　assert宏的原型定义在<assert.h>中，其作用是如果它的条件返回错误，则终止程序执行，原型定义：

#include <assert.h>  
void assert( int expression );

　　assert的作用是现计算表达式 expression ，如果其值为假（即为0），那么它先向stderr打印一条出错信息，然后通过调用 abort 来终止程序运行。请看下面的程序清单badptr.c：

[复制代码](javascript:void(0);)

#include <stdio.h>  
#include <assert.h>  
#include <stdlib.h>  
int main( void )  
{  
 FILE \*fp;  
   
 fp = fopen( "test.txt", "w" );//以可写的方式打开一个文件，如果不存在就创建一个同名文件  
 assert( fp ); //所以这里不会出错  
 fclose( fp );  
   
 fp = fopen( "noexitfile.txt", "r" );//以只读的方式打开一个文件，如果不存在就打开文件失败  
 assert( fp ); //所以这里出错  
 fclose( fp ); //程序永远都执行不到这里来  
 return 0;  
}

[复制代码](javascript:void(0);)

[root@localhost error\_process]# gcc badptr.c   
[root@localhost error\_process]# ./a.out   
a.out: badptr.c:14: main: Assertion `fp' failed.

　　已放弃使用assert()的缺点是，频繁的调用会极大的影响程序的性能，增加额外的开销。在调试结束后，可以通过在包含#include <assert.h>的语句之前插入 #define NDEBUG 来禁用assert调用，示例代码如下：

#include <stdio.h>  
#define NDEBUG  
#include <assert.h>

**用法总结与注意事项：**

　　1）在函数开始处检验传入参数的合法性如：

[复制代码](javascript:void(0);)

int resetBufferSize(int nNewSize)  
{  
　　//功能:改变缓冲区大小,  
　　//参数:nNewSize 缓冲区新长度  
　　//返回值:缓冲区当前长度   
　　//说明:保持原信息内容不变 nNewSize<=0表示清除缓冲区  
　　assert(nNewSize >= 0);  
　　assert(nNewSize <= MAX\_BUFFER\_SIZE);  
　　...  
}

[复制代码](javascript:void(0);)

　　2）每个assert只检验一个条件，因为同时检验多个条件时，如果断言失败，无法直观的判断是哪个条件失败，如：

　　不好：

assert(nOffset>=0 && nOffset+nSize<=m\_nInfomationSize);

　　好：

assert(nOffset >= 0);  
assert(nOffset+nSize <= m\_nInfomationSize);

　　3）不能使用改变环境的语句，因为assert只在DEBUG个生效，如果这么做，会使用程序在真正运行时遇到问题，如：

　　错误：

assert(i++ < 100);

　　这是因为如果出错，比如在执行之前i=100，那么这条语句就不会执行，那么i++这条命令就没有执行。

　　正确：

assert(i < 100);  
 i++;

　　4）assert和后面的语句应空一行，以形成逻辑和视觉上的一致感。  
  
　　5）有的地方，assert不能代替条件过滤。