**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: Киреев Александр Константинович

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. **Постановка задачи**

Вариант 12. (Список, Очередь, Трапеция)

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**

В файле main.cpp представлен интерактивный ввод/вывод для работы со списком. В файле trapezoid.hpp написан класс “Трапеция”, который может быть считан из потока, а также при построении присутствует проверка на валидность данных. Также в файле есть метод подсчета площади и перегруженный оператор для вывода трапеции на экран. В файле list.hpp реализован односвязный список, forward\_iterator для него, а также методы для удаления, вставки, доступа к списку через итераторы. Для выделения памяти написан аллокатор, в котором хранится буфер заданного размера и очередь указателей на свободные элементы.

1. **Набор тестов**

Тест 1:

1 0

0 0 2 6 4 6 6 0

1 0

0 0 1 2 3 2 4 0

3

4

7

2 1

3

1 0

0 10 0 0 10 10 10 20

2 0

3

6

Тест 1 проверяет работоспособность программы в целом, а также показывает, что программа правильно обрабатывает случаи, когда ей не подают равнобедренную трапецию.

Тест 2:

1 0

0 0 1 2 3 2 4 0

1 10

2 0

2 0

3

1 0

0 0 2 6 4 6 6 0

asdac

1 1

0 0 0 0 0 0 0 0

4 0

6

В тесте 2 мы проверяем реакцию программы на ошибки: выход за границы, неверные команды, попытка удаления несуществующего элемента.

1. **Результаты выполнения тестов**

Тест 1:

MacBook-Air-K:src AK$ ./lab5 < ../tests/test\_1.txt

Command -- Description

1 [idx] -- Add figure

2 [idx] -- Delete figure by index from vector

3 -- Display vector

4 -- Display count of figures with less area

5 -- Display help

6 -- End program

=========START OF VECTOR=========

[ (0, 0), (1, 2), (3, 2), (4, 0) ]

[ (0, 0), (2, 6), (4, 6), (6, 0) ]

==========END OF VECTOR==========

Count = 1

=========START OF VECTOR=========

[ (0, 0), (1, 2), (3, 2), (4, 0) ]

==========END OF VECTOR==========

=========START OF VECTOR=========

[ (0, 0), (1, 2), (3, 2), (4, 0) ]

==========END OF VECTOR==========

Тест 2:

Cmd: MacBook-Air-K:src./lab5 < ../tests/test\_2.txt

Command -- Description

1 [idx] -- Add figure

2 [idx] -- Delete figure by index from vector

3 -- Display vector

4 -- Display count of figures with less area

5 -- Display help

6 -- End program

ERROR: Out of bounds

Invalid command!

ERROR: Out of bounds

Count = 0

1. **Листинг программы**

**main.cpp**

// 12 вариант

// трапеция, список, очередь

// М8О-206Б-19 Киреев А.К.

#include <iostream>

#include <vector>

#include <sstream>

#include <string>

#include <algorithm>

#include "list.hpp"

#include "trapezoid.hpp"

void print(Trapezoid<double> t) {

if (t.valid) {

std::cout << t;

}

}

// мануал

void help() {

std::cout << "Command -- Description\n" <<

"1 [idx] -- Add figure \n" <<

"2 [idx] -- Delete figure by index from vector\n" <<

"3 -- Display vector\n" <<

"4 -- Display count of figures with less area\n" <<

"5 -- Display help\n" <<

"6 -- End program" << std::endl;

}

void mainLoop() {

int command = 0, idx = 0;

List<Trapezoid<double>, custom\_allocator<Trapezoid<double>, 100>> l;

std::string s;

while (std::cout << "Cmd: " && std::cin >> s) {

if (s.length() > 1) {

std::cout << "Invalid command." << std::endl;

continue;

}

std::stringstream ss(s);

ss >> command;

if (command == 6)

break;

if (command == 1) {

std::cin >> idx;

List<Trapezoid<double>, custom\_allocator<Trapezoid<double>, 100>>::ListIterator it(l, idx);

Trapezoid<double> tmp(std::cin);

try {

l.insert(it, tmp);

} catch (std::logic\_error& e) {

std::cerr << "ERROR: " << e.what() << "\n";

continue;

}

} else if (command == 2) {

std::cin >> idx;

List<Trapezoid<double>, custom\_allocator<Trapezoid<double>, 100>>::ListIterator it(l, idx);

try {

l.erase(it);

} catch (std::logic\_error& e) {

std::cerr << "ERROR: " << e.what() << "\n";

continue;

}

} else if (command == 3) {

std::cout << "=========START OF VECTOR=========" << std::endl;

std::for\_each(l.begin(), l.end(), print);

std::cout << "==========END OF VECTOR==========" << std::endl;

} else if (command == 4) {

std::cout << "Enter max val:\n";

int max\_val;

std::cin >> max\_val;

int ans = std::count\_if(l.begin(), l.end(), [max\_val](Trapezoid<double> t){return t.area() < max\_val;});

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

} else if (command == 5) {

help();

} else {

std::cout << "Invalid command!" << std::endl;

}

}

}

int main() {

help();

mainLoop();

return 0;

}

**list.hpp**

// М8О-206Б-19 Киреев А.К.

#pragma once

#include <memory>

#include <iostream>

#include <exception>

#include <iterator>

#include <queue>

template <class T, size\_t CAP = 10>

struct custom\_allocator {

using value\_type = T;

using pointer = T \*;

using const\_pointer = const T \*;

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

static const size\_t cap = CAP;

std::size\_t size = 0;

pointer buf;

std::size\_t free\_size = 0;

std::queue<pointer> free\_mem;

custom\_allocator() noexcept {}

template <class U, size\_t CU>

struct rebind {

using other = custom\_allocator<U, CU>;

};

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

if (size == 0) {

buf = reinterpret\_cast<T\*>(operator new(CAP \* sizeof(T)));

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

free\_mem.push(&(buf[i]));

free\_size++;

}

size = CAP;

}

T\* to\_return = free\_mem.front();

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

free\_mem.pop();

free\_size--;

}

return to\_return;

}

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

free\_mem.push(p);

free\_size++;

if (free\_size >= CAP) {

::operator delete(buf);

free\_mem.~queue();

size--;

}

}

template <typename U, typename... Args>

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

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

}

void destroy(pointer p){

p->~T();

}

};

template <class T, class Allocator>

class List {

private:

struct ListNode {

using allocator\_type = typename Allocator::template rebind<ListNode, Allocator::cap>::other;

std::unique\_ptr<ListNode> \_next;

T \_val;

T& get(std::size\_t idx);

void insert(ListNode &prev, std::size\_t idx,const T& val);

void erase(ListNode &prev, std::size\_t idx);

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

return get\_allocator().allocate(1);

}

void operator delete(void \*p) {

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

get\_allocator().deallocate((ListNode \*)p, 1);

}

static allocator\_type get\_allocator() {

static allocator\_type allocator;

return allocator;

}

};

std::unique\_ptr<ListNode> \_head;

std::size\_t \_size;

public:

using value\_type = T;

class ListIterator {

private:

List& \_list;

size\_t \_idx;

friend class List;

public:

using difference\_type = int;

using value\_type = List::value\_type;

using reference = List::value\_type&;

using pointer = List::value\_type\*;

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

ListIterator(List& l, std::size\_t idx) : \_list(l), \_idx(idx) {}

ListIterator& operator++();

reference operator\*();

pointer operator->();

bool operator!=(const ListIterator& other);

bool operator==(const ListIterator& other);

};

List<T, Allocator>::ListIterator begin();

List<T, Allocator>::ListIterator end();

ListIterator insert(ListIterator& it, T& val);

ListIterator erase(ListIterator& it);

T& operator[](size\_t idx);

size\_t size();

};

template <class T, class Allocator>

size\_t List<T, Allocator>::size() {

return \_size;

}

template <class T, class Allocator>

typename List<T, Allocator>::ListIterator& List<T, Allocator>::ListIterator::operator++() {

++\_idx;

return \*this;

}

template <class T, class Allocator>

typename List<T, Allocator>::ListIterator::reference List<T, Allocator>::ListIterator::operator\*() {

return \_list[\_idx];

}

template <class T, class Allocator>

typename List<T, Allocator>::ListIterator::pointer List<T, Allocator>::ListIterator::operator->(){

return &\_list[\_idx];

}

template <class T, class Allocator>

bool List<T, Allocator>::ListIterator::operator!=(const ListIterator& other){

if(\_idx != other.\_idx) {

return true;

}

if(&\_list != &(other.\_list)) {

return true;

}

return false;

}

template <class T, class Allocator>

bool List<T, Allocator>::ListIterator::operator==(const ListIterator& other){

if(\_idx != other.\_idx) {

return false;

}

if(&\_list != &(other.\_list)) {

return false;

}

return true;

}

template <class T, class Allocator>

T& List<T, Allocator>::ListNode::get(size\_t idx){

if(idx == 0) {

return \_val;

}

if(\_next) {

return \_next->get(--idx);

}

throw std::logic\_error("Out of bounds");

}

template <class T, class Allocator>

void List<T, Allocator>::ListNode::insert(ListNode &prev, size\_t idx, const T& val) {

if(idx == 0) {

prev.\_next = std::make\_unique<ListNode>(ListNode{std::move(prev.\_next), val});

return;

} else {

return \_next->insert(\*this, --idx, val);

}

throw std::logic\_error("Out of bounds");

}

template <class T, class Allocator>

void List<T, Allocator>::ListNode::erase(ListNode &prev, size\_t idx) {

if(idx == 0) {

prev.\_next = std::move(\_next);

return;

} else {

if(\_next) {

return \_next->erase(\*this, --idx);

}

throw std::logic\_error("Out of bounds");

}

}

template<class T, class Allocator>

typename List<T, Allocator>::ListIterator List<T, Allocator>::begin() {

return ListIterator(\*this, 0);

}

template<class T, class Allocator>

typename List<T, Allocator>::ListIterator List<T, Allocator>::end() {

return ListIterator(\*this, \_size);

}

template<class T, class Allocator>

typename List<T, Allocator>::ListIterator List<T, Allocator>::insert(typename List<T, Allocator>::ListIterator& it, T& val) {

if (it.\_idx > size()) {

throw std::logic\_error("Out of bounds");

}

if (it.\_idx == 0) {

\_head = std::make\_unique<ListNode>(ListNode{std::move(\_head), val});

\_size++;

} else {

\_head->\_next->insert(\*\_head,it.\_idx-1, val);

\_size++;

}

if (it.\_idx <= size()) {

return it;

}

return ListIterator(\*this, size());

}

template<class T, class Allocator>

T& List<T, Allocator>::operator[](size\_t idx){

if(!\_head) {

throw std::logic\_error("Out of bounds");

}

return \_head->get(idx);

}

template <class T, class Allocator>

typename List<T, Allocator>::ListIterator List<T, Allocator>::erase(ListIterator& iter) {

if (iter.\_idx >= size()) {

throw std::logic\_error("Out of bounds");

}

if (iter.\_idx == 0){

\_head = std::move(\_head->\_next);

\_size--;

} else {

if (\_head->\_next) {

\_head->\_next->erase(\*\_head,iter.\_idx-1);

\_size--;

}

}

if(iter.\_idx < size()) {

return iter;

}

return ListIterator(\*this, size());

}

**trapezoid.hpp**

// М8О-206Б-19 Киреев А.К.

#pragma once

#include <iostream>

#include <algorithm>

#include <vector>

#include <cmath>

// вспомогательные функции для сортировки по полярному углу

template <class T>

std::pair<T, T> get(const std::pair<T, T>& lhs, const std::pair<T, T> rhs) { // получаем вектор по двум точкам

return std::make\_pair(rhs.first-lhs.first, rhs.second-lhs.second);

}

template <class T>

T cross(const std::pair<T, T>& lhs, const std::pair<T, T> rhs) { // cross product

return lhs.first \* rhs.second - lhs.second \* rhs.first;

} // конец вспомогательных функций

template <class T>

class Trapezoid {

public:

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

vertex a, b, c, d;

bool valid;

Trapezoid() = default;

Trapezoid(std::istream& is);

friend std::ostream& operator<<(std::ostream& os, const Trapezoid& t);

double area();

~Trapezoid() = default;

};

template <class T>

Trapezoid<T>::Trapezoid(std::istream& is) {

is >> a.first >> a.second

>> b.first >> b.second

>> c.first >> c.second

>> d.first >> d.second;

std::vector<vertex> points = {a, b, c, d};

std::sort(points.begin(), points.end(), [](auto& lhs, auto& rhs) {

if (lhs.first == static\_cast<T>(0) &&

lhs.second == static\_cast<T>(0)) {

return true;

}

if (rhs.first == static\_cast<T>(0) &&

rhs.second == static\_cast<T>(0)) {

return false;

}

std::pair<T, T> p0 = std::make\_pair(static\_cast<T>(0), static\_cast<T>(0));

std::pair<T, T> l = get(p0, lhs);

std::pair<T, T> r = get(p0, rhs);

return cross(l, r) < 0;

});

a = points[0];

b = points[1];

c = points[2];

d = points[3];

double diag1 = (c.first - a.first)\*(c.first - a.first) +

(c.second - a.second)\*(c.second - a.second);

double diag2 = (d.first - b.first)\*(d.first - b.first) +

(d.second - b.second)\*(d.second - b.second);

double segm1 = (c.first - b.first)\*(c.first - b.first) +

(c.second - b.second)\*(c.second - b.second);

double segm2 = (d.first - a.first)\*(d.first - a.first) +

(d.second - a.second)\*(d.second - a.second);

double segm3 = (a.first - b.first)\*(a.first - b.first) +

(a.second - b.second)\*(a.second - b.second);

double segm4 = (d.first - c.first)\*(d.first - c.first) +

(d.second - c.second)\*(d.second - c.second);

if ((diag1 == diag2) && ((segm1 != segm2 && segm3 == segm4) ||

(segm1 == segm2 && segm3 != segm4))) {

valid = true;

} else {

valid = false;

}

}

template <class T>

std::ostream& operator<<(std::ostream& os, Trapezoid<T>& t) {

if (t.valid) {

os << "["

<< " (" << t.a.first << ", " << t.a.second << "), "

<< "(" << t.b.first << ", " << t.b.second << "), "

<< "(" << t.c.first << ", " << t.c.second << "), "

<< "(" << t.d.first << ", " << t.d.second << ") "

<< "]\n";

}

return os;

}

template <class T>

double Trapezoid<T>::area() {

return 0.5 \* abs(a.first \* b.second + b.first \* c.second + c.first \* d.second + d.first \* a.second -

b.first \* a.second - c.first \* b.second - d.first \* c.second - a.first \* d.second);

}

1. **Вывод**

Аллокаторы в С++ позволяют гибко настроить выделение памяти и интерфейс работы с ней. Данная обертка позволит избежать излишних обращений к операционной системе для запроса выделения памяти, позволит уменьшить дефрагментацию, повысить производительность.

**Список литературы**

1. “Язык программирования C++. Краткий курс”, Бьярне Страуструп, 2-е издание, (перевод Игоря Красикова), бумага офсетная-белая, твердый переплет, 320 стр., ISBN 978-5-907144-12-5.