我们知道头文件<assert.h>唯一的目的就是提供宏assert的定义。让我们在程序不应该出现的问题的地方使用这个宏进行断言。如果一处断言被证明非真，我们希望程序在标准错误流中输出一条适当的提示信息。并且使程序异常终止。

看到这里也许会有很多疑问，什么算是合适的地方？要是在不合适的地方使用断言会怎么样？如何写到标准错误流中的？下面我们会一点点来谈这方面的内容。

在编写代码时，使用断言代码的确会变得更简单，但是有之而引发的问题，其实还是比较严重，前面说到，断言是用来断言程序不应该出现问题的地方使用，若这个问题出现了，断言成功，试想这个问题是在用户端使用过程出现，那结果将是非常糟糕的，我们不希望，用户在使用过程，突然中止程序，应该是发出一个信息，提示用户来进行相关的操作。

但是这个对编写代码的人员来说，却是十分方便的，可以很快速的找到问题所在。 那么，如何在程序写好以后如何让断言不起作用？？也许能想得到的方法是替换到assert,或者注释掉assert，但是C标准库中给我们提供了十分方便的操作。 使用宏定义 NDEBUG ，至于这里为什么使用NDEBUG 这里，目前我还不是很清楚。但是他的作用是毋庸置疑的。

关闭断言

#define NDEBUG  // 让assert不起作用

#include<assert.h>

打开断言

#undef NDEBUG//让assert起作用 #include <assert.h>

这里值得一提的是，即使宏NDEBUG已经被定义了。我们仍然可以安全地定义它。这是一个良性的重定义。

这里简单说一下良性重定义的含义:在一个翻译单元内包含这样的宏定义的多个实例不会造成任何损失。在标准C中允许这样的宏的良性重定义。具有相同宏名的两个定义必须具有相同的记号序列。它们只能是记号之间的空白不同（这里的空白指空格符和制表符）。没有必要阻止包含两个符合这样情况的定义。

但是，你必须在多个地方提供相同的定义。这是一个比较麻烦的维护问题。 这里提供两种解决办法：

1. 在多个地方编写相同的定义。定义改变时，就要找出这个定义的所有实例。
2. 把定义 放在一个独立的头文件中，给这个文件起一个不会和程序员创建的文件名冲突的名字。在每个用到该定义的头文件中包含这个头文件

在大多数情况下，建议选择第二种，因为它简化了使库适用于不同的编译器所做的工作。（这里特殊情况，我还没有想到。）

下面我们来看一下<assert.h>的实现

为了对NDEBUG做出正确的回应，该头文件一定要有一个总体结构：

#undef assert /\*remove existing defintion\*/

#ifdef NDEBUG

#define assert (test) ((void) 0) //passive form

#else

#define assert(test).... //active form

#endif

如果当前的宏assert的定义不存在，那么开头的#undef 预处理指令就不会有任何副作用。总是可以#undef 一个名字，不论它是不是被定义为一个宏，（这里可以把它看做是

良性取消定义）但是，如果定义可能改变的话，这条处理指令还是很有必要的。

这里简单说一下#undef命令。

#undef name

这个命令使预处理器忘记name的所有宏定义。取消一个当前未定义宏的定义并不是错误。当一个名称被取消定义之后，就可以向他提供一个全新的定义（使用#define），而不会产生任何错误。在#undef命令内部，并不会执行宏替换。