>>(fake->next);

}

iterator<node<T>> end() { //Конец стека

return iterator<node<T>>(fake);

}

iterator<node<T>> at(int pos) { //Не попадание в стек

if (pos >= size\_ || pos < 0) {

throw std::logic\_error("Wrong position iterator");

}

if (is\_empty()) {

throw std::logic\_error("Can not get iterator from empty stack");

}

int k = 0;

if (pos < size\_ / 2) {

iterator<node<T>> temp(begin());

while (k < pos) {

temp++;

k++;

}

return temp;

} else {

iterator<node<T>> temp(--end());

while (size\_ - k - 1 > pos) {

temp--;

k++;

}

return temp;

}

}

private:

using node\_allocator = typename Allocator::rebind\_to\_node::other;

node\_allocator all; //Блок

std::shared\_ptr<node<T>> fake; //Указатель на элемент

size\_t size\_; //Размер стека

std::shared\_ptr<node<T>> nodeAllocate() { //Блок в стеке

node<T>\* temp = all.allocate(1);

all.construct(temp);

std::shared\_ptr<node<T>> ptr(temp, [this](node<T>\* a) {

this->all.destroy(a);

this->all.deallocate(a, 1);

});

return ptr;

}

std::shared\_ptr<node<T>> nodeAllocate(const T& item) { //Блок в стеке

node<T>\* temp = all.allocate(1);

all.construct(temp, item);

std::shared\_ptr<node<T>> ptr(temp, [this](node<T>\* a) {

this->all.destroy(a);

this->all.deallocate(a, 1);

});

return ptr;

}

};

void menu() {

std::cout << "Menu: \n"; //Меню

std::cout << "1. Push figure to stack\n"; //Добавление в стек

std::cout << "2. Pop figure from stack\n"; //Удаление из стека

std::cout << "3. Print stack\n"; //Печать стека

std::cout << "4. Insert item to pos N\n"; //Вставка на позицию

std::cout << "5. Delete item from pos N\n"; //Удаление с позиции

std::cout << "6. Count figures with area less then area\n"; //Подсчет, сколько фигур имеют площадь, меньше заданной

std::cout << "0. Exit\n"; //Выход из программы

}

int main() {

stack<hexagon<double>, myAllocator<hexagon<double>, 100>> st;

int action = 0;

menu();

std::cout << "Select menu item: ";

std::cin >> action;

while (action > 0 && action < 7) {

if (action == 1) { //Вставка в стек

std::cout << "Enter figure cords: ";

hexagon<double> temp;

std::cin >> temp;

st.push(temp);

}

if (action == 2) { //Удаление из стека

try {

st.pop();

}

catch (std::exception& ex) {

std::cout << "Exception with pop(): " << ex.what() << "\n";

}

}

if (action == 3) { //Печать стека

std::for\_each(st.begin(), st.end(), [](hexagon<double>& p) { std::cout << p << "\n"; });

}

if (action == 4) { //Вставка на позицию

int k;

std::cout << "Enter position: ";

std::cin >> k;

try {

auto it = st.at(k);

std::cout << "Enter figure cords: ";

hexagon<double> temp;

std::cin >> temp;

st.insert(it, temp);

}

catch (std::exception& ex) {

std::cout << "Exception with insert(): " << ex.what() << "\n";

}

}

if (action == 5) { //Удаление с позиции

int k;

std::cout << "Enter position: ";

std::cin >> k;

try {

auto it = st.at(k);

st.erase(it);

}

catch (std::exception& ex) {

std::cout << "Exception with erase(): " << ex.what() << "\n";

}

}

if (action == 6) { //Подсчет, сколько фигур имеют площадь, меньше заданной

double k;

std::cout << "Enter area: ";

std::cin >> k;

int ans = std::count\_if(st.begin(), st.end(), [k](hexagon<double>& pd) -> bool { return pd.area() < k; });

std::cout << "Result = " << ans << "\n";

}

std::cout << "Select menu item: ";

std::cin >> action;

}

return 0;

}

**6. Выводы**

В результате выполнения данной лабораторной работы я получила навыки реализации шаблонных контейнеров, а также научилась работать с умными указателями, аллокаторами и создавать свои итераторы.

**Список литературы**

1. Стандартные библиотеки шаблонов/Электронный ресурс/Режим доступа: <https://tproger.ru/articles/stl-cpp/> (дата обращения: 20.11.2020).
2. Справочник по языку С++/Электронный ресурс/Режим доступа: <https://en.cppreference.com/w/> (дата обращения: 22.11.2020).
3. C++ reference - std::iterator\_traits/Электронный ресурс/Режим доступа: [std::iterator\_traits - cppreference.com](https://en.cppreference.com/w/cpp/iterator/iterator_traits) (дата обращения: 22.11.2020).
4. Аллокаторы С++/Электронный ресурс/Режим доступа: <https://ru.cppreference.com/w/cpp/memory/allocator> (дата обращения: 30.11.2020).