**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 weak\_ptr指针变成实验 指导教师 王勇杰

实验项目编号 6 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. 认识并理解weak\_ptr的概念及用法
3. 理解weak\_ptr与shared\_ptr之间的关系
4. 实验要求：阅读课本第12章，完成练习12.19，定义你自己版本的StrBlobPtr，更新StrBlob类，加入恰当的friend声明及begin和end成员。
5. 实验报告要求：
   1. 实验目的
   2. 实验原理
   3. 在程序中适当地作注释
   4. 实验结果的截图
   5. 实验评估
6. **实验原理**
7. weak\_ptr是一种不控制所指向对象生存期的智能指针，它指向由一个shared\_ptr管理的对象。将一个weak\_ptr绑定到一个shared\_ptr不会改变shared\_ptr的引用计数。一旦最后一个指向对象的shared\_ptr被销毁，对象就会被释放。及时有weak\_ptr指向对象，对象也还是会被释放，因此，weak\_ptr的名字抓住了这种智能指针“弱”共享的特点。
8. weak\_ptr的用法：

w.reset(); //将w置空

w.use\_count(); //计算与w共享对象的shared\_ptr的数量

w.expired(); //弱w.use\_count()为0，返回true，否则返回false

w.lock(); //若果expired为true，返回空shared\_ptr；苟泽返回一个指向w对象的shared\_ptr

* 1. 初始化：当我们创建一个weak\_ptr时，要用一个shared\_ptr来初始化它：

auto p = make\_shared<int>(42);

weak\_ptr<int> wp(p); //wp弱共享p，p的引用计数未改变

* 1. 访问：由于对象可能不存在，我们不能使用weak\_ptr直接访问对象，而必须要调用lock，次函数检查weak\_ptr指向的对象是够仍然存在

if (shared\_ptr<int> np = wp.lock())

{

cout << \*np << endl;

}

* 1. 使用weak\_ptr，不会影响其绑定的shared\_ptr指向的对象的生存期，但是可以组织用户访问一个不存在的对象的企图

1. 本实验要求自己设计一个StrBlob类的伴随指针类，其中涉及的要素如下：
   1. 数据成员：wptr，要么为空，要么指向一个StrBlob中的vector；curr，保存当前对象所表示的元素的下标
   2. 成员函数：

I． StrBlobPtr(): 默认构造函数，将curr显示初始化为0，将wptr隐式初始化为空weak\_ptr（因为未在初始化列表中初始化wptr，wptr自动使用其默认初始化方式，即weak\_ptr的默认初始化方式——其数据类型的默认构造函数初始化的方法）

II. StrBlobPtr(StrBlob &a, size\_t sz = 0): 含参数的构造函数，使用参数列表中传进来的实参进行初始化，如果不把sz传进来则默认初始化为0；另外，由于这里引用了StrBlob类的私有数据成员data，所以要在StrBlob类体里面声明StrBlobPtr为StrBlob类的友元类，并且在StrBlob类定义之前，前置声明StrBlobPtr类

III. string& deref() const: 解引用，首相通过check检查vector是否为空，若不空返回(\*p)[curr]，\*p运算后得到vector，[curr]作为下标运算符

IV. StrBlobPtr& incr(): 前缀递增，返回递增后的对象的引用

V. check(): 检查使用vector是否安全以及curr是否在合法的范围内

* 1. 非成员函数：

I. bool eq(const StrBlobPtr &lhs, const StrBlobPtr &rhs): 对比两个StrBlobPtr是否相等

II. bool neq(const StrBlobPtr &lhs, const StrBlobPtr &rhs): 对比两个StrBlobPtr是否不等

* 1. 另外，还要在StrBlob类里面定义两个返回指向首元素和尾喉元素的StrBlobPtr的函数begin()和end()

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**

写出程序的源程序。并且注释代码中每条语句

myStrBlobPtr.h

#include <iostream>

#include <vector>

#include <string>

#include <memory>

using namespace std;

//对于StrBlob中的友元声明来说，此前置声明是必要的

class StrBlobStr;

//-----------------------------StrBlob类-----------------------------------------------

class StrBlob

{

friend class StrBlobPtr;

public:

typedef vector<string>::size\_type size\_type;//为vector<string>::size\_type定义别名size\_type

StrBlob();//默认构造函数

StrBlob(initializer\_list<string> i1);//含参数的构造函数

size\_type size() const { return data->size(); }//返回智能指针指向的vector的大小（计算vector中含有多少个string）

bool empty() const { return data->empty(); }//判断该智能指针指向的vector是否为空

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

void pop\_back();//删除元素

//元素访问：目前未搞懂这几个函数的意义

string& front();

const string& front() const;//可以引用数据成员（常与非常都可以），返回常string

string& back();

const string& back() const;//可以引用数据成员（常与非常都可以），返回常string

//返回指向首元素和尾后元素的StrBlobPtr

StrBlobPtr begin();

StrBlobPtr end();

private:

shared\_ptr<vector<string>> data;//智能指针data：类型为——类型为string的vector容器

void check(size\_type i, const string &msg) const;//如果data[i]不合法，跑出异常信息msg

};

StrBlob::StrBlob() :data(make\_shared<vector<string>>()) {}//使用make\_shared初始化智能指针

StrBlob::StrBlob(initializer\_list<string> i1) : data(make\_shared<vector<string>>(i1)) {}//make\_shared<数据类型>(初始化参数)

void StrBlob::check(size\_type i, const string &msg) const//常成员函数，只能引用常数据成员和非常数据成员，而不能改变他们

{

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

throw out\_of\_range(msg);

}

string& StrBlob::front()//返回智能指针所指向vector<string>当前的前一个string数据

{

check(0, "front on empty StrBlob");//如果vector为空，check会抛出一个异常

return data->front();

}

const string& StrBlob::front() const//返回智能指针所指向vector<string>当前的前一个string数据

{

check(0, "front on empty StrBlob");//如果vector为空，check会抛出一个异常

return data->front();

}

string& StrBlob::back()//返回智能指针所指向vector<string>当前的后一个string数据

{

check(0, "back on empty StrBlob");//如果vector为空，check会抛出一个异常

return data->back();

}

const string& StrBlob::back() const//返回智能指针所指向vector<string>当前的后一个string数据

{

check(0, "back on empty StrBlob");//如果vector为空，check会抛出一个异常

return data->back();

}

void StrBlob::pop\_back()//删除vector里面所有数据

{

check(0, "pop\_back on empty StrBlob");//如果vector为空，check会抛出一个异常

data->pop\_back();

}

//-------------------------------------------------------------------------------------

//----------------------------------StrBlobPtr类---------------------------------------

class StrBlobPtr

{

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

public:

StrBlobPtr() :curr(0) {}

StrBlobPtr(StrBlob &a, size\_t sz = 0) :wptr(a.data), curr(sz) { }

string& deref() const;//解引用

StrBlobPtr& incr();//前缀递增

private:

shared\_ptr<vector<string>> check(size\_t, const string&) const;//所检查成功，check返回一个指向vector的shared\_ptr

weak\_ptr<vector<string>> wptr;//保存一个weak\_ptr，意味着底层vector可能被销毁

size\_t curr;//保存数组中的当前位置

};

shared\_ptr<vector<string>> StrBlobPtr::check(size\_t i, const string &msg) const

{

auto ret = wptr.lock();//检查vector还在吗？

if (!ret)//vector已经不存在

throw runtime\_error("unbound StrBlobPtr");

if (i >= ret->size())//传入的索引值无效

throw out\_of\_range(msg);

return ret;

}

string& StrBlobPtr::deref() const

{

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

return (\*p)[curr];//\*p为vector，curr为索引值

}

StrBlobPtr& StrBlobPtr::incr()//前缀递增：返回递增后的对象的引用

{

check(curr, "increment past end of StrBlobPtr");//如果curr已经指向容器的尾后位置，就不能递增它

++curr;//推进当前的位置

return \*this;

}

//对比两个StrBlobPtr是否相等

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

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

{

return !eq(lhs, rhs);

}

StrBlobPtr StrBlob::begin()//StrBlob的对象的第一个元素

{

return StrBlobPtr(\*this);

}

StrBlobPtr StrBlob::end()//StrBlob的对象的最后一个元素

{

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

return ret;

}

useMyStrBlobPtr.cpp

#include <iostream>

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

#include <string>

using std::string;

#include "myStrBlobPtr.h"

int main()

{

StrBlob b1;

{

initializer\_list<string> temp = { "a", "ab", "abc" };

StrBlob b2(temp);//定义一个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("abcd");//在智能指针b2指向的vector末尾添加元素"abcd"

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

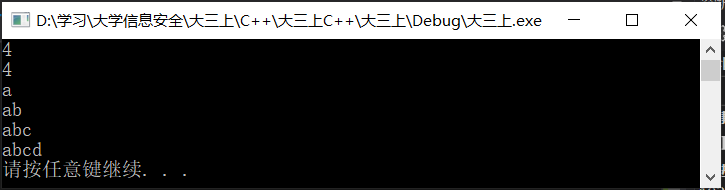
for (StrBlobPtr it = b1.begin(); neq(it,b1.end()); it.incr())//遍历b1对象

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

system("pause");

return 0;

}



初始化一个temp，装有三个字符串（a, ab, abc），将智能指针分别与之绑定，调用push\_back函数，输出b2.size()，b1.size()，以及遍历b1指向的vector的元素，结果符合预期。

1. **实验评估**

本实验综合性较强，难度较大，收获也比较大。通过本实验，我理解了weak\_ptr的概念和使用方法以及weak\_ptr与shared\_ptr的关系，深深地体会到了weak\_ptr弱共享的特性，并且知道weak\_ptr的初始化以及访问的方式都与shared\_ptr不太相同，了解到他们之间的区别以及原理。通过设计StrBlobPtr类（此任务较难），我查阅了相关的书籍和网站，初步理解了这个类中数据成员和成员函数的功能及使用（另外包括几个非成员函数、友元类的使用）。