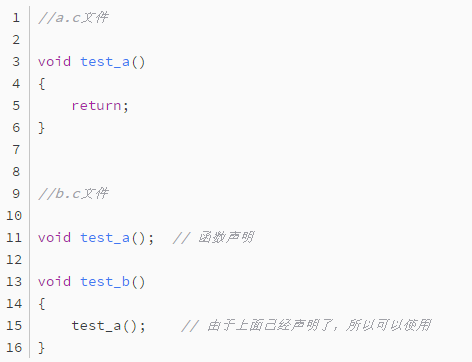
# 深入理解include预编译原理

**1.  #include 命令的作用**

**1.1  什么情况不使用 include**

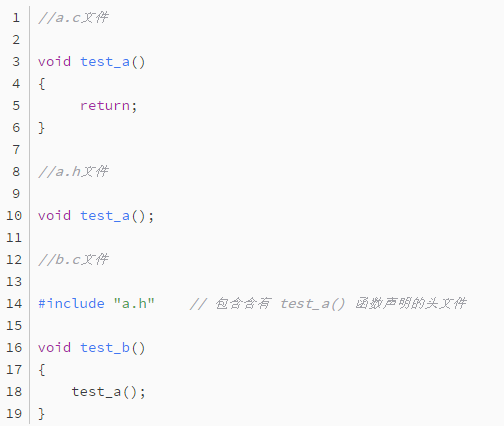


其实，这样的工程，可以不用使用 include 预编译命令。

**1.2  什么情况使用 include**

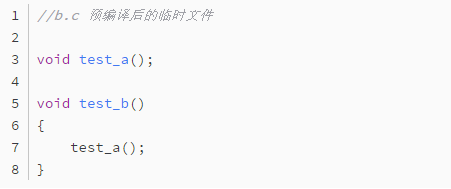
如果工程里面的函数特别多，那么按照上面的做法，则必须在每一个 .c 文件的开头列出所有本文件调用过的函数的声明，这样很不高效，而且一旦某个函数的形式发生变化，又得一个一个改 .c 开头的函数声明。

因此，#include 预编译命令诞生。



**1.3  #include 起到什么效果**

上述代码在编译器进行预编译的时候，遇到 #include "a.h" ，则会把整个 a.h 文件都copy到 b.c 的开头，因此，在实际编译 b.c 之前，b.c 已经被修改为了如下形式：

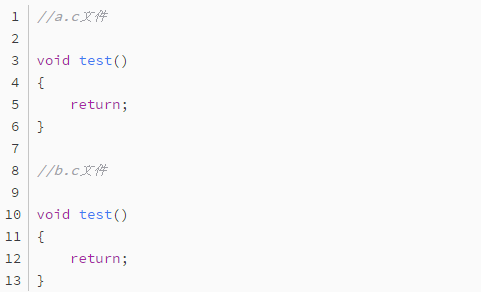


由此可见，得到的效果和手动加 test\_a() 函数声明时的效果相同。  
  
#tips# 在[Linux](http://lib.csdn.net/base/linux)下，可以使用 gcc -E b.c 来查看预编译 b.c 后的效果。

**2. static 关键词的使用**

**2.1  什么叫函数重复定义**

我们经常会遇到报错，说变量或者函数重复定义。那么，在此，首先我举例说明一下什么叫函数的重复定义。



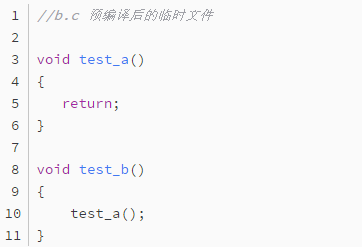
那么，在编译的时候是不会报错的，但是，在链接的时候，会出现报错：  
multiple definition of `test'，因为在同一个工程里面出现了两个test函数的定义。

**2.2  在.h里面写函数实现**

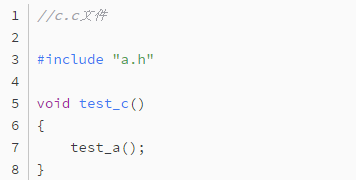
如果在 .h 里面写了函数实现，会出现什么情况？



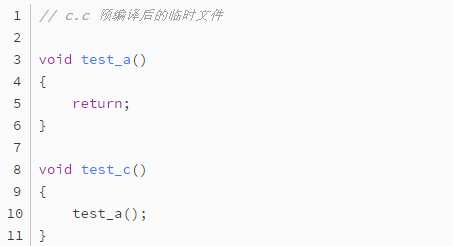
预编译后，会发现，b.c 被修改为如下形式：



当然，这样目前是没有什么问题的，可以正常编译链接成功。但是，如果有一个 c.c 也包含的 a.h 的话，怎么办？



同上，c.c 在预编译后，也形成了如下代码：

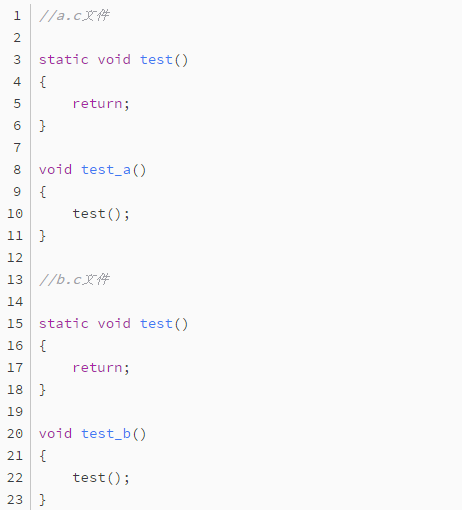


那么，在链接器进行链接（link）的时候，会报错：  
multiple definition of `test\_a'  
  
因此，在 .h 里面写函数实现的弊端就暴露出来了。但是，经常会有这样的需求，将一个函数设置为 内联（inline） 函数，并且放在 .h 文件里面，那么，怎样才能防止出现上述 重复定义的报错呢？

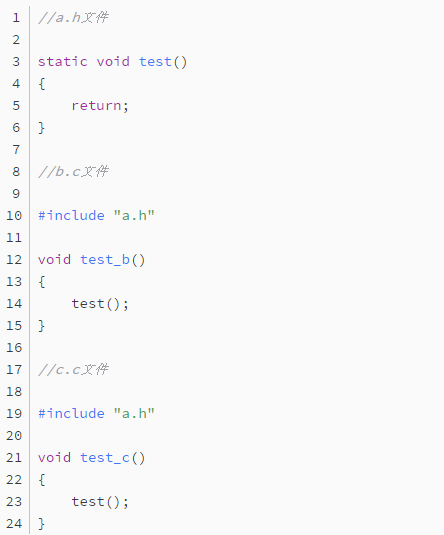
**2.3  static 关键词**

应对上面的情况，static关键词很好地解决了这个问题。

用static修饰函数，则表明该函数只能在本文件中使用，因此，当不同的文件中有相同的函数名被static修饰时，不会产生重复定义的报错。例如：



编译工程时不会报错，但是test()函数只能被 a.c 和 b.c 中的函数调用，不能被 c.c 等其他文件中的函数调用。  
  
那么，用static修饰 .h 文件中定义的函数，会有什么效果呢？



这样的话，在预编译后，b.c 和 c.c 文件中，由于 #include "a.h" ，故在这两个文件开头都会定义 static void test() 函数，因此，test\_b() 和 test\_c() 均调用的是自己文件中的 static void test() 函数 ， 因此不会产生重复定义的报错。  
  
因此，结论，在 .h 文件中定义函数的话，建议一定要加上 static 关键词修饰，这样，在被多个文件包含时，才不会产生重复定义的错误。