**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: Лисин Роман Сергеевич

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

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:

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.

1. **Описание программы**

*Класс rhombus*

Класс rhombus представляет собой структуру для хранения фигур-ромбов. Ромб задаётся координатами центра и длинами диагоналей. Первая введённая диагональ параллельна оси абсцисс, вторая - оси ординат.

В классе переопределены операторы сравнения на равенство и неравенство, а также определены функции считывания и печати ромба.

*Класс stack*

В классе stack реализованы все методы для работы с коллекцией “Стек”. По заданию необходимо уметь вставлять элементы на любую позицию стека и удалять их, поэтому мой стек реализован на основе линейного однонаправленного списка.

В качестве вспомогательной структуры для элемента стека я реализовал класс item, в котором содержится текущий элемент стека и ссылка на следующий элемент. Для ссылок используется std::unique\_ptr. В структуре переопределены операторы присваивания и сравнения элементов стека.

Класс стек хранит в себе указатель head на вершину стека.

Внутри класса “Стек” реализован класс iterator. Для итератора переопределены все необходимые операторы (разыменование, сравнение, инкремент) для его совместимости со стандартными алгоритмами. Также в классе описаны все поля, необходимые для корректной работы итератора (iterator\_traits). Итератор использует умные указатели std::shared\_ptr.

В стеке реализованы следующие методы:

● T& top() - возвращает элемент на вершине стека. Если стек пустой, то генерируется исключение.

● void pop() - удаление вершины стека. Если стек пустой, то генерируется исключение.

● void push(T) - добавление элемента на вершину стека.

● iterator begin() const, iterator end() const - возвращает итераторы на начало и конец стека соответственно.

● void insert(iterator&, T) - вставляет элемент на позицию итератора.

● void erase(iterator&) - удаление элемента на позиции итератора.

*Класс allocator*

В классе allocator реализованы все методы для работы аллокатора. В классе определены следующие атрибуты: T\* buffer - указатель на буфер памяти, std::vector<T\*> free\_blocks - динамический массив, в котором хранятся указатели на свободные блоки памяти.

Для того чтобы аллокатор был совместим со стандартными структурами данных, в нём определены специальные типы - allocator\_traits, шаблонная функция rebind для изменения аллоцируемого типа данных, функции для явного вызова конструктора и деструктора аллоцируемого типа данных и функции для непосредственного выделения и освобождения памяти.

Функция allocate при первом вызове выделяет память для буфера и заполняет вектор свободными блоками. Так как вектор позволяет быстро удалять элементы из своего конца, то блоки тоже будут выделяться с конца. Если свободных блоков не останется, то будет сгенерировано исключение. Функция deallocate добавляет освободившийся блок в вектор.

Класс allocator реализован по технике “Singleton”. В программе создаётся только один объект класса, который удаляется только после завершения работы программы.

*main.cpp*

В функции main реализован интерфейс для взаимодействия с пользователем согласно заданию.

Печать всех фигур реализована при помощи стандартной функции std::for\_each. Для упрощения реализации использовалось лямбда-выражение.

1. **Набор тестов**

Тест

1 0 0 2 5 // добавить ромб с центром (0,0) и диагоналями 2 и 5

1 -1 1 10 2 // добавить ромб с центром (-1, 1) и диагоналями 10 и 2

1 50 -100 25 45 // добавить ромб с центром (50, -100) и диагоналями 25 и 45

3 // посмотреть вершину стека

6 // печать всех фигур

2 // удалить элемент из вершины стека

3 // посмотреть вершину стека

6 // печать всех фигур

2 // удалить элемент из вершины стека

2 // удалить элемент из вершины стека (стек становится пустым)

2 // удалить элемент из вершины стека (должна появиться ошибка)

3 // посмотреть вершину стека (стек пустой -> будет ошибка)

6 // печать всех фигур

0 // завершение работы

Результат

Allocator's constructor

Allocating buffer

Allocating 0x1147008

Pushed

Allocating 0x1146fe8

Pushed

Allocating 0x1146fc8

Pushed

Top: Rhombus {(37.5; -100), (50; -77.5), (62.5; -100), (50; -122.5)}

Rhombus {(37.5; -100), (50; -77.5), (62.5; -100), (50; -122.5)}

Rhombus {(-6; 1), (-1; 2), (4; 1), (-1; 0)}

Rhombus {(-1; 0), (0; 2.5), (1; 0), (0; -2.5)}

Invalid cmd

Deallocating 0x1146fc8

Popped

Deallocating 0x1147008

Deallocating 0x1146fe8

Allocator's destructor

Process finished with exit code 0

Тест

1 10 10 3 1 // добавить ромб с центром (10, 10) и диагоналями 3 и 1

1 2 4 20 9 // добавить ромб с центром (2, 4) и диагоналями 20 и 9

1 -10 -5 5 10 // добавить ромб с центром (-10, -5) и диагоналями 5 и 10

6 // печать всех фигур

4 1 4 2 2 2 // добавить ромб с центром (1, 4) и диагоналями 2 и 2 на позицию 2

6 // печать всех фигур

5 3 // удалить ромб с 3 позиции

6 // печать всех фигур

4 0 0 10 15 0 // добавить ромб с центром (0,0) и диагоналями 10 и 15 на позицию 0

6 // печать всех фигур

5 2 // удалить ромб с 2 позиции

5 0 // удалить ромб с 0 позиции

5 0 // удалить ромб с 0 позиции

5 1000 // удалить ромб с 1000 позиции (должна быть ошибка)

6 // печать всех фигур

0 // завершение работы

Результат

Allocator's constructor

Allocating buffer

Allocating 0xfd7008

Pushed

Allocating 0xfd6fe8

Pushed

Allocating 0xfd6fc8

Pushed

Rhombus {(-12.5; -5), (-10; 0), (-7.5; -5), (-10; -10)}

Rhombus {(-8; 4), (2; 8.5), (12; 4), (2; -0.5)}

Rhombus {(8.5; 10), (10; 10.5), (11.5; 10), (10; 9.5)}

Allocating 0xfd6fa8

Inserted

Rhombus {(-12.5; -5), (-10; 0), (-7.5; -5), (-10; -10)}

Rhombus {(-8; 4), (2; 8.5), (12; 4), (2; -0.5)}

Rhombus {(0; 4), (1; 5), (2; 4), (1; 3)}

Rhombus {(8.5; 10), (10; 10.5), (11.5; 10), (10; 9.5)}

Deallocating 0xfd6fa8

Erased

Rhombus {(-12.5; -5), (-10; 0), (-7.5; -5), (-10; -10)}

Rhombus {(-8; 4), (2; 8.5), (12; 4), (2; -0.5)}

Rhombus {(0; 4), (1; 5), (2; 4), (1; 3)}

Allocating 0xfd6fa8

Inserted

Rhombus {(-5; 0), (0; 7.5), (5; 0), (0; -7.5)}

Rhombus {(-12.5; -5), (-10; 0), (-7.5; -5), (-10; -10)}

Rhombus {(-8; 4), (2; 8.5), (12; 4), (2; -0.5)}

Rhombus {(0; 4), (1; 5), (2; 4), (1; 3)}

Deallocating 0xfd7008

Erased

Deallocating 0xfd6fa8

Erased

Deallocating 0xfd6fc8

Erased

Invalid position

Rhombus {(0; 4), (1; 5), (2; 4), (1; 3)}

Deallocating 0xfd6fe8

Allocator's destructor

Process finished with exit code 0

1. **Листинг программы**

***rhombus.h***

#ifndef OOP\_LAB5\_RHOMBUS\_H

#define OOP\_LAB5\_RHOMBUS\_H

#include <iostream>

template<class T>

class rhombus {

private:

std::pair<T, T> center;

double diagonal1;

double diagonal2;

public:

rhombus() : diagonal1(0), diagonal2(0) {

center = std::make\_pair(0, 0);

}

rhombus(T x\_center, T y\_center, double diag1, double diag2) : diagonal1(diag1), diagonal2(diag2) {

center = std::make\_pair(x\_center, y\_center);

}

bool operator==(rhombus<T> &other) {

if (center != other.center) {

return false;

}

if (diagonal1 != other.diagonal1) {

return false;

}

if (diagonal2 != other.diagonal2) {

return false;

}

return true;

}

bool operator!=(rhombus<T> &other) {

return !(\*this == other);

}

template<class T1>

friend std::ostream &operator<<(std::ostream &out, rhombus<T1> &r);

template<class T1>

friend std::istream &operator>>(std::istream &in, rhombus<T1> &r);

};

template<class T>

std::istream &operator>>(std::istream &in, rhombus<T> &r) {

in >> r.center.first >> r.center.second >> r.diagonal1 >> r.diagonal2;

}

template<class T>

std::ostream &operator<<(std::ostream &out, rhombus<T> &r) {

out << "Rhombus {(" << r.center.first - r.diagonal1 \* 0.5 << "; " << r.center.second << "), (";

out << r.center.first << "; " << r.center.second + r.diagonal2 \* 0.5 << "), (";

out << r.center.first + r.diagonal1 \* 0.5 << "; " << r.center.second << "), (";

out << r.center.first << "; " << r.center.second - r.diagonal2 \* 0.5 << ")}";

}

#endif //OOP\_LAB5\_RHOMBUS\_H

***stack.h***

#ifndef OOP\_LAB5\_STACK\_H

#define OOP\_LAB5\_STACK\_H

#include <memory>

#include <exception>

template<class T,class ALLOCATOR>

class stack {

private:

struct item {

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

T value;

std::unique\_ptr<item> next = nullptr;

item() = default;

item(T val) : value(val) {}

item &operator=(item const &other) noexcept {

value = other.value;

next = std::move(other.next);

return \*this;

}

bool operator!=(item &other) {

return &value != &other.value;

}

// singleton allocator

static allocator\_type& get\_allocator() {

static allocator\_type allocator;

return allocator;

}

void\* operator new(size\_t size) {

return get\_allocator().allocate();

}

void operator delete(void\* p) {

get\_allocator().destroy((item\*)p);

get\_allocator().deallocate((item\*)p);

}

};

std::unique\_ptr<item> head;

public:

class iterator {

private:

item\* ptr;

friend class stack;

public:

// iterator traits

using difference\_type = ptrdiff\_t;

using value\_type = T;

using reference = T &;

using pointer = T\*;

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

iterator() {

ptr = nullptr;

}

iterator(item\* node) {

ptr = node;

}

iterator &operator++() {

if (ptr != nullptr) {

ptr = ptr->next.get();

}

else {

ptr = nullptr;

}

return \*this;

}

reference operator\*() {

return ptr->value;

}

pointer operator->() {

return &(ptr->value);

}

iterator& operator=(iterator const& other) {

ptr = other.ptr;

}

bool operator!=(iterator &other) {

return ptr != other.ptr;

}

bool operator!=(iterator &&other) {

return ptr != other.ptr;

}

bool operator==(iterator &other) {

return ptr == other.ptr;

}

bool operator==(iterator &&other) {

return ptr == other.ptr;

}

};

T top() {

if (head) {

return head->value;

}

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

}

void pop() {

if (head) {

std::unique\_ptr<item> tmp = std::move(head);

head = std::move(tmp->next);

} else {

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

}

}

void push(T val) {

std::unique\_ptr<item> new\_head = std::make\_unique<item>(val);

if (head) {

new\_head->next = std::move(head);

head = std::move(new\_head);

}

else {

head = std::move(new\_head);

}

}

// iterator to the top

iterator begin() const {

if (head == nullptr) {

return nullptr;

}

return head.get();

}

// iterator to the last element

iterator end() const {

return iterator();

}

// insert to iterator's pos

void insert(iterator &it, T val) {

if (it == begin()) {

push(val);

}

else {

std::unique\_ptr<item> new\_node = std::make\_unique<item>(\*it);

new\_node->next = std::move(it.ptr->next);

\*it = val;

it.ptr->next = std::move(new\_node);

}

}

// erase from iterator's pos

void erase(iterator &it) {

if (it == begin()) {

pop();

}

else if (it.ptr->next == nullptr) {

auto iter = begin();

while (iter.ptr->next->next != nullptr) {

++iter;

}

iter.ptr->next = nullptr;

}

else {

\*it = it.ptr->next->value;

it.ptr->next = std::move(it.ptr->next->next);

}

}

};

#endif //OOP\_LAB5\_STACK\_H

***allocator.h***

#ifndef OOP\_LAB6\_ALLOCATOR\_H

#define OOP\_LAB6\_ALLOCATOR\_H

#include <vector>

template<class T, size\_t BLOCKS\_AMOUNT>

class allocator {

private:

T\* buffer;

std::vector<T\*> free\_blocks;

public:

// allocator traits

using value\_type = T;

using pointer = T \*;

using const\_pointer = const T \*;

using size\_type = size\_t;

allocator() noexcept {

std::cout << "Allocator's constructor" << std::endl;

buffer = nullptr;

}

~allocator() {

std::cout << "Allocator's destructor" << std::endl;

free(buffer);

}

// allocator type conversion

template<class U>

struct rebind {

using other = allocator<U, BLOCKS\_AMOUNT>;

};

pointer allocate() {

// at first we should allocate memory for buffer

if (!buffer) {

std::cout << "Allocating buffer" << std::endl;

buffer = (pointer)malloc(BLOCKS\_AMOUNT \* sizeof(T));

for (int i = 0; i < BLOCKS\_AMOUNT; ++i) {

free\_blocks.push\_back(buffer + i);

}

}

if (free\_blocks.empty()) {

throw std::bad\_alloc();

}

pointer p = free\_blocks[free\_blocks.size() - 1];

free\_blocks.pop\_back();

std::cout << "Allocating " << p << std::endl;

return p;

}

void deallocate(pointer p) {

std::cout << "Deallocating " << p << std::endl;

free\_blocks.push\_back(p);

}

template<typename U, typename... Args>

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

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

}

void destroy(pointer p) {

p->~T();

}

};

***main.cpp***

#include <iostream>

#include <algorithm>

#include "rhombus.h"

#include "stack.h"

#include "allocator.h"

/\*Лабораторная работа №6

Вариант 5: стек с ромбами, аллокатор на динамическом массиве

Выполнил: Лисин Р.С., группа М8О-206Б-20

\*/

void print\_menu() {

std::cout << "1. Push rhombus to stack" << std::endl;

std::cout << "2. Pop rhombus from the stack" << std::endl;

std::cout << "3. Check the element on the top of the stack" << std::endl;

std::cout << "4. Insert rhombus to the position" << std::endl;

std::cout << "5. Erase rhombus from the position" << std::endl;

std::cout << "6. Print all figures" << std::endl;

std::cout << "0. Exit" << std::endl;

std::cout << std::endl;

std::cout << "(to add a rhombus type the in coords of the center and lengths of diagonals)" << std::endl;

std::cout << std::endl;

}

int main() {

stack<rhombus<int>, allocator<rhombus<int>, 10>> s;

print\_menu();

char cmd;

while (std::cin >> cmd) {

if (cmd == '1') {

try {

rhombus<int> r;

std::cin >> r;

s.push(r);

std::cout << "Pushed" << std::endl;

}

catch (std::exception& ex) {

std::cout << "Not enough memory: " << ex.what() << std::endl;

}

}

else if (cmd == '2') {

try {

s.pop();

std::cout << "Popped" << std::endl;

}

catch (std::exception& ex) {

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

}

}

else if (cmd == '3') {

try {

auto t = s.top();

std::cout << "Top: " << t << std::endl;

}

catch (std::exception& ex) {

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

}

}

else if (cmd == '4') {

rhombus<int> r;

std::cin >> r;

unsigned int pos;

std::cin >> pos;

auto iter = s.begin();

try {

if (iter == s.end() && pos != 0) {

throw std::runtime\_error("Invalid position");

}

for (unsigned int i = 0; i < pos; ++i) {

++iter;

if (iter == s.end() && i != pos - 1) {

throw std::runtime\_error("Invalid position");

}

}

s.insert(iter, r);

std::cout << "Inserted" << std::endl;

}

catch (std::runtime\_error& ex) {

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

}

catch (std::bad\_alloc& ex) {

std::cout << "Not enough memory: " << ex.what() << std::endl;

}

}

else if (cmd == '5') {

unsigned int pos;

std::cin >> pos;

auto iter = s.begin();

try {

if (iter == s.end()) {

throw std::runtime\_error("Invalid position");

}

for (unsigned int i = 0; i < pos; ++i) {

++iter;

if (iter == s.end() && i != pos) {

throw std::runtime\_error("Invalid position");

}

}

s.erase(iter);

std::cout << "Erased" << std::endl;

}

catch (std::exception& ex) {

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

}

}

else if (cmd == '6') {

std::for\_each(s.begin(), s.end(), [](rhombus<int> r) {

std::cout << r << std::endl;

});

}

else if (cmd == '0') {

break;

}

else {

std::cout << "Invalid cmd" << std::endl;

}

}

}

**ЛИТЕРАТУРА**

1. Аллокаторы памяти[Электронный ресурс].

URL: https://habr.com/ru/post/505632/ (дата обращения: 03.12.2021)