**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 动态内存 指导教师 王勇杰

实验项目编号 3 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. 巩固C语言中的指针的概念以及用法，理解指针作用域的概念
3. 认识动态内存的概念，以及理解内存的生命周期的概念
4. 认识C++中面向对象编程中类的设计（数据成员、成员函数、构造函数、析构函数、友元类、友元函数）
5. 理解C++中动态内存与智能指针的概念，认识shared\_ptr类和weak\_ptr类，理解shared\_ptr类和weak\_ptr类的关系，区分shared\_ptr类与unique\_ptr类的异同
6. 实验要求：

阅读课本p400-p407

* 1. 调试运行“程序1”文件夹中的源代码。
  2. 调试运行“程序2”文件夹中的源代码。
  3. 调试运行“程序3”文件夹中的源代码。

1. 实验报告要求：
   1. 实验目的
   2. 实验原理
   3. 注释代码中的每条语句
   4. 实验结果的截图
   5. 实验评估
2. **实验原理**
3. 指针：

指针通过指向内存中的一块逻辑地址，可以读取或者修改该内存中的数据，用法是：

数据类型 \*指针变量=某地址；//定义指针

\*指针变量 //取指针的数据

1. 动态内存：

通过一对运算符——new（为对象分配内存并返回一个指向该对象的指针，我们可以对此对象进行初始化）、delete（收回指针，销毁对象，释放内存）

1. 类：

本实验的程序3主要涉及以下几个类成员：数据成员、成员函数、构造函数、析构函数、友元类、友元函数

* 1. 数据成员：用于描述类的属性
  2. 成员函数：是类提供的功能
  3. 构造函数：分为默认构造函数、含参数的构造函数、复制构造函数、转换构造函数，主要作用是对类的对象进行初始化，其中前两个可以使用初始化列表来初始化
  4. 析构函数：控制此对象在销毁时进行的操作（本实验未涉及析构函数）
  5. 友元类/友元函数：作为类的友元类或者友元函数时，可以在类外引用类的私有成员，使用friend关键字来声明友元。注意如果声明友元类，要在类体中声明友元之前提前声明友元类的类名或者提前定义友元类的类体

1. 智能指针：

分为两大类（shared\_ptr和unique\_ptr），本实验只使用了shared\_ptr类，而C++标准库还定义了一个weak\_ptr类（作为软引用，指向shared\_ptr所管理的对象）。shared\_ptr类中有一个计数器（引用计数），用于计算该智能指针指向的内存中的数据有多少个智能指针一起共享。当智能指针离开了其作用域时，引用计数减一，当引用计数为零时，智能指针自动释放该指针指向的内存，相应的对象也被销毁，智能指针也被收回。

1. **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**

**程序1：**

**Foo.h**

#ifndef FOO\_H

#define FOO\_H

#include <iostream>

typedef int T;//为int定义别名T

struct Foo { //使用struct定义一个类Foo

Foo(T t) : val(t) { }//Foo类的构造函数，使用传进来的T类型的数据对数据成员val进行初始化（使用了类的初始化列表）

T val;

};

std::ostream&

print(std::ostream &os, const Foo &f)//定义一个print函数，返回类型是一个ostream类，参数类型是标准输出流、Foo类的引用

{

os << f.val;

return os;

}

#endif

**allocPtr.cpp**

#include <vector>

using std::vector;

#include <string>

using std::string;

#include <iostream>

using std::istream; using std::ostream;

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

#include "Foo.h"

//定义一个函数：传入一个T类型数据，返回一个Foo类对象指针

Foo\* factory(T arg)

{

return new Foo(arg); //返回一个新的Foo类对象

}

Foo\* use\_factory(T arg)

{

Foo \*p = factory(arg);//调用factory函数

print(cout, \*p);//输出p指针指向的对象的值

cout << endl;

return p; //返回p指针

}

//动态内存实现

int main()

{

T arg;

while (cin >> arg) //当遇到不符合cin条件的时候结束循环（比如输入非T类型的数据）

{

Foo \*p = use\_factory(arg);

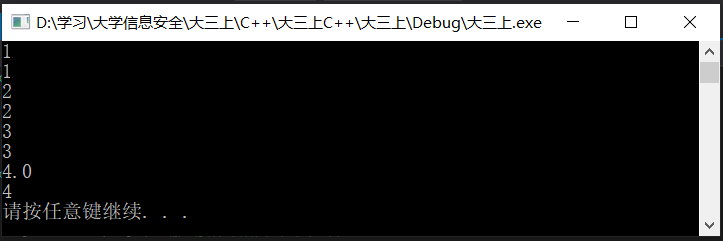
delete p; //使用完指针之后记得删除指针，否则会导致内存泄漏

}

system("pause");

return 0;

}

****

分别输入1、2、3、4.0，由于前三个数据符合T类型，第四个数据4.0其实编译器识别了4这个T型数据，而.作为非T类型，使while循环结束，于是程序结束。

**程序2：**

**Foo.h（与程序1相同）**

#ifndef FOO\_H

#define FOO\_H

#include <iostream>

typedef int T;//为int定义别名T

struct Foo { //使用struct定义一个类Foo

Foo(T t) : val(t) { }//Foo类的构造函数，使用传进来的T类型的数据对数据成员val进行初始化（使用了类的初始化列表）

T val;

};

std::ostream&

print(std::ostream &os, const Foo &f)//定义一个print函数，返回类型是一个ostream类，参数类型是标准输出流、Foo类的引用

{

os << f.val;

return os;

}

#endif

**allocSP.cpp**

#include <vector>

using std::vector;

#include <string>

using std::string;

#include <memory>

using std::shared\_ptr;

#include <iostream>

using std::istream; using std::ostream;

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

#include "Foo.h"

//Factory函数返回一个shared\_ptr

shared\_ptr<Foo> factory(T arg)

{

return shared\_ptr<Foo>(new Foo(arg));//返回一个指向Foo类的共享指针

}

shared\_ptr<Foo> use\_factory(T arg)

{

shared\_ptr<Foo> p = factory(arg);

print(cout, \*p);

cout << endl;

return p; //当我们返回p时，引用计数会递增

}

int main()

{

T arg;

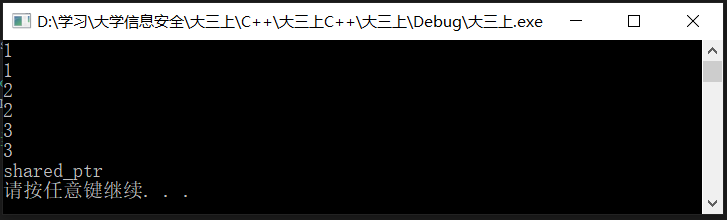
while (cin >> arg)

use\_factory(arg);

system("pause");

return 0;

}



输入1,2,3，shared\_ptr之后，显示运行结果如上。

**程序3：**

**StrBlob.h**

#ifndef STRBLOB\_H

#define STRBLOB\_H

#include <vector>

#include <string>

#include <memory>

#include <stdexcept>

class StrBlobPtr;//由于StrBlobPtr类是StrBlob的友元类，所以这里要提前声明StrBlobPtr类

//StrBlob类

class StrBlob {

friend class StrBlobPtr;//声明StrBlobPtr类是StrBlob的友元类

public:

typedef std::vector<std::string>::size\_type size\_type;//为命名空间std中的vector<string>::size\_type类型声明别名为size\_type

//构造函数

StrBlob() : data(new std::vector<std::string>()) { }//默认构造函数

StrBlob(const std::string\*, const std::string\*);//含参数的构造函数

size\_type size() const { return data->size(); }//返回StrBlob类中数据成员data的大小

bool empty() const { return data->empty(); }//判断data是否为空

void push\_back(const std::string &t) { data->push\_back(t); }//增加元素

void pop\_back();//删除元素

// 元素存取

std::string& front();

std::string& back();

//StrBlobPtr类的接口，用于遍历类中的元素

StrBlobPtr begin();

StrBlobPtr end();

private:

std::shared\_ptr<std::vector<std::string> > data;//数据成员data

void check(size\_type i, const std::string &msg) const;//如果data为空，抛出异常信息msg

};

//定义含参数的构造函数

inline//inline关键字用于表示内置成员函数，使用此关键字可以在类外定义成员函数，否则，也可以在雷替里面用inline声明

StrBlob::StrBlob(const std::string \*beg, const std::string \*end) :

data(new std::vector<std::string>(beg, end)) { }//使用两个string参数beg、end初始化数据成员data（即vector容器data里面是指针beg，end之间的数据）

//StrBlobPtr类

class StrBlobPtr {

friend bool eq(const StrBlobPtr&, const StrBlobPtr&);//声明友元函数

public:

StrBlobPtr() : curr(0) { }//默认构造函数

StrBlobPtr(StrBlob &a, size\_t sz = 0) : wptr(a.data), curr(sz) { }//含参数的构造函数，可以传入两个或者一个参数，如果不传入第二个参数，则curr默认为零

//成员函数声明如下：

std::string& deref() const;

StrBlobPtr& incr(); // prefix version？

StrBlobPtr& decr(); // prefix version？

private:

std::shared\_ptr<std::vector<std::string> > check(std::size\_t, const std::string&) const;//私有成员函数check：若size\_t=0，则抛出参数string所传进来的异常信息；否则，返回shared\_ptr指针

std::weak\_ptr<std::vector<std::string> > wptr;//数据成员wptr，weak\_ptr类型（是一个伴随类，是一种弱引用，指向shared\_ptr多管理的对象）

std::size\_t curr;//数据成员curr，size\_t类型（无符号整型）

};

inline

std::string& StrBlobPtr::deref() const//常成员函数：只能引用本类中的数据成员，而不能修改他们

{

std::shared\_ptr<std::vector<std::string> >

p = check(curr, "dereference past end");

return (\*p)[curr]; //返回vector的第curr个string对象

}

inline

std::shared\_ptr<std::vector<std::string> >

StrBlobPtr::check(std::size\_t i, const std::string &msg) const

{

//判断wptr的生命周期还在吗，若在，返回到ret里面

std::shared\_ptr<std::vector<std::string> > ret = wptr.lock();//weak\_ptr的成员函数lock():生命期未结束，则返回一个shared\_ptr，指向同样的对象。否则返回一个指向null的shared\_ptr

if (!ret)

throw std::runtime\_error("unbound StrBlobPtr");

if (i >= ret->size())

throw std::out\_of\_range(msg);

return ret; //返回ret

}

//Prefix: 返回对递增对象的引用

inline

StrBlobPtr& StrBlobPtr::incr()

{

check(curr, "increment past end of StrBlobPtr");//如果不存在指向该对象的智能指针，抛出异常信息increment past end of StrBlobPtr

++curr;//递增curr

return \*this;//返回vector的下一个元素

}

inline

StrBlobPtr& StrBlobPtr::decr()

{

--curr;//递减curr

check(-1, "decrement past begin of StrBlobPtr");

return \*this;//返回vector的上一个元素

}

inline

StrBlobPtr

StrBlob::begin()//StrBlob的对象的第一个元素

{

return StrBlobPtr(\*this);

}

inline

StrBlobPtr

StrBlob::end()//StrBlob的对象的最后一个元素

{

StrBlobPtr ret = StrBlobPtr(\*this, data->size());

return ret;

}

//对比两个StrBlobPtr是否相等

inline

bool eq(const StrBlobPtr &lhs, const StrBlobPtr &rhs)

{

std::shared\_ptr<std::vector<std::string> >

l = lhs.wptr.lock(), r = rhs.wptr.lock();

if (l == r)

return (!r || lhs.curr == rhs.curr);//如果两个StrBlobPtr相等，要么都为空，要么指向相同的内存地址

else

return false; //如果他们指向不同的vector容器，返回false

}

//对比两个StrBlobPtr是否不等

inline

bool neq(const StrBlobPtr &lhs, const StrBlobPtr &rhs)

{

return !eq(lhs, rhs);

}

#endif

**useBlob.cpp**

#include <iostream>

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

#include <string>

using std::string;

#include "StrBlob.h"

int main()

{

StrBlob b1;

{

string temp[] = { "a", "an", "the" };

StrBlob b2(temp, temp + sizeof(temp) / sizeof(\*temp));//sing以一个StrBlob对象，并且用在区间[temp,temp+sizeof(temp)/sizeof(\*temp)]的数据来初始化

b1 = b2;//b1指向b2所指向的内容（两者指向了同一个vector容器）

b2.push\_back("about");//在智能指针b2指向的vector末尾添加元素"about"

cout << b2.size() << endl;//输出b2指向的vector大小

}//在这之后b2已经被销毁，因为b2在大括号里面声明的

cout << b1.size() << endl;//这里b1的size和b2的size一样

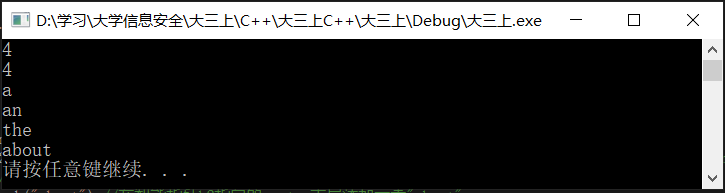
for (StrBlobPtr it = b1.begin(); neq(it, b1.end()); it.incr())//遍历b1对象

cout << it.deref() << endl;//返回当前指向的元素

system("pause");

return 0;

}

****

运行结果符合预期：输出b2的size（四个string），输出b1的size（四个string，因为b1、b2执行同一个vector容器，虽然智能指针b2已经被销毁，但是所指向的vector容器还有智能指针b1的指向，一直到b1也销毁了，vector容器的这段内存空间才被释放），输出四个字符串。

1. **实验评估**

本实验是较难的一个实验，考察了动态内存的相关知识，并着重介绍了C++新增的智能指针的知识。通过本上机实验，我巩固了指针的使用，深入理解了动态内存及其生命周期的概念，初步认识了C++中重要内容——类的设计（面向对象编程）。本实验最重要的是智能指针的理解以及运用（本实验涉及shared\_ptr和weak\_ptr）。由于这方面的知识十分复杂，通过本实验我理解并且运用到的仅仅是一些基础的用法，通过以后的深入学习与研究，希望能够更深入地理解关于动态内存与智能指针的知识。