**暨南大学本科实验报告专用纸**

课程名称 C++程序设计 成绩评定

实验项目名称 allocator类编程实验 指导教师 王勇杰

实验项目编号 7 实验项目类型 编程型 实验地点 C105

学生姓名 李炜鹏 学号 2017052544

学院 智能科学与工程学院 系 计算机 专业 信息安全

实验时间2019年12月10日 午～12月13日 午 温度 ℃湿度

1. **实验目的**
2. 认识并且理解allocator类的概念及其用法
3. 认识并且理解拷贝和填充未初始化内存的算法
4. 实验要求：阅读课本第12章的“12.2 动态数组”内容，试分别调试运行实验七中的两个源代码
5. 实验报告要求：
   1. 实验目的
   2. 实验原理
   3. 注释必要的代码
   4. 实验结果的截图
   5. 实验评估
6. **实验原理**
7. allocator类：
   1. 使用new和delete的缺陷性：（1）new将内存分配和对象构造组合了在一起，delete将对象析构和内存释放组合在了一起。当分配一大块内存是，我们通常计划在这块内存上按需构造对象，在这种情况下，我们希望内存分配和对象构造分离开来（2）使用new将内存分配和对象构造产生的问题：可能创建了一些永远用不到的对象、每个使用到的元素被赋值了两次、那些没有默认构造函数的类就不能动态分配数组了
   2. allocator类解决了上面缺陷
   3. allocator类的使用：分配内存、构造对象、销毁对象、释放内存
8. 拷贝和填充未初始化内存的算法：这是标准库为allocator定义的两个伴随算法uninitialized\_copy(b,e,b2)、uninitialized\_copy\_n(b,n,b2)、uninitialized\_fill(b,e,t) 、uninitialized\_fill\_n(b,n,t)
9. **主要仪器设备**

**仪器：**计算机

**实验环境：**Visual Studio 2017或Dev-C++

1. **源程序**

**usealloc.cpp**

#include <string>

using std::string;

#include <memory>

using std::allocator;

#include <cstddef>

using std::size\_t;

#include <iostream>

using std::cout; using std::endl;

#include <fstream>

using std::ifstream;

int main()

{

//四个步骤：

//1.分配内存

const size\_t n = 100;

allocator<string> alloc; //能够分配string对象的allocator类对象alloc

string \*p = alloc.allocate(n); //分配n个未初始化的string，返回首指针到p中

string \*q = p; //指针q指向最后一个被构造的元素，即p

//2.构造对象

//C++11之前的分配器构造必须显式传递一个已分配类型的对象，因此，我们显式传递一个空的string（）

alloc.construct(q++, string()); // \*q为空字符串

cout << "对象q1：" << \*(q - 1) << endl;

alloc.construct(q++, string(10, 'c')); // \*q为cccccccccc

cout << "对象q2：" << \*(q - 1) << endl;

alloc.construct(q++, string("hi")); // \*q为hi

cout << "对象q3：" << \*(q - 1) << endl;

//销毁对象

cout << "对象p：" << \*p << endl;

while (q != p)

alloc.destroy(--q); // 释放我们实际分配的字符串

//4.释放内存

alloc.deallocate(p, n); // 返回我们分配的内存

p = alloc.allocate(n); // 分配n个未初始化的string

string s;

q = p; // 指针q指向第一个string

ifstream in("abc.txt");

while (in >> s && q != p + n)

alloc.construct(q++, s); // 只构造从in输入的字符串

size\_t size = q - p; // 计算读取的string数量

cout << "read " << size << " strings" << endl;

for (q = p + size - 1; q != p; --q)

alloc.destroy(q); // 释放我们实际分配的字符串

alloc.deallocate(p, n); // 返回我们分配的内存

in.close();

in.open("abc.txt");

p = new string[n]; // 初始化n个空字符串

q = p; // 指针q指向第一个string

while (in >> s && q != p + n)

\*q++ = s; // 向指针q指向的内存赋一个新的值

size = q - p; // 计算读取的string数量

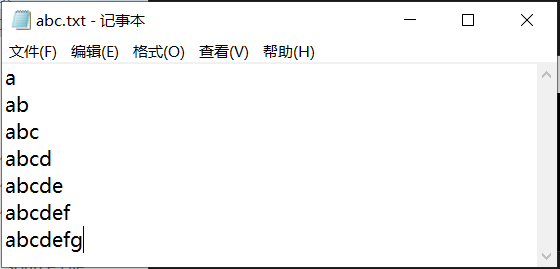
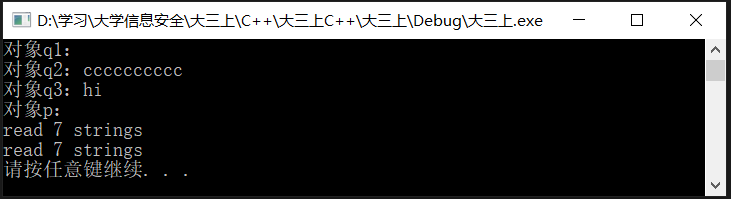
cout << "read " << size << " strings" << endl;

delete[] p; // 记得要使用delete []来释放内存

system("pause");

return 0;

}

本程序通过allocator类分配动态内存，经过四个步骤：分配内存、构造对象、销毁对象、释放内存。在这个代码里面，我们将对象q分表初始化为：null、cccccccccc、hi，而对象p则仅仅分配了内存而未初始化，于是没有输出；后面则是通过读文件的方式传出string再将动态指针指向的内存初始化，结果符合预期。

**usealloc2.cpp**

#include <cstddef>

using std::size\_t;

#include <string>

using std::string;

#include <vector>

using std::vector;

#include <memory>

using std::uninitialized\_copy;

using std::allocator; using std::uninitialized\_fill\_n;

#include <iostream>

using std::cout; using std::endl;

int main()

{

int temp[] = { 1,2,3,4,5,6,7,8,9 };//定义并且初始化一个int型数组

vector<int> vi(temp, temp + sizeof(temp) / sizeof(\*temp));//用int型数组的数据对vector进行初始化

allocator<int> alloc;

int \*p = alloc.allocate(vi.size() \* 2); //分配比vi中元素所占空间大一倍的动态内存

int \*q = uninitialized\_copy(vi.begin(), vi.end(), p);//通过拷贝vi中的元素来构造从p开始的元素

uninitialized\_fill\_n(q, vi.size(), 42);//将剩余元素初始化为42

//输出p指针指向的元素

cout << "p: ";

for (size\_t i = 0; i != vi.size(); ++i)

cout << \*(p + i) << " ";//p指向拷贝数据的开始（数据1）

cout << endl;

//输出q指针指向的元素

cout << "q: ";

for (size\_t i = 0; i != vi.size(); ++i)

cout << \*(q + i) << " ";//q指向填充数据的开始（第一个数据42）

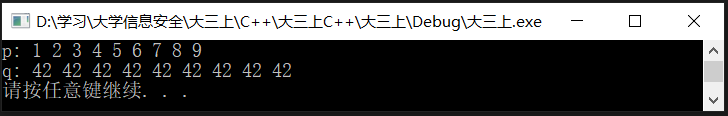
cout << endl;

alloc.deallocate(p, vi.size());//释放内存

system("pause");

return 0;

}



本程序主要使用了拷贝和填充未初始化内存的算法：uninitialized\_copy、uninitialized\_fill\_n，需要注意的是：经过uninitialized\_copy函数之后，p指向拷贝数据的开始（数据1）；经过uninitialized\_fill\_n之后，q指向填充数据的开始（第一个数据42），运行结果符合预期。

1. **实验评估**

本实验是主要考察了allocator类的基本操作，通过本实验，我初步理解了allocator类分配内存构造对象的方法，以及这种方法与new之间的差异，还练习了allcoator类里面的两个伴随算法，本实验属于概念理解型的实验，尽管操作相对简单，但是对理解概念还是有很大的帮助。