**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: Гребенков Дмитрий Игоревич

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Разработать шаблоны классов согласно варианту задания. Параметром шаблона должен являться скалярный тип данных задающий тип данных для оси координат. Классы должны иметь публичные поля. Фигуры являются фигурами вращения, т.е. равносторонние (кроме трапеции и прямоугольника). 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:

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.

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.

Вариант задания (12): трапеция, список, очередь.

1. Описание программы

Созданы шаблонные классы для хранения трапеций по координатам трех вершин, для хранения элемента списка, состоящего из объекта класса “Трапеция” и ссылки на следующий элемент, для линейного аллокатора, хранящего указатели на свободные блоки памяти в очереди, для реализации итератора и списка.

Классы для элемента списка, списка и итератора написаны с использованием умных указателей std::shared\_ptr и std::weak\_ptr.

В классе списка реализованы требуемые в варианте задания методы.

В классе итератора перегружены все необходимые для совместимости со стандартными алгоритмами операторы.

Реализована шаблонная функция для вывода вершин трапеций, а также вариант этой функции для использования в стандартных алгоритмах.

Класс аллокатора представляет собой линейный аллокатор со списком свободных блоков, хранящемся в виде очереди. В этом классе применяются методы для инициализации аллокатора, аллокации и деаллокации.

Взаимодействие с пользователем осуществляется при помощи меню.

При выполнении некорректных операций происходит генерация исключений и вывод информации об ошибке в поток вывода std::cerr.

Примечание: Трапеция равнобедренная. Для ускорения вычислений и упрощения хранения на вводимые фигуры установлены некоторые ограничения. Во-первых, основания вводимых трапеций должны быть параллельны оси Ox. Во-вторых, большее основание трапеции должно быть снизу.

Трапеция задается координатами трех вершин - левой верхней, левой нижней и правой нижней.

Для добавления элемента в пустой список необходимо вставлять элемент на место с индексом 0.

1. Наборы тестов и результаты их выполнения

Демонстрация работы основных функций программы:

0. Show menu.

1. Insert figure.

2. Delete figure.

3. Print element.

4. Print list.

5. Quit program.

Enter option: 1

Enter index: 0

Enter coordinates of left-top, left-bottom and right-bottom vertices:

2 2 1 0 5 0

Enter option: 1

Enter index: 0

Enter coordinates of left-top, left-bottom and right-bottom vertices:

1 2 0 0 3 0

Enter option: 1

Enter index: 2

Enter coordinates of left-top, left-bottom and right-bottom vertices:

2 2 1 1 4 1

Enter option: 4

(1, 2)(0, 0)(3, 0)(2, 2)

(2, 2)(1, 0)(5, 0)(4, 2)

(2, 2)(1, 1)(4, 1)(3, 2)

Enter option: 2

Enter index: 1

Enter option: 4

(1, 2)(0, 0)(3, 0)(2, 2)

(2, 2)(1, 1)(4, 1)(3, 2)

Enter option: 1

Enter index: 2

Enter coordinates of left-top, left-bottom and right-bottom vertices:

1 4 0 1 3 1

Enter option: 4

(1, 2)(0, 0)(3, 0)(2, 2)

(2, 2)(1, 1)(4, 1)(3, 2)

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

Enter option: 3

Enter index: 0

(1, 2)(0, 0)(3, 0)(2, 2)

Enter option: 3

Enter index: 1

(2, 2)(1, 1)(4, 1)(3, 2)

Enter option: 3

Enter index: 2

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

Enter option: 2

Enter index: 1

Enter option: 4

(1, 2)(0, 0)(3, 0)(2, 2)

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

Enter option: 2

Enter index: 1

Enter option: 4

(1, 2)(0, 0)(3, 0)(2, 2)

Enter option: 2

Enter index: 0

Enter option: 4

List is empty.

Enter option: 0

0. Show menu.

1. Insert figure.

2. Delete figure.

3. Print element.

4. Print list.

5. Quit program.

Enter option: 5

Работа с некорректными командами:

0. Show menu.

1. Insert figure.

2. Delete figure.

3. Print element.

4. Print list.

5. Quit program.

Enter option: 1

Enter index: 2

ERROR: incorrect index.

Enter option: 2

Enter index: 0

ERROR: incorrect index.

Enter option: 3

Enter index: 1

ERROR: incorrect index.

Enter option: 4

List is empty.

Enter option: 1

Enter index: 0

Enter coordinates of left-top, left-bottom and right-bottom vertices:

1 1 1 1 1 1

Enter option: 1

Enter index: 4

ERROR: incorrect index.

Enter option: 2

Enter index: 3

ERROR: incorrect index.

Enter option: 3

Enter index: 2

ERROR: incorrect index.

Enter option: 4

(1, 1)(1, 1)(1, 1)(1, 1)

Enter option: 2

Enter index: 0

Enter option: 4

List is empty.

Enter option: 2

Enter index: 0

ERROR: incorrect index.

Переполнение памяти:

0. Show menu.

1. Insert figure.

2. Delete figure.

3. Print element.

4. Print list.

5. Quit program.

Enter option: 1

Enter index: 0

Enter coordinates of left-top, left-bottom and right-bottom vertices:

1 1 1 1 1 1

Enter option: 1

Enter index: 1

Enter coordinates of left-top, left-bottom and right-bottom vertices:

2 2 2 2 2 2

Enter option: 1

Enter index: 2

Enter coordinates of left-top, left-bottom and right-bottom vertices:

3 3 3 3 3 3

Enter option: 1

Enter index: 3

Enter coordinates of left-top, left-bottom and right-bottom vertices:

4 4 4 4 4 4

Enter option: 1

Enter index: 4

Enter coordinates of left-top, left-bottom and right-bottom vertices:

5 5 5 5 5 5

Enter option: 1

ERROR: cannot allocate - no empty space.

1. Листинг программы

/\*

Гребенков Д.И.

Группа М8О-207Б-19.

Вариант задания: 12 (трапеция, список, очередь).

Разработать шаблоны классов согласно варианту задания. Параметром шаблона должен являться скалярный тип данных задающий тип данных для оси координат. Классы должны иметь публичные поля. Фигуры являются фигурами вращения, т.е. равносторонние (кроме трапеции и прямоугольника).

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:

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.

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.

\*/

#include <iostream>

#include <queue>

#include <memory>

#include <cmath>

#include <algorithm>

const int MAX\_SIZE = 5;

template <typename T>

class Trapezoid {

public:

using type = T;

std::pair <T, T> left\_top, left\_bottom, right\_bottom;

Trapezoid () {}

Trapezoid (T ltx, T lty, T lbx, T lby, T rbx, T rby) {

left\_top.first = ltx;

left\_top.second = lty;

left\_bottom.first = lbx;

left\_bottom.second = lby;

right\_bottom.first = rbx;

right\_bottom.second = rby;

};

};

template <typename T>

void Vertices (Trapezoid <T> tr) {

std::cout << "(" << tr.left\_top.first << ", " << tr.left\_top.second << ")"

<< "(" << tr.left\_bottom.first << ", " << tr.left\_bottom.second << ")"

<< "(" << tr.right\_bottom.first << ", " << tr.right\_bottom.second << ")"

<< "(" << tr.right\_bottom.first - tr.left\_top.first + tr.left\_bottom.first <<

", " << tr.left\_top.second << ")" << std::endl;

}

template <class T, int MAX\_SIZE>

class Allocator {

public:

size\_t sizeOne = sizeof (T);

std::queue <T\*> que;

T memory [MAX\_SIZE];

using value\_type = T;

using pointer = T \*;

using const\_pointer = const T\*;

using size\_type = size\_t;

int count = 0;

void initialize () {

for (int i = 0; i < MAX\_SIZE; i ++) {

que.push (&memory [i]);

}

}

T\* allocate (int s) {

T\* res;

if ((count + s - 1) < (MAX\_SIZE)) {

res = que.front ();

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

que.pop ();

count ++;

}

return (res);

}

else {

throw "ERROR: cannot allocate - no empty space.";

}

return (res);

}

void deallocate () {

int size = que.size ();

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

que.pop ();

}

count = 0;

initialize ();

}

template <class U>

struct rebind {

using other = Allocator <U, MAX\_SIZE>;

};

};

template <class T>

class ListMember {

public:

T figure;

std::shared\_ptr <ListMember <T> > next;

};

template <class T>

class Iterator {

public:

std::weak\_ptr <T> it;

Iterator (std::shared\_ptr <T> al) {

it = al;

}

Iterator \* operator++ () {

if (it.lock ()) {

it = it.lock ()->next;

}

else {

throw "ERROR: trying to go out of range.";

}

return this;

}

T operator\* () {

if (!it.lock ()) {

throw "ERROR: trying to dereference nullptr.";

}

return (\*(it.lock ().get ()));

}

bool operator!= (Iterator <T> sec) {

bool res;

if ((it.lock ().get ()) != (sec.it.lock ().get ())) {

res = true;

}

else {

res = false;

}

return (res);

}

};

template <class T, class Allocator>

class List {

public:

std::shared\_ptr <T> first;

Allocator al;

int count = 0;

List () {

al.initialize ();

}

struct deleter {

void operator() (void \* ptr) {}

};

deleter del;

Iterator <T> begin () {

return (Iterator <T> (first));

}

Iterator <T> end () {

std::shared\_ptr <T> temp = first;

while (temp) {

temp = temp->next;

}

return (Iterator <T> (temp));

}

void insert () {

int idx;

int ltx, lty, lbx, lby, rbx, rby;

T\* ptr = al.allocate (1);

std::cout << "Enter index: ";

std::cin >> idx;

if ((idx > count) || (idx < 0)) {

throw "ERROR: incorrect index.";

}

std::cout << "Enter coordinates of left-top, left-bottom and right-bottom vertices: " <<

std::endl;

std::cin >> ltx >> lty >> lbx >> lby >> rbx >> rby;

std::shared\_ptr <T> elem (new (ptr) T, del);

elem->figure.left\_top.first = ltx;

elem->figure.left\_top.second = lty;

elem->figure.left\_bottom.first = lbx;

elem->figure.left\_bottom.second = lby;

elem->figure.right\_bottom.first = rbx;

elem->figure.right\_bottom.second = rby;

if (idx == 0) {

if (first) {

elem->next = first;

first = elem;

}

else {

first = elem;

}

count ++;

}

else {

std::shared\_ptr <T> temp = first;

std::shared\_ptr <T> prev;

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

prev = temp;

temp = temp->next;

}

prev->next = elem;

elem->next = temp;

count ++;

}

}

void erase () {

int idx;

std::cout << "Enter index: ";

std::cin >> idx;

if ((idx >= count) || (idx < 0)) {

throw "ERROR: incorrect index.";

}

if (idx == 0) {

if (first) {

first = first->next;

count --;

}

else {

throw "ERROR: list is empty.";

}

}

else {

std::shared\_ptr <T> temp = first;

std::shared\_ptr <T> prev;

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

prev = temp;

temp = temp->next;

}

prev = temp->next;

count --;

}

}

T operator[] (int idx) {

if ((idx >= count) || (idx < 0)) {

throw "ERROR: incorrect index.";

}

std::shared\_ptr <T> temp = first;

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

temp = temp->next;

}

return (\*(temp.get ()));

}

};

void Print (ListMember <Trapezoid <int> > lm) {

Vertices (lm.figure);

}

void ShowMenu () {

std::cout << " 0. Show menu." << std::endl;

std::cout << " 1. Insert figure." << std::endl;

std::cout << " 2. Delete figure." << std::endl;

std::cout << " 3. Print element." << std::endl;

std::cout << " 4. Print list." << std::endl;

std::cout << " 5. Quit program." << std::endl;

}

int main () {

List <ListMember <Trapezoid <int> >, Allocator <ListMember <Trapezoid <int> >, MAX\_SIZE > > list;

int option = 7;

ShowMenu ();

while (option != 5) {

std::cout << "Enter option: ";

std::cin >> option;

switch (option) {

case 0: {

ShowMenu ();

break;

}

case 1: {

try {

list.insert ();

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 2: {

try {

list.erase ();

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 3: {

try {

int idx;

std::cout << "Enter index: ";

std::cin >> idx;

Vertices (list [idx].figure);

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 4: {

try {

if (list.first) {

if (list.begin () != list.end ()) {

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

}

else {

Vertices (list [0].figure);

}

}

else {

std::cout << "List is empty." << std::endl;

}

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 5: {

break;

}

default: {

std::cout << "ERROR: incorrect option." << std::endl;

break;

}

}

}

return 0;

}

1. Вывод

Изучены основы работы с аллокаторами. Закреплены навыки использования умных указателей.

Список литературы

1. Аллокаторы памяти / Хабр [Электронный ресурс] URL: <https://habr.com/ru/post/505632/> (дата обращения 12.12.2020).
2. std::queue - cppreference.com [Электронный ресурс] URL: <https://en.cppreference.com/w/cpp/container/queue> (дата обращения 12.12.2020).
3. GitHub - mtrebi/memory-allocators: Custom memory allocators in C++ to improve the performance of dynamic memory allocation [Электронный ресурс] URL: <https://github.com/mtrebi/memory-allocators> (дата обращения 12.12.2020).