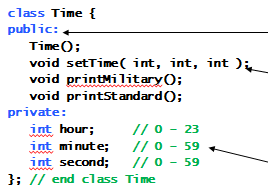
Classes and Data Abstraction （类和数据抽象）

class：

·声明：



Access Authority (访问权限)：

·public：对外接口

·private：只允许类的成员函数访问

·protected：

构造函数：创建对象本身

·名字与类的名字相同

·主要用来在创建对象时初始化对象

·即为对象成员变量赋初始值

·不能有返回类型

·一般情况下只有一个构造函数被调用

·可以设定初始值也可以不设定

【类似这种就是什么都不做不设定初始值的构造函数】

Default constructor（默认构造函数）：

·一个类里面只有一个

析构函数：

·类的成员函数

·delete系统对象：在对象生命周期结束时回收占用的内存空间

·会有默认的析构函数存在（生命期结束时，会自动执行析构函数）

·如果 显示定义：加~

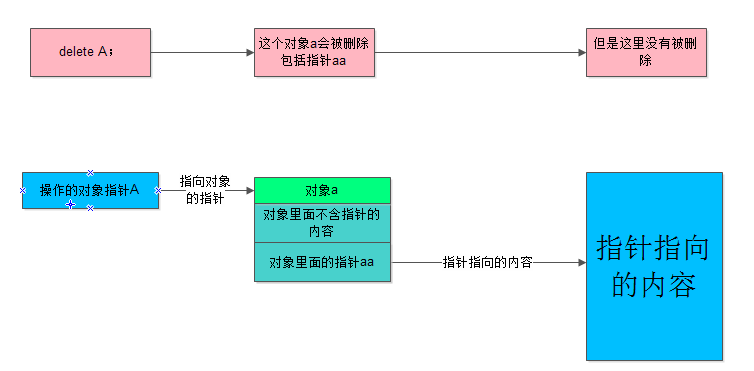
·一定无参数

·无重载 每一个类只有一个析构函数

析构函数的作用正好与构造函数相反。当对象超出其作用范围，对应的内存空间被系统回收或被程序用delete删除时，析构函数将被调用。

当对象为指针对象时， delete指针变量，析构函数只释放本身空间，不能主动释放指针指向的外部的空间

**这里看配图吧：**



构造函数与析构函数的调用：

构造函数和析构函数都是自动被调用，顺序取决于作用域

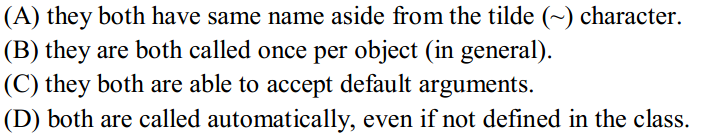
(1) 在全局范围中定义的对象，它的构造函数在文件中的所有函数(包括main函数)执行之前调用。但如果一个程序中有多个文件，而不同的文件中都定义了全局对象，则这些对象的构造函数的执行顺序是不确定的。当main函数执行完毕或调用exit函数时(此时程序终止)，调用析构函数。

(2) 如果定义的是局部自动对象，则在建立对象时调用其构造函数。如果函数被多次调用，则在每次建立对象时都要调用构造函数。在函数调用结束、对象释放时先调用析构函数。

(3) 如果在函数中定义静态(static)局部对象，则只在程序第一次调用此函数建立对象时调用构造函数一次，在调用结束时对象并不释放，因此也不调用析构函数，只在main函数结束或调用exit函数结束程序时，才调用析构函数。

+++++++++++++++++++++++++++++++试卷中曾经出现的题目++++++++++++++++++

以下说法哪个是错误的？



·A 它们有相同的名字，除了符号（~）

·B 它们一般在一个对象里只会被调用一次

·C 它们都能接受默认参数（错误）（析构函数不接受任何参数）

·D 即使没有在类里面定义，也可以自动调用

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

拷贝构造函数必须以引用的形式传递(参数为引用值)

其原因如下：当一个对象以传递值的方式传一个函数的时候，拷贝构造函数自动的被调用来生成函数中的对象。如果一个对象是被传入自己的拷贝构造函数，它的拷贝构造函数将会被调用来拷贝这个对象这样复制才可以传入它自己的拷贝构造函数，这会导致无限循环直至栈溢出（Stack Overflow）。除了当对象传入函数的时候被隐式调用以外，拷贝构造函数在对象被函数返回的时候也同样的被调用。

引用

定义引用的表示方法与定义指针相似，只是用&代替了\*

Point &pt2=pt1; 定义了pt2为pt1的引用。通过这样的定义，pt1和pt2表示同一对象。

需要特别强调的是引用并不产生对象的副本，仅仅是对象的同义词。【所以说其实pt1和pt2是一个地址，改变了一个就会改变另一个？】

* 引用不可以改变指向，对一个对象"至死不渝"；但是指针可以改变指向，而指向其它对象
* 引用的大小是所指向的变量的大小，因为引用只是一个别名而已；指针是指针本身的大小，4个字节
* 指针指向一块内存，它的内容是所指内存的地址；而引用则是某块内存的别名，引用不改变指向。
* 指针用于指向对象，保存的是另一个对象的地址；

引用and指针的区别【答案来自百度】

1. 引用访问一个变量是直接访问，而指针是间接访问。

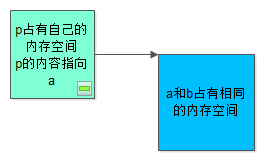
【引用一步到位，指针要经过两步的意思？】

比如：

int a = 5;

int &b = a; //变量名是b，而b也是a，所以b跟a占有相同的内存空间

int \*p = &a; //变量名是p，是一个指针，占有自己的内存空间，然后内容是指向a的指针，要对a操作必须用\*p才能操作，而上面直接用b就可以操作



1. 引用是一个变量的别名，本身不单独分配自己的内存空间，而指针有自己的内存空间。

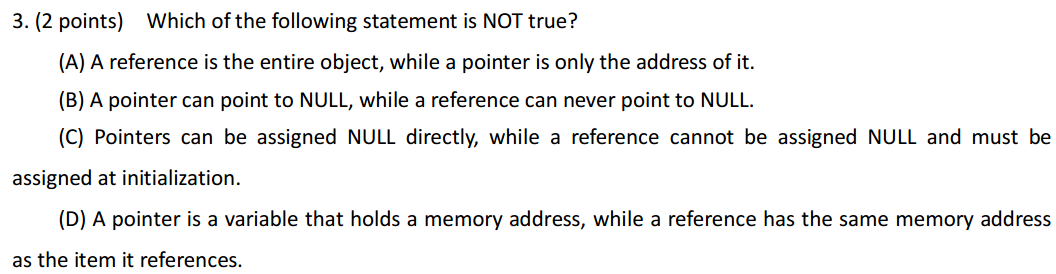
【变量的别名的意思是，通过引用可以直接到达变量的内存地址？】

意思是两个是同样的东西，在内存中占有相同的内存单元，相当于你名字是A，乳名是B，A相当于变量名，B相当于别名，A也是你，B也是你，AB是完全一样的东西。

3.引用在开始的时候就绑定到了一个内存空间(开始必须赋初值),所以他只能是这个内存空间的名字,而不能改成其他的。指针在任何时刻都可以改变自己的指向

+++++++++++++++++++++++++++++试卷中出现过的题目++++++++++++++++++++++++

引用与指针的区别：



（11年）

A引用指向整个对象（？？），而指针仅仅指向对象的地址（错误）

B指针可以指向NULL，引用不能

C指针可以被分配NULL，直接，而引用不能被指定为NULL，必须在初始化时被分配。

D指针是一个变量，可以存储一个内存地址，引用与它指向的对象有相同的内存地址

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

拷贝构造函数：基于同一类的其他对象的构建及初始化

·形参类型为该类类型本身且参数传递方式为按引用传递。

·用一个已存在的该类对象初始化新创建的对象。

 ·每个类都必须有拷贝构造函数：

 ·用户可根据自己的需要显式定义拷贝构造函数。

 ·若用户未提供，则该类使用由系统提供的缺省拷贝构造函数。

 ·缺省拷贝构造函数使用逐位复制方式利用已存在的对象来初始化新创建的对象（相当于赋值= ）。

·拷贝构造函数必须以引用的形式传递(参数为引用值)

其原因如下：当一个对象以传递值的方式传一个函数的时候，拷贝构造函数自动的被调用来生成函数中的对象。如果一个对象是被传入自己的拷贝构造函数，它的拷贝构造函数将会被调用来拷贝这个对象这样复制才可以传入它自己的拷贝构造函数，这会导致无限循环直至栈溢出（Stack Overflow）。除了当对象传入函数的时候被隐式调用以外，拷贝构造函数在对象被函数返回的时候也同样的被调用。

【简单一句话，若不是传地址，递归就是无限循环】

const

当变量只作为值存在 而不需要进行改变时 在编程习惯上 加const

const指定的对象是不能修改的（要有初始值） 只要尝试改变const的都会被看做编程语法错误

只有被声明为const的成员函数才能被一个const类对象调用

【就是有const的函数才能调用类中有const的数据？】

比如你写一个Student类，然后这样定义：

const Student stu;

那么，这个stu就是一个常对象，它只能调用const的函数，不能调用不是const的成员函数，自己打一段代码编译一下就明白了。

const对象只能调用const函数

const函数只能调用const对象 不能改变值

const函数是不能改变对象的内容，比如还是Student类

Student stu;

stu.getId() 这个获取学号的过程，stu的内容是不变的。;

对于在类体之外定义的const成员函数我们必须在它的定义和声明中同时指定关键字const

格式



当只是做返回值的函数时 返回值函数 设为const（好习惯）

构造函数和析构函数 不能加const 因为这两个函数要改变变量的内存地址

构造函数要创建对象，析构函数要删除对象，对象本身肯定要发生改变，所以不能是const的，然后内存方面，创建对象要分配内存，删除对象要回收内存。

不涉及对象修改的函数也定义为const

就是我前面说的，如果不改变stu自己的内容，那么就定义为const函数，如果要改变到对象的内容，就不能定义成const函数

不能采用在构造函数中对常数据成员赋初值的方法 要采用成员初始化表

成员初始化表

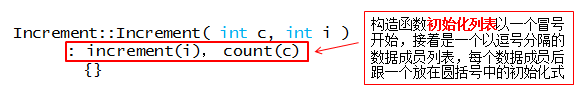
成员初始化表是由逗号分隔的成员、名字实参对

成员初始化表跟在构造函数的原型后，由冒号开头

成员初始化表和使用构造函数对数据成员进行赋值的操作结果是一样的，但前者是初始化后者是赋值

任何数据成员的初始化都可以使用这种方式 但const的赋值方式是只有这一种

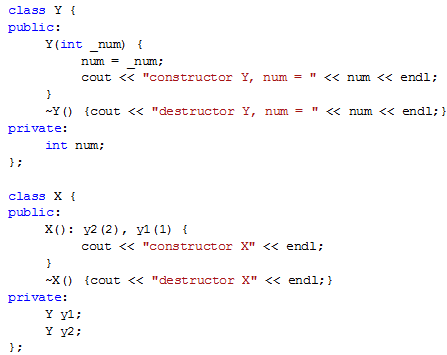
例子如下



组合（Composition）

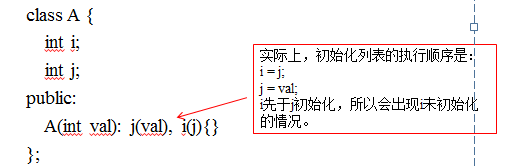
把对象作为类的成员 对象之间的嵌套关系 对象中包含对象

例子如下（eg. 一个类里面再嵌套一个类）



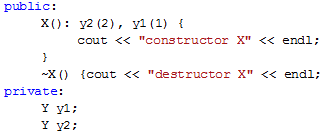
初始化顺序【按照定义的先后顺序初始化】

就是你在A里面是先声明i再声明j的，那么无论你的初始化列表是先初始化i还是j，它都是按照你上面的声明顺序来进行初始化，就是先i后j，跟你的初始化列表的初始化顺序无关。



构造函数从里到外初始化

析构函数从外到里初始化



内置或复合类型的变量依赖于其作用域：局部作用域则不被初始化，全局作用域则被初始化为0【什么叫内置或复合类型？】

内置类型是指已经给你预留好的变量类型，像int， double， long之类的，复合类型是指数组、指针、结构体、类这几种

在main函数外面声明一个变量，并且在main函数里面声明另外一个变量，调试看不对这两个变量进行赋值的时候，这两个变量的初始值是多少。

而变量的初始化顺序是按照其在类中声明的顺序。所以虽然在成员初始化列表中y2的位置比y1前，但是由于y1比y2早声明，所以先初始化y1再初始化y2，然后再初始化该类对象，析构的时候顺序则相反。

友元

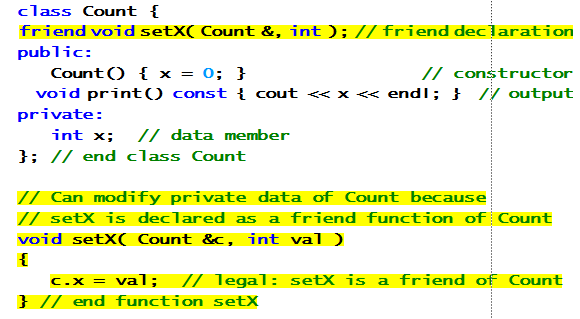
友元声明以关键字friend 开头 它只能出现在类的声明中

友元函数：声明在类内 不是成员函数 定义在类的外部 不写作用域 可以访问私有对象 保护对象

属性（性质）：主动给予 不对称 不传播

在A类里面声明函数friend void f(); 就是声明f是A类的友元函数，就是说A告诉编译器，f是A的朋友，所以f可以访问A里面私有成员。

使用！



这里友元函数在类里面声明

setX定义时，可以直接用变量val的值改变count类的私有成员x的值

this

·指向调用对象的地址，它的值是当前被调用的成员函数所在的

·函数的返回值是对象的地址

·其他函数可以访问它的对象

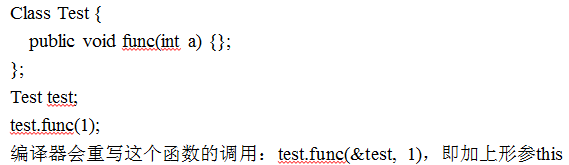
·成员函数里，this指针访问对象私有数据成员

·作用：级连赋值

·this引用被隐式地用于引用一个对象的实例变量和方法。另一个this引用的用法是实现“连接方法调用”，也叫“级连方法的调用”。如：t.setHour(18).setMinute(30).setSecond(22);

“.”运算符从左至右的结合，因此上述表达式首先计算t.setHour(18)，然后此方法返回一个对象t的引用。

·每个成员函数(除了static成员函数)都有一个额外的、隐含的形参this



【是把test的地址传给形参this 然后按this的指针引用其他数据成员】

额，这里就是想告诉你，类里面有个this指针，指向它的对象本身，谁调用函数，this指向的就是谁，这个谁指的是对象，而不是整个类。

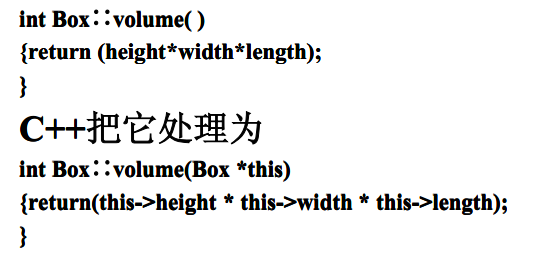
比如你再定义一个Test test2;然后调用：

test2.func(1);

那么这里的this指针就是指向test2，而不是原来的test1了

这个test.func(&test, 1)到这里就变成test2.func(&test2, 1)

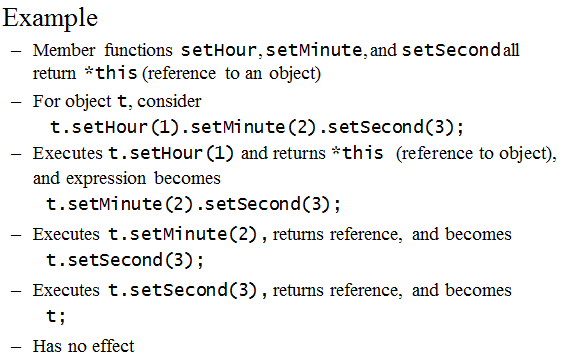
不管是&test还是&test2，到函数的定义里面都是传递给一个叫this的指针变量，所以可以用this访问对象的内容。



this指针是隐式使用



使用this指针可以一次性给hour minute second 赋值



要在定义set的时候把返回值改为 \*this

因为返回的是引用

前面说了，this指针是指向那个对象本身，所以返回值是那个对象的引用，具体的到时候做作业就会有了…

定义的时候要这样声明定义

这里是声明了一个setTime的函数，然后它的返回值是Time类型的引用。

主函数中使用：

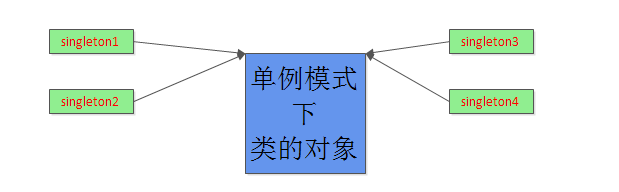


单例模式（sicily上出现过）

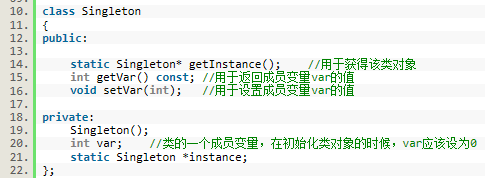
其意图是保证一个类仅有一个实例，并提供一个访问它的全局访问点，该实例被所有程序模块共享。

【就是一个类只有一个正常的对象a，其他对象的数据都跟a一样？】

就是它其实只有一次创建的过程，其他的指针指向这个对象而已



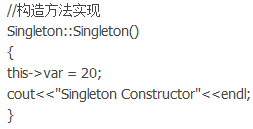
单例模式的要点有三个；一是某个类只能有一个实例；二是它必须自行创建这个实例；三是它必须自行向整个系统提供这个实例。



【为什么构造函数要私有？】

私有是不让外部直接构造对象，如果你不定义为私有的，那么main函数里面可以用new来构造对象，就跟不用单例模式一样，可以构造很多个对象，那就不是单例模式了…

百度的：



自己写的：

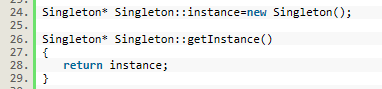


【用不用this都可以过的。但是据说单例模式就是让我们尝试用this？

用this和不用this的区别在哪？】

在这里用不用this是没有区别的，this只是一个指向对象本身的指针而已，需要用到本对象的指针或者引用时，才必须用this…其他情况不用this是没有问题的…

//初始化静态成员

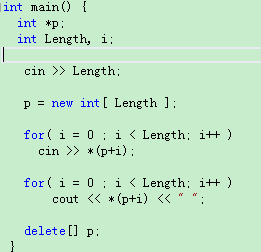


【就是给它开辟个内存空间？？】

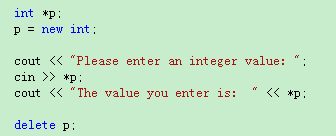
是的，创建一个对象就是要给它分配一块内存空间…判断两个对象是不是同一个对象的标准也是从内存空间来判断的，占据相同的内存空间才是同一个对象…

new 和 delete的使用

·使用new和delete动态分配和释放内存空间： 



·也可仅动态分配一个变量：



new的使用：

指针 = new 类型名；//动态创建一个变量

指针 = new 类型名[ 数组长度] ；//用于动态分配数组

指针 = new 类型名（初始化表）；//动态创建对象

 ·初始化表及其括号为可选

 ·类型可为基本类型，也可为类类型，若为类类型，则初始化表相当于将实际参数传递给该类的构造函数

 ·new运算返回一个指针，指向分配到的内存空间

 ·若内存分配失败，则返回NULL（0 ）

注意：

·动态分配的内存空间使用完毕后，应该“ 释放” 掉这块空间，即使得这块内存空

间可以被操作系统回收以作它用。

·假如程序中动态分配了很多内存空间，但使用完毕后都不释放，则这些空间无法用于存储别的数据，造成严重的内存浪费。

delete的使用：

 ·用new分配的内存在不再使用时，要用delete释放。

 ·delete释放的是指针所指对象占据的内存。

 ·用delete释放空间后，指针的值仍是原来指向的地址指针已无效（重复释放将出错）。

 ·delete对象指针，会调用该对象的析构函数。

·若使用new运算分配的是数组（尤其是类类型对象的组），则用delete释放时必须注意加上方括号。

注意：

·由于delete一个指针即等同于释放掉该指针所指向的内存空间，而在C++中同一内存空间的释放只能进行一次；

·已delete过的指针再次delete会产生错误。因此，最好不要让多个指针指向同一个对象（多个指针同时向同一内存空间的现象称为“ 指针别名” ）