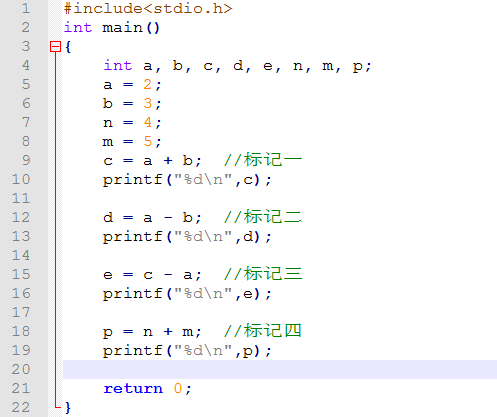
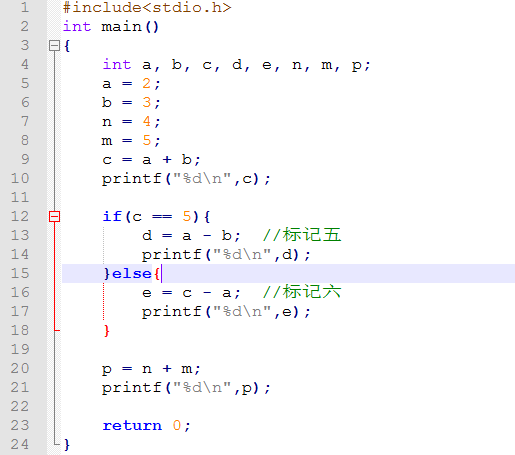
Foreword：最近项目需求要我去看别人看的代码质量，我之前的对代码质量的理解还在表层，比如：规范（缩进，逗号后面记得空格等），看起来工整等等。

华清的时候老师推荐过《c/c++高质量编程》，夏超也在我在华清学到数据结构的时候推荐过这本书，当时只是简单的看了一下，但是记住了一个词，“高内聚低耦合”。后来笔试题还碰到了，选正确了，但是不知道是什么意思。工作了，刚到公司老总就一直在强调代码质量，到现在强调不止四五次了，公司推荐的是这本书《代码大全2》.这儿我通过资料的学习就简单讲一下高内聚和低耦合，尽管薛雨之前讲过耦合了。额，废话太多了。

**内聚：**

**概念：** 内聚性主要是衡量一个模块的各个组成元素结合的紧密性，紧密性越高，内聚性越高，模块独立性也就越高。理想的高内聚是指功能明确，单一，即一个模块只做一个事情。

Example：



我通过上面的例子讲一下各种内聚，当然，命名方式请勿学习，下面的几种内聚模式：**越往下，内聚性越高。**

**偶然内聚：**各个成分间并没有联系，只是把各种功能集合在一起。主要是为了单纯完成任务或项目。内聚性最低的一种内聚。标记一和标记四就是毫无联系，所以把他们两个卸载一起属于偶然内聚。

Xy：通过标记一和标记四的例子来说明偶然内聚，还特意使用不同颜色，让人一下子知道你的重点在于举例来说明概念。这是很好的做法。不过你的错别字不忍直视。

**逻辑内聚：**仅仅只是因为逻辑功能相关才组合在一起，其本身并没有什么联系，这种比较常见。可以写一个仅仅只有逻辑的程序，不同逻辑来调用不用的子程序来提高内聚性。标记五和标记六写在一起就属于逻辑内聚。

**时间内聚：**把需要同时执行的动作组合在一起形成的模块为时间内聚模块。（代码大全归类为临时内聚，例子是startup（），shutdown（））；//我理解还不够透彻，主要感觉比较少见，一般都是顺序的。

**过程内聚：**模块具有多个功能，这些功能需要按一定的步骤一次完成。即使两者功能之间没有数据进行传递。这个例子暂时还没想好，不过字面应该够理解。

**通信内聚：**程序中不同的操作使用了相同的数据，仅仅因为这个关系（不存在其他联系）而写在一个程序里。标记一和标记二就是属于通信内聚，都使用了a和b。

**顺序内聚：**是指一个功能的输出正好是另一个的输入，因为这个关系把他们合起来。标记一和标记三就属于顺序内聚。其中标记三需要的c是由标记一产生的。

**功能内聚：**只完成单一的功能，里面单独的标记就属于功能内聚，已经不能再拆分了。比如标记一可以写成单独的求和函数。这样就属于功能内聚，也是最理想的内聚，耦合也低，高内聚低耦合。

还有，谢敏在薛雨写的那篇<解耦>里面说没理解复用，其实最常见的复用就是printf函数 ，这个函数哪都在用。上面例子的标记一写成的求和函数很多地方也可以用到，只不过有局限性。只能整型求和。

Xy：这里多加说明一下，例如整型求和可以写成：

ULONG OPT\_IntGetSum(ULONG \*ulSum, ULONG ulNumOne, ULONG ulNumTwo);

这样下次对于整型求和就可以直接代入函数：

ULONG ulXmSb = 1;

ULONG ulYbSb = 1;

ULONG LiangGeErB;

if (OK == OPT\_ IntGetSum(&LiangGeErB , ulXmSb, ulYbSb))

{

printf(“The Summary is %d”, LiangGeErB);

}

函数返回值OK的情况下，在里面进行判断成功后的操作。

也可以是：

if (ERR == OPT\_ IntGetSum(&LiangGeErB , ulXmSb, ulYbSb))

{

printf（“出错！！！两个人凑到一起不是SB，直接退出！”）；

return ERR；

}

先到这儿，明天再写耦合。

参考链接 <http://wenku.baidu.com/view/958424848762caaedd33d40a.html?re=view>

<http://baike.baidu.com/link?url=F1jHJYqO91FUITi35tHBjR8F5YDIoAPbn-9oJxyxhhoBGZl049M2PIaWk0tL_gNu> <http://wenku.baidu.com/link?url=4Q9W_uPmVeLLUtBdZDG_GXRh3KPVpqD_N6LcKyEmOFtXSiZPL1-9xYHCwVIMav_KU7-v0b8TKF-VuN2E1X5BzxqnfNu8pDQLcluhegGAHDy>

xxxxxxxxxxxxxxxxxxx