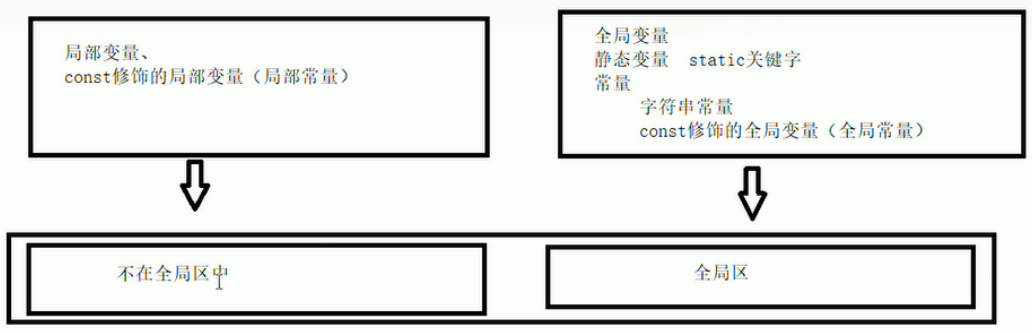
#C++内存分区：

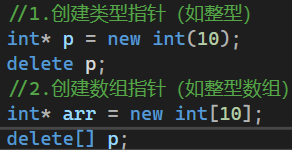
1. 代码区：存放函数体的二进制代码，由操作系统管理；
2. 全局区：存放**全局变量**和**静态变量**和**常量**，在程序结束后由操作系统释放；
3. 栈区：存放**函数的参数值**和**局部变量**，由编译器自动分配释放；
4. 堆区：由程序员分配和释放。



注：

不要返回局部变量的地址（第一次编译器做了保留，第二次编译器已经自动释放）；

#new关键字：



#引用（本质：指针常量，指向不变，值可以改变）

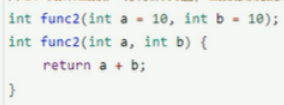
注意事项：

1. 引用必须初始化；
2. 引用在初始化之后不可改变；
3. 不要返回局部变量的引用；
4. 如果函数的返回值是引用，则这个函数调用可以作为左值；
5. 常量引用：如可以修饰形参，防止形参改变实参（使a在addup函数里面值不变，无法修改）；
6. 

#函数默认参数：

注意事项：

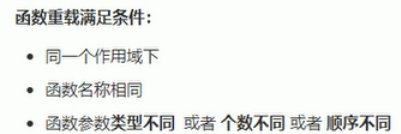
1. 如果某个位置有默认参数，则这个位置往后都必须要有默认值；
2. 如果函数声明的时候有默认值，那么函数实现的时候，就不能有默认参数，如：



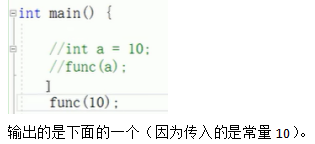
#函数重载：

注意事项：

1.



2.当引用作为重载条件时：



调用原因：，。

#C++面向对象：

封装、继承、多态。

#类和对象

注意事项：

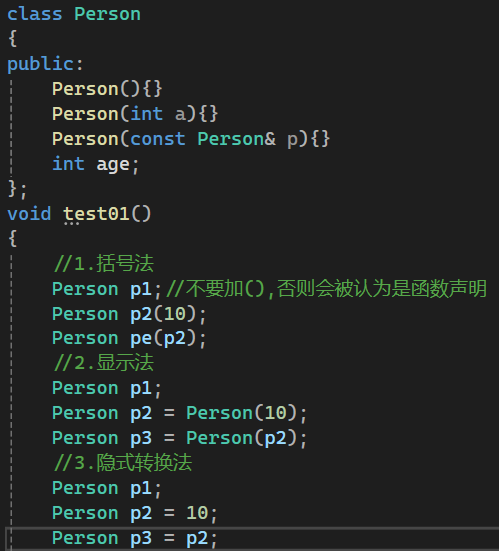
1.类的分文件编写：

在.h头文件中写函数声明（全局函数和成员函数）和成员参数；

在.cpp分文件中包括头文件.h，然后写函数实现（注意：格式为<返回值类型> <作用域>::<函数名>(参数列表) ）；

在.cpp主文件中包括头文件.h。

2.构造函数的三种调用：



注意：拷贝构造函数最好都用**const <类型>& p**的形式，防止改变原来的数值。

3.

4.深拷贝与浅拷贝：

浅拷贝：简单的复制拷贝操作，会把指针也拷贝，指向同一个内存，造成**内存重复释放**；

深拷贝：自己写拷贝构造函数，**在堆区重新开辟一个空间**，进行拷贝操作。

深拷贝的写法：



5.静态成员：

（1）**类内声明，类外初始化；**

（2）在编译阶段分配内存；

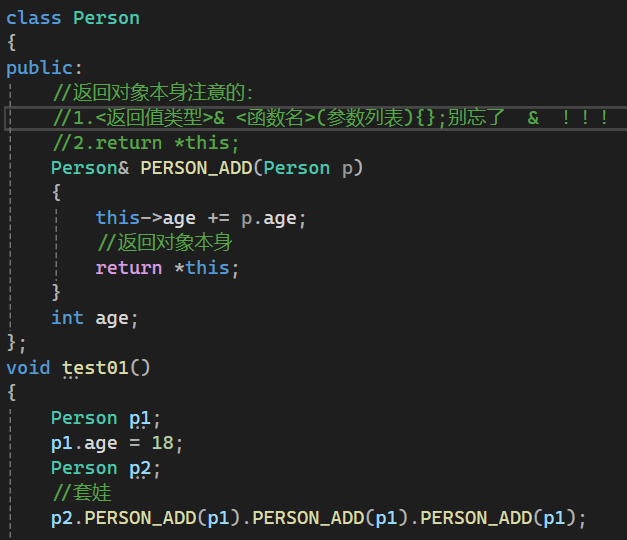
（3）所有对象共享一份数据（变量或函数）；

（4）（专对静态成员函数）静态成员函数只能访问静态成员变量；

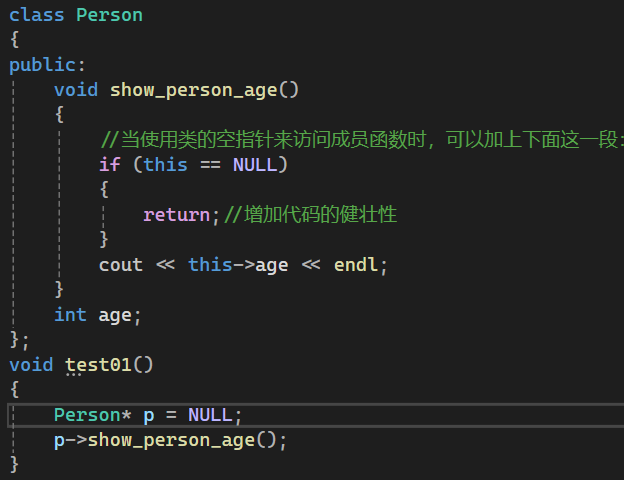
（5）静态成员两种访问方式：1.通过对象；**2.通过类名（作用域）**；

6.空对象占用内存空间为1（编译器给每个空对象都分配一个字节空间，以区分空对象占用内存的位置）；

7.在类的非静态成员函数中返回对象自己，可以使用return \*this（\*this就是对象自己），如下：



8.空指针访问成员函数：



9.const修饰成员函数/对象（常函数/常对象）：

（1）常函数：成员函数后加const；常对象：声明对象前加const；

（2）常函数内不可以修改成员属性，除非成员属性声明为mutable；

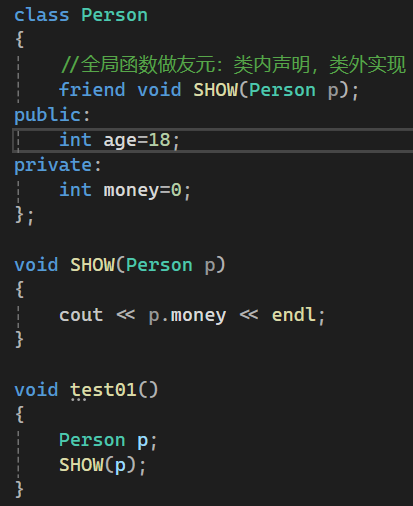
（3）常对象只能调用常函数。

（4）常函数本质：this指针本质是指针常量，常函数让this指针变成了**常量指针常量**。

10.友元（有缘 ~）：

三种实现：

（1）全局函数做友元：类内声明，类外实现



（2）类做友元：最好先在外面声明一下类名，防止编译器不认识。

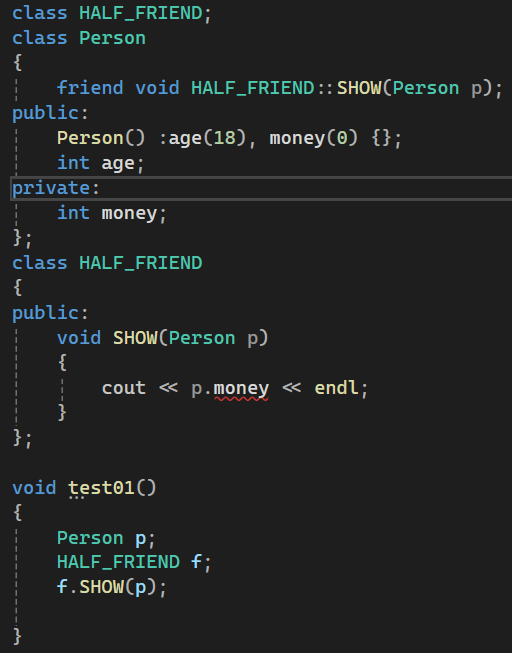
注：A把B作为友元，意味着B可以访问A的私有属性，**不意味着A能够访问B的私有属性！**

（友谊的小船说翻就翻）



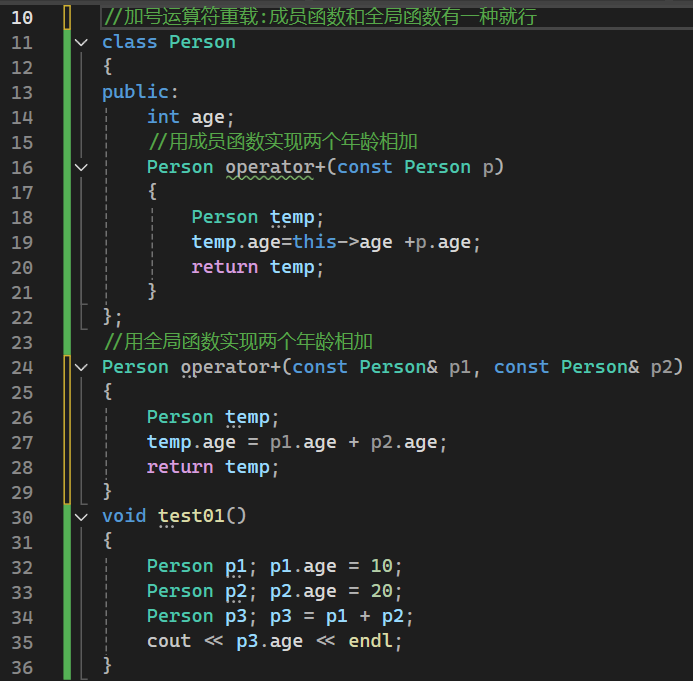
（3）成员函数做友元：同样最好先在外面声明一下类名，防止编译器不认识。

**留有一个疑惑：为什么money还是无法声明?**



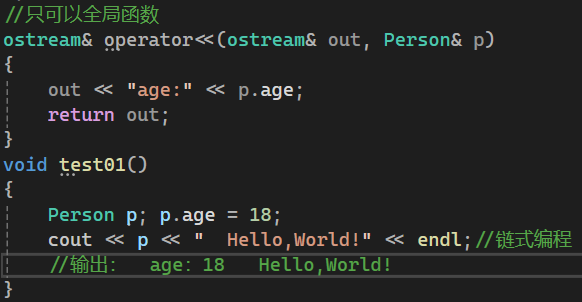
#运算符重载：

1.加号运算符重载（自定义数据类型相加）：



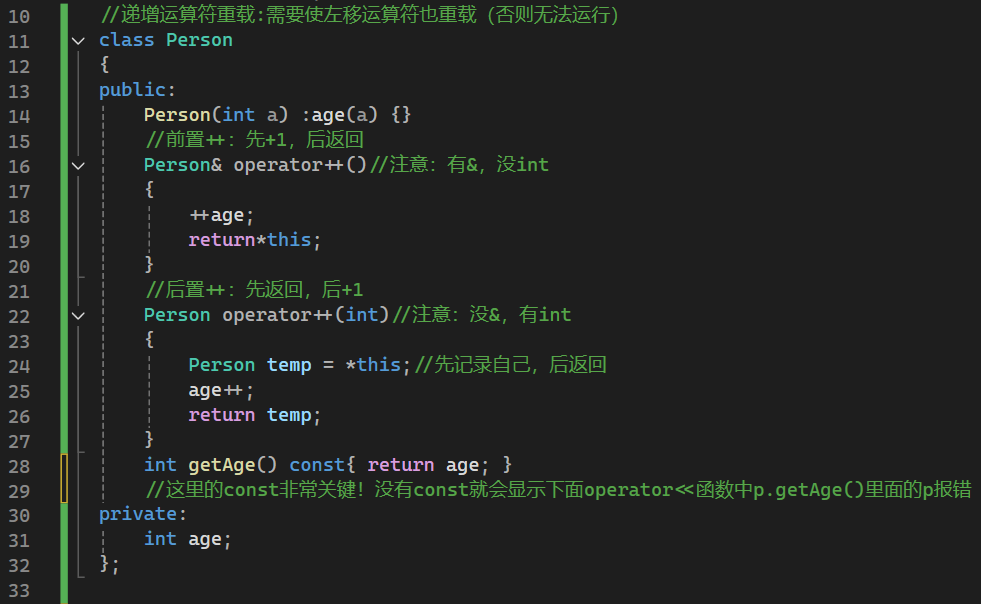
注：上面的忘了写一个&。

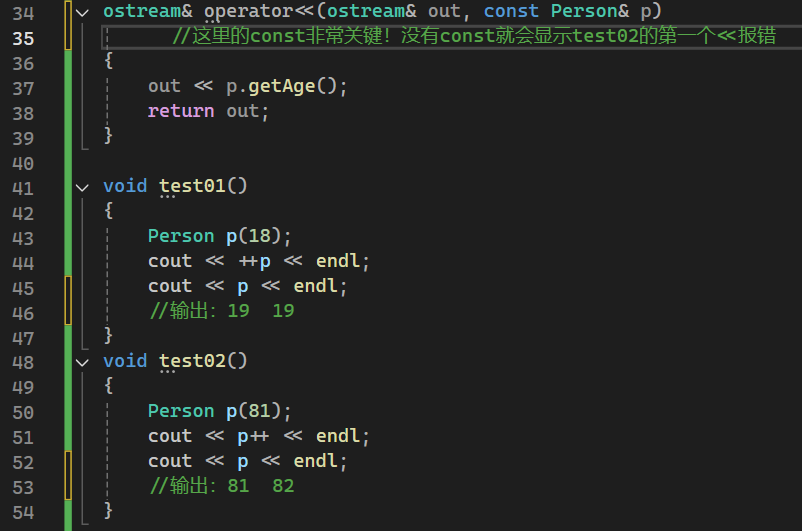
2.左移运算符重载（输出自定义数据类型）：



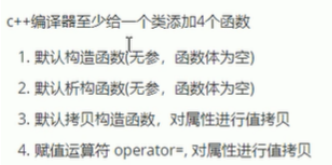
对它的理解：这个运算，经过处理输出了然后返回了out（等价于cout），然后继续呼出后面的（等价于输出了前面之后继续输出）

3.递增运算符重载：分为前置++和后置++：

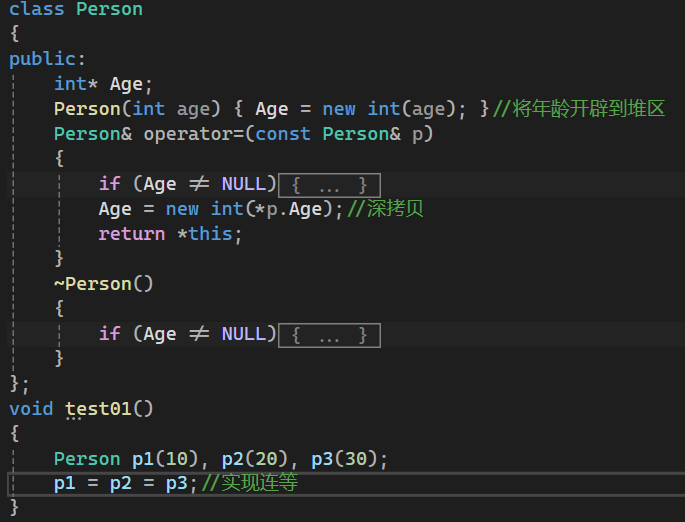




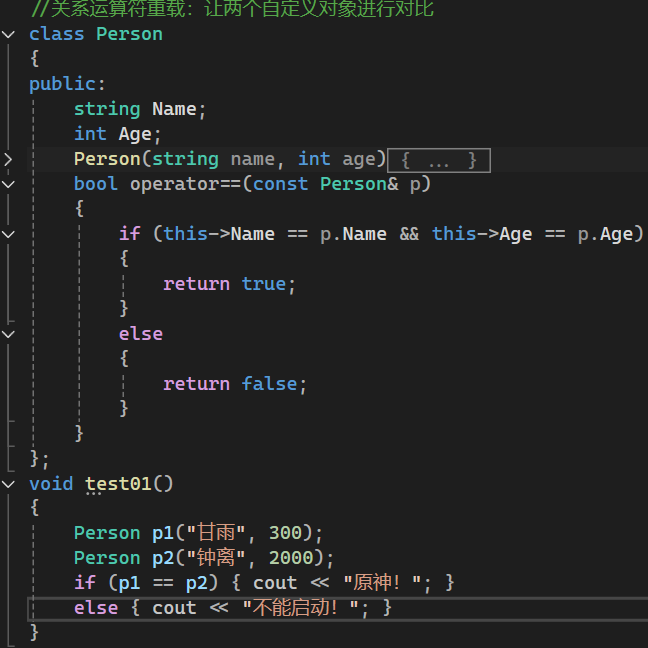
4.赋值运算符重载：



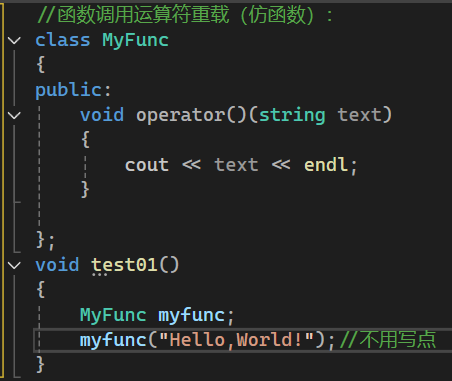
如果类中有属性指向堆区，则也会有深浅拷贝的问题。



5.关系运算符重载（让两个自定义对象进行对比）：



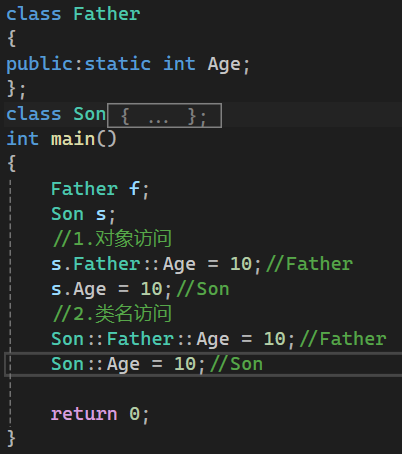
6.函数调用运算符重载（仿函数）（仿函数没有固定写法）：



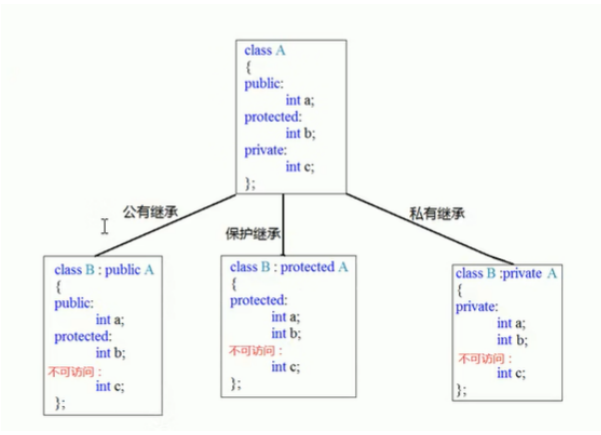
#继承

注意事项：

1.继承中静态成员的两种访问方式：



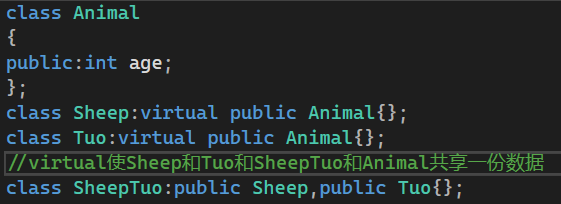
2.三大继承方式：公共继承、保护继承、私有继承



注：这里面的不可访问是指 **继承了但不可访问（被隐藏了）**，而不是没有继承。

3.继承中构造函数和析构函数的调用：**构造函数先父后子，析构函数先子后父。**

4.菱形继承和多继承：记得带virtual（虚继承）：



#多态：

注意事项：

1.静态多态：函数重载和运算符重载，编译阶段确定函数地址（地址早绑定）；

动态多态：子类和虚函数实现实时多态，运行阶段确定函数地址（地址晚绑定）。

2.多态满足条件：

（1）有继承关系；

（2）子类**重写**父类的**虚函数（****）**（父类中的虚函数往往写成纯虚函数，这个时候父类就变成了抽象基类）；

（3）**父类的指针或引用指向子类对象**（常用的是指针，如；记得删除）。

3.抽象类：

（1）类中只要有一个纯虚函数就是抽象类；

（2）抽象类无法实例化对象；

（3）子类必须重写父类（抽象基类）的纯虚函数，否则也属于抽象类。

4.虚析构和纯虚析构：

（1）优点：可以用父类指针释放子类对象（如果不用虚析构可能导致释放不干净，造成内存泄露）；

（2）纯虚析构无法实例化对象。

#fstream库（文件操作）

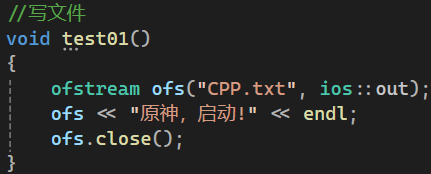
注意事项：

1.

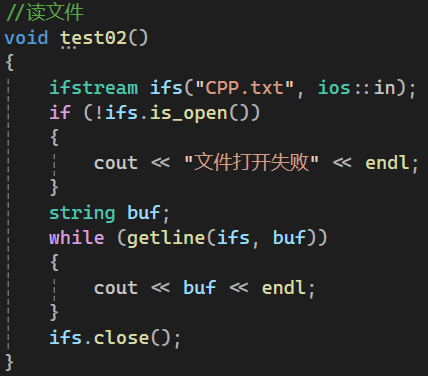


2.文件操作

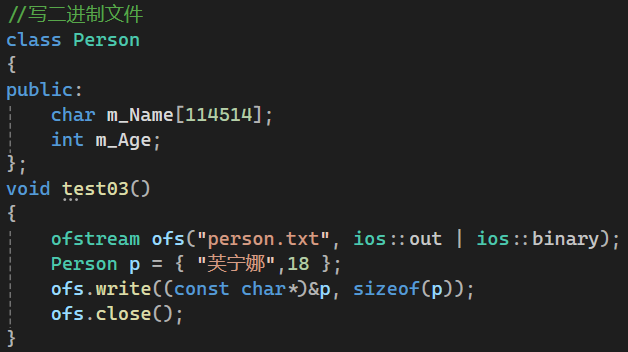
（1）写文件：



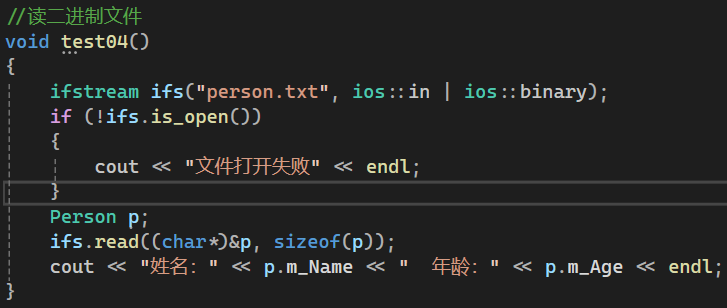
（2）读文件：



（3）写二进制文件：

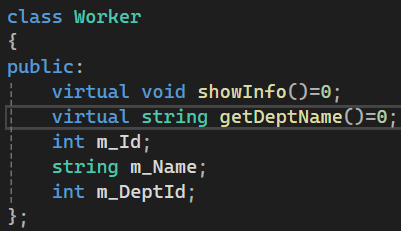


（4）读二进制文件：



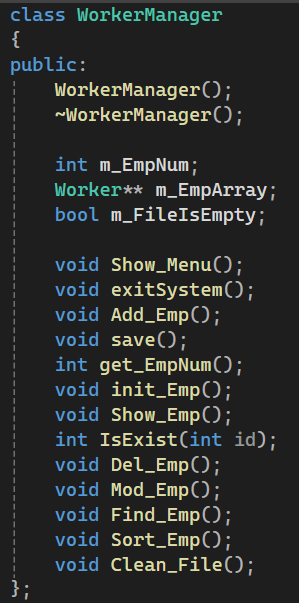
#职工管理系统：

抽象基类：



三个子类employee、manager、boss，分别继承Worker类，并且各有各的有参构造函数（给自己的参数赋值）。

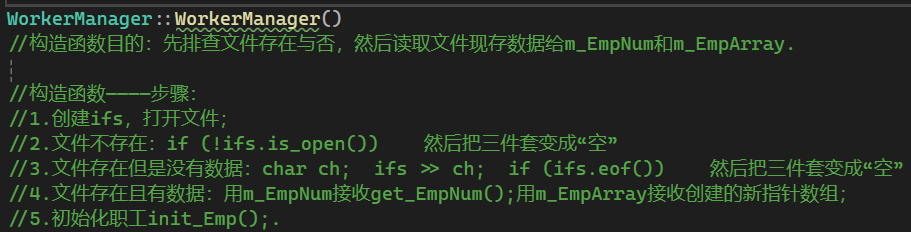
中介类：

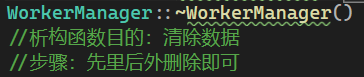


中介类各个函数的目的与步骤：

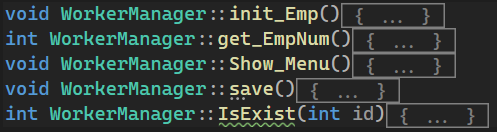
（1）构造函数与析构函数：

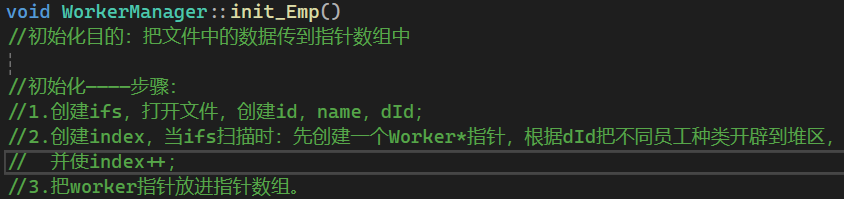


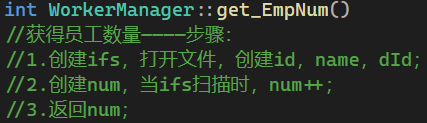




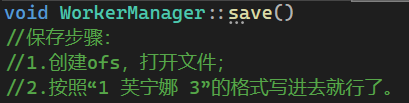
（2）工具函数：













（3）主体函数

