**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: Терво Александр Александрович

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. Постановка задачи

Создать шаблон динамической коллекции, согласно варианту задания:

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

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

3. Коллекция должна содержать метод доступа: 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. Динамическая коллекция должна соответствовать варианту задания (Динамический массив, Список, Стек, Очередь);

5. Коллекция должна использовать аллокатор для выделения иосвобождения памяти для своих элементов;

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

7. Реализовать программу, которая:

• позволяет вводить с клавиатуры фигуры (с типом int в качествепараметра шаблона фигуры) и добавлять в коллекцию;

• позволяет удалять элемент из коллекции по номеру элемента

• выводит на экран введённые фигуры с помощью std::for\_each

Вариант 1. Треугольник, стек, динамический массив.

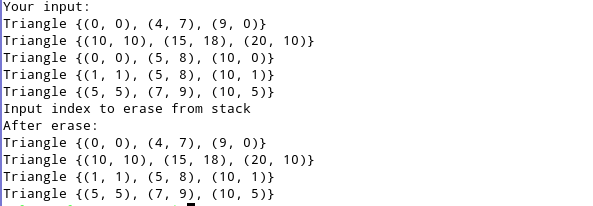
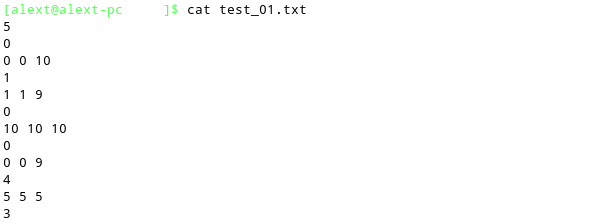
1. Описание программы

Будем использовать шаблонный класс треугольника и стека из предыдущей лабораторной работы. В классе стека добавим ещё один шаблонный аргумент изаменим всю работу с new и delete на работу с алокатором. Так как стекреализован с использованием std::shared\_ptr, то для корректного обращения спамятью необходимо создать отдельный класс-интерфейс deleter, которыйнужно передать в конструктор умного указателя. Реализуем класс линейногоаллокатора, который выделяет блок фиксированного размера и затемвозвращает указатели на свободные блоки. Освобождать память такойаллокатор не может, поэтому метод deallocate ничего не делает. Вместо этогодеструктор аллокатора освобождает весь выделенный блок памяти. Аллокаторне может вызывать конструктор для элементов, поэтому используем методconstruct для инициализации динамических объектов.

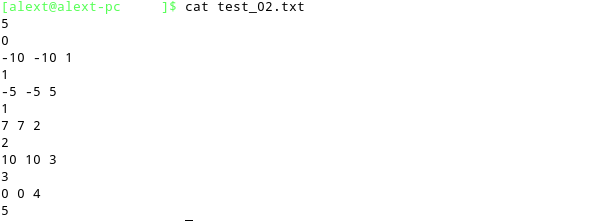
1. Набор тестов и результаты их выполнения

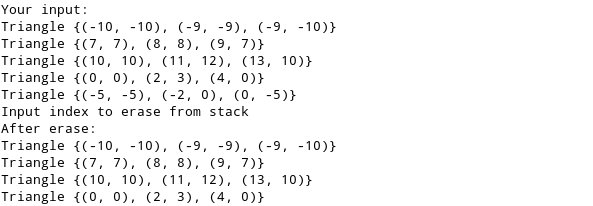
Программа принимает на вход положительное число N – количество фигур в стеке. В следующих N строках следует описание каждого треугольника: координаты левого нижнего угла и длина стороны квадрата. После этих строк программа принимает на вход индекс элемента для удаления из стека.

Тест №1

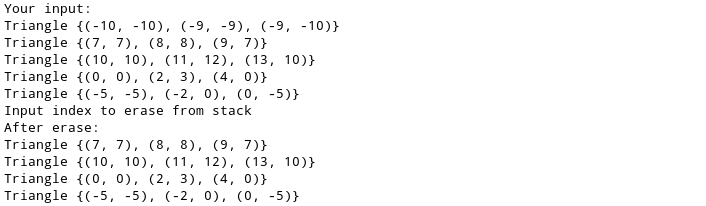
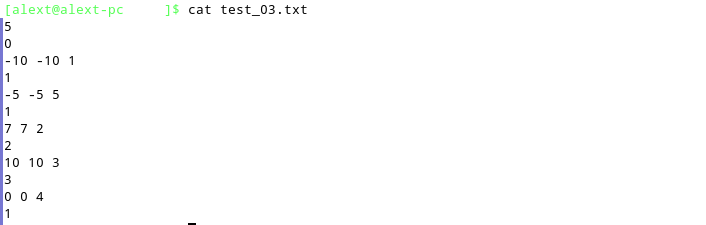


Тест №2

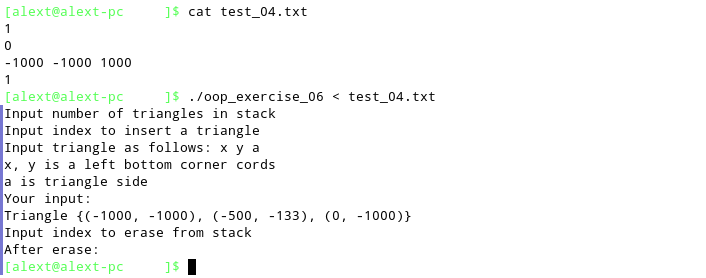




Тест №3



Тест №4



1. Листинг программы

**allocator.hpp:**

#ifndef ALLOCATOR\_HPP

#define ALLOCATOR\_HPP

#include <iostream>

#include <vector>

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

class allocator {

private:

std::vector<T\*> vec;

T\* buffer = nullptr;

public:

using allocator\_type = allocator;

using value\_type = T;

using pointer = T\*;

using reference = T&;

using const\_reference = const T&;

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

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

//std::cout << "WARNING: n = " << n << std::endl;

if (buffer == nullptr) {

buffer = new T[BLOCK\_SIZE];

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

vec.push\_back(&buffer[i]);

}

}

//std::cout << "WARNING: vec.size() = " << vec.size() << std::endl;

if (vec.size() < n) {

throw(std::bad\_alloc());

} else {

T\* p = vec.front();

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

vec.erase(vec.begin());

}

return p;

}

}

template<class OTHER\_T, class... ARGS>

void construct(OTHER\_T\* p, ARGS... arguments) {

\*p = OTHER\_T(std::forward<ARGS>(arguments)...);

}

void deallocate(pointer, std::size\_t) {

;

}

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

buffer = new T[BLOCK\_SIZE];

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

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

vec.push\_back(&buffer[i]);

}

}

allocator() : vec(), buffer(nullptr) {

static\_assert(BLOCK\_SIZE > 0);

}

template <class OTHER\_T>

class rebind {

public:

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

};

~allocator() {

delete[] buffer;

}

};

#endif

**stack.hpp:**

#ifndef STACK\_HPP

#define STACK\_HPP

#include <iostream>

#include <memory>

template <class T, class ALLOCATOR>

class stack\_t {

private:

struct stack\_node\_t;

using allocator\_type = typename ALLOCATOR::template rebind<stack\_node\_t>::other;

struct deleter {

allocator\_type stack\_node\_deleter;

deleter() : stack\_node\_deleter() {};

deleter(allocator\_type\* another\_deleter) : stack\_node\_deleter(another\_deleter) {}

void operator() (void\* ptr) {

stack\_node\_deleter.deallocate((stack\_node\_t\*)ptr, 1);

}

};

struct stack\_node\_t {

T data;

std::shared\_ptr<stack\_node\_t> next;

stack\_node\_t() noexcept : data(), next(nullptr) {};

explicit stack\_node\_t(const T & elem) noexcept : data(elem), next(nullptr) {};

friend bool operator != (const stack\_node\_t & lhs, const stack\_node\_t & rhs) {

return &lhs.data != &rhs.data;

}

friend bool operator==(const stack\_node\_t & lhs, const stack\_node\_t & rhs) {

return &lhs.data == &rhs.data;

}

friend std::ostream & operator << (std::ostream & out, const stack\_node\_t & node) {

out << node.data;

return out;

}

};

public:

class iterator {

private:

std::shared\_ptr<stack\_node\_t> ptr;

public:

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

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

using value\_type = T;

using pointer = T\*;

using reference = T&;

using const\_reference = const T&;

iterator() : ptr(nullptr) {}

iterator(const std::shared\_ptr<stack\_node\_t> & another\_ptr) : ptr(another\_ptr) {}

bool is\_null() {

return ptr == nullptr;

}

void unvalidate() {

ptr = nullptr;

}

iterator & operator ++ () {

if (this->ptr != nullptr) {

this->ptr = this->ptr->next;

return \*this;

} else {

throw(std::runtime\_error("Iterator points to nullptr!"));

}

}

bool operator != (const iterator & other\_iterator) {

return &other\_iterator.ptr->data != &this->ptr->data;

}

friend std::ostream & operator << (std::ostream & out, const iterator & it) {

out << \*(it.ptr);

return out;

}

stack\_node\_t & operator \* () {

return \*ptr;

}

};

private:

std::shared\_ptr<stack\_node\_t> top\_node;

deleter stack\_deleter;

public:

stack\_t() noexcept : top\_node() {};

iterator begin() {

return iterator(top\_node);

}

iterator end() {

return iterator(nullptr);

}

void pop() {

if (top\_node) {

top\_node = top\_node->next;

} else {

throw(std::runtime\_error("Stack is empty!"));

}

}

void push(const T & elem) {

//std::cout << "WARNING: sizeof(stack\_node\_t) = " << sizeof(stack\_node\_t) << std::endl;

stack\_node\_t\* new\_node = stack\_deleter.stack\_node\_deleter.allocate(sizeof(stack\_node\_t));

stack\_deleter.stack\_node\_deleter.construct(new\_node, elem);

std::shared\_ptr<stack\_node\_t> new\_node\_shared(new\_node, stack\_deleter);

new\_node\_shared->next = top\_node;

top\_node = new\_node\_shared;

}

T top() {

if (top\_node) {

return top\_node->data;

} else {

throw(std::runtime\_error("Stack is empty!"));

}

}

void erase(iterator it) {

if (it.is\_null()) {

throw(std::runtime\_error("Iterator points to nullptr!"));

} else {

if (\*it == \*top\_node) {

top\_node = top\_node->next;

} else {

std::shared\_ptr<stack\_node\_t> prev\_node = top\_node;

while (\*prev\_node->next != \*it) {

prev\_node = prev\_node->next;

}

prev\_node->next = prev\_node->next->next;

(\*it).next = nullptr;

}

it.unvalidate();

}

}

void insert(iterator it, const T & elem) {

//std::cout << "WARNING: sizeof(stack\_node\_t) = " << sizeof(stack\_node\_t) << std::endl;

stack\_node\_t\* new\_node = stack\_deleter.stack\_node\_deleter.allocate(sizeof(stack\_node\_t));

stack\_deleter.stack\_node\_deleter.construct(new\_node, elem);

std::shared\_ptr<stack\_node\_t> new\_node\_shared(new\_node, stack\_deleter);

if (top\_node) {

if (\*it == \*top\_node) {

new\_node\_shared->next = top\_node;

top\_node = new\_node\_shared;

it.unvalidate();

return;

}

std::shared\_ptr<stack\_node\_t> prev\_node = top\_node;

while (\*prev\_node->next != \*it) {

prev\_node = prev\_node->next;

}

if (it.is\_null()) {

prev\_node->next = new\_node\_shared;

} else {

new\_node\_shared->next = prev\_node->next;

prev\_node->next = new\_node\_shared;

// std::swap(prev\_node->data, prev\_node->next->data);

}

} else {

top\_node = new\_node\_shared;

}

it.unvalidate();

}

};

#endif

**triangle.hpp:**

#ifndef TRIANGLE\_HPP

#define TRIANGLE\_HPP

#include <iostream>

#include <tuple>

#include <cmath>

template<class T>

struct TTriangle {

/\* Cords of left bottom corner, side \*/

std::pair<T, T> Cord;

T Side;

TTriangle() : Cord(), Side() {}

TTriangle(const std::pair<T, T> & cord, T side) : Cord(cord), Side(side) {}

};

template<class T>

T CalcSquare(const TTriangle<T> & Tri) {

return Tri.Side \* Tri.Side \* pow(3, 0.5) / 4;

}

template<class T>

std::ostream & operator << (std::ostream & out, const TTriangle<T> & tri) {

out << "Triangle {";

out << std::pair<T, T>(tri.Cord.first, tri.Cord.second) << ", ";

out << std::pair<T, T>(tri.Cord.first + 0.5 \* tri.Side, tri.Cord.second + tri.Side \* (std::pow(3, 0.5) / 2)) << ", ";

out << std::pair<T, T>(tri.Cord.first + tri.Side, tri.Cord.second);

out << "}";

return out;

}

template<class T1, class T2>

std::ostream & operator << (std::ostream & out, const std::pair<T1, T2> & p) {

out << "(" << p.first << ", " << p.second << ")";

return out;

}

#endif

**main.cpp:**

#include <algorithm>

#include "allocator.hpp"

#include "stack.hpp"

#include "triangle.hpp"

const std::size\_t BLOCK\_SIZE = 256;

int main() {

auto Print = [](const auto & elem) {

std::cout << elem << std::endl;

};

size\_t n;

std::cout << "Input number of triangles in stack" << std::endl;

std::cin >> n;

int cordX, cordY, side;

stack\_t< TTriangle<int>, allocator< TTriangle<int>, BLOCK\_SIZE > > st;

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

size\_t n;

std::cout << "Input index to insert a triangle" << std::endl;

std::cin >> n;

std::cout << "Input triangle as follows: x y a" << std::endl;

std::cout << "x, y is a left bottom corner cords" << std::endl;

std::cout << "a is triangle side" << std::endl;

std::cin >> cordX >> cordY >> side;

try {

stack\_t< TTriangle<int>, allocator< TTriangle<int>, BLOCK\_SIZE > >::iterator it = st.begin();

while (n--) {

++it;

}

st.insert(it, TTriangle<int>(std::pair<int, int>(cordX, cordY), side));

} catch (std::runtime\_error & exception) {

std::cout << exception.what() << std::endl;

}

}

std::cout << "Your input:" << std::endl;

std::for\_each(st.begin(), st.end(), Print);

std::cout << "Input index to erase from stack" << std::endl;

std::cin >> n;

try {

stack\_t< TTriangle<int>, allocator< TTriangle<int>, BLOCK\_SIZE > >::iterator it = st.begin();

while (n > 1) {

++it;

--n;

}

st.erase(it);

} catch (std::runtime\_error & exception) {

std::cout << exception.what() << std::endl;

}

std::cout << "After erase:" << std::endl;

std::for\_each(st.begin(), st.end(), Print);

return 0;

}

5. Выводы

В ходе выполнения лабораторной работы я узнал, что такое аллокатор, какие бывают виды аллокаторов и изучил принципы их работы. Я реализовал свой линейный аллокатор и стек, работающий с ним. Проверил совместимость моего аллокатора со стандартными коллекциями std::set, std::vector и std::list.

6. Список литературы

• std::shared\_ptr<T>::shared\_ptr — cppreference.com

URL: <https://en.cppreference.com/w/cpp/memory/shared_ptr/shared_ptr>

• std::allocator\_traits — cppreference.com

URL: <https://en.cppreference.com/w/cpp/memory/allocator_traits>

• Аллокаторы памяти / Хабр — Habr

URL: <https://habr.com/ru/post/505632/>