

西安电子科技大学计算机科学与技术学院

1. **实验目的**

完成第十章练习题15、17。

1. **实验环境**

IDE：clion cmake mingw

os：windows 10

硬件：Dell g15 5511笔记本

1. **实验内容**

1、main函数保证只有一个cout<<“Hello,world!\n”;但要求输出

Initialize

Hello,world!

Clean up

2、定义两个类，各有一个静态变量，每个静态变量都涉及到对另一个类的对象的引用调用。说明实际中何处会出现这样的构造。说明如何修改以解决构造顺序依赖问题。

1. **实验步骤**

最直观的方法是重载<<，不过根据额外要输出的英文来看，这题使用构造析构输出额外字符串更符合作者题意。

新建一个类，构造函数里cout<<”Initialize\n”;析构函数里cout<<”Clean up\n”;然后在main函数前新建此类的一个对象。

那么，main函数执行前，此对象调用构造函数，输出Initialize。

然后main函数执行，输出Hello,world!

然后程序将要结束，要删除所有对象，于是调用该对象的析构函数，输出Clean up。

2、

这种方式提供了不同类的实例对象间的通信，使得任意一个类的对象都可以获得另一个类的对象的引用调用入口。另外，这个对象也能调用另一个类的公共成员函数。通过这种方式，所有老师都能随时访问学生信息，也能“命令”学生；反之学生也能进行类似操作。

而这也就是问题的来源——两个集合体之间互相包含，都需要使用对方的东西。

cpp中对类的编译机制使得这种逻辑能够实现，但这些类必须定义在**同一个文件**中。cpp在编译类时，若遇到未声明的名称，则暂时搁置，留下一点空间存储它；往下编译时，会不断丰富每个类；若原先未处理的名称被声明（还是定义？不太清楚）了，那就可以让原本搁置的空间与之发生联系，使得互相包含的逻辑编译成功。

然而，如果两个类被定义在**不同的文件**里，通过一个包含另一个、或者两个被平行包含的方式链接，则无法编译成功。因为这样的链接方式有单方向的问题，一个文件编译时只能访问它include 的文件，而无法访问到include它的文件。在代码中，Teacher被Student包含，则Teacher会被先编译，Student里的Teacher::getName()在Teacher编译之后出现，因此不会报错；但Teacher里的Student::getName()在编译时无法访问到Student，便认为getName不存在，于是报错。

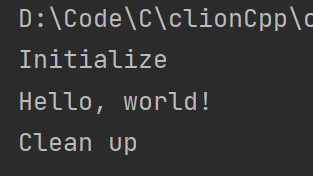
若不得不让每个类存在不同的文件里，我想到并实现了的一个**解决方案**是：新建preStudent.h，开头声明类Teacher，然后定义类preStudent作为Student的父类，里面包含所有Teacher类里面必须要用的Student类的成员函数。我在preStudent里面包含了构造函数、getName()、m\_name、static Teacher \*s\_teacher。这两个类成员变量设为protected而非private，使其能够被继承。

然后，Teacher.h就#include”preStudent.h”，并且修改static Student \*s\_student;的指针类型，变成static preStudent \*s\_student;。另外，将Student::...调用统一改成preStudent::...。

之后，Student照旧#include”Teacher.h”，并public继承preStudent，使用父类构造函数，并补充剩余成员函数，如setTeacher()和getInfo()。这样相当于把“互相包含”所必要的东西都提到了两个类的最前面，形成*12*-**12345**-*345*这样的结构。这种方法更奇妙的一点是，对于外界来说，是和把两个类写在一起有同样效果的（父类指针可以适配子类对象）。但是，在编辑Teacher类时，不能写Student只能写preStudent，代码一致性没有达到极致。

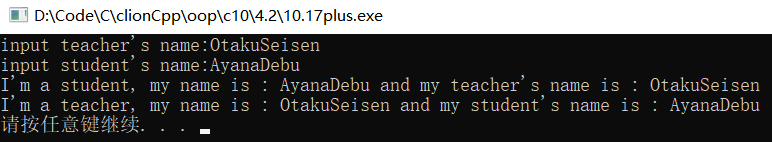
所以如果条件允许，还是把两个类写在同一个文件里面吧。

1. **实验结果**

1、

满足题目要求。

2、

能够正常输出，说明逻辑基本没有问题。

1. **实验总结**

第一题使我对对象的构造函数、析构函数的理解加深了，它更是强调了对象的周期，让两个函数的地位更加凸显。在复杂的逻辑代码里，这两个函数（特别是析构，这在下一章的String类里有很大体现）能够在一些地方起到很大的简化作用。

第二题题意不明，暂时也不清楚这种代码在哪种模块设计、实际工程里面会用到。但是通过摸索，我对cpp的编译机制的理解加深了不少，还顺便学了父子类继承的知识。