**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: Ивенкова Любовь Васильевна

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

**Содержание**

[**1. Постановка задачи**](#_q3xiyr9tk7jn)3

[**2. Описание программы**](#_fsgxdsto112)3

[**3. Набор тестов**](#_2bvtr239grk9)5

[**4. Результат выполнения тестов**](#_zh5gzj573xha)5

[**5. Листинг программы**](#_7vl5juvf0yhv)7

[**6. Вывод**](#_s13bec61cwdb)14

[**Список используемых источников**](#_y4bprx2szmjz)14

# 

# **1. Постановка задачи**

**Вариант:** 2

**Задача:**

Создать шаблонный класс квадрата с публичными полями. Создать шаблонную коллекцию “стек”, который реализован с помощью умных указателей. В качестве параметров в него должны поступать фигуры. Реализовать forward iterator по стеку, а так же begin(), end(), pop(), push(), top(). Реализовать метод вставки на позицию итератора inrase и метод удаления из позиции итератора erase. Реализовать аллокатор, выделяющий фиксированный размер памяти. Внутри него должны храниться указатель на выделенную память и список, хранящий указатели на свободные блоки. Стек должен использовать аллокатор для выделения и освобождения памяти под свои элементы.

Аллокатор должен быть совместим с контейнерами std::map и std::list.

Реализованная программа должна:

* позволять добавлять фигуры в стек;
* позволять удалять фигуры из стека;
* выводить все введенные фигуры с помощью std::for\_each.

# **2. Описание программы**

Программа принимает в себя данные из консоли и из файла, при перенаправлении потока ввода вывода, и выполняет заданные действия. При запуске появляется меню выбора операций, после выбора которой вводятся данные. В список операций входит *добавление фигуры* *в стек*, *удаление фигуры из стека*, *вывод всех координат фигур в стеке, вывод количества фигур, площадь которых меньше заданной*.

Программа состоит из четырех файлов: main.cpp, stack.h, allocator.h и figure.h:

* figure.h - описание классов фигур. Они представлены шаблонными классами, в которых присутствуют лишь публичные поля. Функция get\_coords передает в класс координаты на основе стороны, которую ввел пользователь.
* allocator.h - реализация аллокатора. Содержит конструктор и деструктор аллокатора.

Структура rebind служит для получения из аллокатора одного типа аллокатор другого типа.

T\* allocate() - возвращает указатель на свободную ячейку памяти.

void deallocate(T\* ptr, long long n = 1) - добавляет ячейку памяти в список свободных ячеек.

* stack.h - реализация стека с помощью умных указателей.

Класс Node - элемент стека. Он содержит фигуру и указатели на предыдущий и следующий элементы. Также содержит перегрузку операторов new и delete для использования аллокатора в дальнейшем.

Класс Forward\_iterator - реализация итератора, который может совместим со стандартными алгоритмами. Содержит перегрузку операторов: ++, ->, \*, ==, !=. Является дружественным классом для класса Stack.

Класс Stack. Содержит указатели на первый и последний элементы стека, а также его размер.

void Push(T& elem) - добавление нового элемента elem в конец стека.

void Insert(Forward\_iterator iter, T& obj) - добавление элемента obj на позицию итератора iter.

void Erase(Forward\_iterator iter) - удаление элемента из позиции итератора iter.

bool Empty() - проверка стека на наличие элементов в нем.

void Pop() - удаление последнего элемента стека.

Forward\_iterator Top() - возвращает верхний элемент стека.

Forward\_iterator Begin() - возвращает первый элемент стека.

Forward\_iterator End() - возвращает последний элемент стека (NULL).

* main.cpp - главный файл. В нем создается стек. Считывается выбор пользователя, исходя из которого выбираются дальнейшие действия.

void menu() - вывод меню.

void error() - вывод сообщения об ошибке.

**Переменные классов**

class Figure

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

coords a, b, c, d; - координаты квадрата.

class Stack

std::shared\_ptr<Node> head, tail; - указатели на верхний и нижний(первый) элементы.

long long size; - размер стека.

class Forward\_iterator

std::shared\_ptr<Node> node; - указатель на фигуру.

class Node

std::shared\_ptr<Node> next, previous; - указатель на следующий и предыдущий элементы.

T value; - фигура.

class TAllocator

std::list<T\*> Free\_blocks - список указателей на свободные ячейки памяти;

T\* Used\_blocks - указатель на начало выделенной памяти.

# **3. Набор тестов**

Таблица 1. Тест 1

|  |  |
| --- | --- |
| 1 5 2 3  1 3 1  1 5 2 1  3  2 1  3  0 | -добавить на 2 позицию стека новый квадрат со стороной 5 (ошибка: такого итератора в стеке нет);  -добавить в конец стека новый квадрат со стороной 3;  -добавить новый квадрат со стороной 5 на 1 позицию в стеке;  -вывести все фигуры в стеке (выведет квадраты со сторонами 3 и 6);  -удалить 1 элемент стека;  -вывести все фигуры в стеке (выведет квадрат со стороной 6);  -выход из программы |

Таблица 2. Тест 2

|  |  |
| --- | --- |
| 2 1  1 4 1  1 5 1  1 7 1  3  4 10  4 26  0 | -удалить 1 элемент стека (ошибка: такого итератора в стеке нет);  -добавить в конец стека новый квадрат со стороной 4;  -добавить в конец стека новый квадрат со стороной 5;  -добавить в конец стека новый квадрат со стороной 7;  -вывести все фигуры в стеке (выведет квадраты со сторонами 4, 5 и 7);  -вывести количество фигур, площадь которых меньше 10  -вывести количество фигур, площадь которых меньше 26  -выход из программы |

# **4. Результат выполнения тестов**

Тест 1:

Enter 0-4 to:

1 - add square to stack

2 - delete figure from stack

3 - print all coordinates of figures in stack

4 - print the number of figures with an area less than...

0 - exit

Your choice: 1

Enter the side of the square: 5

Insert at the end of the stack [1] or at a specific position [2]? 2

Enter the position number on the stack 3

There is no such iterator

What's next?

>>1

Enter the side of the square: 3

Insert at the end of the stack [1] or at a specific position [2]? 1

What's next?

>>1

Enter the side of the square: 5

Insert at the end of the stack [1] or at a specific position [2]? 2

Enter the position number on the stack 1

What's next?

>>3

(0, 0), (5, 0), (5, 5), (0, 5)

(0, 0), (3, 0), (3, 3), (0, 3)

What's next?

>>2

Enter the position number on the stack 1

What's next?

>>3

(0, 0), (3, 0), (3, 3), (0, 3)

What's next?

>>0

The program is closed, goodbye!

Тест 2:

Enter 0-4 to:

1 - add square to stack

2 - delete figure from stack

3 - print all coordinates of figures in stack

4 - print the number of figures with an area less than...

0 - exit

Your choice: 2

Enter the position number on the stack 1

There is no such iterator

What's next?

>>1

Enter the side of the square: 4

Insert at the end of the stack [1] or at a specific position [2]? 1

What's next?

>>1

Enter the side of the square: 5

Insert at the end of the stack [1] or at a specific position [2]? 1

What's next?

>>1

Enter the side of the square: 7

Insert at the end of the stack [1] or at a specific position [2]? 1

What's next?

>>3

(0, 0), (4, 0), (4, 4), (0, 4)

(0, 0), (5, 0), (5, 5), (0, 5)

(0, 0), (7, 0), (7, 7), (0, 7)

What's next?

>>4

Enter area: 10

The number of figures with an area less than 10 is 0

What's next?

>>4

Enter area: 26

The number of figures with an area less than 26 is 2

What's next?

>>0

The program is closed, goodbye!

# **5. Листинг программы**

main.cpp

/\* Ивенкова Любовь Васильевна, М8О-208Б-19

https://github.com/Li-Iven/OOP/tree/main/oop\_exercise\_06

\*/

#include <iostream>

#include "stack.h"

#include "figure.h"

#include "allocator.h"

#include <algorithm>

//allocator - list

//collection - stack

void menu() {

std::cout << "Enter 0-4 to:" << std::endl;

std::cout << "1 - add square to stack" << std::endl;

std::cout << "2 - delete figure from stack" << std::endl;

std::cout << "3 - print all coordinates of figures in stack" << std::endl;

std::cout << "4 - print the number of figures with an area less than..." << std::endl;

std::cout << "0 - exit" << std::endl;

}

void error() {

std::cout << "Incorrect value!" << std::endl;

}

int main()

{

Stack<Figure<int>> stack;

int a, b, choice;

long double area;

Figure<int> sq;

menu();

std::cout << "Your choice: ";

do {

std::cin >> choice;

switch (choice) {

case 1:

std::cout << "Enter the side of the square: ";

std::cin >> a;

sq = Figure<int>(a);

std::cout << "Insert at the end of the stack [1] or at a specific position [2]? ";

std::cin >> choice;

if (choice == 1) {

stack.Push(sq);

}

else if (choice == 2) {

std::cout << "Enter the position number on the stack ";

try {

std::cin >> b;

if (stack.Size() == 0) {

b--;

if (b<0 || b>stack.Size()) {

throw b;

}

}

}

catch (const int a) {

std::cout << "There is no such iterator\n";

break;

}

Stack<Figure<int>>::Forward\_iterator it = stack.Begin();

int i = 1;

if (i != b) {

for (i; i < b; ++i) {

++it;

}

}

stack.Insert(it, sq);

}

else {

error();

}

break;

case 2: {

std::cout << "Enter the position number on the stack ";

std::cin >> b;

try {

if (b<0 || b>stack.Size()) {

throw b;

}

}

catch (const int a) {

std::cout << "There is no such iterator\n";

break;

}

Stack<Figure<int>>::Forward\_iterator it = stack.Begin();

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

++it;

}

stack.Erase(it);

break;

}

case 3: {

auto print = [](Figure<int>& elem) {

std::cout << "(" << elem.a.first << ", " << elem.a.second << "), ";

std::cout << "(" << elem.b.first << ", " << elem.b.second << "), ";

std::cout << "(" << elem.c.first << ", " << elem.c.second << "), ";

std::cout << "(" << elem.d.first << ", " << elem.d.second << ")" << std::endl;

};

std::for\_each(stack.Begin(), stack.End(), print);

break;

}

case 4:

{

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

std::cin >> area;

try {

if (area < 0) {

throw area;

}

}

catch (const long double a) {

std::cout << "You entered negative area\n";

break;

}

long double count = std::count\_if(stack.Begin(), stack.End(), [area](Figure<int>& elem) {return area > elem.d.second \* elem.d.second; });

std::cout << "The number of figures with an area less than " << area << " is " << count << std::endl;

break;

}

case 0:

std::cout << "The program is closed, goodbye!\n";

return false;

break;

default:

std::cout << "Incorrect values! Try again.\n";

break;

}

std::cout << "What's next?\n";

std::cout << ">>";

} while (choice);

}

figure.h

#pragma once

template<typename T>

struct Figure {

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

coords a, b, c, d;

Figure(T a) {

get\_coords(\*this, a);

}

Figure() {}

~Figure() {}

};

template<typename T, typename S>

void get\_coords(T& elem, S side) {

elem.a.first = elem.a.second = elem.d.first = elem.b.second = 0;

elem.b.first = elem.c.first = elem.c.second = elem.d.second = side;

}

stack.h

#pragma once

#include<iostream>

#include<memory>

#include"allocator.h"

template<typename T>

struct Stack {

class Node {

private:

std::shared\_ptr<Node> next, previous;

T value;

public:

Node() : value(), next(nullptr), previous(nullptr) {}

Node(T val) : value(val), next(nullptr), previous(nullptr) {}

~Node() {}

static TAllocator<Node, 20>& get\_allocator() {

static TAllocator<Node, 20> al;

return al;

}

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

void\* point = get\_allocator().allocate();

if (point == nullptr)

throw - 1;

return point;

}

void operator delete(void\* point) {

get\_allocator().deallocate((Node\*)point);

}

friend struct Stack;

};

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

long long size;

Stack() : head(nullptr), tail(nullptr), size(0) {}

~Stack() {}

long long Size() {

return size;

}

class Forward\_iterator {

public:

using value\_type = T;

using reference = T&;

using pointer = T\*;

using difference\_type = ptrdiff\_t;

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

Forward\_iterator(std::shared\_ptr<Node> elem = nullptr) :node(elem) {}

~Forward\_iterator() {}

friend Stack;

bool operator==(const Forward\_iterator& other) const {

return node == other.node;

}

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

return node != other.node;

}

Forward\_iterator operator++() {

node = node->next;

return \*this;

}

Forward\_iterator operator++(int index) {

Forward\_iterator tmp(node);

node = node->next;

return tmp;

}

std::shared\_ptr<Node> operator->() {

return this->node;

}

T& operator\*() {

return this->node->value;

}

private:

std::shared\_ptr<Node> node;

};

void Push(T& elem) {

Node\* l = new Node(elem);

std::shared\_ptr<Node> el(l);

if (size == 0) {

tail = el;

head = tail;

}

else {

el->previous = head;

head->next = el;

head = el;

}

size++;

}

void Insert(Forward\_iterator iter, T& obj) {

if (iter == Begin()) {

if (tail == nullptr) {

Push(obj);

return;

}

else {

std::shared\_ptr<Node> tmp = std::make\_shared<Node>(obj);

tail->previous = tmp;

tmp->next = tail;

tail = tmp;

}

}

else if (iter == End()) {

std::shared\_ptr<Node> tmp = std::make\_shared<Node>(obj);

head->next = tmp;

tmp->previous = head;

head = tmp;

}

else {

std::shared\_ptr<Node> tmp = std::make\_shared<Node>(obj);

tmp->next = iter.node;

iter->previous->next = tmp;

tmp->previous = iter->previous;

iter->previous = tmp;

}

size++;

}

void Erase(Forward\_iterator iter) {

if (iter == tail) {

try {

if (size == 0) {

throw "List is empty\n";

}

}

catch (const char a[14]) {

std::cout << a;

return;

}

if (size == 1) {

head = nullptr;

tail = nullptr;

}

else {

tail->next->previous = nullptr;

tail = tail->next;

}

size--;

return;

}

else if (iter == head) {

std::shared\_ptr<Node> tmp = head->previous;

head->previous->next = nullptr;

head = tmp;

size--;

return;

}

else {

iter->next->previous = iter->previous;

iter->previous->next = iter->next;

size--;

return;

}

}

bool Empty() {

return size == 0;

}

void Pop() {

Erase(head);

}

Forward\_iterator Top() {

return head;

}

Forward\_iterator Begin() {

return tail;

}

Forward\_iterator End() {

return Forward\_iterator{};

}

};

allocator.h

#pragma once

#include<iostream>

#include<list>

template<typename T, std::size\_t Alloc\_Size>

struct TAllocator {

using value\_type = T;

using pointer = T\*;

using const\_pointer = const T\*;

using reference = T&;

using const\_reference = const T&;

TAllocator() : Used\_blocks(nullptr) {}

~TAllocator() {

while (!Free\_blocks.empty()) {

Free\_blocks.pop\_back();

}

delete Used\_blocks;

}

template<typename U>

struct rebind {

using other = TAllocator<U, Alloc\_Size>;

};

T\* allocate() {

if (Used\_blocks == nullptr) {

Used\_blocks = new T[Alloc\_Size];

for (size\_t i = 0; i < Alloc\_Size; ++i)

Free\_blocks.push\_back(Used\_blocks + i);

}

if (!Free\_blocks.empty())

{

T\* point = Free\_blocks.back();

Free\_blocks.pop\_back();

return point;

}

else {

return nullptr;

}

}

void deallocate(T\* ptr, long long n = 1) {

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

Free\_blocks.push\_front(ptr + i);

}

}

std::list<T\*> Free\_blocks;

T\* Used\_blocks;

}

# 

# **6. Вывод**

Благодаря данной лабораторной работе я ознакомилась с аллокаторами, смогла реализовать списочный аллокатор. Мой аллокатор совместим со стандартными коллекциями, например, std::map и std::list.

# **Список используемых источников**

1. std::for\_each [Электронный ресурс]. URL: <https://en.cppreference.com/w/cpp/algorithm/for_each> (Дата обращения: 15.04.2021)
2. Аллокаторы памяти [Электронный ресурс]. URL: <https://habr.com/ru/post/505632/> (Дата обращения: 15.04.2021)
3. Альтернативные аллокаторы памяти [Электронный ресурс]. URL: <https://habr.com/ru/post/274827/> (Дата обращения: 15.04.2021)
4. Forward iterators in C++ [Электронный ресурс]. <https://www.geeksforgeeks.org/forward-iterators-in-cpp/> (Дата обращения: 15.04.2021)