引言  
    在C＋＋的消息公告板上经常可以看到出现**ASSERT**错误的求助信息。虽然这通常是希望消除**ASSERT**错误的帮助请求，但是几乎所有的求助者都认为**ASSERT**本身是罪恶的。我完全能理解一个**ASSERT**错误给程序员新手带来的沮丧。你的程序正在运行，通常如你所愿，突然一声巨响——一个**ASSERT**错误！  
    那么就让我们来看看**ASSERT**们，为什么他们会出现在那里以及我们能从他们那里得到什么信息。我应该强调一下，这篇文章讨论**MFC**如何处理**ASSERT**。

语言学  
    打开google搜索，输入“define **assert**”，然后单击搜索。WordNet中“**ASSERT**”有四种意思：  
    1.断言、声称、主张……（直接了当的陈述）  
    2.确认、查证、承认、宣誓……（正式而严肃的宣告某事属实，如：“Before God I swear I am innocent”——“在上帝面前我发誓我是清白的”）  
    3. 坚持己见大胆地或强有力地提出（自己）的观点，以使其为大家所知。（“Women should **assert** themselves more!”）  
    4.强调、表明（表明事实，“The letter asserts a free society”）  
    上面的意思都很接近，但是第4个意思字面上更加接近我们的**ASSERT**。**ASSERT**表明条件已经被强调为真。如果条件不为真，则程序将处于严重故障中，而你——程序员，则应该收到这个警告。

**ASSERT**在代码中的意思  
    当一个程序员写一条**ASSERT**就表明他在说“这个条件必须为真，否则我们将发生错误”。比如你正在写一个函数，希望得到一个字符串指针。  
void CMyClass::MyFunc(LPCTSTR szStringPtr)  
{  
    if (\*szStringPtr == ’7’)  
        DoSomething();  
}  
这个函数读取这个指针所指向的内存段，因此它最好是指向一段合法有效的内存。否则你的程序将崩溃！如果你传递一个空指针给这个函数，那么任何程序在调用这个函数时都将发生错误。如果现在那个指针就是空的，而且你的程序崩溃了，你就没有足够的信息出处理它。仅仅只有一个消息框，告诉你一个代码地址和你曾尝试读取0x00000000段的内存。关联这个代码地址到你实际代码的那一行并不是一件轻松的事，尤其你对于处理这种情况还很生疏。  
    那么让我们对这个函数做一个小小的修改。  
void CMyClass::MyFunc(LPCTSTR szStringPtr)  
{  
    **ASSERT**(szStringPtr);

    if (\*szStringPtr == ’7’)  
        DoSomething();  
}  
这里所做的就是测试szStringPtr。如果为空则马上崩溃。崩溃？是的。但是如果你正用DEBUG编译程序它会以一种受控的方式崩溃。**MFC**拥有一些build-in plumbing能够捕捉受控的崩溃，并将其和编译器的副本联系到一起。如果你正运行这个程序的DEBUG，并且**ASSERT**失败，你将会看到一个与此类似的消息框。

这向你表明哪个文件哪一行触发了**ASSERT**.你可以选择abort，中止程序；也可以选择ignore忽略这个错误，有时这样做也奏效；或者你可以选择retry，重试。只要你的计算机上安装了调试器，即使单独运行这个程序这条语句也会起作用。

release和debug之间的区别  
    懂得**ASSERT**是编译助手很重要。如果你用DEBUG编译你的程序，编译器就会包含**ASSERT**(...)中的所有代码。如果你选择RELEASE编译，那么**ASSERT**本身以及圆括号中的所有代码都会消失。它认为你已经测试过你程序的DEBUG版了，并且捕获了所有可能的错误。如果你不幸漏掉了一个错误，并且发布了一个带有BUG的程序，你只能祈祷它能蹒跚着一路平安了。有时乐观也是一件好事！  
    有时也会有这种情况出现，你希望在DEBUG版里强调某条件为真，而有一些你又希望不论是在DEBUG版里还是在RELEASE版里都希望你所强调条件的代码被编译到程序里去。这时就要用到VERIFY了。在DEBUG版里VERIFY能让你跳到编译器中条件不满足的事件那里。在RELEASE版里，VERIFY中的代码仍然被包含在可执行文件中。举例来说：  
VERIFY(MoveFile(szOriginalFilename, szNewFileName));  
以上语句在DEBUG版里，如果MoveFile()函数失败则会产生一个调试**ASSERT**.无论选择那种方式编译程序，MoveFile()函数的调用都会被包含到程序中。但是在RELEASE版里，调用失败会简单地被忽略。与下面这条语句作个比较：  
**ASSERT**(MoveFile(szOriginalFilename, szNewFileName));  
在DEBUG版里，MoveFile()会被编译和执行。而在RELEASE版里这一行就完全消失了，没有任何文件试图被移动。

在**MFC**中跟踪**ASSERT**  
    如果你现在还在继续往下读，希望从中了解**ASSERT**错误，比较大的可能是你正在调试的**ASSERT**并不是来自你自己的代码。如果你够幸运，它会来自**MFC**，因为你拥有它的源代码。  
    需要记住的第一件事是它不太可能是由**MFC**中的BUG造成的。我不否认**MFC**中存在BUG，但是在我使用**MFC**的十多年中，我从来没有遇到过一个**ASSERT**是由**MFC**的BUG造成的。  
    第二件需要记住的事是**ASSERT**的出现一定有原因。你需要检查触发**ASSERT**错误的那一行代码，并且弄明白它在测试什么。  
    举例说来，相当一部分**MFC**类是围绕WINDOWS控件的。许多情况下，封装函数将一个SendMessage调用转化得看起来像一般函数，以此来简化你的编码。比如，CTreeCtrl::SortChildren()函数，接受一个树形句柄，并对其子项进行分类。在你的代码中可能类似下面的语句：  
m\_myTreeCtrl.SortChildren(hMyNode);  
事实上，这个类向树形控件发送了一条消息。你只是简单的调用了一个函数，而**MFC**则具体传递那条消息需要用到的参数。下面是**MFC**源码中重新格式过的函数代码：  
\_AFXCMN\_INLINE BOOL CTreeCtrl::SortChildren(HTREEITEM hItem)  
{  
    **ASSERT**(::IsWindow(m\_hWnd));  
    return (BOOL)::SendMessage(m\_hWnd, TVM\_SORTCHILDREN,0, (LPARAM)hItem);  
}  
它做的第一件事是表明你的CTreeCtrl对象中的窗口句柄是合法有效的！现在我真的不知道如果你试图送一个TVM\_SORTCHILDREN消息到一个不存在的窗口会有什么坏事情发生。我所知道的是：在我正在尝试那么做的时候，我希望被告知。如果我正在做一些没有成功希望的事情，**ASSERT**在这里可以直接警告我。  
    因此如果你在调用类似那样的函数时遇到了**ASSERT**错误，你最好看看失败的那一行，并会发现它正在强调窗口句柄必须是一个存在的窗口。那是它唯一强调的事情。如果它失败，唯一的可能就是那个句柄窗口并不存在。那些就是你跟踪错误的线索。  
    上面是一个简洁的例子说明**MFC**是如何使用**ASSERT**的，以及你如何在你的工程中发现**MFC**的**ASSERT**原因。现在让我们来看看如何在自己的代码中应用**ASSERT**。  
void CMyClass::MyFunc(LPCTSTR szStringPtr)   
还是来看这个函数，前面我提到用一个简单的检查来使所传递的指针非空。事实上我们可以比上面做得更好。**MFC**和WINDOWS本身都提供了一串函数，我们可以使用它们来测定一个指针是否指向一段有效内存。  
void CMyClass::MyFunc(LPCTSTR szStringPtr)  
{  
    **ASSERT**(szStringPtr);  
    **ASSERT**(AfxIsValidString(szStringPtr));

    if (\*szStringPtr == ’7’)  
        DoSomething();  
}  
第一条语句仅仅捕获一种错误，即是否空指针。加上第二条语句我们就可以测试指针是否指向有效内存。这个测试检查你是否拥有读取该段内存的权限，并且该内存段包含字符串的结束符。一个相关的函数是AfxIsValidAddress，它检查你是否有权访问前面调用中声明过大小的内存块。你也能够检查是否有这个块的读或者写权限。

其它的**ASSERT**检查  
    除了前面提到的两种检查，还可能用到检查一个传递的对象是否某个特定类型。如果你正在写一个处理CEmployee对象和CProduct对象的程序，显然他们不能互换。因此需要确认处理CEmployee对象的函数只接受相应类型的对象。在**MFC**中你可以像下面这样做：  
void CMyClass::AnotherFunc(CEmployee \*pObj)  
{  
    **ASSERT**(pObj);    //it can’t be a NULL  
    **ASSERT**\_KINDOF(CEmployee, pObj);  
}  
像前面一样，我们首先确认这个指针不是空的。然后我们检查这个对象指针的类型是不是CEmployee型。只有对于从CObject派生的类才能这样处理，并且需要添加runtime支持。幸运的是添加runtime支持真的微不足道。  
    你必须已经声明这个对象至少是dynamic。解释一下，在**MFC**中你能够声明一个类包含runtime类信息。你可以通过在类声明中包含DECLARE\_DYNAMIC(ClassName)宏和在执行处包含IMPLEMENT\_DYNAMIC(ClassName,BaseClassName)来完成这一操作。  
class CMyReallyTrivialClass : public CObject  
{  
    DECLARE\_DYNAMIC(CMyReallyTrivialClass)  
public:

    //  Various class members and functions...  
};  
and the implementation file  
IMPLEMENT\_DYNAMIC(CMyReallyTrivialClass,CObject);  
.  
.  
.  
// Other class functions...

    如果你只是想使用**ASSERT**\_KINDOF宏，这两行已经足够了。现在，当你写程序时，你可以在任何地方使用**ASSERT**\_KINDOF宏来检查一个传递给你的对象指针是否你要的类型。如果不是，你的程序将如前所述以一种受控的方式崩溃，并收到调试器给的一个**ASSERT**失败。  
    如果你的对象已经包含DECLARE\_DYNCREATE宏或者DECLARE\_SERIAL宏，你就不需要使用DECLARE\_DYNAMIC了，因为这些宏包含了**ASSERT**\_KINDOF宏需要的runtime类信息。

结论  
    以上阐述了如何用**ASSERT**捕获runtime错误并将你带到调试器中导致**ASSERT**错误的行。我们看到了如何追溯**ASSERT**失败原因的方法。沿着这个思路，我们学会了如何测试我们自己的代码中指向内存的指针的有效性；如何检查我们得到的对象指针是否是代码需要的。  
    最近几年，我一直在我的代码中使用**ASSERT**作为runtime检查器，感觉受益匪浅。我很少遇到需要处理由空指针或者错误类型的对象指针所造成的程序崩溃。