Author：Ananaxs

Time：2016-03-17

Foreword：

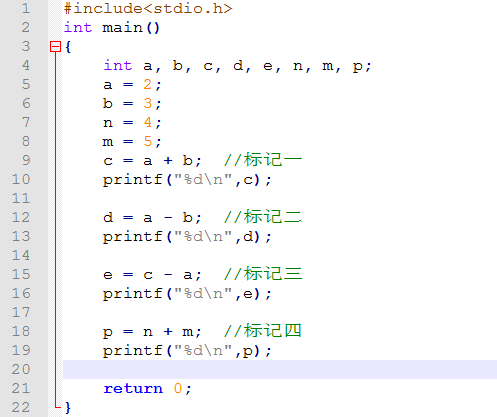
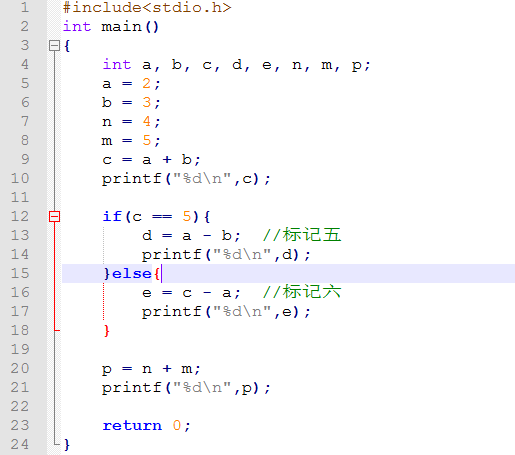
最近项目需求要我去看别人看的代码质量，我之前的对代码质量的理解还在表层，比如：规范（缩进，逗号后面记得空格等），看起来工整等等。

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自身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，所以把他们两个写在一起属于偶然内聚。

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。标记一和标记二写在一起就是属于通信内聚，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。标记一和标记三写在一起就属于顺序内聚。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；

}

Yb：嗯，把返回值留出来判断程序是否成功，这是很常规也是很正确的做法，但是参数排序上有点小问题，根据代码大全上描述的，函数参数应该按 输入-修改-输出 （修改：既是输入也是输出的参数）的顺序排列，这种排列方法暗含函数内部执行顺序先输入数据-修改数据-输出数据，所以上面的顺序需要改一下，还有命名规则的问题，书上建议是 输入-修改-输出 前面分别加上 i\_ m\_ o\_ 不嫌麻烦可以加 Input\_ Modify\_ Output\_ 我觉得这个参数命名规则倒是很少见到。所以我就提议一下，不用这么按他这么来（ps:以后我回复就这个颜色了，表抢我的）

**耦合:**

**概念：**前面内聚性是模块指自身的，耦合性则是指模块之间的联系，耦合性是对一个软件结构内不同模块间的相互依赖程度的度量，耦合性越弱（即模块间联系越少），模块独立性越高。以下几种耦合：**越往下，耦合性越弱。**

**内容耦合：**1.一个模块直接修改另一个模块的内容，2.一个模块不通过正常入口而转入另一个模块，3.两个模块有部分程序代码重叠，4.一个模块有多个入口，其实这种耦合完全是特指汇编吧，汇编不熟悉想不出例子。第1点还有迹可循，比如两个模块调用了共享内存等，但是后来发现下面这个分类(**公共耦合**)，就是特指全局共享区的，第2，3，4想不出来了，这种耦合是：耦合性最强的，模块独立性最弱的。（[百度百科：现在大多数高级程序设计语言已经设计成不允许出现内容耦合，他一般出现在汇编语言中](http://baike.baidu.com/link?url=qB7RiQm69RIvnDWWJREdoaSfyA0nZKM4Lb0tSwYtncv0c29MVz9qn8CmMFfk8KAxdxytQa_WD01h_uCljsAXdq#2_8)）

**公共耦合：** 一组模块访问的同一个公共数据坏境，公共数据环境可以是全局数据结构、共享的通信区、内存的公共覆盖区等，如果模块只是从公共环境里取出数据，这种属于松散的公共耦合，如果模块既向公共数据环境输入数据又从公共数据环境取出数据，这属于较紧密的公共耦合。

公共耦合会引起以下问题：

1. 无法控制各个模块对公共数据的存取，严重影响了软件模块的可靠性和适应性。

2. 使软件的可维护性变差。若一个模块修改了公共数据，则会影响相关模块。

3. 降低了软件的可理解性。不容易清楚知道哪些数据被哪些模块所共享，排错困难。

一般地，仅当模块间共享的数据很多且通过参数传递很不方便时，才使用公共耦合。

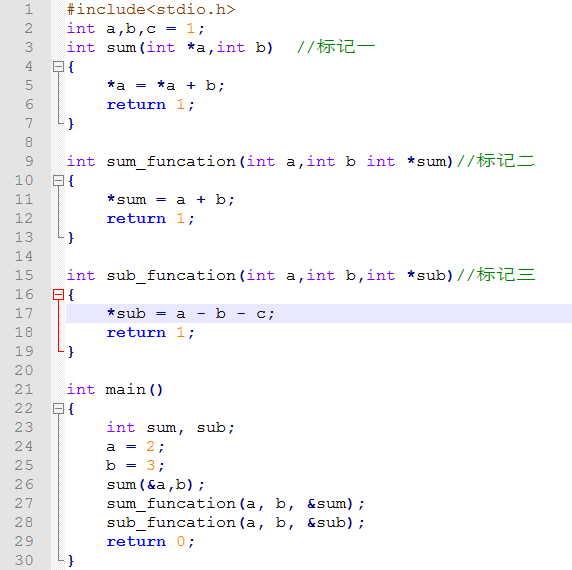
Example：

1.进程间通信中的共享内存通信算是比较常见的例子了吧

Ps：本来想说标记一和标记二是紧密型公共耦合，标记二和标记三是松散型的公共耦合，但是自己揣摩发现，公共耦合一般是特指同组模块访问公共环境（没说包括简单全局变量），但是发现下面外部耦合访问的是简单全局变量且不通过参数传递的才叫外部耦合，那通过参数传递的简单全局变量是该归那一类呢，想想还是归公共耦合吧

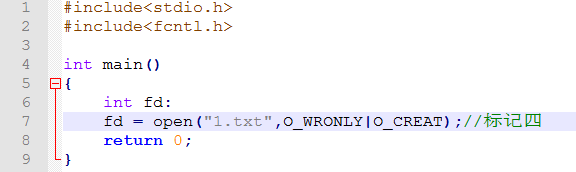
**外部耦合：** 一组模块都访问同一全局简单变量，而且不通过参数表传递该全局变量的信息，则称之为外部耦合。

Example：下图中标记三就是直接调用了c这个全局变量，并非传入的参数，若有多种标记三这种模块，那他们之间的联系就是外部耦合的吧。



**控制耦合：**模块之间传递的不是数据信息，而是控制信息例如标志、开关量等，一个模块控制了另一个模块的功能，这个你们应该常用到的，标志位，

Example：



Main函数通过传入标志位改变open函数生成的fd；所以main和open属于控制耦合。

**标记耦合：**调用模块和被调用模块之间传递数据结构而不是简单数据，同时也称作特征耦合。表就和的模块间传递的不是简单变量，而是像高级语言中的数据名、记录名和文件名等数据结果，这些名字即为标记，其实传递的是地址。

Example:

见上图，main函数传入的是文件名1.txt 这时他们之间传递的不是简单的数据而是具有特征的数据。其实和公共耦合还是有相同之处的。

**数据耦合：**调用模块和被调用模块之间只传递简单的数据项参数。这个比较好理解，就是传值进去，这个值不应该是变量的地址，简单的说就是你传值进去，他的模块怎么改都影响不到你的模块，这种是比较常用的耦合。

**非直接耦合：**两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的。耦合度最弱，模块独立性最强。

**总结：**

个人觉得耦合要难理解一点，可能理解有稍许偏差，总的来说都是遵循“一个模块，一个功能”的原则，主要就是功能尽量内聚，功能间联系尽量少。高内聚低耦合

**参考链接 ：**

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

<http://baike.baidu.com/link?url=F1jHJYqO91FUITi35tHBjR8F5YDIoAPbn9oJxyxhhoBGZl049M2PIaWk0tL_gNu>

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

http://baike.baidu.com/link?url=DUicDNFu3-V-yxIvlDaMG1Ki9XnGoWnJ-FhDUFn1lEBUxANFlUhJlGHGzOdeppBCUz6gDBeqKrBO4VjHOVvmMa