为什么要学习汇编语言？

如果把学习比作远行的话，旅行者要解决的不只是一条路的问题，还有两座山的问题。这两座山的存在会使得你无法看清远方的目标，会使得你一路上不断怀疑方向，失去信心。这也就是为什么很多人买了厚厚的编程书籍然后让他们永远的在书架上落灰，这也就是为什么很多人能够饶有兴趣的搜集 n 多的电子书，然后永远的让他们以各种各样的电子状态永久的停滞在存储器上……

一座山是：为什么要学习汇编语言？我的观点是：因为汇编语言是计算机软件技术的“原子学说”。我非常喜欢电脑的重要原因是，他上面所有的事情背后都有着明确的原因。也许正是因为这个原因，所以你看轮船下水，或者房屋开工都会有着杀猪点香的仪式，可是程序员写程序或者建立新的工程，从来不会有烧香拜佛的情形---如果拜祭的话，是不是要拜拜电力之神法拉利？汇编语言可以解释 vc 生成的程序为什么没有按照预期结果输出，可能是优化编译出现了问题，也可能是链接程序的问题，更有可能是马虎导致的程序逻辑上的问题。但是这一切都可以在汇编语言层次得到完美的解决。如果你用汇编语言学完某项知识，当用它来完成这个程序的时候，也就意味着你已经完全的掌握了这门知识，你再看 vc 或者其他高级语言例程的时候，会觉得脉络明晰，甚至你会觉得为什么其他语言要啰嗦那么久。此谓“游刃有余”或“庖丁解牛”。

另外一座山是：我怎么下手学习？说道这里，我另外提一句，很多人学习编程的初衷是想“写一个病毒木马”或者“自己设计一款游戏”。对于前者，我并不反对，毕竟“唯一一项”“由免费软件”支撑起来的庞大产业是杀毒行业，此外还希望如果真有一天设计出来了，千万不要做的太过分，比如：专门删除 ghost 文件等等，这是心理缺陷的表现。

关于教材，我的观点是教科书有两种，一种是看起来像字典一样，实际上也是字典---小时候爸爸经常带着赞美的语气说他知道的 xxx 通过背字典的方式来学习，好在爸爸只是知道，我也肯定不会这样学习的。字典通常都是面目可憎的，仿佛古板的教授，比如说：字典中“虎”；哺乳动物，毛黄褐色，有黑色条纹，性凶猛，力大。骨和血及内脏均可入药（通称“老虎”）；还有一种看起来轻松得多，娓娓道来，非常不幸这样的书籍通常都是国外翻译过来的，而翻译人员通常是那些编字典的人员…. ….

这篇文章不会一项一项的介绍汇编语言的知识点，也不会就某个方面深入展开，这是一本讲 howto 的书，不是讲 what is 的书。读完此书，你应该学会如何查找资料，如何阅读 msdn，甚至可以明白如何讲述清楚自己遇到的问题。读完此书，你在去看罗云彬的那本经典，就丝毫没有问题。

版本	日期	修改内容
V0.1	2010-9-4	第一版

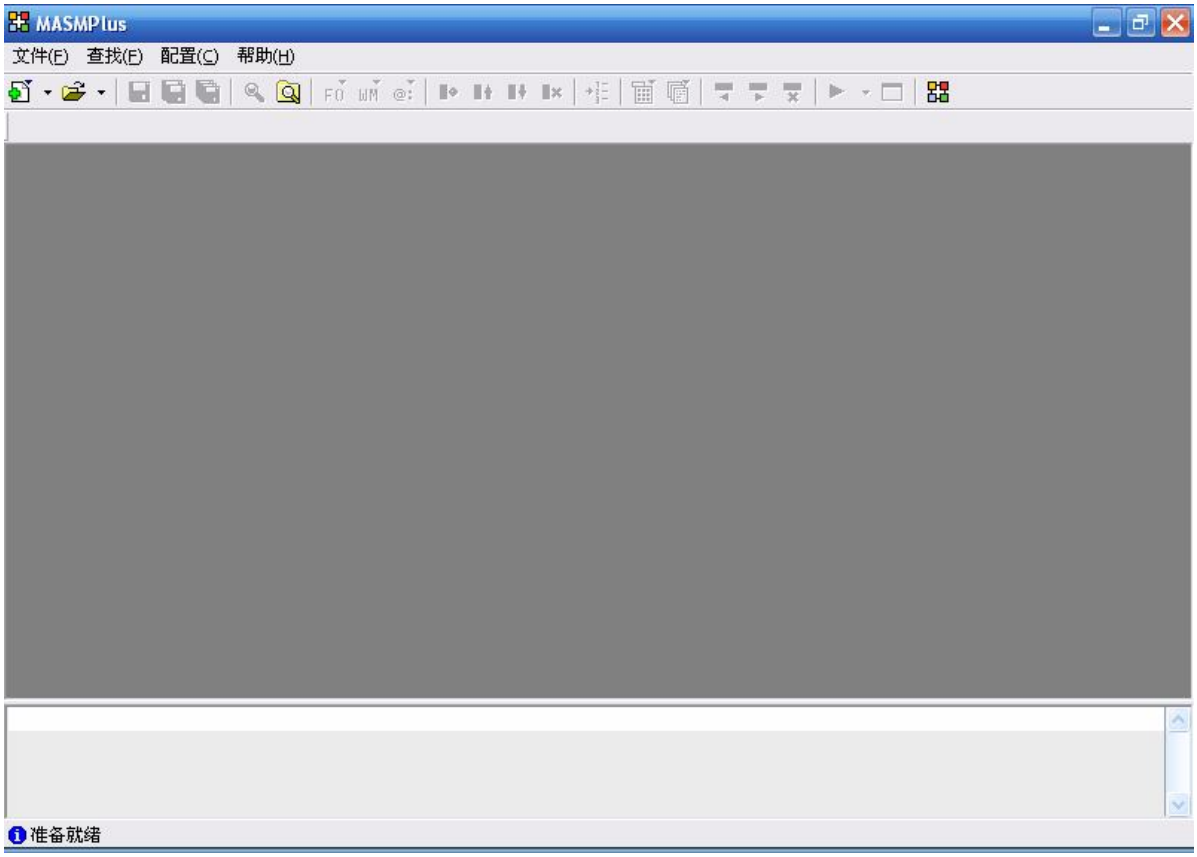
用 MasmPlus 学习 Win32 汇编（1）

汇编语言给人的印象就是黑黑的窗口，一行行的字符，不断上滚的界面……初学者必须弄清楚同汇编语言本身无关的很多东西也正因如此，初学者视汇编语言为畏途。今天开始，我将讲述如何使用 MasmPlus 进入 Win32 的汇编语言世界。

MasmPlus 的使用

我们在这里推荐使用 MasmPlus 编辑器，这是一款由 Aogo 开发的编辑器，有着诸多的优点。不过我想对于初学者来说，最重要的优点就是：使用简单不需要配置以及全中文支持。

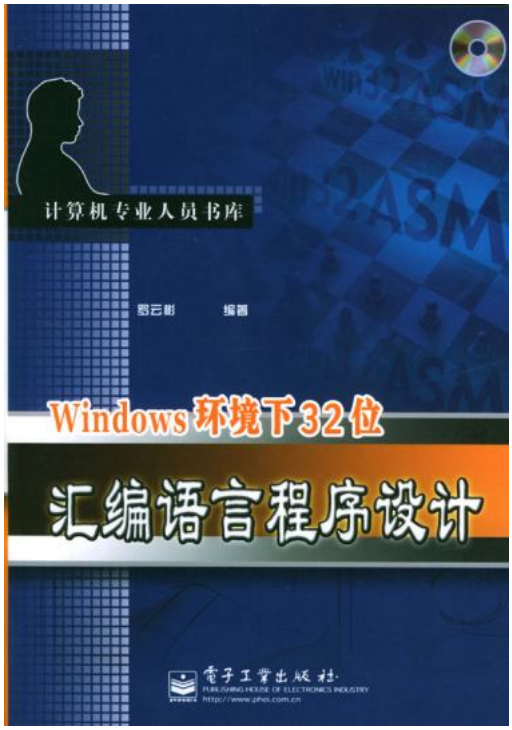
下载地址为： http://www.aogosoft.com/masmplus/ 下载之后即可安装。



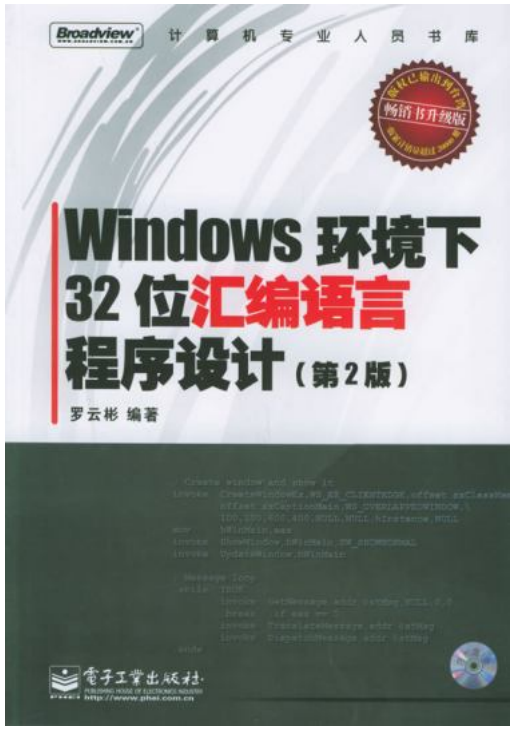
MasmPlus 全貌

另外，我推荐电子版的《Win32ASM 教程》，在 http://www.aogosoft.com/ 首页上有下载。

纸质的图书 《Windows 环境下 32 位汇编程序设计（第 2 版）》，这本书的最大特点就是“厚”！



第一版



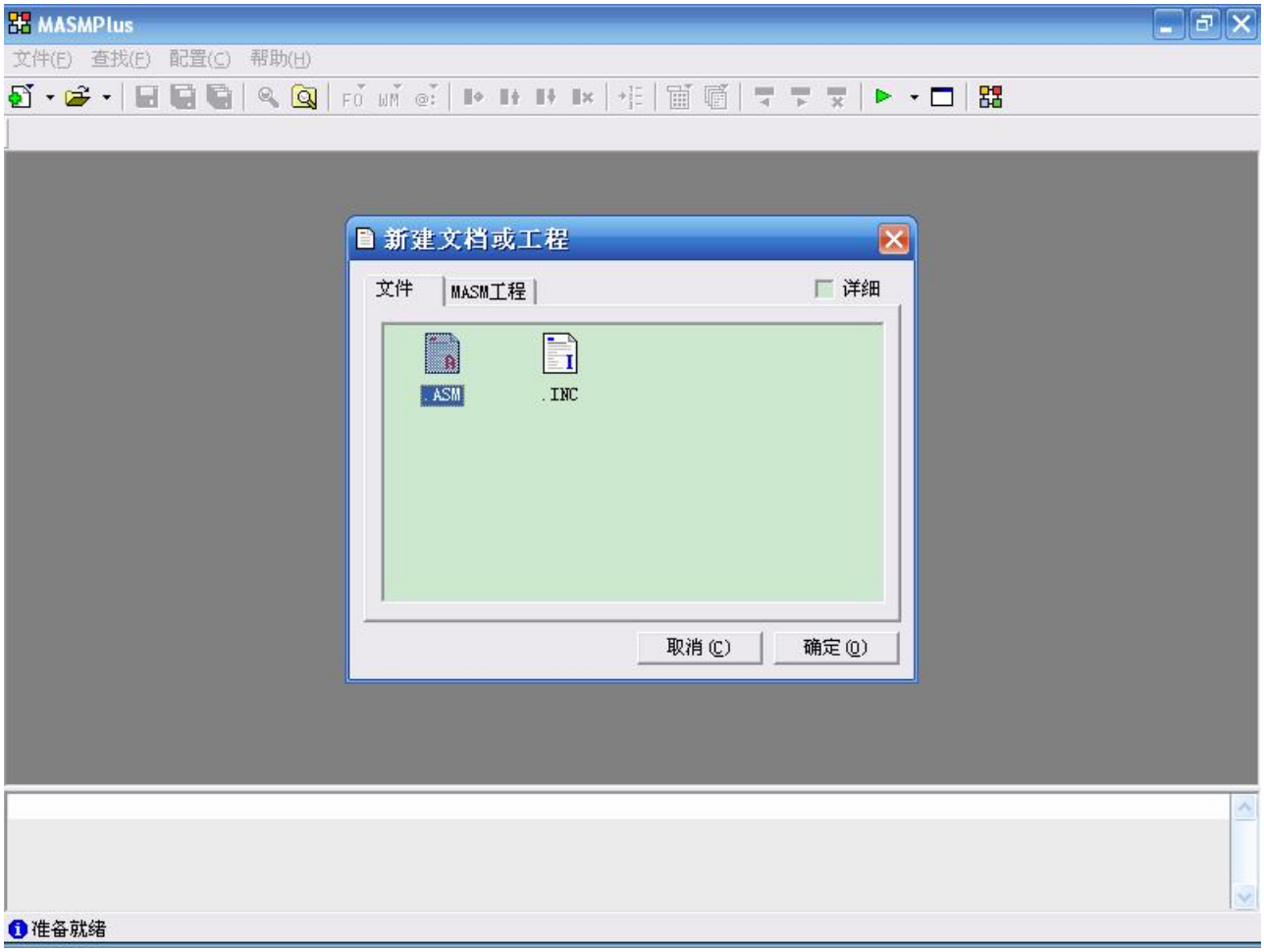
第二版

我说“厚”的意思是此书涵盖面非常宽，并且出版时间很长，你手持此书在各个汇编论坛上都能找到“同志”。后面我还会使用此书的例子作为讲解。这本书在很多网上书店都有前 4 章的试读，我强烈推荐各位读读先，也强烈推荐读者购买一本作为“收藏”。

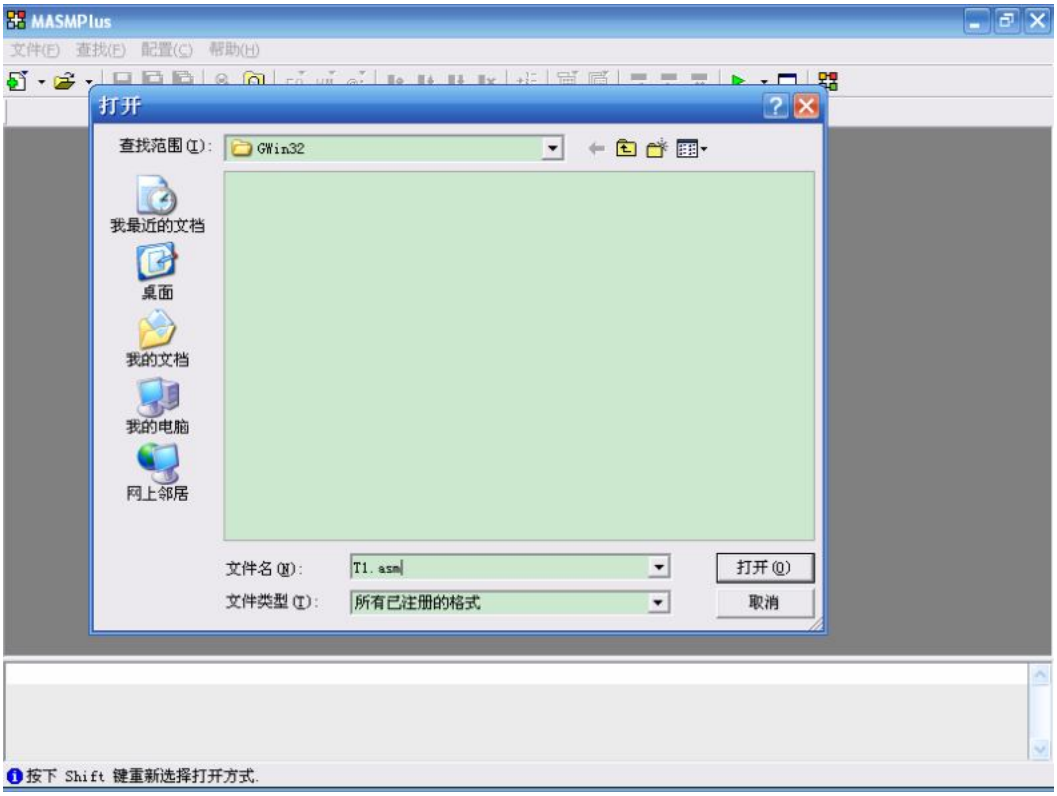
我的第一个程序

前面说了，MasmPlus 下载安装之后即可使用~ 马上我们就进入到实战阶段。第一个例子来自《Win32ASM 教程》。具体说是来自 《Iczelion 的 Win32 汇编教程》的“第二课 消息框”。

使用 “文件” → “新建…”在新弹出的对话框中选择 .Asm

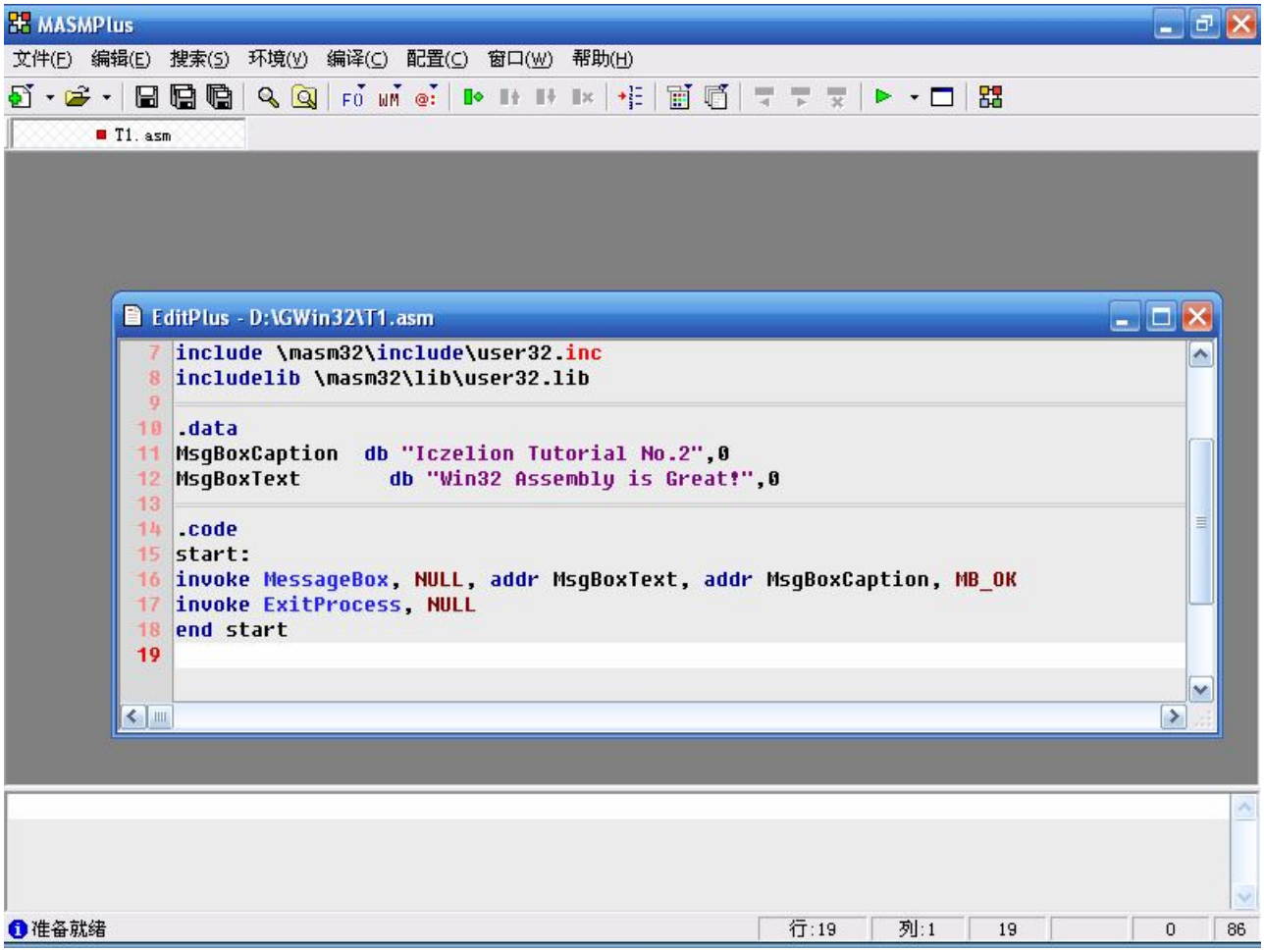


下面会提示你保存的文件名，我建立了一个 Gwin 的目录，将这个文件保存为 t1.asm。需要注意的是文件后缀必须是 .ASM 否则后面有些操作可能无法完成。

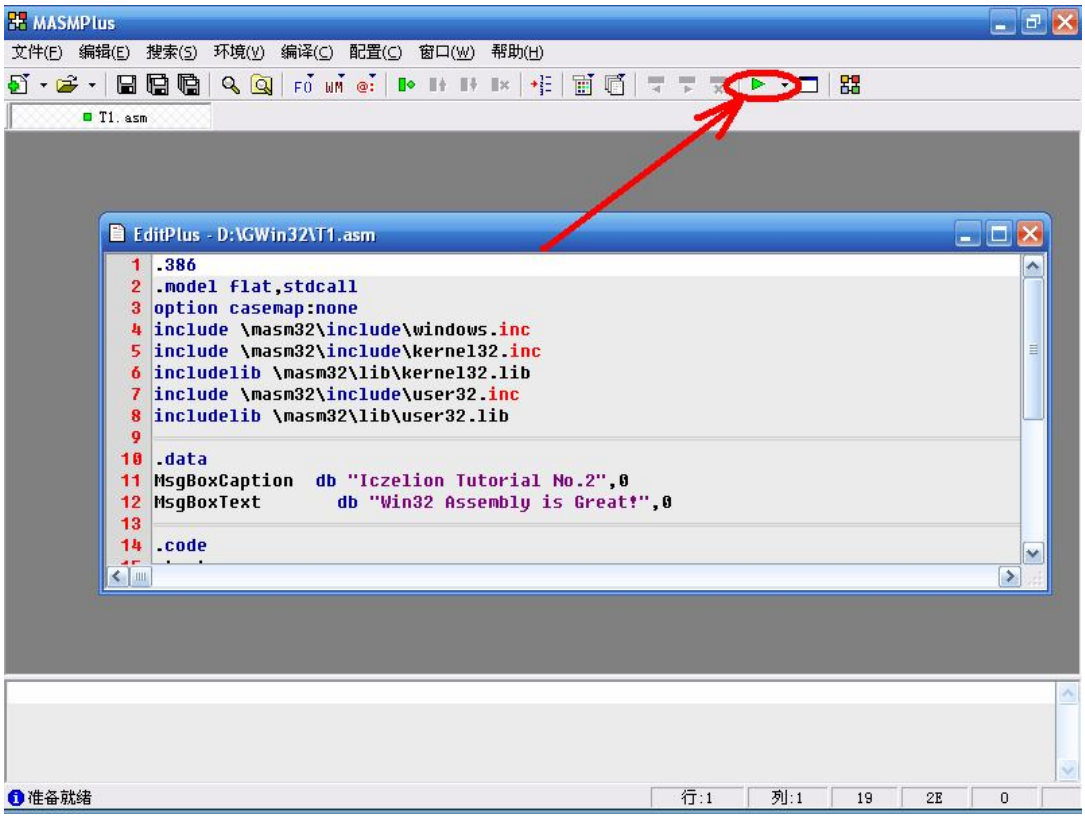


然后将下面的程序段拷贝到新建的文件中：

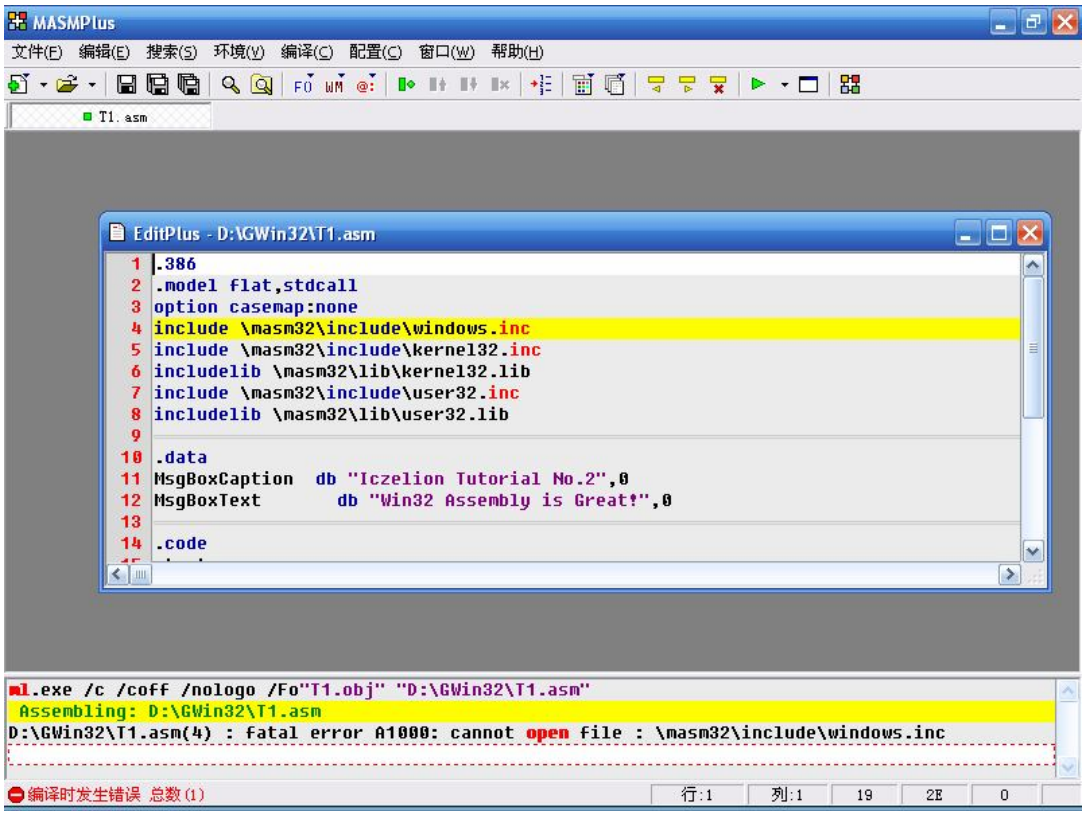
```
.386
.model flat,stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib

.data
MsgBoxCaption    db "Iczelion Tutorial No.2",0
MsgBoxText           db "Win32 Assembly is Great!",0

.code
start:
invoke MessageBox, NULL, addr MsgBoxText, addr MsgBoxCaption, MB_OK
invoke ExitProcess, NULL
end start
```



点击下图指示的按钮即可一次性完成编译连接。

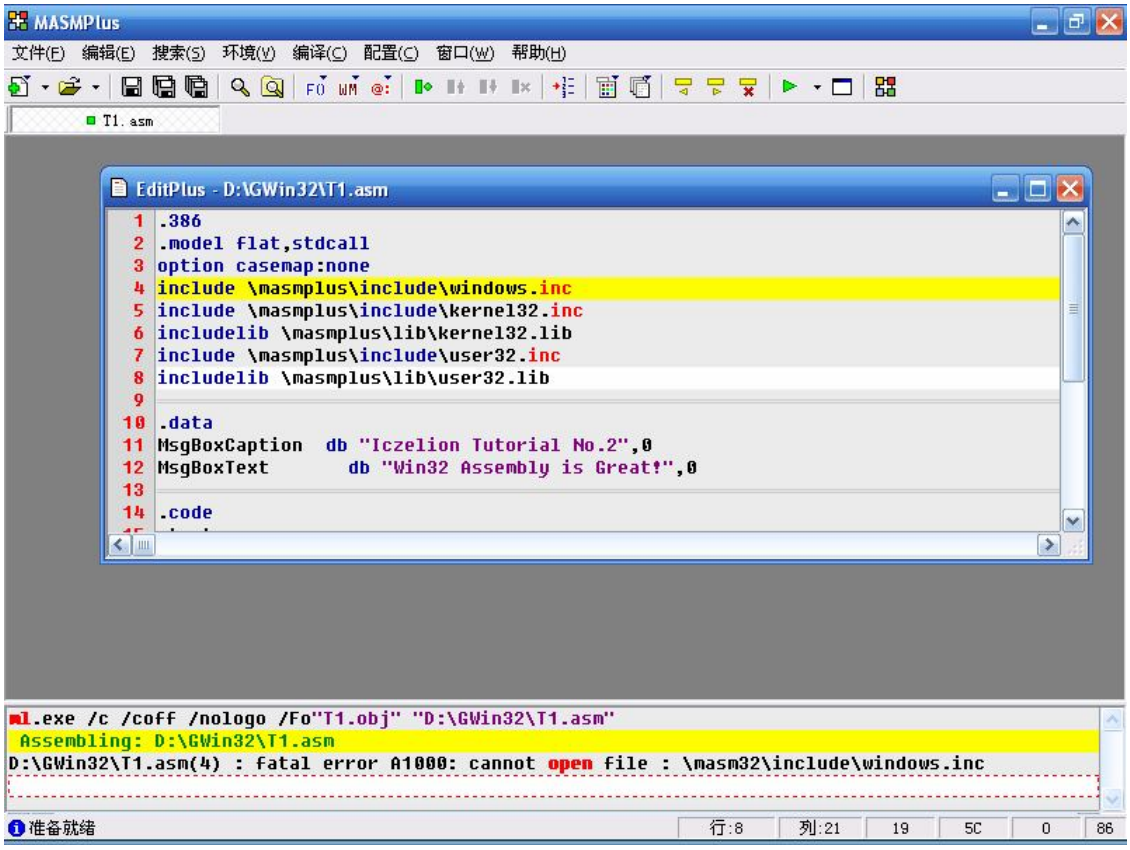


结果编译出错。我们一定要坚信：错误会让我们学习到比正确更多的东西!

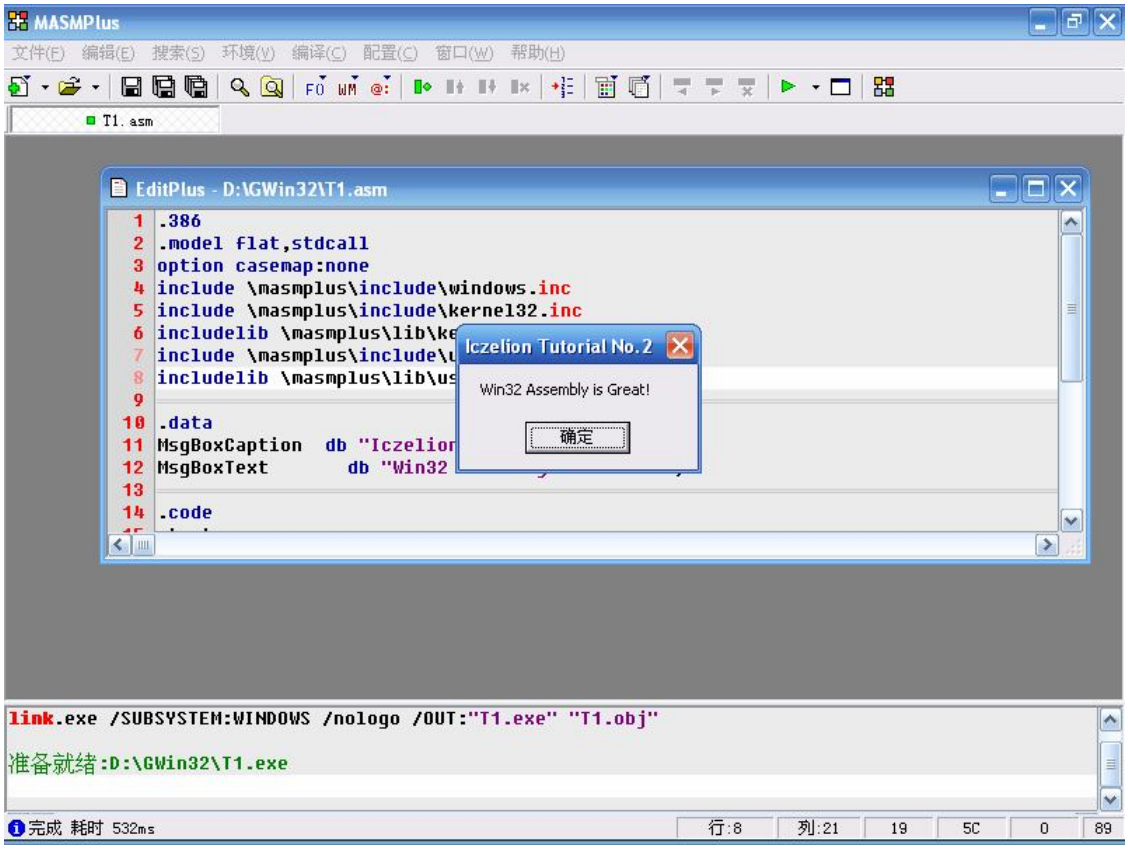


错误的意思是“无法打开 include \masm32\include\windows.inc 文件”。这是因为那本教程使用的是 Masm32 。例子都是按照那个编译器配置的。我们将程序中的

masm32 换做 MasmPlus 即可。



再次编译：



编译之后，顺便还会执行（如果有一天，你再编写格式化硬盘的程序，千万要注意噢~）编译结果可以在源程序目录下找到：

程序的修改

如果你不是靠 Win32 混饭吃（天杀 and Aogo 属于此类），或者心理扭曲（年轻时候的 Z.t），写程序基本上就是为了好玩，下面就是各种奇奇怪怪的试验了。

看看教程，上面说

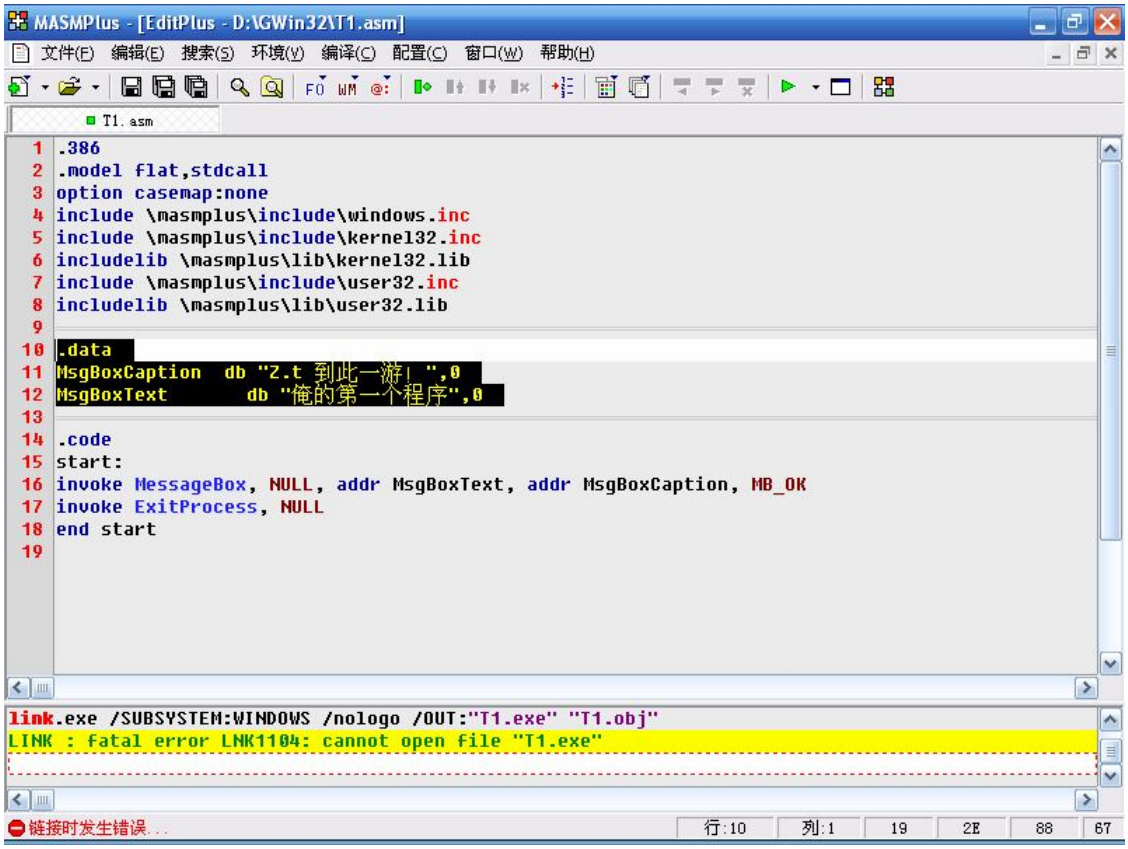
```
.386
.model flat,stdcall
option casemap:none
include \masmplus\include\windows.inc
include \masmplus\include\kernel32.inc
includelib \masmplus\lib\kernel32.lib
include \masmplus\include\user32.inc
includelib \masmplus\lib\user32.lib
```

这个段就是声明引用，没什么意思。（提问：我减少引用可能会导致无法编译，那我增加引用的呢？会不会冲突？文件大小是否会变化？）

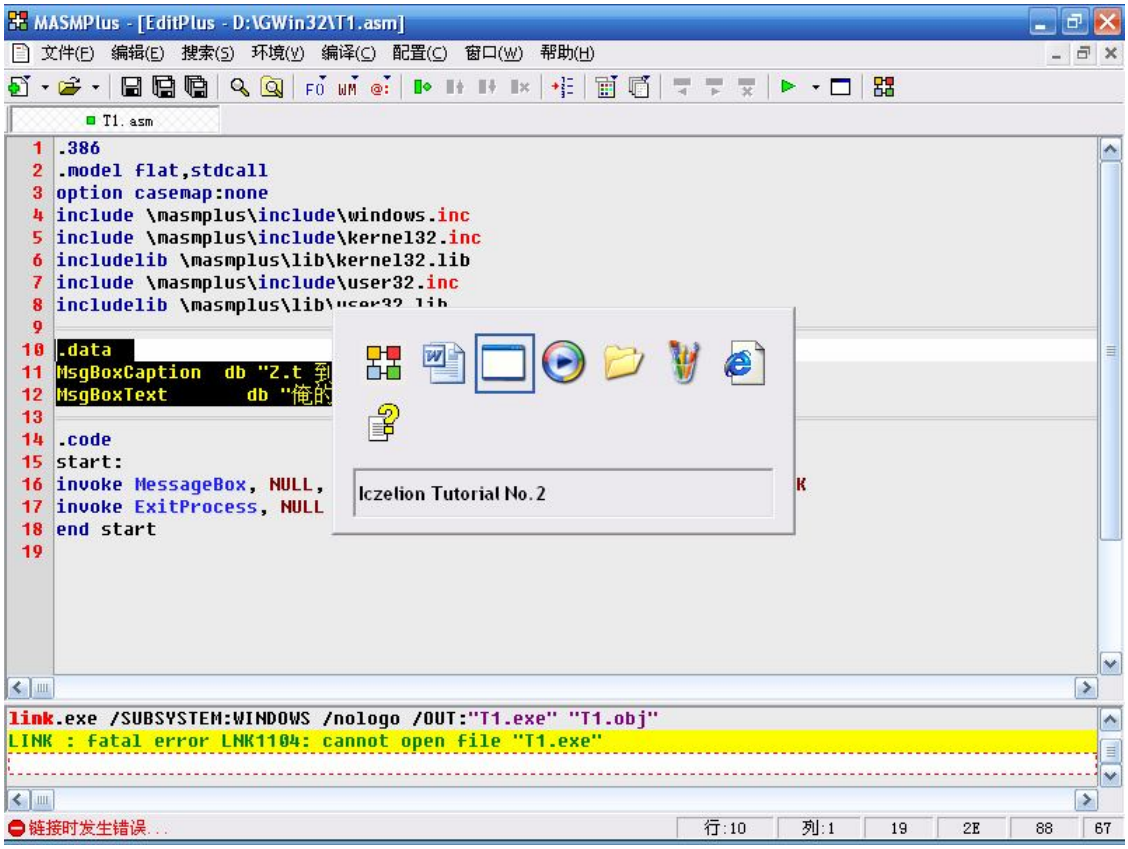
那就用 .data 开刀吧：改为

```
.data
MsgBoxCaption    db "Z.t 到此一游！",0
MsgBoxText           db "俺的第一个程序",0
```

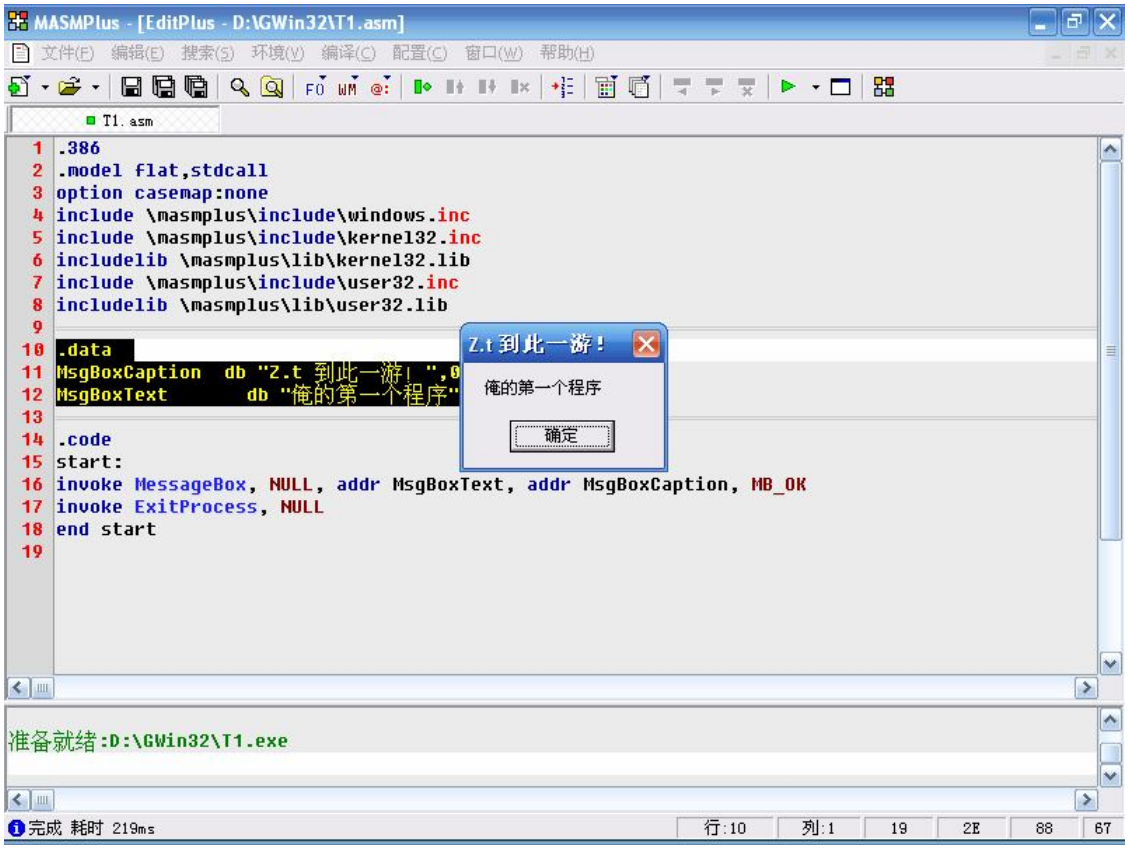
编译：



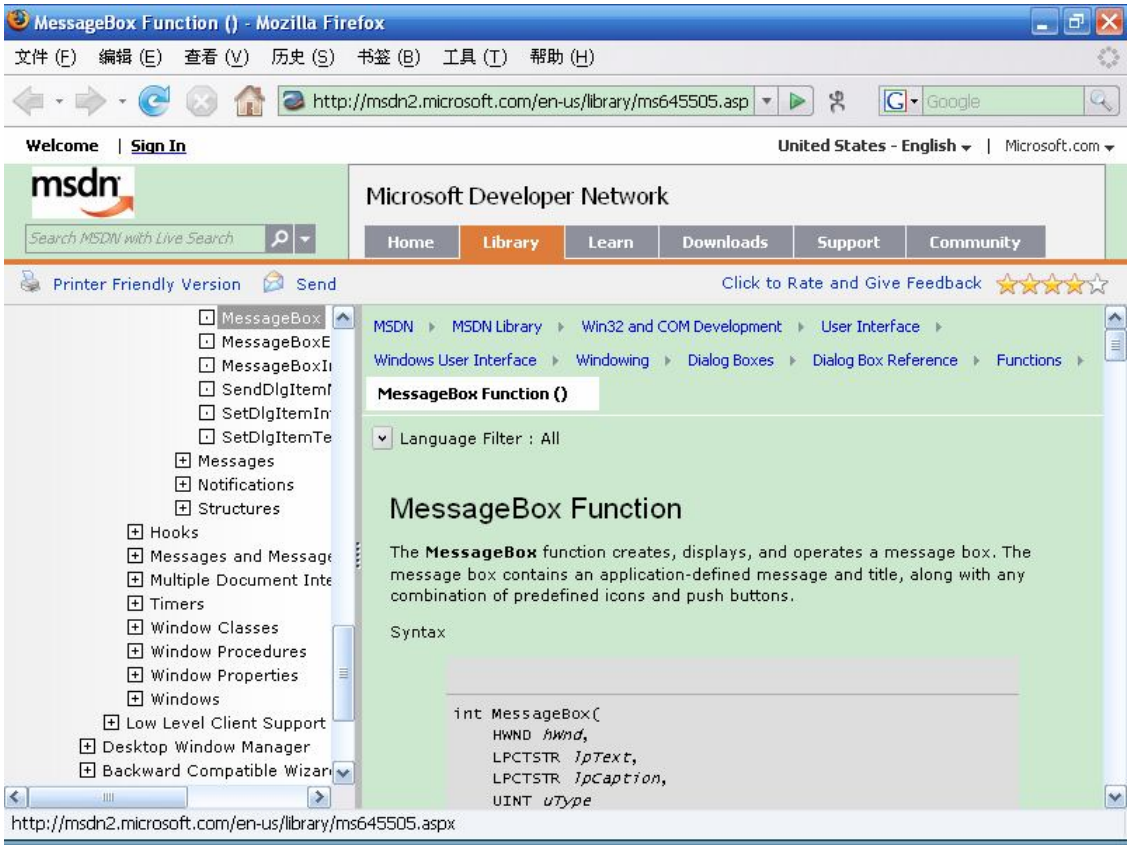
这次的错误是：无法打开 T1.EXE。这个是刚才编译生成的文件，为什么有问题？原来是刚才我一高兴，又执行了一次 t1.exe 并且没有关闭，下图中可以清楚的看到。



关闭那个程序，继续玩，编译执行：



下面该弄一下 .code 段了。教程上说了 MessageBox 是一个 Windows API 。那么到哪里知道这些 API 呢？买一本 API 大全吧~ 这是开玩笑，API 大全一般都不会“全”的。最好的方法是在网上查找 MSDN。我通常的做法是在 google 中输入“API 函数名”出现的第一个连接通常都是微软的 MSDN。



invoke MessageBox, NULL, addr MsgBoxText, addr MsgBoxCaption, MB_OK

第一个参数太复杂，暂时不理会它。第二个参数是标题栏，我们前面已经修改过了（提问：注意到字符串末尾的 0 了吧？去掉 MsgBoxCaption 末尾的 “0”试试看？再去掉 MsgBoxText 的“0” 再试试看？）MB_OK 似乎是控制显示的按钮。 MSDN 参考如下：

MB_ABORTRETRYIGNORE

The message box contains three push buttons: **Abort**, **Retry**, and **Ignore**.

MB_CANCELTRYCONTINUE

Microsoft Windows 2000/XP: The message box contains three push buttons: **Cancel**, **Try Again**, **Continue**. Use this message box type instead of MB_ABORTRETRYIGNORE.

MB_HELP

Windows 95/98/Me, Windows NT 4.0 and later: Adds a **Help** button to the message box. When the user clicks the **Help** button or presses F1, the system sends a WM_HELP message to the owner.

MB_OK

The message box contains one push button: **OK**. This is the default.

MB_OKCANCEL

The message box contains two push buttons: **OK** and **Cancel**.

MB_RETRYCANCEL

The message box contains two push buttons: **Retry** and **Cancel**.

MB_YESNO

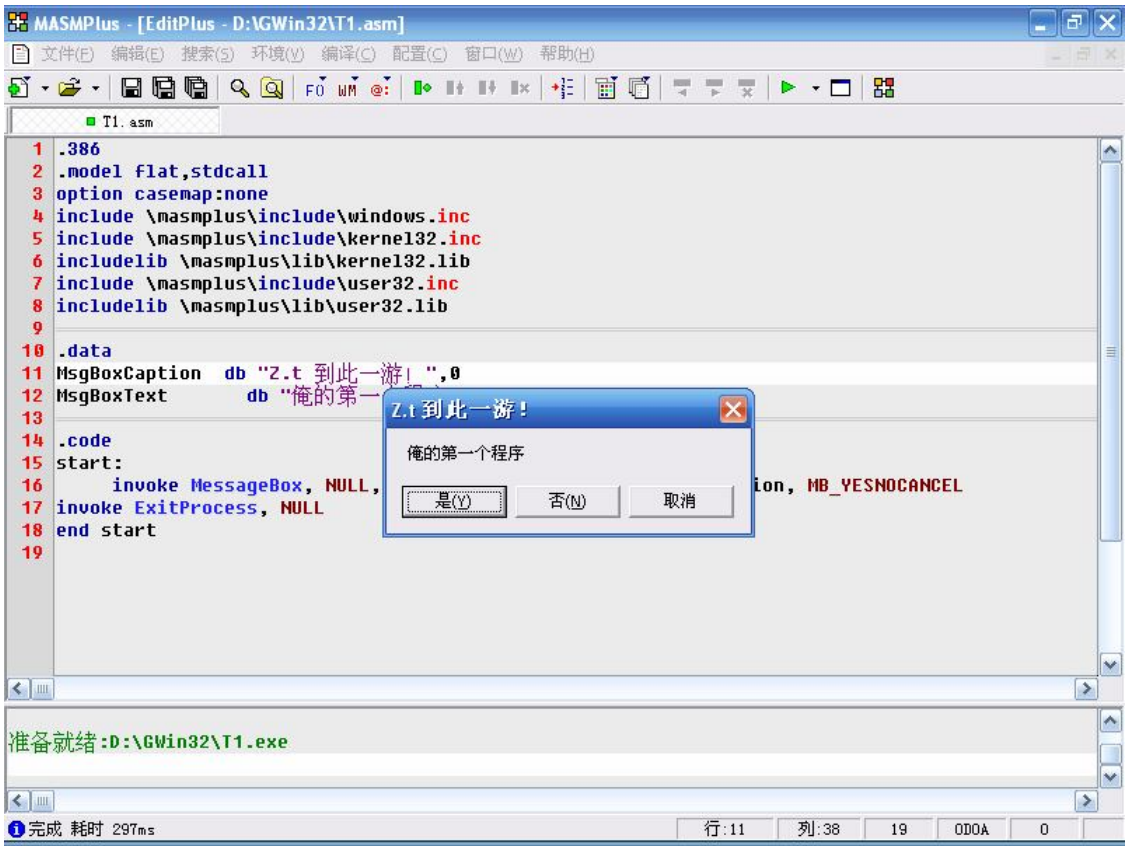
The message box contains two push buttons: **Yes** and **No**.

MB_YESNOCANCEL

The message box contains three push buttons: **Yes**, **No**, and **Cancel**.

换上一个 MB_YESNOCANCEL，就是将这句话修改为：

invoke MessageBox, NULL, addr MsgBoxText, addr MsgBoxCaption, MB_YESNOCANCEL



和我们的想象差不多。选择任何一个按钮的结果都是退出程序。

继续看 MSDN

To display an icon in the message box, specify one of the following values.

MB_ICONEXCLAMATION

An exclamation-point icon appears in the message box.

MB_ICONWARNING

An exclamation-point icon appears in the message box.

MB_ICONINFORMATION

An icon consisting of a lowercase letter *i* in a circle appears in the message box.

MB_ICONASTERISK

An icon consisting of a lowercase letter *i* in a circle appears in the message box.

MB_ICONQUESTION

A question-mark icon appears in the message box. The question-mark message icon is no longer recommended because it does not clearly represent a specific type of message and because the phrasing of a message as a question could apply to any message type. In addition, users can confuse the message symbol question mark with Help information. Therefore, do not use this question mark message symbol in your message boxes. The system continues to support its inclusion only for backward compatibility.

MB_ICONSTOP

A stop-sign icon appears in the message box.

MB_ICONERROR

A stop-sign icon appears in the message box.

MB_ICONHAND

A stop-sign icon appears in the message box.

还有图标功能哦~ 注意到 MSDN 介绍这个函数的时候，

```
int MessageBox(
    HWND hWnd,
    LPCTSTR lpText,
    LPCTSTR lpCaption,
    UINT uType
);
```

uType 是 UINT，所以我们可以这样用：

```
invoke MessageBox, NULL, addr MsgBoxText, addr MsgBoxCaption, MB_YESNOCANCEL
+ MB_ICONQUESTION
```

结果多了一个图标：

（提问：我要是 MB_YESNOCANCEL + MB_ICONQUESTION + MB_ICONHAND 或者加上更多的，结果会怎么样呢？暂时不要问为什么了：）

最后一个参数留给读者自己试验了，如果不明白欢迎到论坛上进行讨论。

前面说了 ： int MessageBox 意思是 这个 API 有返回值啊！翻翻教程， 返回值应该在 EAX 中。

那就再来一段吧：

```
start:
invoke MessageBox, NULL, addr MsgBoxText, addr MsgBoxCaption, MB_YESNOCANCEL
.if eax == IDCANCEL
 invoke MessageBox, NULL, addr MsgBoxText, addr MsgBoxCaption, MB_OK
.endif
invoke ExitProcess, NULL
end start
```

意思是，你要是选 cancel 那就再显示一次对话框。（提问：要是我想显示 5 次 10 次……应该怎么写？）

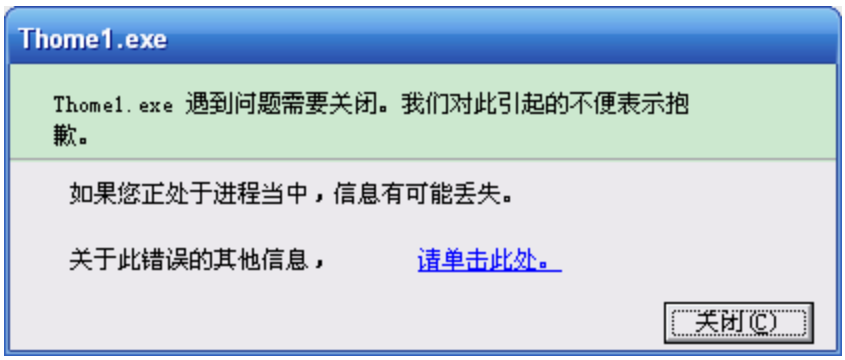
今天就讲到这里，我回去看看书，下次继续讲。总结本课内容：

1. 如何使用 MasmPlus 编译 Win32 汇编程序；
2. 如何查找 API 函数。

改错联系：下面是一个程序，我希望能够显示 AL 中的数值，但是程序在运行期会出现错误，

```
.386
.model flat,stdcall
option casemap:none
include \masmplus\include\windows.inc
include \masmplus\include\kernel32.inc
includelib \masmplus\lib\kernel32.lib
include \masmplus\include\user32.inc
includelib \masmplus\lib\user32.lib
```

```
.data
MsgBoxCaption    db "结果！",0
MsgBoxText       db 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
.code
start:
mov al,97h
call   ShowAL


invoke MessageBox, NULL, addr MsgBoxText, addr MsgBoxCaption, MB_YESNOCANCEL
invoke ExitProcess, NULL

ShowAL proc
            mov    dl,al          ;保存 AL
            mov    cl,04
            shr    al,cl          ;AL 高 4 位移至低 4 位
            mov    cx,02h         ;循环 2 次
            xor    ebx,ebx
            mov    esi,offset MsgBoxText

Low4bit:
             cmp    al,09
             jbe    larger        ;低 4 位超过 9
             add    al,07
larger:
             add    al,30h

             mov    [esi],al
             inc     esi

             mov    al,dl          ;恢复保存
             and    al,0Fh
             loop   Low4bit

              ret
ShowAL endp

end start
```



正确的结果应该是：

错误的处理

错误是必可避免的，好在现代的编译器能为我们发现很多错误。这里我也郑重推荐有志于深入学习计算机的朋友们，有机会研究一下“编译原理”这门课程，虽然这门课程是计算机系的必修课，也是历经了 n 十年的发展，但是截止目前仍然读起来枯燥无谓面目可憎，自学起来非常困难…..

继续正题，Masm 实际上是返回一个错误值，然后根据 ml.err 中的表格对应输出简单的提示信息。远一点说，Masm 套件中不只是汇编编译器，还有管理编译工程的软件 nmake，还有调试的 codeview，他们也都有着完整的对应错误值，这一点上非常值得在设计大型程序上借鉴。

比如下面几个是经常会遇到的错误值和提示信息。

A1000 cannot open file: filename 意思是无法打开文件，原因多种多样，出现这样的情况通常是因为前面一个程序并没有关闭，比如说前面一个程序没有正确响应关闭的消息，界面虽然消失，但实际上仍然在内存中存在。我们用 Masmplus 再次编译的时候，因为之前生成的 exe 还被 process 占用着，所以无法再次生成，就会出现这样的提示。通常的解决方法就是 ctrl+alt+del 调出任务管理器手工 kill 一下就好了。

A1011 directive must be in control block 可以理解为你使用的 .else .endif 等等并没有配对。出现这样的情况只能上去一段一段的数，看看是否有遗漏的匹配。从经验上来说最简单的方法是暂时大段移除代码，然后不断尝试，定位问题之后再贴回来就好了。这里顺便提一下 Masmplus 有很强的恢复能力，即便你移除之后忘记原来的样子，多按几次 ctrl+Z 也是可以找回来的。

A1012 error count exceeds 100; stopping assembly 就是目前编译中错误数量超过 100 条，自动停止 这是闷头写程序，然后最后全部编译最容易遇到的结果，通常这是最后一条提示信息，没什么好说的根据其他信息慢慢调试吧。不过不要灰心，实际问题通常不会这么

多，你忘记定义一个变量很可能导致 99 个问题。

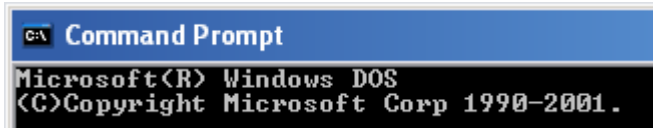
A2044 invalid character in file 这个是很容易遇到的问题，意思是文件中无效的字符。比如你把中文的“;”当作了英文字符的 “；”，即便我写出来也是非常不容易区分的，仿佛凤姐和整容之后的凤姐一般。

用 MasmPlus 学习 Win32 汇编（2）

控制台 Console

控制台，就是 Windows 下面那个黑色的窗口。进入控制台有两种方式，一种是运行 Command.com：

另一种是运行 Cmd.exe：



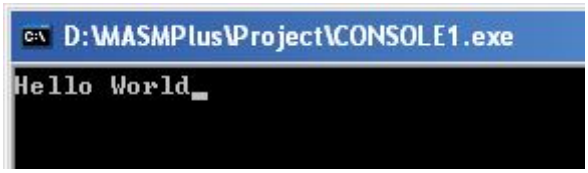
这两种方式上存在一些差别……我们要运行 Win32 Console 程序，建议选择后者。之所以要介绍这种就是因为很轻松的看到结果。

一.第一个 console 程序

MasmPlus 已经为我们准备好了模板：新建

选择 Masm 工程标签页：

选中 “Win32 Console”确定即可。我们将它存为 Console1.asm 之后出现的这个模板本身就是一个可以编译通过的小程序，加载之后我们立即就可以运行看到结果。



按下回车即结束。在保存目录下，我们可以看到编译生成了的 console.exe 文件。

（可以尝试一下从运行 cmd 或者 command 先进入 dos box 再进入这个目录运行 console 看看结果有什么不同）

下面我们对这个简单的程序进行一下分析：

```
    invoke StdOut,CTXT("Hello World!")
; StdOut 并不是 Windows 的 API，而我暂时找不到 Aogo 把 Macro 藏到什么地方了
;所以就不讲述了，知道后面跟着字符串的地址就可以了


    invoke StdIn,addr buffer,sizeof buffer
;暂停显示,回车键关闭
;同样，这个我也不解释了，输入的应该是 ascii 码，存放在前面定义的 buffer 中

    invoke ExitProcess,0
;这个就是退出结束的意思。
```

修改，我输入，然后显示输入：

```
    invoke StdIn,addr buffer,sizeof buffer
    invoke StdOut,addr buffer
    invoke ExitProcess,0
```

编译运行程序会一闪而过，但是如果你在 dos box 下面看就能看到整个过程。

二．Ascii 码

我发现实际上很多人无法理解 Ascii 十六进制 二进制 ……这些乱其八糟的东西，在这里我觉得有必要讲述一下。

ASCII 全称是 American Standard Code for Information Interchange 美国信息交换标准码。起始于 50 年代后期，在 1967 年定案。ASCII 码使用指定的 7 位或 8 位二进制数组

合来表示 128 或 256 种可能的字符。一般情况下我们使用的是 8 位的。一个字节也是 8 位的，因此正好能够将它们按照数字对字符一一对应起来。

比如： mov al,25h (注意，是十六进制的 25h) 然后将 al 直接输出，结果就是“%”。这就是简单的对应。额外的：

十六进制	显示出来的字符
30h	0
31h	1
32h	2
33h	3
34h	4
35h	5
36h	6
37h	7
38h	8
39h	9

就是说 mov al,31h 显示出来就是 “1”。如果还不明白，不妨动手实验下面的程序：

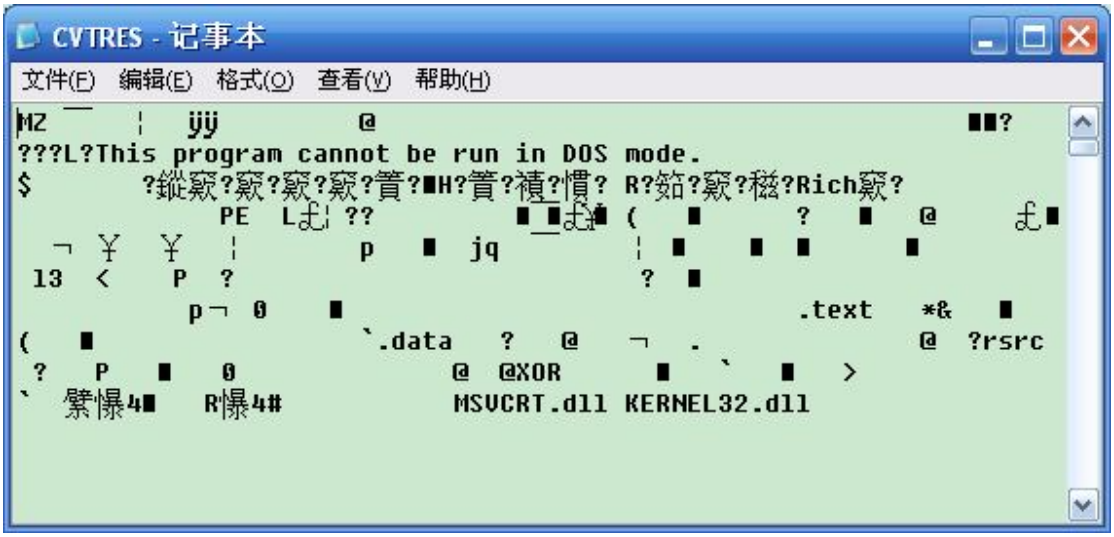
注意：.data 不是 .data？ 字符串放在后者显示不出来

```
.386
.model flat, stdcall
option casemap :none

include windows.inc
```

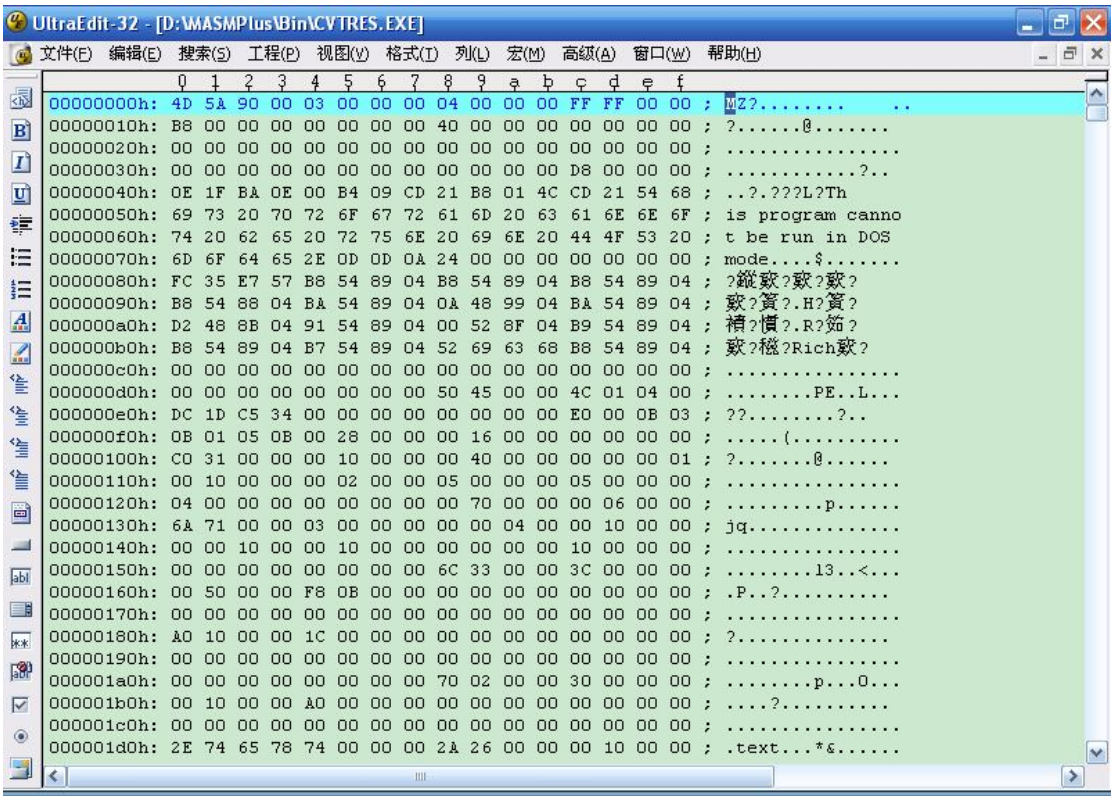
```
include user32.inc
include kernel32.inc
include masm32.inc

includelib user32.lib
includelib kernel32.lib
includelib masm32.lib
include macro.asm

.data
   buffer db 31h,32h,33h,34h,35h,36h,37h,38h,0
.CODE
START:

   invoke StdOut,addr buffer
   invoke ExitProcess,0

end START
```

需要注意的是不是所有的 Ascii 都是可见的，因此我们直接用记事本打开一个可执行文件时会看到很多乱码，比如我们使用记事本打开

D:\MASMPlus\Bin\CVTRES.exe 结果如下：



而使用十六进制编辑软件 ultraEdit 打开这个文件，看到的就是：



左边是按照十六进制表示的，右边是左边数值对应的 Ascii，很多无法显示的 Ascii 都用“.”来表示了。可以看出来和用 Notepad 打开的看起来很“像”。

观察上面那个十六进制对 Ascii 的表可以发现一点点“规律”，一个数字 N 加上 30h 之

后显示出来就是这个数字。如果我们要按照十六进制显示 AL 中的数值，就可以将 Al 的高 4 位加上 30h 显示一下，再按照同样的方式处理低 4 位。不过 0-9 的数值和 A-F 的并不连续，这里需要特殊处理一下。下面的 dos 程序段就是这个原理（非常不幸的是 Dos Box 中，我们的 conole 程序无法直接调用 int 21h）.

```
;Input AL,Bl,No Output
ShowAL proc
            pusha
            mov  dl,al      ;保存 AL
            mov  cl,04
            shr  al,cl      ;AL 高 4 位移至低 4 位
            mov  cx,02h     ;循环 2 次

Low4bit:    cmp  al,09
            jbe  larger     ;低 4 位超过 9
            add  al,07
larger:     add  al,30h

            push  cx
            mov  bh,0h
            mov  cx,1h
            mov  ah,09h
            int  10h
            pop  cx

            add  shX,1
            call SetCur
            mov  al,dl      ;恢复保存
            and  al,0Fh
            loop  Low4bit
            popa
         ret
ShowAL endp
```

如果有兴趣的话，可以将这个程序改造为你自己的 console 的 showal.

另外，有些书上说 一个字节（Byte）是 9 位(bit)，这样的说法中多出来的这一位是奇偶校验位，是硬件上使用的，对我们软件来说是透明的。我教过一个人编写 DDR2 内存检测软件，需要找出内存哪个芯片坏了，他写的程序就使用了 9 位，我在查看他代码的时候

觉得非常崩溃，他委屈的说这是从网上查到的……

三.更简单的解决方法

上面是典型的 DOS 时代的思想：了解原理，同硬件做最亲密的接触。但是，用小刀在硬盘上刻操作系统的时代已经过去了……上网搜索一下，有一个 API

http://msdn2.microsoft.com/en-us/library/ms647550.aspx

wsprintf Function

The **wsprintf** function formats and stores a series of characters and values in a buffer. Any arguments are converted and copied to the output buffer according to the corresponding format specification in the format string. The function appends a terminating null character to the characters it writes, but the return value does not include the terminating null character in its character count.

Syntax

```
int wsprintf(
    LPTSTR lpOut,
    LPCTSTR lpFmt,
     ...
);
```

Parameters

lpOut

[out] Pointer to a buffer to receive the formatted output. The maximum size of the buffer is 1024 bytes.
[输出]指向一个输出的缓冲，最大 1024 字节

lpFmt

[in] Pointer to a null-terminated string that contains the format-control specifications. In addition to ordinary ASCII characters, a format specification for each argument appears in this string. For more information about the format specification, see the Remarks section.
[输入] 指向一个包含格式控制字符的非 null 结尾的字符串。

...

[in] Specifies one or more optional arguments. The number and type of argument parameters depend on the corresponding format-control specifications in the *lpFmt* parameter.

Return Value

If the function succeeds, the return value is the number of characters stored in the output buffer, not counting the terminating null character.

If the function fails, the return value is less than the length of the expected output. To get extended error information, call GetLastError.

Remarks

Security Alert Using this function incorrectly can compromise the security of your application. The string returned in *lpOut* is not guaranteed to be NULL-terminated. Also, avoid the %s format -- it can lead to a buffer overrun. If an access violation occurs it causes a denial of service against your application. In the worse case, an attacker can inject executable code. Consider using one of the following alternatives: StringCbPrintf, StringCbPrintfEx, StringCbVPrintf, StringCbVPrintfEx, StringCchPrintf, StringCchPrintfEx, StringCchVPrintf, or StringCchVPrintfEx. You should review Security Considerations: Windows User Interface before continuing.

The format-control string contains format specifications that determine the output format for the arguments following the *lpFmt* parameter. Format specifications, discussed below, always begin with a percent sign (%). If a percent sign is followed by a character that has no meaning as a format field, the character is not formatted (for example, %% produces a single percent-sign character).

The format-control string is read from left to right. When the first format specification (if any) is encountered, it causes the value of the first argument after the format-control string to be converted and copied to the output buffer according to the format specification. The second format specification causes the second argument to be converted and copied, and so on. If there are more arguments than format specifications, the extra arguments are ignored. If there are not enough arguments for all of the format specifications, the results are undefined.

A format specification has the following form:

%[-][#][0][*width*][.*precision*]*type*

Each field is a single character or a number signifying a particular format option. The *type* characters that appear after the last optional format field determine whether the associated argument is interpreted as a character, a string, or a number. The simplest format specification contains only the percent sign and a type character (for example, **%s**). The optional fields control other aspects of the formatting. Following are the optional and required fields and their meanings.

Field	Meaning
-	Pad the output with blanks or zeros to the right to fill the field width, justifying output to the left. If this field is omitted, the output is padded to the left, justifying it to the right.
#	Prefix hexadecimal values with 0x (lowercase) or 0X (uppercase).
0	Pad the output value with zeros to fill the field width. If this field is omitted, the output value is padded with blank spaces.
width	Copy the specified minimum number of characters to the output buffer. The *width* field is a nonnegative integer. The width specification never causes a value to be truncated; if the number of characters in the output value is greater than the specified width, or if the *width* field is not present, all characters of the value are printed, subject to the precision specification.
.*precision*	For numbers, copy the specified minimum number of digits to the output buffer. If the number of digits in the argument is less than the specified precision, the output value is padded on the left with zeros. The value is not truncated when the number of digits exceeds the specified precision. If the specified precision is 0 or omitted entirely, or if the period (.) appears without a number following it, the precision is set to 1. For strings, copy the specified maximum number of characters to the output buffer.
type	Output the corresponding argument as a character, a string, or a number. This field can be any of the following values. **Value** **Meaning** c Single character. This value is interpreted as type

	WCHAR if the calling application defines Unicode and as type **__wchar_t** otherwise.
C	Single character. This value is interpreted as type **__wchar_t** if the calling application defines Unicode and as type **WCHAR** otherwise.
d	Signed decimal integer. This value is equivalent to i.
hc, hC	Single character. The **wsprintf** function ignores character arguments with a numeric value of zero. This value is always interpreted as type **__wchar_t**, even when the calling application defines Unicode.
hd	Signed short integer argument.
hs, hS	String. This value is always interpreted as type **LPSTR**, even when the calling application defines Unicode.
hu	Unsigned short integer.
i	Signed decimal integer. This value is equivalent to d.
lc, lC	Single character. The **wsprintf** function ignores character arguments with a numeric value of zero. This value is always interpreted as type **WCHAR**, even when the calling application does not define Unicode.
ld	Long signed integer. This value is equivalent to li.
li	Long signed integer. This value is equivalent to ld.
ls, lS	String. This value is always interpreted as type **LPWSTR**, even when the calling application does not define Unicode. This value is equivalent to ws.
lu	Long unsigned integer.
lx, lX	Long unsigned hexadecimal integer in lowercase or uppercase.
p	**Windows 2000/XP:** Pointer. The address is printed using hexadecimal.
s	String. This value is interpreted as type **LPWSTR** when the calling application defines Unicode and as type **LPSTR** otherwise.
S	String. This value is interpreted as type **LPSTR** when the calling application defines Unicode and as type **LPWSTR** otherwise.
u	Unsigned integer argument.

x, X	Unsigned hexadecimal integer in lowercase or uppercase.

Note It is important to note that **wsprintf** uses the C calling convention (**_cdecl**), rather than the standard call (**_stdcall**) calling convention. As a result, it is the responsibility of the calling process to pop arguments off the stack, and arguments are pushed on the stack from right to left. In C-language modules, the C compiler performs this task.

To use buffers larger than 1024 bytes, use **_snwprintf**. For more information, see the documentation for the C run-time library.

Example

For an example, see Reading from a Mailslot.

Function Information

Minimum DLL Version	user32.dll
Header	Declared in Winuser.h, include Windows.h
Import library	User32.lib
Minimum operating systems	Windows 95, Windows NT 3.1
Unicode	Implemented as ANSI and Unicode versions.

试验一下：

```
.386
.model flat, stdcall
option casemap :none

include windows.inc
include user32.inc
include kernel32.inc
include masm32.inc

includelib user32.lib
includelib kernel32.lib
includelib masm32.lib
include macro.asm

.data?
   buffer db 100 dup(?)
   lpszSize  db 50 dup(?)
```

```
.CODE
START:

    mov eax,1235
    invoke wsprintf,offset lpszSize,CTXT("%d"),eax
        ;将 EAX 按照 Signed decimal integer 格式化，然后放到
        ;lpszSize 中
    invoke StdOut, offset lpszSize

    ;暂停显示,回车键关闭
    invoke StdIn,addr buffer,sizeof buffer
    invoke ExitProcess,0

end START
```

运行结果：

（之后建议读者试验一下如何使用此 API 显示十六进制）

除了 ASCII 之外，还有 Unicode 编码。思想上来说，就是 128 或者 256 太少不够用，我们就用更多的编码来表示。比如，“风”的 unicode16 编码就是十进制的 39118。不过这样做，很明显占用了更多的字节，刚开始接触的时候会觉得很别扭，习惯就好了。23601 26159 36825 26679。

三. WinExec

上面的内容可能让人觉得乏味，下面就介绍 1 个好玩的 API。

WinExec

Runs the specified application.

Note This function is provided only for compatibility with 16-bit Windows. Applications should use the CreateProcess function.

```
UINT WINAPI WinExec(
  LPCSTR lpCmdLine,
```

```
  UINT uCmdShow
);
```

Parameters

lpCmdLine　　*这个给定启动程序名称和参数*

[in] The command line (file name plus optional parameters) for the application to be executed. If the name of the executable file in the *lpCmdLine* parameter does not contain a directory path, the system searches for the executable file in this sequence:

1. The directory from which the application loaded.
2. The current directory.
3. The Windows system directory. The **GetSystemDirectory** function retrieves the path of this directory.
4. The Windows directory. The **GetWindowsDirectory** function retrieves the path of this directory.
5. The directories listed in the PATH environment variable.

uCmdShow 这个在后面介绍，我们现在使用SW_SHOWNORMAL 就可以了

[in] The display options. For a list of the acceptable values, see the description of the *nCmdShow* parameter of the **ShowWindow** function.

Return Value

If the function succeeds, the return value is greater than 31.

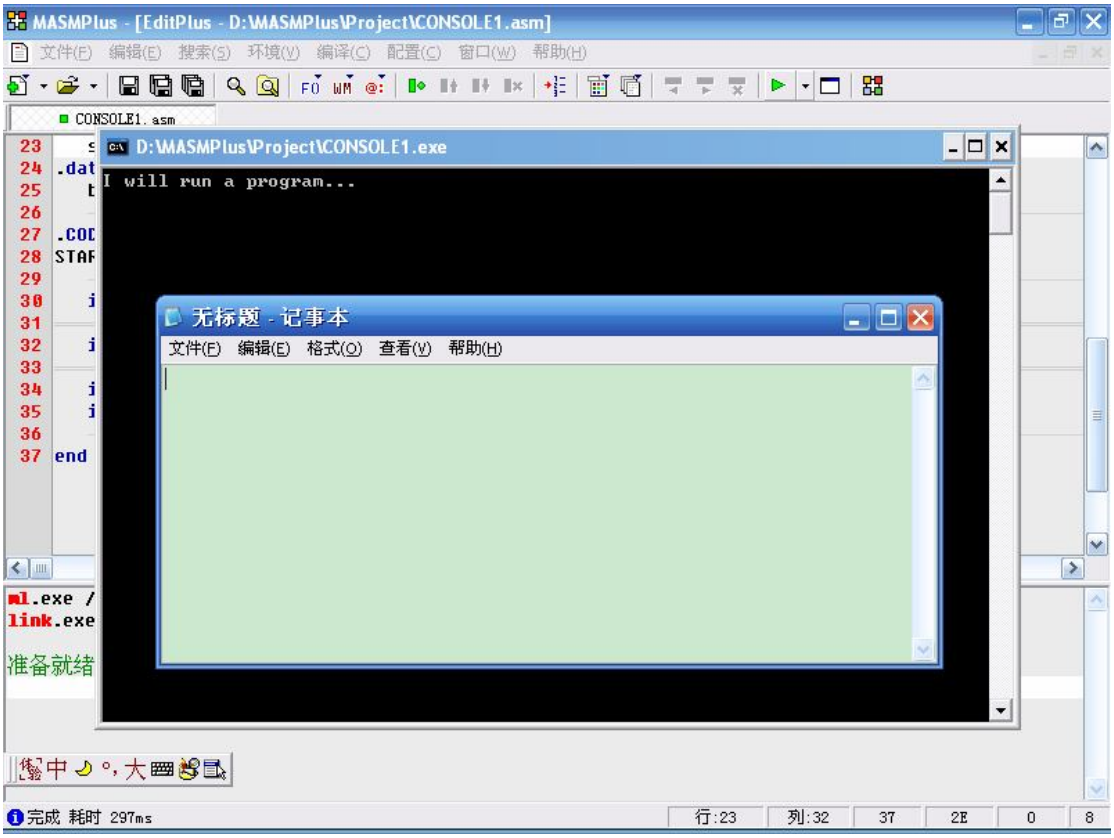
If the function fails, the return value is one of the following error values.

Return code/value	Description
0	The system is out of memory or resources.
ERROR_BAD_FORMAT	The .exe file is invalid.
ERROR_FILE_NOT_FOUND	The specified file was not found.
ERROR_PATH_NOT_FOUND	The specified path was not found.

试验一下：

```
.386
.model flat, stdcall
option casemap :none
```

```
include windows.inc
include user32.inc
include kernel32.inc
include masm32.inc

includelib user32.lib
includelib kernel32.lib
includelib masm32.lib
include macro.asm

.data
   szCmdline db '\windows\notepad.exe',0
.data?
   buffer db 100 dup(?)

.CODE
START:

   invoke StdOut,CTXT("I will run a program...")
   invoke WinExec,ADDR szCmdline,SW_SHOWNORMAL
   invoke StdIn,addr buffer,sizeof buffer
   invoke ExitProcess,0

end START
```



前面提到，可以添加参数，我们可以实验一下

```
szCmdline      db    '\windows\notepad.exe c:\boot.ini',0
```

另外，还可以实验一下返回值，看看错误的情况下返回值是多少。

掌握了控制台的输入输出方式，我们就可以很容易的验证一些 API，很容易的就能看到结果。

用 MasmPlus 学习 Win32 汇编（3）

更改“开始”按钮


```
;#Mode=CON
.386
.model flat, stdcall
option casemap :none

include windows.inc
include user32.inc
include kernel32.inc
include masm32.inc

includelib user32.lib
includelib kernel32.lib
includelib masm32.lib
include macro.asm

.data?
    Handle1   dd   ?
    Handle2   dd ?
     buffer     db 100 dup(?)

.CODE
START:
        invoke FindWindow,CTEXT('Shell_TrayWnd'),NULL      ;get the handle of the taskbar
        mov   Handle1,eax
        invoke FindWindowEx,Handle1,0,CTEXT('Button'),NULL ;get the handle of the start button from the taskbar
        mov         Handle2,eax
       invoke SetWindowText,Handle2,CTEXT('Z..T')              ;set the text
       invoke SendMessage,Handle2,WM_MOUSEMOVE,0,0      ;used to refresh the caption of the start button
```

运行之后

```
    ;暂停显示,回车键关闭
    invoke StdIn,addr buffer,sizeof buffer
```

```
    invoke ExitProcess,0

end START
```

运行之后，就变为：

下面对程序进行简单的分析：

```
    invoke FindWindow,CTEXT('Shell_TrayWnd'),NULL      ;查找任务栏的 handle
    mov   Handle1,eax
    invoke FindWindowEx,Handle1,0,CTEXT('Button'),NULL
                ;开始键是任务栏上的 button，取得它的 handle
    mov         Handle2,eax
    invoke SetWindowText,Handle2,CTEXT('Z..T')              ;替换掉原来的文字
    invoke SendMessage,Handle2,WM_MOUSEMOVE,0,0     ;这个地方，我们给 button
```

发一个消息，假装鼠标从上面经过，重新绘制新的文字就会显示出来

从这个程序我们进行一下推广：

1.如果去掉最后一句，我们用鼠标滑过 button 字迹也会出现，可以试试看哦~

2.文字被改动之后，什么情况下会恢复为原来的呢？不妨自己手动试试看~

3. invoke FindWindow,CTEXT('Shell_TrayWnd'),NULL 其中的 Shell_TrayWnd 是 Class name。不必关心这个是怎么来的，如果我告诉你 xp 中的“计算器”的 Class name 是 SciCalc，可否试验一下如何修改它的标题呢？

4. invoke SendMessage,Handle2,WM_MOUSEMOVE,0,0 是发送“消息”，我们可以试试看发送一个“点击”的消息，WM_LBUTTONDOWN。运行之后，开始菜单会自动弹出来的哦~

继续控制台

上一次我们介绍了 Console 程序，不过我们仍然是沿用 DOS 下的编写方法，下面我们要开始学习一下 Windows 中的消息。

下面一段是我从 Iczelion 的 win32 汇编教程中“抄袭”过来的，原本是使用在 GUI 下面的。

```
.WHILE TRUE
invoke GetMessage,  ADDR msg, NULL, 0, 0
.BREAK .IF (!eax)
invoke TranslateMessage,  ADDR msg
invoke DispatchMessage,  ADDR msg
.ENDW
```

这时候我们的窗口已显示在屏幕上了。但是它还不能从外界接收消息。所以我们必须给它提供相关的消息。我们是通过一个消息循环来完成该项工作的。每一个模块仅有一个消息循环，我们不断地调用 GetMessage 从 Windows 中获得消息。GetMessage 传递一个 MSG 结构体给 Windows ，然后 Windows 在该函数中填充有关的消息，一直到 Windows 找到并填充好消息后 GetMessage 才会返回。在这段时间内系统控制权可能会转移给其他的应用程序。这样就构成了 Win16 下的多任务结构。如果 GetMessage 接收到 WM_QUIT 消息后就会返回 FALSE，使循环结束并退出应用程序。TranslateMessage 函数是一个是实用函数，它从键盘接受原始按键消息，然后解释成 WM_CHAR，在把 WM_CHAR 放入消息队列，由于经过解释后的消息中含有按键的 ASCII 码，这比原始的扫描码好理解得多。如果您的应用程序不处理按键消息的话，可以不调用该函数。DispatchMessage 会把消息发送给负责该窗口过程的函数。“

就是说，我们用 GetMessage 送 msg 这样一张小纸条给 Windows，等他批阅之后我再拿回来看看。如果返回的是要退出，那我就主动的“自我了断”。

定时器

许多时候，我们需要“每隔” XX 秒来做一件事情，比如我通常每隔一天检查一次信箱，或者每隔几个小时站起来找个 mm 聊天。在 DOS 下或者说面向过程的时候我们通常这样写：

```
开始计时
Repeat
    工作;
```

Until (计时器到点了);
计时结束

不过这样的写法很明显是比较浪费的，我们不得不每隔一段时间检查一下计时器，老师们肯定会因为学生经常看表而大发雷霆，在约会时不断看表也是赶走他人的最好方法。

最好的方法是到了时间有人来叫你，就好比中午我趴着睡觉的时候有人叫我“上班都半个小时了，快点起来~”也或者是突然之间觉得自己肚子饿了，大脑收到肚子饿这个消息，转而执行“觅食”的动作。

下面介绍的是 Windows 中的一种定时器

```
UINT_PTR SetTimer(        //创建一个定时器
  HWND hWnd,              // 所在线程的 handler，如果这个值为 Null 而下面 nIDEvent
                             这个参数不为 0，表示替换 nIDEvent 这个定时器
  UINT_PTR nIDEvent,      // 定时器 ID，多个定时器时，可以通过该 ID 判断是哪个定时器
  UINT uElapse,           // 时间间隔,单位为毫秒
  TIMERPROC lpTimerFunc   // 回调函数
);

BOOL KillTimer(           //释放一个定时器
   HWND hWnd,             //释放 handle 线程的定时器
   UINT_PTR uIDEvent      //释放编号为 uIDEvent 的定时器
);
```

上面 2 个 API 通常要配对使用，有创建就要有释放。来也匆匆去也冲冲才是好同志。

下面我们就要具体使用一下了，我们先实验一个简单的：

在控制台下，每隔 200 毫秒输出一个字符”.”。

```
;#Mode=CON

.586P
.MODEL Flat,StdCall
OPTION CASEMAP:NONE

INCLUDE       WINDOWS.INC
INCLUDE       KERNEL32.INC
INCLUDE       USER32.INC
INCLUDELIB    KERNEL32.LIB
INCLUDELIB    USER32.LIB
include       macro.asm
```

```
include         masm32.inc
includelib      masm32.lib

.DATA

Counter         db   20
lpMsg           MSG <>
TimerID         dd   ?
buffer      db 100 dup(?)

.CODE

TimerProc     proc
      invoke wsprintf,offset buffer,CTXT("."),NULL
            invoke StdOut, offset buffer
          dec Counter                                    ;计数减 1
          jnz NoKill                                        ;到 0 了没
          invoke    KillTimer, NULL, TimerID       ;释放定时器
          invoke    PostQuitMessage,NULL
NoKill:
            ret
TimerProc     endp

Start:
   invoke    SetTimer, NULL, 0, 200, addr TimerProc  ;200ms 一次，触发 TimerProc
   mov TimerID,eax                          ;存一下，释放的时候还要用

   .WHILE TRUE
            invoke GetMessage, ADDR lpMsg,NULL,0,0
            .BREAK .IF (!eax)
            invoke TranslateMessage, ADDR lpMsg
            invoke DispatchMessage, ADDR lpMsg
     .ENDW

     invoke    ExitProcess, NULL
END         Start
```

给笔记本做一个自动存盘功能

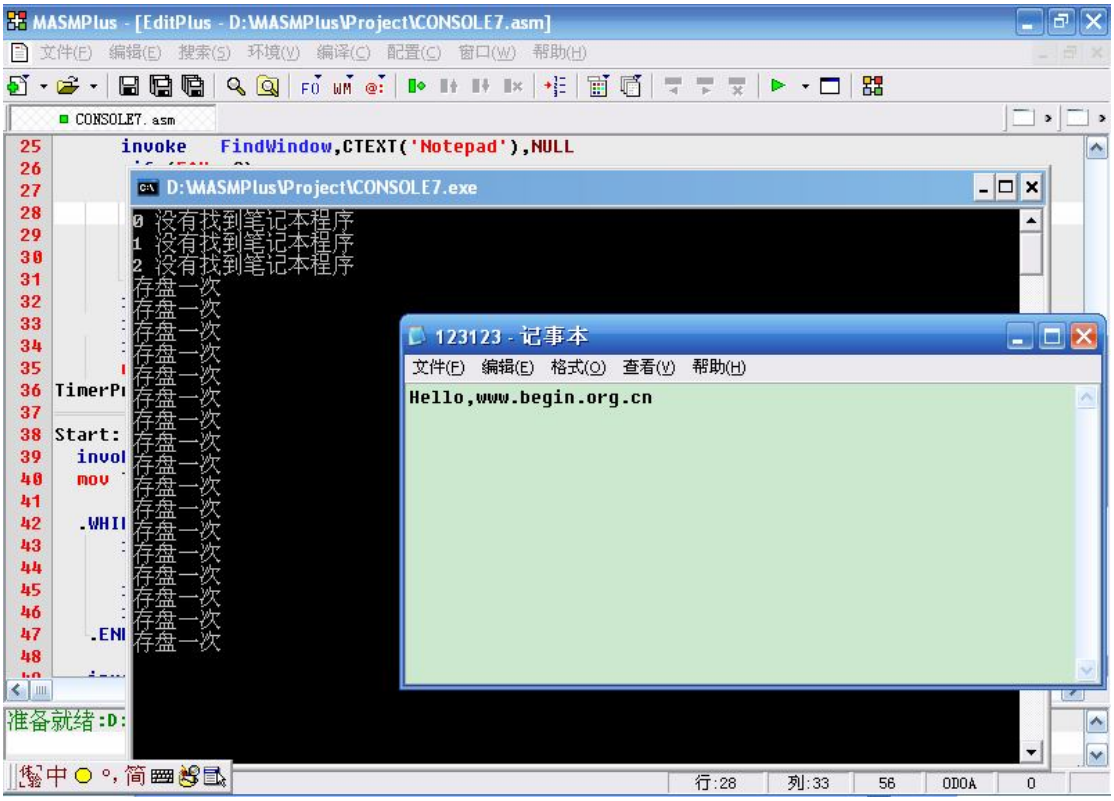
我们平时使用的 word 都有自动存盘的功能（特别是一边调试程序一边用 word 写文章的时候一定要记得打开…否则不知道什么时候就欲哭无泪了）。而 Windows 自带的 NotePad

没有这个功能。下面就让我们做一个能让它自动存盘的程序。

程序的基本思想是：使用 FindWindow API 取得 NotePad 的 handle，然后使用 sendmessage 功能给它发 WM_COMMAND 消息。

```
;#Mode=CON

.586P
.MODEL Flat,StdCall
OPTION CASEMAP:NONE

INCLUDE       WINDOWS.INC
INCLUDE       KERNEL32.INC
INCLUDE       USER32.INC
INCLUDELIB    KERNEL32.LIB
INCLUDELIB    USER32.LIB
include       macro.asm
include       masm32.inc
includelib    masm32.lib

.DATA

lpMsg         MSG <>
TimerID       dd   ?
iCount        dd    0
buffer        db 100 dup(?)
.CODE

TimerProc     proc
        invoke FindWindow,CTEXT('Notepad'),NULL
        .if (EAX==0)
            invoke wsprintf,offset buffer,CTEXT("%d 没有找到笔记本程序",10,13),iCount
                invoke StdOut, offset buffer
                inc iCount
                ret
          .endif
          invoke SendMessage,EAX,WM_COMMAND,3,0
        invoke wsprintf,offset buffer,CTEXT("存盘一次",13,10),NULL
          invoke StdOut, offset buffer
          ret
TimerProc     endp
```

```
Start:
    invoke    SetTimer, NULL, 0, 1000, addr TimerProc;200ms 一次，触发 TimerProc
    mov TimerID,eax                                    ;存一下，释
放的时候还要用

    .WHILE TRUE
        invoke GetMessage, ADDR lpMsg,NULL,0,0
        .BREAK .IF (!eax)
        invoke TranslateMessage, ADDR lpMsg
        invoke DispatchMessage, ADDR lpMsg
    .ENDW

    invoke    KillTimer, NULL, TimerID          ;释放定时器

    ;暂停显示,回车键关闭
    invoke StdIn,addr buffer,sizeof buffer
    invoke ExitProcess,0

END       Start
```



上面的程序很简单，我就不做更多的解释了，至于为什么我们知道发送 WM_COMMAND 消息，且听下回分解。在下一期中，我们会介绍一些工具的使用。

参考：

中文 SetTimer 这个 API 函数

1. http://blog.csdn.net/junhua198310/archive/2007/07/22/1702173.aspx

英文 MSDN 上关于 Timer 的专题介绍

2. http://msdn2.microsoft.com/en-us/library/ms632592.aspx

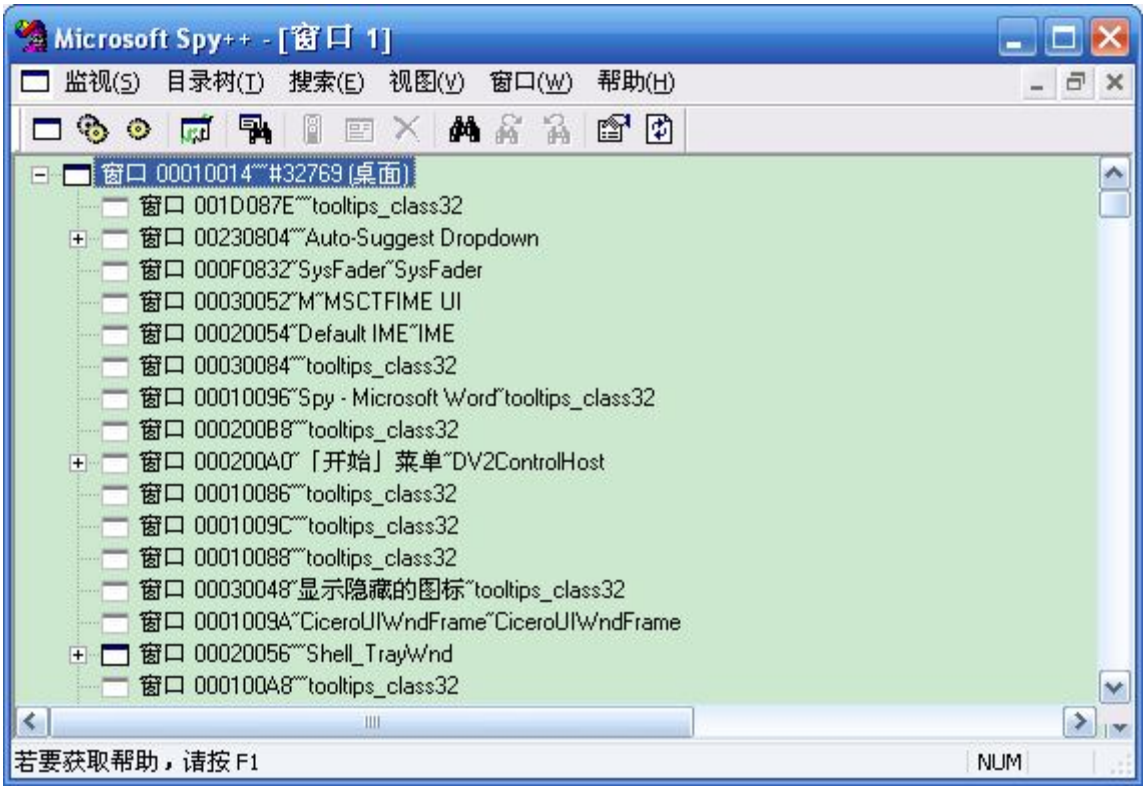
用 MasmPlus 学习汇编语言（4）

介绍一个工具 Spy++

上一期的最后，我卖了关子，没有告诉大家如何使知道记事本窗口的类。这一期我将揭晓谜底，就是 微软的 Spy++

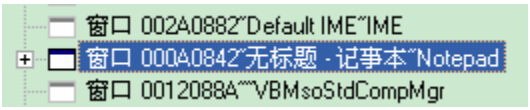
这款软件是用来查看 Windows 消息窗口等等的利器，当然除此之外还有很多类似的工具，比如国产的软件 Spy4Win 等等。 MS Spy++ 是老牌工具，并且也是其他软件模仿的对象，学会它一通百通，下面我将介绍这个软件。

启动之后，软件界面如下：



上面列出了当前系统中的全部窗口信息。

比如，当前我打开了一个“记事本程序”，在列出的信息中就能看到：

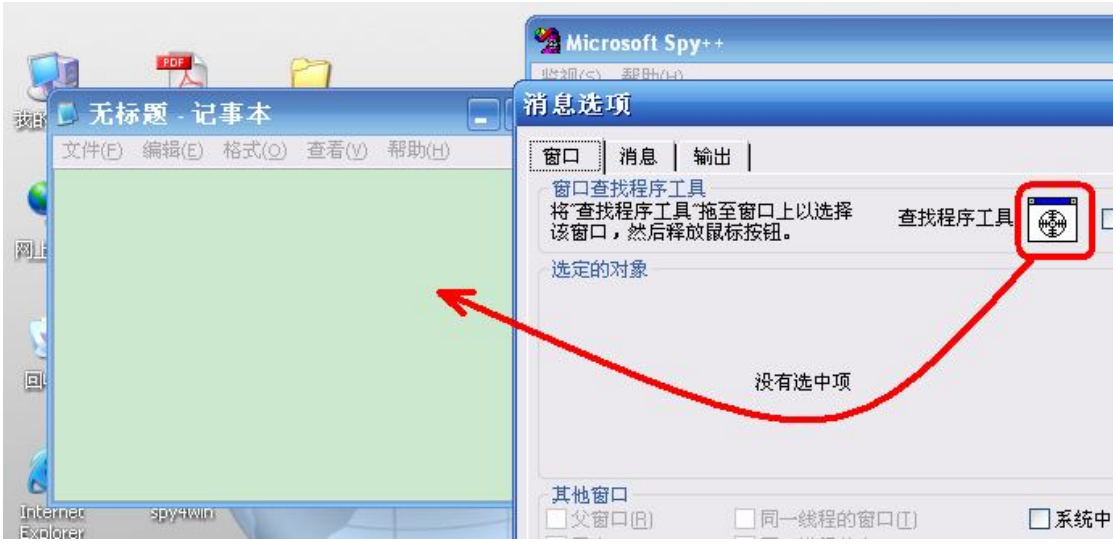


双击会弹出它的属性

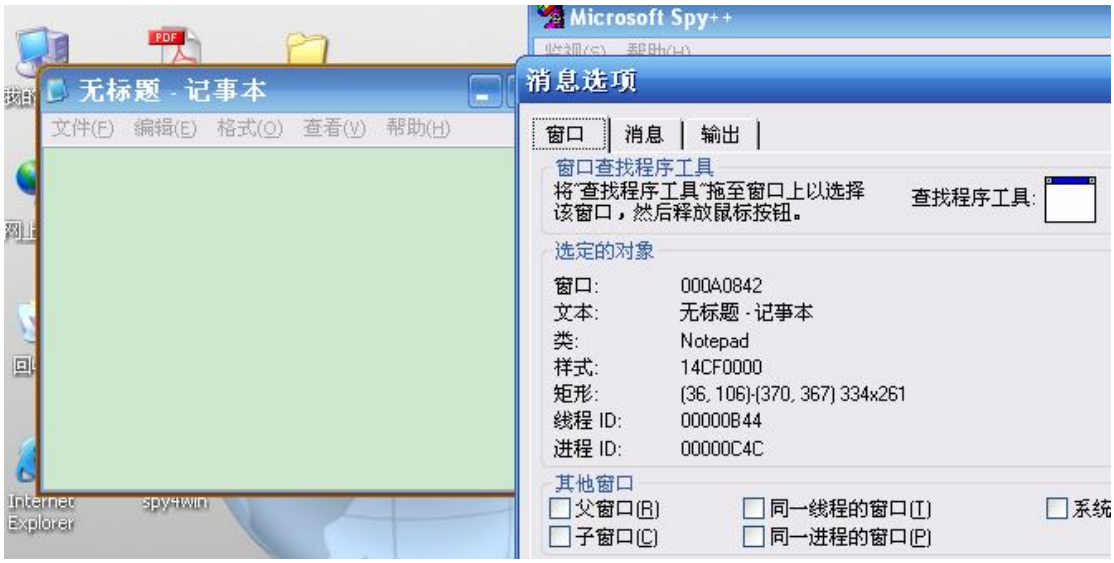
我们可以清楚地看到这个窗口的各种信息。

更多的操作可以参考它的帮助文件（中文的哦~）下面介绍查看一个窗口更简单的方法：选择“监视”下面的“日志消息”

拖动瞄准  镜到你要查看的窗口上：



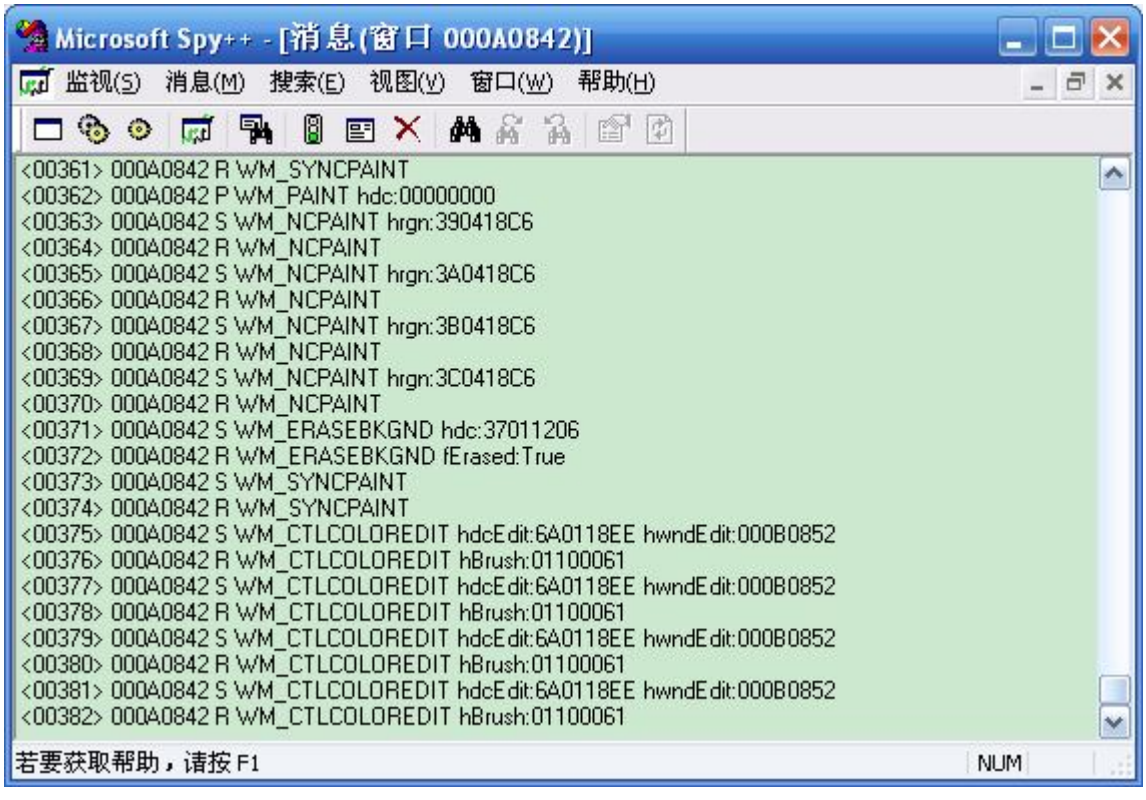
就可以显示出来关于这个窗口的信息：



在这里我们可以选择可以监视的“消息”，当有消息发送给我们监视的窗口时，Spy++会自动记录下来。点确定之后，就开始了监视，你可以随便监视一下 NotePad。


消息选项
窗口
消息
输出
要查看的消息(V):
ACM_OPENA
ACM_OPENW
ACM_PLAY
ACM_STOP
BM_CLICK
BM_GETCHECK
BM_GETIMAGE
BM_GETSTATE
BM_SETCHECK
BM_SETIMAGE
BM_SETSTATE
1009 选定的消息。
全选(A)
全部清除(C)
保存为默认设置(S)
消息組
键盘
鼠标
剪贴板
对话框
笔
非工作区
IME
MDI
DDE
AFX/MFC
常规
已注册
未知
WM_USER
编辑字段
按钮
列表框
组合框
滚动条
静态
列表视图
TreeView
工具提示
工具栏
标头
状态栏
选项卡控件
热键
UpDown
进度
跟踪条
MonthCal
DateTime
动画
ComboEx
页导航
ReBar
IP 地址
系统链接
确定
取消
帮助

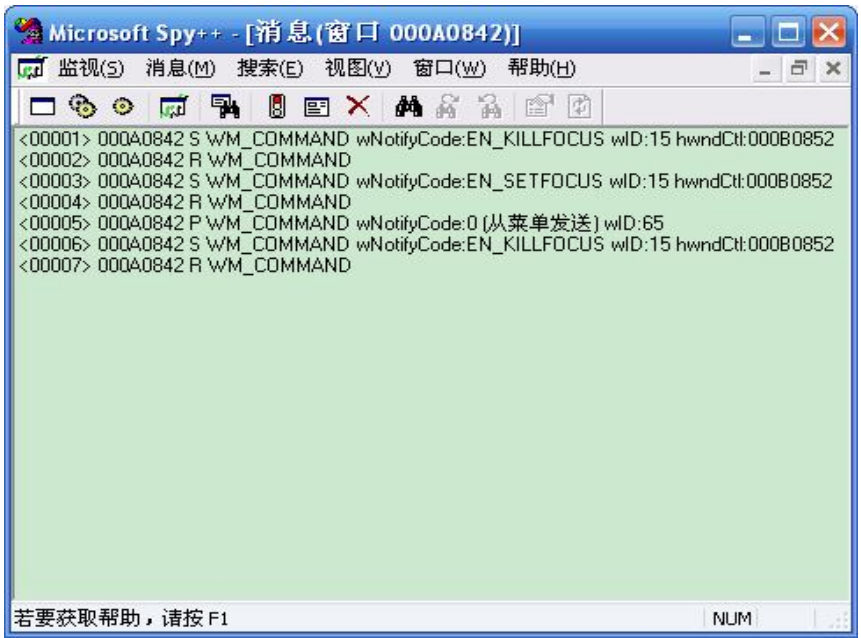
消息像潮水一般。


Microsoft Spy++ - [消息(窗口 000A0842)]
监视(S)
消息(M)
搜索(E)
视图(V)
窗口(W)
帮助(H)
<00361> 000A0842 R WM_SYNCPAINT
<00362> 000A0842 P WM_PAINT hdc:00000000
<00363> 000A0842 S WM_NCPAINT hrgn:390418C6
<00364> 000A0842 R WM_NCPAINT
<00365> 000A0842 S WM_NCPAINT hrgn:3A0418C6
<00366> 000A0842 R WM_NCPAINT
<00367> 000A0842 S WM_NCPAINT hrgn:3B0418C6
<00368> 000A0842 R WM_NCPAINT
<00369> 000A0842 S WM_NCPAINT hrgn:3C0418C6
<00370> 000A0842 R WM_NCPAINT
<00371> 000A0842 S WM_ERASEBKGND hdc:37011206
<00372> 000A0842 R WM_ERASEBKGND fErased:True
<00373> 000A0842 S WM_SYNCPAINT
<00374> 000A0842 R WM_SYNCPAINT
<00375> 000A0842 S WM_CTLCOLOREDIT hdcEdit:6A0118EE hwndEdit:000B0852
<00376> 000A0842 R WM_CTLCOLOREDIT hBrush:01100061
<00377> 000A0842 S WM_CTLCOLOREDIT hdcEdit:6A0118EE hwndEdit:000B0852
<00378> 000A0842 R WM_CTLCOLOREDIT hBrush:01100061
<00379> 000A0842 S WM_CTLCOLOREDIT hdcEdit:6A0118EE hwndEdit:000B0852
<00380> 000A0842 R WM_CTLCOLOREDIT hBrush:01100061
<00381> 000A0842 S WM_CTLCOLOREDIT hdcEdit:6A0118EE hwndEdit:000B0852
<00382> 000A0842 R WM_CTLCOLOREDIT hBrush:01100061
若要获取帮助，请按 F1
NUM

点这个位置，在开始/暂停之间切换。

消息太多了，我们必须过滤一下，留下我们需要的。比如，我们只想监视菜单的消息。在“消息”页面选择 WM_COMMAND 就够了。

确定之后，我们在上选于”，再听到的



NotePad“关查看监消息：

可以知道发送 65 弹出关于等等。之后，我们用下面的语句替换上一期程序中自动存盘的命令，就会让 NotePad 定期弹出关于窗口。

```
invoke SendMessage,EAX,WM_COMMAND,65,0
```

有了 Spy++, 我们就可以充分发挥自己的好奇心，东看看西看看，看看我们平常经常使用的各种软件究竟是什么东西。就像第一次见到望远镜的孩子。

关于这个工具的工作原理，在参考[1]上有一点介绍。现在看不明白不要紧，慢慢玩下去。我也希望读者每一次看都有新的收获。需要注意的是，如果你安装了杀毒软件，它们之间可能会有冲突，你在观测一些消息的时候也许会有一些差别，另外，可能导致 CPU 占用率居高不下。因此在使用的时候最好关闭之。

涂鸦之作

一个伟人曾经说过“当你手中只有一把榔头的时候，世界上的一切问题都是钉子”。在上一期，我们学习了一点如何绘图，下面就用这些知识做点乱写乱画的事情。在此之前，还要再学一点东西，免得每次只能写“大王八”三个字，太没有创意了（忽然发现这三个字非常简单，故儿童常常使用，并且不容易错。如果中国作协的那个老头给人题这个字，想必也用不到贾平凹出来捧场）。

下面要玩的就是在 NotePad 上“写字”，不过不是打开敲进去的，而是画上去的。Windows 是图形操作系统，它上面的程序一般都会有窗口，窗口上通常都会有一个叫做 DC 的东西，至于具体是什么，后面会讲~ 反正您就记着，DC 就好比广告牌面向马路的那个面，画点美女汽车都是在它上面。

除此之外，还要介绍一下创建“一种逻辑字体”。这个名字听起来非常专业，实际上就是设定一个我们需要的字体，给它一个编号（handle），然后我们作画的时候直接说“要 XXX 号的字体”（拿着 handle 给出去就可以了），就是这么简单。

```
HFONT CreateFont(
  int  nHeight,              // 字体高度
  int  nWidth,               // 平均宽度
  int  nEscapement,          // 旋转角度
  int  nOrientation,         // 每个字的倾斜度
  int  fnWeight,             // 字体粗细
  DWORD  fdwItalic,          // 要倾斜否
  DWORD  fdwUnderline,       // 要下划线否
  DWORD  fdwStrikeOut,       // 要删除线否
  DWORD  fdwCharSet,         // 字符集
  DWORD fdwOutputPrecision,  // 输出精度。输出精度定义了输出与所要求的
                             //字体高度、宽度、字符方向等的接近程度。
  DWORD  fdwClipPrecision,   //指定剪辑精度。剪辑精度定义了当字符的
                             //一部分超过剪辑区域时对字符的剪辑方式
  DWORD  fdwQuality,         //输出质量
                               包括：
                                      DEFAULT_QUALITY （默认质量）
                                      DRAFT_QUALITY （草稿质量）
                                      PROOF_QUALITY （正稿质量）
DWORD  fdwPitchAndFamily,    //指定字体的字符间距和族
  LPCTSTR  lpszFace          //它指定的所用的字体名。
                             //如果 lfFaceName 为 NULL，
                             //图形设备接口将使用默认的字体名。
);
```

另外一个函数就是 SelectObject ，意思是我要用什么什么作画。这个“什么什么”可以是某个画笔，某张粘贴画，或者某个字体

The SelectObject function selects an object into the specified device context (DC). The new object replaces the previous object of the same type.

```
HGDIOBJ SelectObject(
  HDC  hdc,           // handle to DC
  HGDIOBJ  hgdiobj    // handle to object
```

);

Parameters

hdc

[in] Handle to the DC.

hgdiobj

[in] Handle to the object to be selected. The specified object must have been created by using one of the following functions.

程序如下：

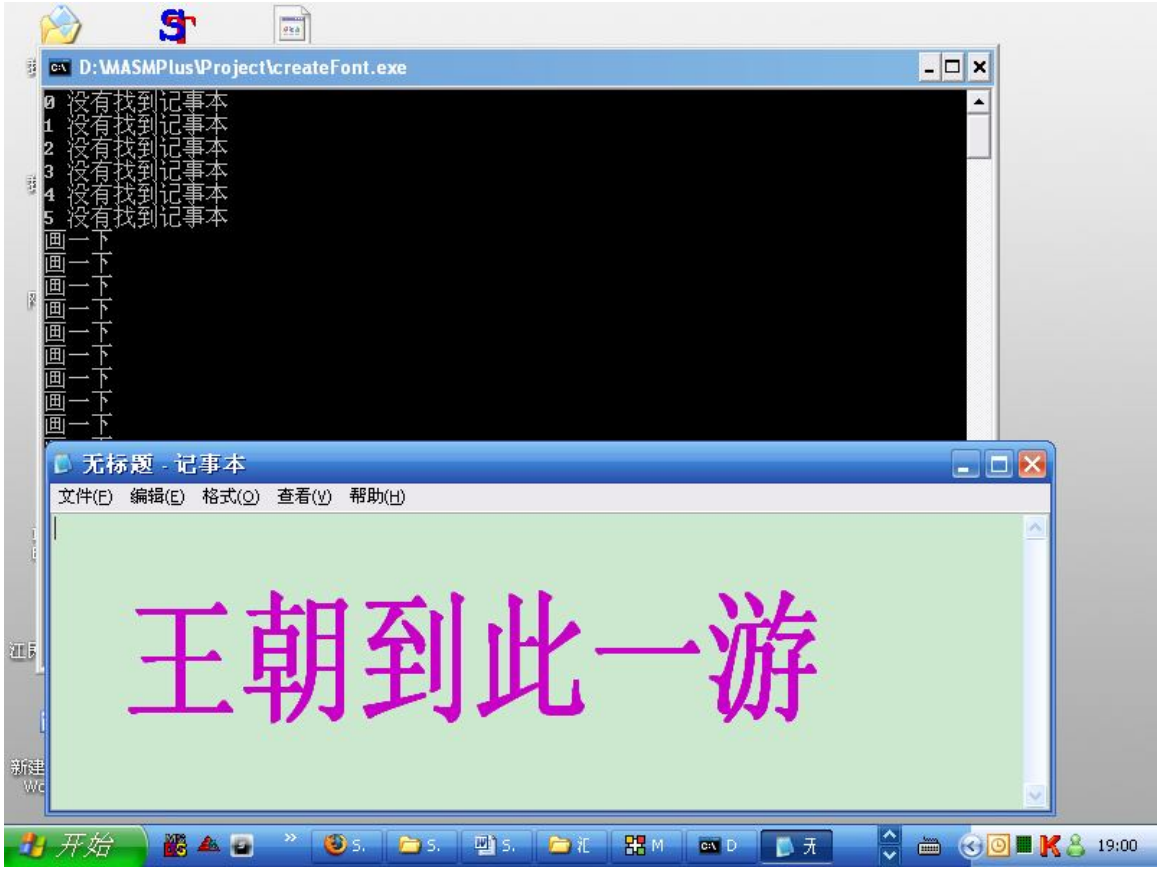
```
;#Mode=CON
;在内存中查找“NotePad”，并且在上面写字
.586P
.MODEL Flat,StdCall
OPTION CASEMAP:NONE

INCLUDE    WINDOWS.INC
INCLUDE    KERNEL32.INC
INCLUDE    USER32.INC
INCLUDELIB  KERNEL32.LIB
INCLUDELIB  USER32.LIB
include      macro.asm
include      masm32.inc
includelib   masm32.lib
include      gdi32.inc
includelib   gdi32.lib

   ; ----------------------------------
   ; INPUT red, green & blue BYTE values
   ; OUTPUT DWORD COLORREF value in eax
   ; ----------------------------------
     RGB MACRO red, green, blue      ;输入 RGB，合成一个 32 位值放在 EAX
中
       xor eax, eax
       mov ah,  blue    ; blue
       mov al,  green   ; green
       rol eax, 16
       mov al,  red     ; red
     ENDM
```

```
.DATA?

lpMsg      MSG <>
TimerID    dd  ?
iCount     dd  ?
buffer     db 100 dup(?)

hMine         dd ?          ;找到的 NotePad 的 handle
hdc        dd ?          ;找到的 NotePad 的 DC 的 Hand
hFont         dd ?          ;创建一种字体
.CODE

TimerProc  proc
       local  cfRGB:COLORREF
       local  fnText:HGDIOBJ
      invoke   FindWindow,CTEXT('notepad'),NULL
      .if (EAX==0)
        invoke  wsprintf,offset  buffer,CTEXT("%d 没有找到记事本
",10,13),iCount
           invoke StdOut, offset buffer
           inc   iCount
           ret
       .endif
       mov hMine,EAX                  ;保存 NotePad 的 Handler

       invoke GetDC,EAX               ;取得 DC
       mov hdc,EAX

       invoke CreateFont,\
                    100,\
                    40,\
                    0,\
                    0,\
                    FW_BLACK,\
                    0,\
                    0,\
                    0,\
                    ANSI_CHARSET,\
                    OUT_DEFAULT_PRECIS,\
                    CLIP_CHARACTER_PRECIS,\
                    DEFAULT_QUALITY,\
                    DEFAULT_PITCH or FF_SWISS,\
                    CTEXT("宋体")
```

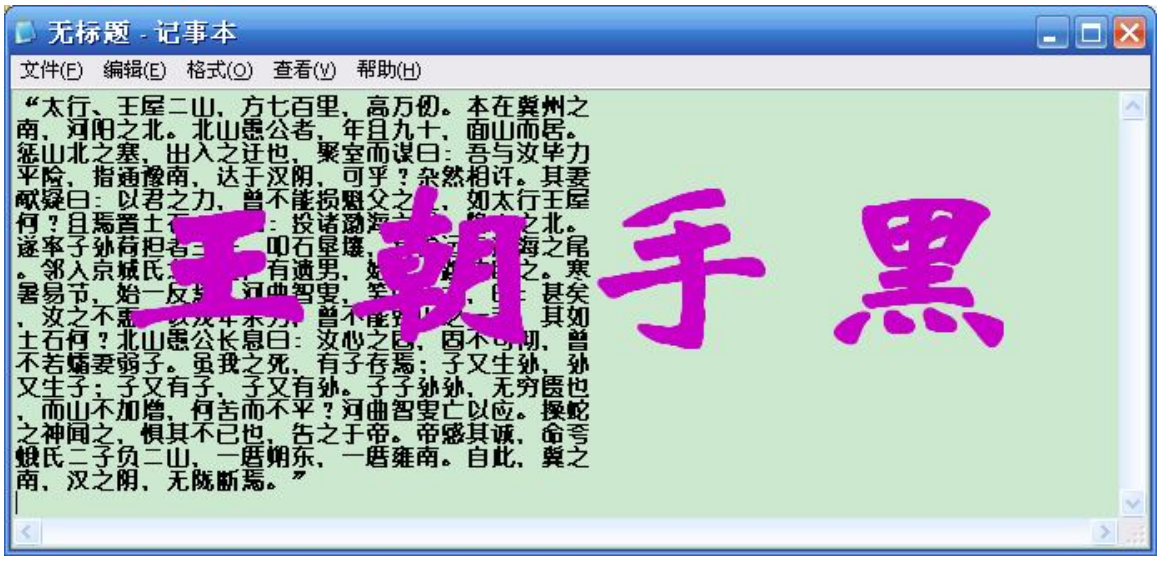
```
        mov  hFont,eax     ;保存一下这种字体的 handle
        ;不知道为什么，有资料说返回值是之前的字体的 handle
        ;害得我调试了好长时间

        invoke SelectObject,hdc,hFont     ;将这种字体设置给 DC

        RGB 200,200,50
        invoke SetTextColor,hdc,eax          ;准备好笔

        RGB 0,0,255
        invoke SetBkMode,hdc,eax          ;准备好背景

        invoke TextOut,hdc,50, 50,CTEXT('王朝到此一游'),12   ;写字啦

        invoke DeleteObject,hFont            ;释放字体
        invoke ReleaseDC,hMine,hdc           ;释放 DC

      invoke wsprintf,offset buffer,CTEXT("画一下",13,10),NULL
        invoke StdOut, offset buffer
        ret
TimerProc   endp

Start:
  invoke  SetTimer, NULL, 0, 1000, addr TimerProc  ;200ms 一次，触发
TimerProc
  mov TimerID,eax                    ;存一下，释放的时候还要用

  .WHILE TRUE
        invoke GetMessage, ADDR lpMsg,NULL,0,0
        .BREAK .IF (!eax)
        invoke TranslateMessage, ADDR lpMsg
        invoke DispatchMessage, ADDR lpMsg
    .ENDW

   invoke  KillTimer, NULL, TimerID      ;释放定时器

    ;暂停显示,回车键关闭
    invoke StdIn,addr buffer,sizeof buffer
    invoke ExitProcess,0

END     Start
```

程序的结构和上一期的没有什么差别，就是找 Notepad-→找 DC—〉准备字体—〉

画，然后就收尾了。



特别需要注意的是 CreateFont API 中 如果使用 **DWORD** *fdwCharSet*, // 字符集设置为 ANSI_CHARSET 是无法输出“华文新魏”这样的字体的，输出始终为“宋体”；如果希望输出其他字体，需要将这项设置为 DEFAULT_CHARSET。



另外，告诉大家可以通过 invoke GetDC,HWND_DESKTOP 取得桌面的 DC。我们就可以在屏幕上写字了。

参考：

1．http://msdn2.microsoft.com/en-us/library/ms534214.aspx

参考：

[1] Visual C++ Team Blog

http://blogs.msdn.com/vcblog/archive/2007/01/16/spy-internals.aspx 关于 SPY++ 内部工作机制的一点介绍

Spy++ Internals

Hi, my name is Pat Brenner and I'm a software design engineer on the Visual C++ libraries team. I recently rejoined the Visual C++ team after almost 10 years in the Visual Studio environment team. I am also the owner of Spy++. While I did not originally write Spy++, I owned it from around 1993 until around 2003, and now that I am back on the Visual C++ team it has been returned to me. Today I'm going to spend some time talking about Spy++ internals.

Spy++ is made to be an observer (and not a modifier) of the system around it. While it can do other things as well, the main purpose of Spy++ is to log messages that are being passed around in Windows. Spy++ accomplishes this by the use of three global message hooks: a WH_GETMESSAGE hook, which hooks a message posted to a window (via PostMessage); a WH_CALLWNDPROC hook, which hooks a message sent to a window (via SendMessage); and a WH_CALLWNDPROCRET hook, which hooks the return of a message sent to a window (via SendMessage).

Because the hooks must be global (so that they can hook messages to any window in the system) they must reside in a DLL. This DLL is loaded into every running process as soon as the hooks are set (via SetWindowsHookEx), and remains loaded in the processes until the hooks are unhooked (via UnhookWindowsHookEx).

The hook DLL communicates the message information with the Spy++ application via a circular queue, so there are a number of synchronization requirements.

- Since the hook DLL is loaded into every process, the circular queue resides in a shared data section, and there is a pair of mutexes that controls access to the queue. A writer mutex is used to synchronize write accesses between the loaded copies of the hook DLL, and an access mutex is used to synchronize access between the writers and the Spy++ application itself, which is reading from the queue.
- Because the queue is circular, there needs to be synchronization between the reading and the writing, so there are two events used for this purpose. A read event is signaled when the Spy++ application reads a packet of message data from the queue, and a written event is signaled when a copy of the hook DLL writes a packet of message data to the queue.
- Synchronization is also required with regard to the current read and write locations in the queue, so there are offsets maintained (as shared data) which specify the next read location and the next write location in the queue. There is also a count maintained (in shared data) which is the number of packets that have been written but not yet read.

So the writers (in the hook DLL), when a message comes through one of their hooks, do the following:

- Take the writer mutex.

- Take the access mutex.
- Obtain a message packet (section of shared queue) large enough to store the data for the message.
- Copy the message data to the packet.
- Increment the count of message packets in the queue.
- Increment the write offset by the size of the message packet taken.
- Set the written event.
- Release the access mutex.
- Release the writer mutex.

The reader, which resides in a loop in a thread in the Spy++ application, does the following.

- Check the written event. Once it is set:
- Take the access mutex.
- Copy the message data from the message packet in the shared queue.
- Decrement the count of message packets in the queue.
- If the message packet count is now zero, reset the written event.
- Release the access mutex.
- Set the read event.
- Sends the copied packet data in a message to a hidden window owned by the main thread.

The hidden window processes the message by looping through the current message loggers (since there may be more than one message logger open) and calling each active logger. The logger will then apply any filtering (since the message may have been for a window or message that the logger is not interested in) and if necessary will display the message in its view.

Once Spy++ starts hooking the messages in the system, we want the writers to stay ahead of the reader in the queue, but we don't want the writers to overtake and pass the reader. So if obtaining a new packet for a writer would overtake the reader, the writer has to loop, doing the following:

- Reset the read event.
- Wait for the read event to be set.
- Check for either of the following conditions:
  1. The message packet count has returned to zero (the reader is fully caught up).
  2. The read offset is far enough ahead (write offset plus new packet size does not pass read offset).

That pretty much covers what I wanted to talk about today. There are a couple of other things worth mentioning.

- You may notice that Spy++ does not log any messages for its own windows. Obviously, this is because this would cause an infinite set of messages to be sent throughout the

system. So Spy++ knows its own process and thread identifiers and filters out any messages sent to windows owned by them.

- All the data that Spy++ displays is a copy of the original data. Since much of the data passed in messages may only be valid for the lifetime of the message, Spy++ copies any data that it needs to display into the queue, and then copies it again when the message is placed in the message log, since the data in the queue will be overwritten as well.
- Ever since Spy++ was first written, using it has occasionally resulted in the hanging of the system, with the only recourse being to cold-boot the computer. After a recent conversation with a co-worker, I (finally!) realized that this was because all the waits (for mutexes or events) were infinite, rather than using timeouts. This could easily cause a hang, particularly if the Spy++ application crashed while it held the access mutex, since then all the copies of the hook DLL would block in their hooks waiting for the access mutex. So I have made a fix for this (by using timeouts on the waits) which should substantially reduce the trouble that Spy++ can cause in the system. This also made Spy++ much easier for me to debug, since I no longer have to resort to powering off/on my computer when a bug is encountered in the message processing.

Thanks, and I hope you found this interesting. I welcome any questions you might have about Spy++. I welcome feature requests for future versions as well, but keep in mind the "Spy++ is an observer, not a modifier" statement I made earlier.

Pat Brenner
Visual C++ Libraries Team
Published Tuesday, January 16, 2007 9:06 PM by vcblog

[2] Spy4Win http://www.ccrun.com/spy4win/

用 MasmPlus 学习汇编语言（5）

这一期，我们会讲述很多关于绘制图形方面的知识，其中最简单的莫过于画线的语句了。MoveToEx 相当于“拿着画笔到”X,Y 位置，LineTo 相当于“落笔，画到”X,Y 位置。凭借着两个简单的语句，我们就能够创建很多有意思的图形。

分形

下面是一个绘图程序，能够绘制一个类似于树的图形。核心是一段递归程序：

```
Tree 起始X坐标，起始Y坐标，长度，方向

    绘制树枝，长度取2/3

    if 达到需要的精度 then

        只绘制树干

    else

        绘制左树干

        绘制右树干

    endif
```

（这个只是基本框架，用来描述算法思想）

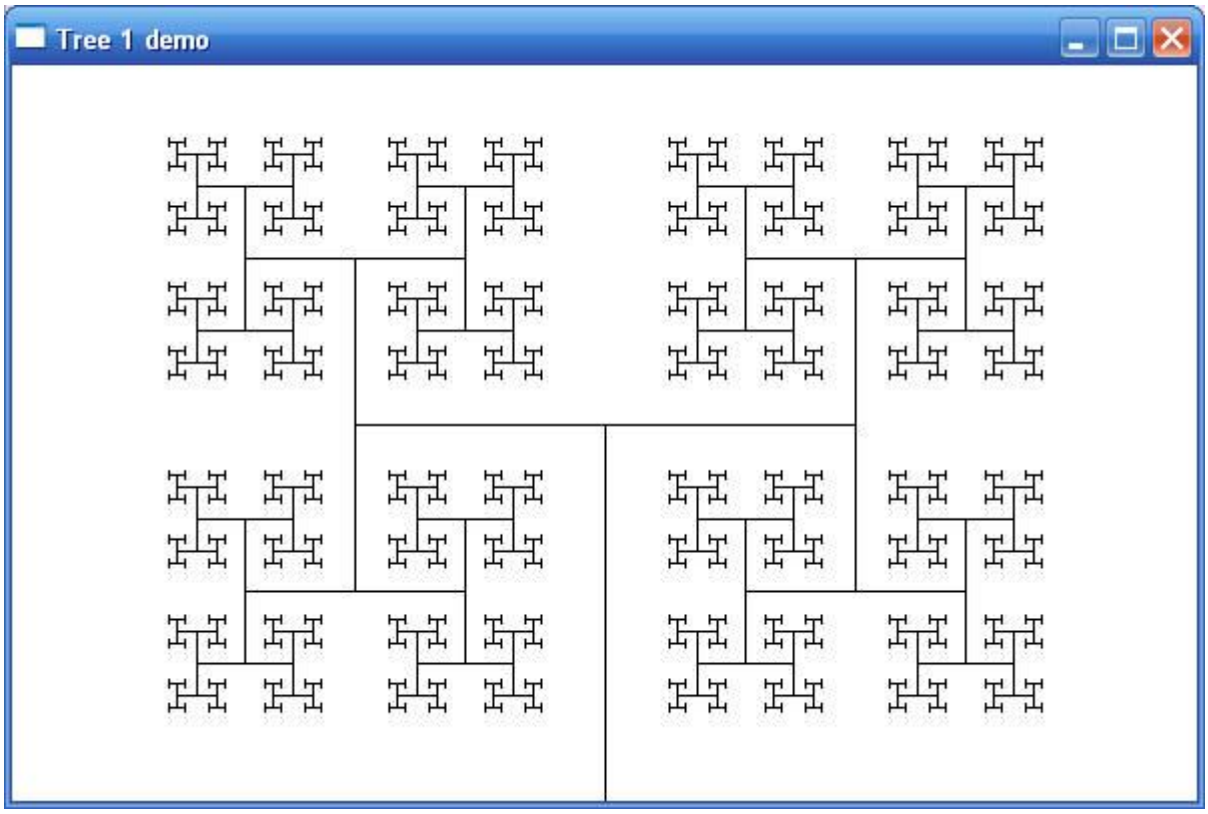
```
;MASMPlus 代码模板 - 普通的 Windows 程序代码
.386
.Model Flat, StdCall
Option Casemap :None

Include windows.inc
Include user32.inc
Include kernel32.inc
Include gdi32.inc

includelib gdi32.lib
IncludeLib user32.lib
IncludeLib kernel32.lib
include macro.asm

WinMain PROTO :DWORD,:DWORD,:DWORD,:DWORD
WndProc PROTO :DWORD,:DWORD,:DWORD,:DWORD


.DATA
szAppName db "Tree 1",0

.DATA?
hInstance dd ?
cxClient dd ?
cyClient dd ?
```

```
hdc dd ?
.CODE
START: ;从这里开始执行

invoke GetModuleHandle,NULL
mov hInstance,eax
invoke WinMain,hInstance,NULL,NULL,SW_SHOWDEFAULT
invoke ExitProcess,0

WinMain proc hInst:DWORD,hPrevInst:DWORD,CmdLine:DWORD,iCmdShow:DWORD
LOCAL wndclass :WNDCLASSEX
LOCAL msg :MSG
local hWnd :HWND
mov wndclass.cbSize,sizeof WNDCLASSEX
mov wndclass.style,CS_HREDRAW or CS_VREDRAW
mov wndclass.lpfnWndProc,offset WndProc

mov wndclass.cbClsExtra,0
mov wndclass.cbWndExtra,0

push hInst
pop wndclass.hInstance

invoke LoadIcon,NULL,IDI_APPLICATION
mov wndclass.hIcon,eax

invoke LoadCursor,NULL,IDC_ARROW
mov wndclass.hCursor,eax

invoke GetStockObject,WHITE_BRUSH
mov wndclass.hbrBackground,EAX

mov wndclass.lpszMenuName,NULL
mov wndclass.lpszClassName,offset szAppName

mov wndclass.hIconSm,0

invoke RegisterClassEx, ADDR wndclass
.if (EAX==0)
invoke MessageBox,NULL,CTXT("This program requires Windows NT!"),addr
szAppName,MB_ICONERROR
ret
.endif
```

```
invoke CreateWindowEx,
NULL,
ADDR szAppName, ;window class name
CTXT("Tree 1 demo"), ;window caption
WS_OVERLAPPEDWINDOW, ;window style
CW_USEDEFAULT, ;initial x position
CW_USEDEFAULT, ;initial y position
CW_USEDEFAULT, ;initial x size
CW_USEDEFAULT, ;initial y size
NULL, ;parent window handle
NULL, ;window menu handle
hInstance, ;program instance handle
NULL ;creation parameters
mov hWnd,eax

invoke ShowWindow,hWnd,iCmdShow
invoke UpdateWindow,hWnd

StartLoop:
invoke GetMessage,ADDR msg,NULL,0,0
cmp eax, 0
je ExitLoop
invoke TranslateMessage, ADDR msg
invoke DispatchMessage, ADDR msg
jmp StartLoop
ExitLoop:

mov eax,msg.wParam
ret
WinMain endp

Tree Proc xPos:DWORD,yPos:DWORD,Len:DWORD,Direction:DWORD
LOCAL Dir1,Dir2:DWORD

xor edx,edx ;Len= Len * 2 / 3
mov eax,Len
shl eax,1
mov ebx,3
div ebx
mov Len,eax

invoke MoveToEx,hdc,xPos,yPos,NULL
```

```
mov ecx,Len ;下面计算要绘制一个枝干的坐标
mov eax,xPos
mov ebx,yPos
.if (Direction==0) ;左
sub eax,ecx
.endif
.if (Direction==1) ;下
add ebx,ecx
.endif
.if (Direction==2) ;右
add eax,ecx
.endif
.if (Direction==3) ;上
sub ebx,ecx
.endif
mov xPos,eax
mov yPos,ebx

invoke LineTo,hdc,xPos,yPos ;绘制一个枝干

.if (Len>1) ;如果这个长度大于 1 就继续绘制

mov eax,Direction ;下一个方向左侧=（方向+5） mod 4
add eax,5
xor edx,edx
mov ebx,4
div ebx
mov Dir1,edx
invoke Tree,xPos,yPos,Len,Dir1
;这个地方可以添加一个循环，为绘制延时，创造动画效果
;xor cx,cx
;@@:
;loop @b
mov eax,Direction ;下一个方向左侧=（方向+3） mod 4
add eax,3
xor edx,edx
mov ebx,4
div ebx
mov Dir2,edx
invoke Tree,xPos,yPos,Len,Dir2
.endif
```

```
ret
Tree Endp

WndProc proc hwnd:DWORD,message:DWORD,wParam :DWORD,lParam :DWORD
LOCAL ps :PAINTSTRUCT

.if message == WM_SIZE
mov eax,lParam ;cxClient = LOWORD (lParam)
and eax,0FFFFh
mov cxClient,eax

mov eax,lParam
shr eax,16
mov cyClient,eax ;cyClient = HIWORD (lParam)

ret
.elseif message == WM_PAINT

invoke BeginPaint,hwnd,addr ps
mov hdc,eax

mov eax,cxClient
shr eax,1

mov ebx,cyClient
shr ebx,1
add ebx,100

invoke Tree,eax,cyClient,ebx,3

invoke EndPaint,hwnd,addr ps
ret
.elseif message == WM_DESTROY

invoke PostQuitMessage,NULL
ret
.endif

invoke DefWindowProc,hwnd, message, wParam, lParam
ret

WndProc endp
END START
```

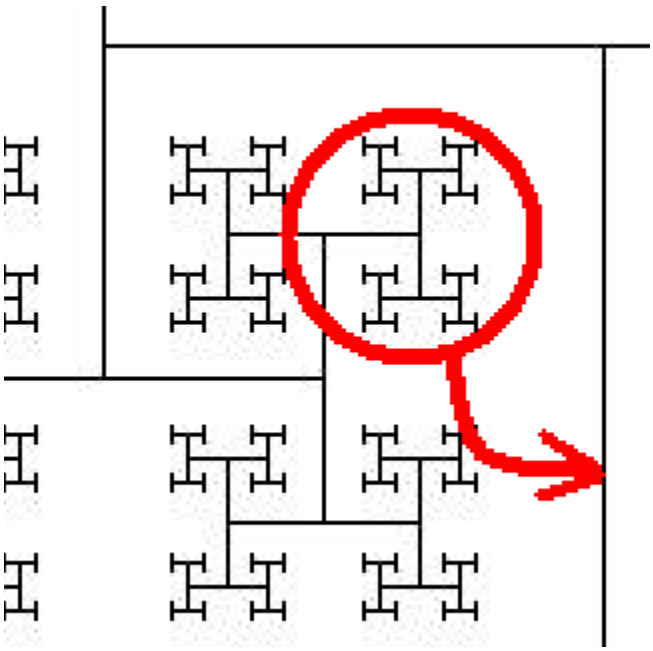


这个程序还不完美，我们还可以调整一下颜色，看得好看一些，还可以改造一下，

改为三叉树，或者改为随机的 n 叉树，每次运行结果都不同... ...

看到这个程序，我提出一个数学问题：绘制的树干长度，为 Len 的多少时，能保证

不发生重叠呢？



具体就是说上图中指出的部分什么情况下不会碰上箭头指出来的那条线。

Koch Snowflake（科赫雪花）

参考 1：http://baike.baidu.com/view/83243.htm

分形

谁创立了分形几何学？

1973 年，曼德勃罗（B.B.Mandelbrot）在法兰西学院讲课时，首次提出了分维和分形几何的设想。分形（Fractal）一词，是曼德勃罗创造出来的，其愿意具有不规则、支离破碎等意义，分形几何学是一门以非规则几何形态为研究对象的几何学。由于不规则现象在自然界是普遍存在的，因此分形几何又称为描述大自然的几何学。分形几何建立以后，很快就引起了许多学科的关注，这是由于它不仅在理论上，而且在实用上都具有重要价值。

分形几何与传统几何相比有什么特点：

⑴从整体上看，分形几何图形是处处不规则的。例如，海岸线和山川形状，从远距离观察，其形状是极不规则的。

⑵在不同尺度上，图形的规则性又是相同的。上述的海岸线和山川形状，从近距离观察，其局部形状又和整体形态相似，它们从整体到局部，都是自相似的。当然，也有一些分形几何图形，它们并不完全是自相似的。其中一些是用来描述一般随即现象的，还有一些是用来描述混沌和非线性系统的。

什么是分维？

在欧氏空间中，人们习惯把空间看成三维的，平面或球面看成二维，而把直线或曲线看成一维。也可以梢加推广，认为点是零维的，还可以引入高维空间，但通常人们习惯于整数的维数。分形理论把维数视为分数，这类维数是物理学家在研究混沌吸引子等理论时需要引入的重要概念。为了定量地描述客观事物的“非规则”程度，1919 年，数学家从测度的角度引入了维数概念，将维数从整数扩大到分数，从而突破了一般拓扑集维数为整数的界限。

分维的概念我们可以从两方面建立起来：一方面，我们首先画一个线段、正方形和立方体，它们的边长都是 1。将它们的边长二等分，此时，原图的线度缩小为原来的 1/2，而将原图等分为若干个相似的图形。其线段、正方形、立方体分别被等分为 2^1、2^2 和 2^3 个相似的子图形，其中的指数 1、2、3，正好等于与图形相应的经验维数。一般说来，如果某图形是由把原图缩小为 1/a 的相似的 b 个图形所组成，有：

a^D=b, D=logb/loga

的关系成立，则指数 D 称为相似性维数，D 可以是整数，也可以是分数。另一方面，当我们画一根直线，如果我们用 0 维的点来量它，其结果为无穷大，因为直线中包含无穷多个点；如果我们用一块平面来量它，其结果是 0，因为直线中不包含平面。那么，用怎样的尺度来量它才会得到有限值哪？看来只有用与其同维数的小线段来量它才会得到有限值，而这里直线的维数为 1（大于 0、小于 2）。与此类似，如果我们画一个 Koch 曲线，其整体是一条无限长的线折叠而成，显然，用小直线段量，其结果是无穷大，而用平面量，其结果是 0（此曲线中不包含平面），那么只有找一个与 Koch 曲线维数相同的尺子量它才会得到有限值，而这个维数显然大于 1、小于 2，那么只能是小数（即分数）了，所以存在分维。其实，Koch 曲线的维数是 1.2618……。

Fractal（分形）一词的由来

据曼德勃罗教授自己说，fractal 一词是 1975 年夏天的一个寂静夜晚，他在冥思苦想之余偶翻他儿子的拉丁文字典时，突然想到的。此词源于拉丁文形容词 fractus，对应的拉丁文动词是 frangere（“破碎”、“产生无规碎片”）。此外与英文的 fraction（“碎片”、“分数”）及 fragment（“碎片”）具有相同的词根。在 70 年代中期以前，曼德勃罗一直使用英文 fractional 一词来表示他的分形思想。因此，取拉丁词之头，撷英文之尾的 fractal，本意是不规则的、破碎的、分数的。曼德勃罗是想用此词来描述自然界中传统欧几里德几何学所不能描述的一大类复杂无规的几何对象。例如，弯弯曲曲的海岸线、起伏不平的山脉，粗糙不堪的断面，变幻无常的浮云，九曲回肠的河流，纵横交错的血管，令人眼花僚乱的满天繁星等。它们的特点是，极不规则或极不光滑。直观而粗略地说，这些

对象都是分形。

分形的定义

曼德勃罗曾经为分形下过两个定义：

（1）满足下式条件

Dim(A)>dim(A)

的集合 A，称为分形集。其中，Dim(A)为集合 A 的 Hausdoff 维数（或分维数），dim(A)为其拓扑维数。一般说来，Dim(A)不是整数，而是分数。

（2）部分与整体以某种形式相似的形，称为分形。

然而，经过理论和应用的检验，人们发现这两个定义很难包括分形如此丰富的内容。实际上，对于什么是分形，到目前为止还不能给出一个确切的定义，正如生物学中对“生命”也没有严格明确的定义一样，人们通常是列出生命体的一系列特性来加以说明。对分形的定义也可同样的处理。

（i）分形集都具有任意小尺度下的比例细节，或者说它具有精细的结构。

（ii）分形集不能用传统的几何语言来描述，它既不是满足某些条件的点的轨迹，也不是某些简单方程的解集。

（iii）分形集具有某种自相似形式，可能是近似的自相似或者统计的自相似。

（iv）一般，分形集的“分形维数”，严格大于它相应的拓扑维数。

（v）在大多数令人感兴趣的情形下，分形集由非常简单的方法定义，可能以变换的迭代产生。

环境变量

以前 DOS 下有一种叫做“环境变量”的东西，最常用的就是“Path=”。比如，我们设定 “path=c:\masm611\bin”，当我们在某一个目录下输入命令“ml a.asm”的时候如果当前目录下没有 “ml.exe” “ml.com” 或者“ml.bat”的时候，系统会自动在 path 给定的路径下搜索这些文件，如果找到就会自动运行。 Windows 也设计了环境变量，在“系统属性”“高级”选项卡中：


系统属性
常规
计算机名
硬件
高级
系统还原
自动更新
远程
要进行大多数改动，您必须作为管理员登录。
性能
视觉效果，处理器计划，内存使用，以及虚拟内存
设置(S)
用户配置文件
与您登录有关的桌面设置
设置(E)
启动和故障恢复
系统启动，系统失败和调试信息
设置(T)
环境变量(N)
错误报告(R)
确定
取消
应用(A)

选择“环境变量”按钮：

上图就是我的环境变量设置，如果你安装过 VC 或者 Delphi 那么还会看到更丰富的 path 设置，这些语言在编译的时候就是在参考 path 给定的路径，调用编译器。

下面，我们就编写一个读取环境变量的程序。程序使用 GetEnvironmentStringsA API,这个 API 的返回当前进程的环境块的起始地址。环境块的结构如下：

Var1=Value1\0
Var2=Value2\0
Var3=Value3\0
...
VarN=ValueN\0\0

这样的结构在 Windows 中很常见，后面我们研究剪切板还会遇到这样结构。

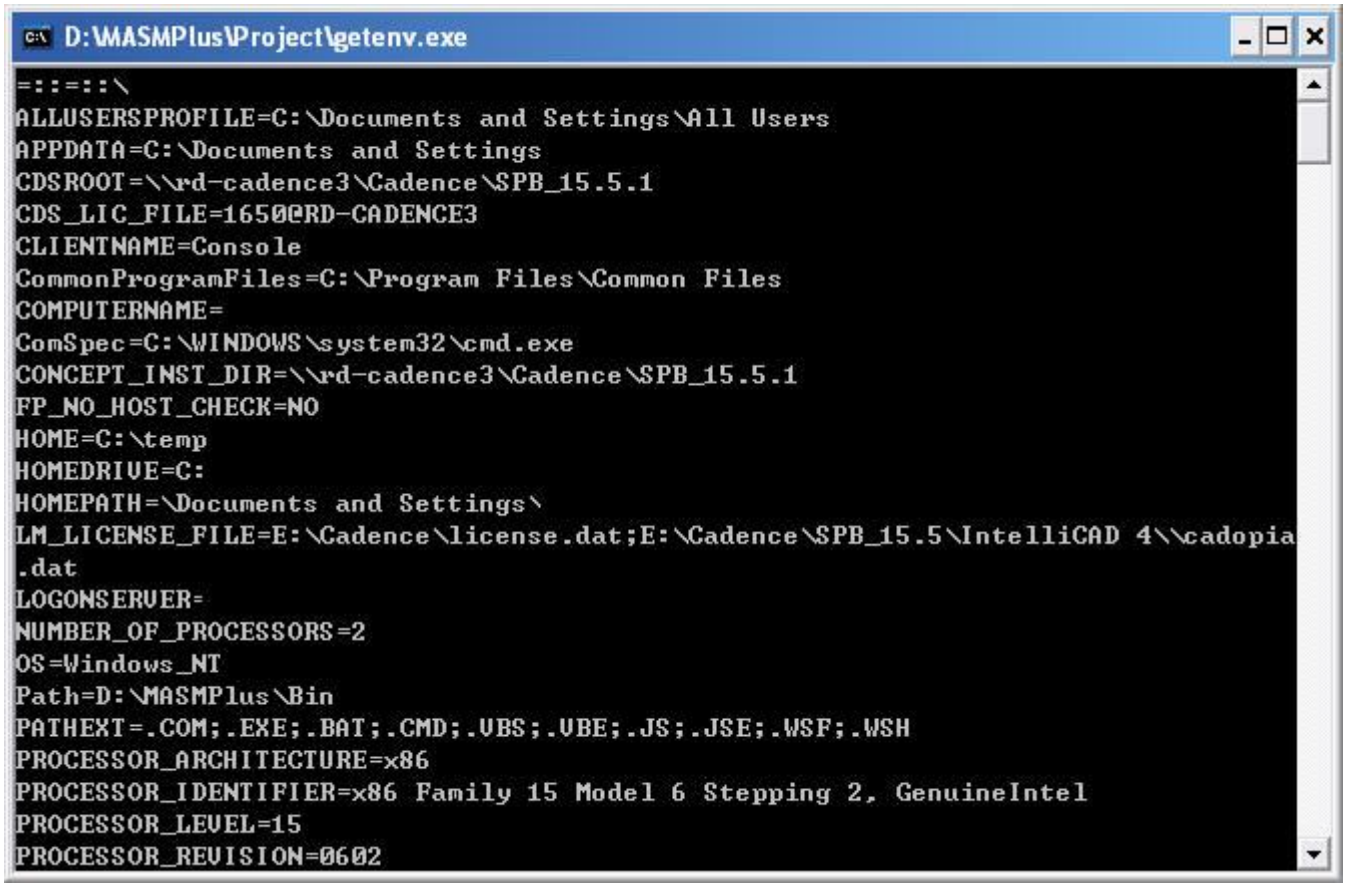
程序的基本思路就是：我们再开辟一块空间，将环境块中的东西拷贝过去，在拷贝的过程中，遇到 \0 就将它换作回车，最后直接显示这块新空间中的内容即可。

```
;#Mode=CON
.386
```

```
.model flat, stdcall
option casemap :none

include windows.inc
include user32.inc
include kernel32.inc
include masm32.inc

includelib user32.lib
includelib kernel32.lib
includelib masm32.lib
include macro.asm

.data?
szBuffer db 100 dup(?)
hStdOut dd ?
hStdin dd ?
hMem dd ?
nwritten dd ?
.CODE
START:

invoke GetStdHandle,STD_OUTPUT_HANDLE
mov hStdOut,eax
invoke GetStdHandle,STD_INPUT_HANDLE
mov hStdin,eax

xor ebx,ebx
invoke VirtualAlloc,ebx,1000,MEM_COMMIT+MEM_RESERVE,PAGE_READWRITE
mov hMem,eax

invoke GetEnvironmentStringsA
mov esi,eax ;esi 指向包含环境变量的内存块
mov edi,hMem
next_symbol: ;edi 指向一个新开辟的内存块
mov al,[esi]
or al,al ;判断一下，是否为 0
jz end_string ;al=0 表示到结尾
mov [edi],al
next_string:
cmpsb
jmp short next_symbol
end_string:
```

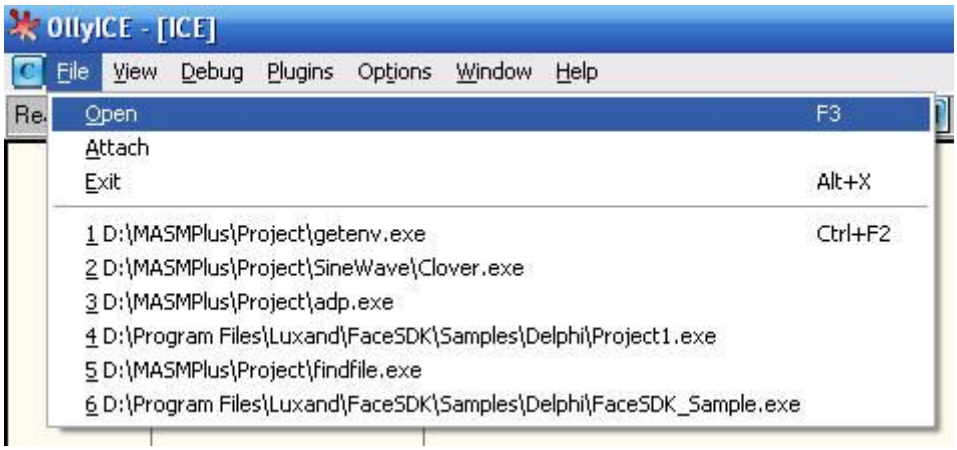
```
mov DWORD ptr [edi],0D0A2020h ;在结尾处我们用回车+空格取代之
;(有兴趣的读者可以试试看写作 0D0A 看看是什么结果)
add edi,3
cmp byte ptr [esi+1],0 ;如果 esi 的下一个还是 0，则说明遇到了 0 0 表示全部结束
jnz next_string ;否则继续输出下一个字符串
inc edi
stosb
invoke StdOut, hMem ;最后显示结果

invoke GetFileSize,[hStdin],ebx
mov edi,hMem
invoke ReadFile,[hStdin],edi, eax,offset nwritten, ebx
add edi,[nwritten]
mov byte ptr [edi],0
invoke StdOut, hMem
invoke VirtualFree,hMem,1000,MEM_RELEASE


;暂停显示,回车键关闭
invoke StdIn,addr szBuffer,sizeof szBuffer
invoke ExitProcess,0

StdOut proc bufOffs:dword
invoke lstrlen,bufOffs
invoke WriteFile,[hStdOut],bufOffs,eax,offset nwritten,0
ret
StdOut endp
end START
```

运行结果：

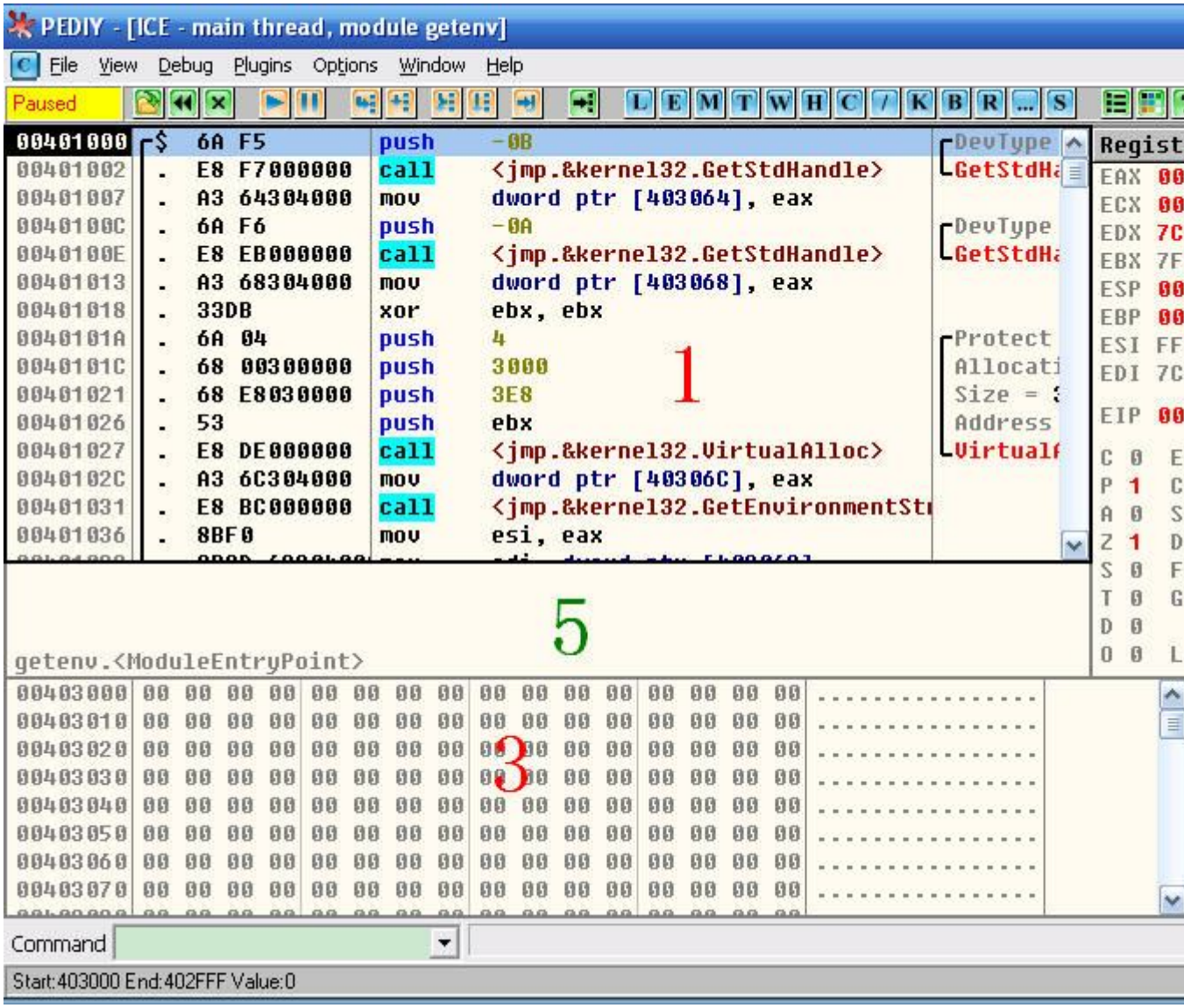


问题是在最上方出现了奇怪的字符 “=:::=::\”是我的程序出错了吗？经过一番努力，还是没有搞清楚，于是祭出利器 OllyDBG。这是一款 Ring3 级别的程序调试器。（此处省略 1000 字有点介绍）

用法很简单，使用 File-->Open 选定程序

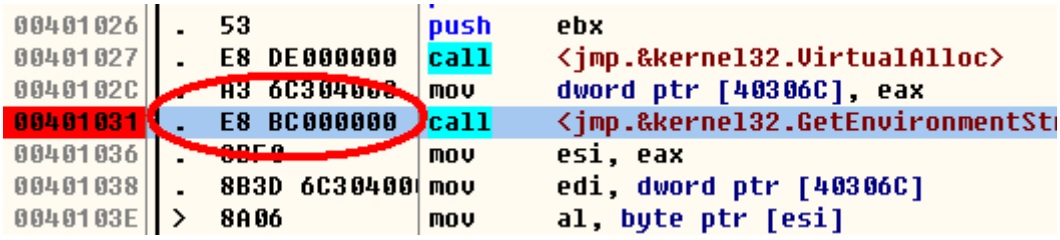


1 号部分是反编译结果，看起来基本上就是我们的 Source Code，2 号部分是寄存器信息，3 号部分是当前内存的信息，4 号部分是堆栈信息，5 号是“显示反汇编窗口中选中的第一个命令的参数及一些跳转目标地址、字串等”。读者可以参考 http://bbs.pediy.com/showthread.php?s=&threadid=21284 《CCDebuger 的 OllyDBG 入门教学》

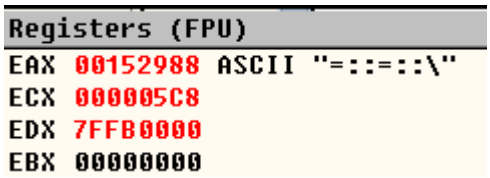


单步执行是 F7 或者 F8 键，他们的区别在于，前者会进入 call xxx 这样的调用，而 F8 则会进入运行后再出来停在 call 语句的下一条。

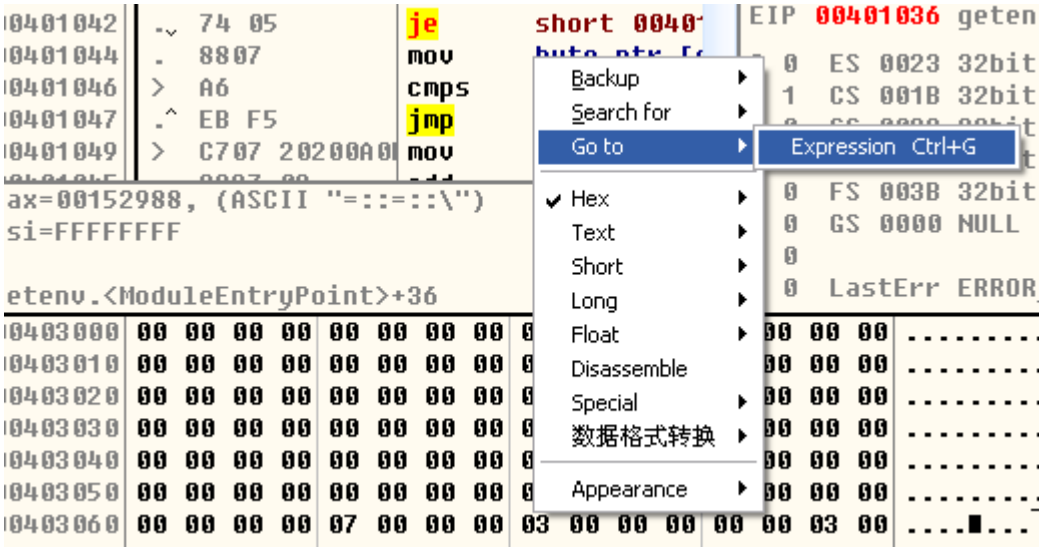
我们想知道，调用过 API 后，返回的内存环境变量是什么样子的，需要在下图的地方下一个断点方法就是双击下图圆圈处，左边就会变为红色，表示已经断在这个地方了。



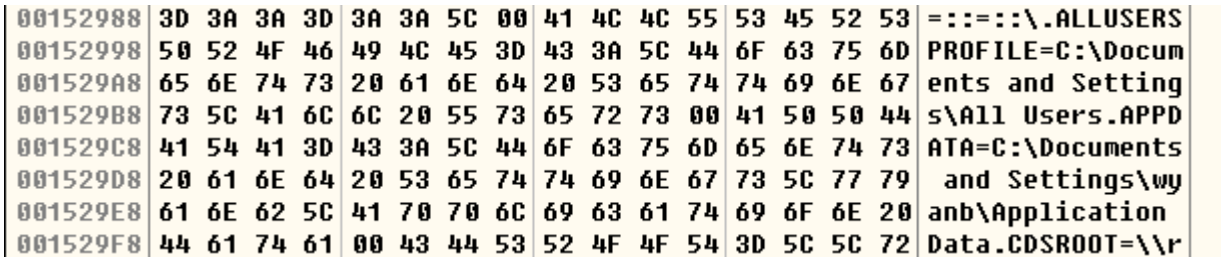
然后我们再使用 F9，让他自动运行，到这个地方会停下来，再按一次 F8 就会调用这个 API 并且停在这条语句后面（当然，更方便的方法是直接在后面的那条语句处下断点，直接运行过去）察看寄存器运行结果：



面熟啊！ EAX 给出块的起始地址（这样说比“EAX 是指向内存块的指针”更容易理解一点吧？）
我们在下面显示内存信息的窗口（3 号部分），点鼠标右键，选择 Go to-->Exprerssion



在弹出的窗口输入 eax，表示我要看 eax 中给出的内存地址的值，然后就是下面的结果：



更面熟！我们确实看到了 “=::=::\”... ...因此，程序没有问题，不过为什么会是这样的结果，

我也不知道了... ...

此外，更方便的方法是使用 GetEnvironmentVariable API，它能够指定返回那个环境变量的值，

比如，可以直接返回“OS”的值“ Windows_NT”。有兴趣的读者可以试试看.....

《用 MasmPlus 学习汇编语言》已经写了 5 期了，每一期我都特意“设计”了很多“不完美”的地方，留给读者来实现，好比我们在丛林中探险，除了指出深处的宝藏之外，还给出了很多的岔路口，我相信尝试每一个岔路也会让你收获无穷。

参考：

http://msdn2.microsoft.com/en-us/library/ms683187.aspx

用 MasmPlus 学习汇编语言 6

一.窗口的动画效果的实现

在 Windows 中要实现窗口的动画效果，只需要一个 API 函数：AnimateWindow 。

MSDN 上定义如下：

```
Syntax
BOOL AnimateWindow(
HWND hwnd,
DWORD dwTime,
DWORD dwFlags
);
```

Parameters
hwnd
[输入] 生成动画效果的窗口的handle。调用这个API的线程必须是这个窗口的。
dwTime
[输入]指定动画的时间，单位是毫秒，通常都是200毫秒。
dwFlags
[输入]指定动画的样式。可以指定一个或者多个。需要注意的是，默认情况下，这些效果是当显示一个窗口的时候。如果需要在隐藏一个窗口时实现这些效果，请指定AW_HIDE并用 or 逻辑再指定其他样式。
AW_SLIDE
滑动效果。当指定了AW_ACTIVE，将忽略这个标志。
AW_ACTIVATE
激活窗口，不要和AW_HIDE同时使用。
AW_BLEND
淡入淡出效果。只有当hwnd给出的窗口处于最上层才有效。
AW_HIDE
隐藏窗口。默认是显示窗口。
AW_CENTER
当指定AW_HIDE时为向中心收缩的效果;未指定AW_HIDE时为向四周扩散的效果。
AW_HOR_POSITIVE
窗口从左到右的效果。
AW_HOR_NEGATIVE
窗口从右到左的效果。
AW_VER_POSITIVE
窗口从上到下的效果。
AW_VER_NEGATIVE
窗口从下到上的效果。

下面就是一个简单的例子，大部分程序都是MasmPlus模板，真正的代码只有四行：

```
;MASMPlus 代码模板 - 普通的 Windows 程序代码

.386
.Model Flat, StdCall
Option Casemap :None

Include windows.inc
Include user32.inc
Include kernel32.inc
Include gdi32.inc

includelib gdi32.lib
```

```
IncludeLib user32.lib
IncludeLib kernel32.lib
include macro.asm

    WinMain PROTO :DWORD,:DWORD,:DWORD,:DWORD
    WndProc PROTO :DWORD,:DWORD,:DWORD,:DWORD

.DATA
    szClassName db "MASMPlus_Class",0

.DATA?
    hInstance    dd ?

.CODE
START:

    invoke GetModuleHandle,NULL
    mov hInstance,eax
    invoke WinMain,hInstance,NULL,NULL,SW_SHOWDEFAULT
    invoke ExitProcess,0

WinMain proc hInst:DWORD,hPrevInst:DWORD,CmdLine:DWORD,CmdShow:DWORD
    LOCAL wc :WNDCLASSEX
    LOCAL msg :MSG
    local hWnd :HWND

    mov wc.cbSize,sizeof WNDCLASSEX
    mov wc.style,CS_HREDRAW or CS_VREDRAW or CS_BYTEALIGNWINDOW
    mov wc.lpfnWndProc,offset WndProc
    mov wc.cbClsExtra,NULL
    mov wc.cbWndExtra,NULL
    push hInst
    pop wc.hInstance
    mov wc.hbrBackground,COLOR_BTNFACE+1
    mov wc.lpszMenuName,NULL
    mov wc.lpszClassName,offset szClassName
    invoke LoadIcon,hInst,100
    mov wc.hIcon,eax
    invoke LoadCursor,NULL,IDC_ARROW
    mov wc.hCursor,eax
    mov wc.hIconSm,0
    invoke RegisterClassEx, ADDR wc
    invoke CreateWindowEx,NULL,ADDR
```

```
szClassName,CTXT("http://www.aogosoft.com"),WS_OVERLAPPEDWINDOW,200,2
00,400,200,NULL,NULL,hInst,NULL
    mov hWnd,eax
    invoke ShowWindow,hWnd,SW_SHOWNORMAL
    invoke UpdateWindow,hWnd

    StartLoop:
        invoke GetMessage,ADDR msg,NULL,0,0
            cmp eax, 0
            je ExitLoop
                invoke TranslateMessage, ADDR msg
                invoke DispatchMessage, ADDR msg
            jmp StartLoop
    ExitLoop:

mov eax,msg.wParam
ret
WinMain endp

WndProc proc hWin:DWORD,uMsg:DWORD,wParam :DWORD,lParam :DWORD
    .if uMsg==WM_CREATE
;真正起作用的是下面的四条语句
            ;从左到右显示窗口
            invoke     AnimateWindow,hWin,1000,AW_SLIDE or
AW_HOR_POSITIVE
    ;当要关闭窗口时会触发这个消息
    .elseif uMsg==WM_CLOSE
            ;淡入效果，窗口消失
            invoke     AnimateWindow,hWin,1000,AW_BLEND or
AW_HIDE
            ;下面要发一条 WM_DESTROY 消息，否则的话程序不会结束
            invoke     SendMessage,hWin,WM_DESTROY,wParam,lParam
    .elseif uMsg == WM_DESTROY
            invoke      PostQuitMessage,NULL
    .else
        invoke DefWindowProc,hWin,uMsg,wParam,lParam
    .endif
    ret
WndProc endp

END START
```

执行后，窗口会从左边滑出来：

关闭时，窗口会变淡直到消失。

参考：

1. MSDN: http://msdn2.microsoft.com/en-us/library/ms632669.aspx
2. <<Visual C++ 6.0 应用编程 150 例>> P18 实例 9 窗口的动画效果

二.禁止用户关闭 Winodws

下面的程序实现阻止用户关闭 Windows。简单的说，当你选择关闭系统时，Windows 会给当前的程序发送 WM_QUERYENDSESSION 消息，意思就是说，我要 shutdown 了，各位还有什么想说的，还有什么想做的... ... 我们在程序中判定这个消息，返回一句话“FALSE”,告诉 Windows，我这还没有完呢~ 于是 Windows 就会停止关闭。

程序非常非常简单，使用 MasmPlus 随便建立一个 Win32 程序，我们在消息循环中插入下面的代码，既可实现。

```
WndProc proc hWin:DWORD,uMsg:DWORD,wParam :DWORD,lParam :DWORD
    .if uMsg==WM_CREATE

    .elseif uMsg == WM_DESTROY
        invoke PostQuitMessage,NULL
```

```
    .elseif uMsg == WM_QUERYENDSESSION ;最关键的就是这里
        mov     eax,FALSE
    .else
        invoke DefWindowProc,hWin,uMsg,wParam,lParam
    .endif
    ret
WndProc endp
```

提醒一点：也许这个功能会有一点副作用，也保不准 Windows 不像城管们，

一样，万一“粗鲁了一下”，也许会使得你的程序强制关闭。最好在具体的应用

环境中饰演一下。

参考：
1.WM_QUERYENDSESSION Message
http://msdn2.microsoft.com/en-us/library/aa376890(VS.85).aspx

用 MasmPlus 学习汇编语言 7

我很喜欢使用 Delphi，至少使用它编程的时候不用担心界面的问题，“拖拖拉拉”就能

完成一个 Windows 标准程序的界面。这次我将介绍如何使用汇编语言设计简单的界面。

一.如何使用 Static ？

这个东西就是在界面上显示一行文字的东西，Delphi 中叫做 Label。创建的方法很简单，

和创建窗口差不多，就是使用 CreateWindowEx API，我们告诉系统，我要建立一种叫做 static

的东西，这个东西放在我建立好的窗口上，窗口的 handler 是 static，程序的 handler 是 hInst，

于是写出来就是这个样子：

```
INVOKE CreateWindowEx, NULL, CTXT("static"), addr szText01,\
WS_CHILD or WS_VISIBLE or SS_NOTIFY,\
53, 66, 200, 69, hWnd, 2001, hInst, NULL
```

对照原形：

```
HWND CreateWindowEx(
    DWORD dwExStyle,
    LPCTSTR lpClassName,
    LPCTSTR lpWindowName,
    DWORD dwStyle,
    int x,
    int y,
    int nWidth,
    int nHeight,
    HWND hWndParent,
    HMENU hMenu,
    HINSTANCE hInstance,
    LPVOID lpParam
);
```

各个含义可以猜得八九不离十。

下面的问题就是怎么向上面写字？我们使用 SetWindowText API,定义如下：

```
BOOL SetWindowText(
    HWND hWnd,
    LPCTSTR lpString
);
```

我们前面一直都在用，ShowMessage API 这个定义一看就清楚。
最后，我们要在上面写点什么呢？就写一下当前的时间吧。用 GetLocalTime API 可以获得当前的日期时间，不过需要留心一下，SYSTEMTIME 结构体中的成员都是 WORD,
直接使用 invoke 压入进行格式转换会出现莫名其妙的错误，具体说来就是：
```
invoke wsprintf,addr szBuffer,CTXT("%u
%u"),lpLocalTime.wYear,lpLocalTime.wMonth
```
这样用是不对的，运行起来除了第一个，其余的都不对。

程序如下，我们将显示时间写在 WM_SIZE 消息中，每次改变窗口都会显示现在的时间。

惯例还要说点需要注意的问题：注意指定的长宽，要是显示不下，有可能出现字符截断
的情况，或者自己转到了下一行的情况哦。
参考：
1.CreateWindowEx
http://msdn2.microsoft.com/en-us/library/ms632680(VS.85).aspx

2.SetWindowText http://msdn2.microsoft.com/en-us/library/ms633546(VS.85).aspx

3.GetLocalTime http://msdn2.microsoft.com/en-us/library/ms724338(VS.85).aspx

二. Button

象上一个程序一样，button 也是使用 CreateWindowEx 建立的。比如，我们用下面的

语句就可以建立 2 个 button，不同的地方是，这里我们特地定义了 ButtonID1 和 ButtonID2。

简单的说，这个只是一个编号，使给我们后面响应命令用的。

```
INVOKE CreateWindowEx, NULL, CTEXT("button"), CTEXT("BT1"),\
WS_CHILD or WS_VISIBLE or WS_TABSTOP,\
15, 18, 80, 25, hWnd, ButtonID1, hInst, NULL
mov hButton1, eax


INVOKE CreateWindowEx, NULL, CTEXT("button"), CTEXT("BT2"),\
WS_CHILD or WS_VISIBLE or WS_TABSTOP,\
103, 19, 80, 25, hWnd, ButtonID2, hInst, NULL
mov hButton2, eax
```

完整的程序：

```
;MASMPlus 代码模板 - 普通的 Windows 程序代码

.386
.Model Flat, StdCall
Option Casemap :None

Include windows.inc
Include user32.inc
Include kernel32.inc
Include gdi32.inc

includelib gdi32.lib
IncludeLib user32.lib
IncludeLib kernel32.lib
include macro.asm

ButtonID1     equ     2003
ButtonID2     equ     2004

    WinMain PROTO :DWORD,:DWORD,:DWORD,:DWORD
    WndProc PROTO :DWORD,:DWORD,:DWORD,:DWORD

.DATA
    szClassName db "MASMPlus_Class",0

.DATA?
    hInstance    dd ?
    hButton1        dd     ?
    hButton2        dd     ?
.CODE
START:

    invoke GetModuleHandle,NULL
    mov hInstance,eax
    invoke WinMain,hInstance,NULL,NULL,SW_SHOWDEFAULT
    invoke ExitProcess,0

WinMain proc hInst:DWORD,hPrevInst:DWORD,CmdLine:DWORD,CmdShow:DWORD
    LOCAL wc :WNDCLASSEX
    LOCAL msg :MSG
    local hWnd :HWND
```

```
    mov wc.cbSize,sizeof WNDCLASSEX
    mov wc.style,CS_HREDRAW or CS_VREDRAW or CS_BYTEALIGNWINDOW
    mov wc.lpfnWndProc,offset WndProc
    mov wc.cbClsExtra,NULL
    mov wc.cbWndExtra,NULL
    push hInst
    pop wc.hInstance
    mov wc.hbrBackground,COLOR_BTNFACE+1
    mov wc.lpszMenuName,NULL
    mov wc.lpszClassName,offset szClassName
    invoke LoadIcon,hInst,100
    mov wc.hIcon,eax
    invoke LoadCursor,NULL,IDC_ARROW
    mov wc.hCursor,eax
    mov wc.hIconSm,0
    invoke RegisterClassEx, ADDR wc
    invoke CreateWindowEx,NULL,ADDR
szClassName,CTXT("http://www.aogosoft.com"),WS_OVERLAPPEDWINDOW,200,2
00,400,200,NULL,NULL,hInst,NULL
    mov hWnd,eax

INVOKE CreateWindowEx, NULL, CTEXT("button"), CTEXT("BT1"),\
WS_CHILD or WS_VISIBLE or WS_TABSTOP,\
15, 18, 80, 25, hWnd, ButtonID1, hInst, NULL
mov hButton1, eax

INVOKE CreateWindowEx, NULL, CTEXT("button"), CTEXT("BT2"),\
WS_CHILD or WS_VISIBLE or WS_TABSTOP,\
103, 19, 80, 25, hWnd, ButtonID2, hInst, NULL
mov hButton2, eax

    invoke ShowWindow,hWnd,SW_SHOWNORMAL
    invoke UpdateWindow,hWnd

    StartLoop:
        invoke GetMessage,ADDR msg,NULL,0,0
            cmp eax, 0
            je ExitLoop
                invoke TranslateMessage, ADDR msg
                invoke DispatchMessage, ADDR msg
            jmp StartLoop
    ExitLoop:
```

```
mov eax,msg.wParam
ret
WinMain endp

WndProc proc hWin:DWORD,uMsg:DWORD,wParam :DWORD,lParam :DWORD
    LOCAL szBuffer[10]    :BYTE
    .if uMsg==WM_CREATE

    .elseif uMsg == WM_DESTROY
        invoke PostQuitMessage,NULL
    .elseif uMsg == WM_COMMAND
        mov eax,wParam

.IF ax==ButtonID1     ;我们在这里响应按键的动作
shr eax,16
.IF ax==BN_CLICKED
        invoke    wsprintf,addr szBuffer,CTXT("%s"),CTEXT("Button1")
        invoke    MessageBox,hWin,addr szBuffer,NULL,NULL
.ENDIF
.ENDIF

.IF ax==ButtonID2
shr eax,16
.IF ax==BN_CLICKED
        invoke    wsprintf,addr szBuffer,CTXT("%s"),CTEXT("Button2")
        invoke    MessageBox,hWin,addr szBuffer,NULL,NULL
.ENDIF
.ENDIF

    .else
        invoke DefWindowProc,hWin,uMsg,wParam,lParam
    .endif
    ret
WndProc endp

END START
```

参考：

1.Iczelion's Win32 Assembly 教程 Tutorial 9: Child Window Controls

三.透明效果的实现

我们要实现透明窗口的效果，必须使得窗口拥有 WS_EX_LAYEDER 扩展属性，当然可以在创建的时候

直接指定，这里我们绕个弯，多讲一些，使用 GetWindowLong 取当前窗口的 GWL_EXSTYLE 属性，加上

WS_EX_LAYERED 属性后再用 SetWindowLong 设置回去。

指定透明的 API 是 SetLayeredWindowAttributes 原形如下：

第一个参数给定窗口的 handler；第二个参数，给出一个眼色值；第三个参数给出透明度的值，

0 是最透明，最大 2 5 5 是最不透明的；后面是标志位，

LWA_COLORKEY ＝ 1 将 crKey 给出的颜色指定为透明色

LWA_ALPHA ＝ 2 将 bAlpha 给出的透明度作为窗口的透明度

我们的程序中只使用了透明的功能。

```
;MASMPlus 代码模板 - 普通的 Windows 程序代码

.386
.Model Flat, StdCall
Option Casemap :None
```

```
Include windows.inc
Include user32.inc
Include kernel32.inc
Include gdi32.inc

includelib gdi32.lib
IncludeLib user32.lib
IncludeLib kernel32.lib
include macro.asm

ButtonID1    equ    2003
ButtonID2    equ    2004

LWA_ALPHA    equ    2
LWA_COLORKEY equ 1
WS_EXD_LAYERED    equ    80000h
    WinMain PROTO :DWORD,:DWORD,:DWORD,:DWORD
    WndProc PROTO :DWORD,:DWORD,:DWORD,:DWORD

.DATA
    szClassName db "MASMPlus_Class",0
    bAlpha         dd     0FFh

.DATA?
    hInstance    dd ?
    hButton1        dd     ?
    hButton2        dd     ?
    hWnd              HWND ?
.CODE
START:

    invoke GetModuleHandle,NULL
    mov hInstance,eax
    invoke WinMain,hInstance,NULL,NULL,SW_SHOWDEFAULT
    invoke ExitProcess,0

WinMain proc hInst:DWORD,hPrevInst:DWORD,CmdLine:DWORD,CmdShow:DWORD
    LOCAL wc :WNDCLASSEX
    LOCAL msg :MSG



    mov wc.cbSize,sizeof WNDCLASSEX
    mov wc.style,CS_HREDRAW or CS_VREDRAW or CS_BYTEALIGNWINDOW
```

```
    mov wc.lpfnWndProc,offset WndProc
    mov wc.cbClsExtra,NULL
    mov wc.cbWndExtra,NULL
    push hInst
    pop wc.hInstance
    mov wc.hbrBackground,COLOR_BTNFACE+1
    mov wc.lpszMenuName,NULL
    mov wc.lpszClassName,offset szClassName
    invoke LoadIcon,hInst,100
    mov wc.hIcon,eax
    invoke LoadCursor,NULL,IDC_ARROW
    mov wc.hCursor,eax
    mov wc.hIconSm,0
    invoke RegisterClassEx, ADDR wc
    invoke CreateWindowEx,NULL,ADDR
szClassName,CTXT("http://www.aogosoft.com"),WS_OVERLAPPEDWINDOW,200,2
00,400,200,NULL,NULL,hInst,NULL
    mov hWnd,eax

INVOKE CreateWindowEx, NULL, CTEXT("button"), CTEXT("虚"),\
WS_CHILD or WS_VISIBLE or WS_TABSTOP,\
15, 18, 80, 25, hWnd, ButtonID1, hInst, NULL
mov hButton1, eax

INVOKE CreateWindowEx, NULL, CTEXT("button"), CTEXT("实"),\
WS_CHILD or WS_VISIBLE or WS_TABSTOP,\
103, 19, 80, 25, hWnd, ButtonID2, hInst, NULL
mov hButton2, eax

    invoke    GetWindowLong,hWnd,GWL_EXSTYLE
    or             eax,WS_EXD_LAYERED
    invoke    SetWindowLong,hWnd,GWL_EXSTYLE,eax
invoke     SetLayeredWindowAttributes,hWnd,0,bAlpha,LWA_ALPHA

    invoke ShowWindow,hWnd,SW_SHOWNORMAL
    invoke UpdateWindow,hWnd

    StartLoop:
        invoke GetMessage,ADDR msg,NULL,0,0
            cmp eax, 0
            je ExitLoop
                invoke TranslateMessage, ADDR msg
                invoke DispatchMessage, ADDR msg
```

```
            jmp StartLoop
    ExitLoop:

mov eax,msg.wParam
ret
WinMain endp

WndProc proc hWin:DWORD,uMsg:DWORD,wParam :DWORD,lParam :DWORD
    LOCAL szBuffer[10]    :BYTE
    .if uMsg==WM_CREATE


    .elseif uMsg == WM_DESTROY
        invoke PostQuitMessage,NULL

    .elseif uMsg == WM_COMMAND
        mov eax,wParam

.IF ax==ButtonID1          ;虚
shr eax,16
.IF ax==BN_CLICKED
        mov    eax,100
        mov    bAlpha,eax
.ENDIF
.ENDIF

.IF ax==ButtonID2             ;实
shr eax,16
.IF ax==BN_CLICKED
        mov    eax,255
        mov    bAlpha,eax
.ENDIF
.ENDIF

invoke     SetLayeredWindowAttributes,hWnd,0,bAlpha,LWA_ALPHA     ;设
置透明

    .else
        invoke DefWindowProc,hWin,uMsg,wParam,lParam
    .endif
    ret
WndProc endp
```

END START

当点击 “虚” 按钮时，窗口会变得透明，当点击 “实” 按钮时，窗口会恢复。

有兴趣的读者还可以试验一下更多的功能。需要注意的是

LWA_ALPHA LWA_COLORKEY WS_EXD_LAYERED 在 SDK 中并没有定义，需要在程序中定义一下。

根据上面的例子，我们还可以编写隐藏任务栏的小程序。同样是设置两个按钮，

一个是隐藏，一个是显示。然后在程序中加入下面的代码即可：

```
        invoke      FindWindow,CTXT('Shell_TrayWnd'),NULL
        mov          hDesktop,eax

.IF ax==ButtonID1           ;隐藏
    invoke     ShowWindow,hDesktop,SW_HIDE
.ENDIF

.IF ax==ButtonID2               ;显示
    invoke     ShowWindow,hDesktop,SW_RESTORE
.ENDIF
```

至于放到什么地方，请读者发挥自己的想象力~

参考：

1. SetLayeredWindowAttributes http://msdn2.microsoft.com/en-us/library/ms633540(VS.85).aspx

SetLayeredWindowAttributes Function

The **SetLayeredWindowAttributes** function sets the opacity and transparency color key of a layered window.

Syntax

```
BOOL SetLayeredWindowAttributes(
    HWND hwnd,
    COLORREF crKey,
    BYTE bAlpha,
    DWORD dwFlags
);
```

Parameters

hwnd

[in] Handle to the layered window. A layered window is created by specifying WS_EX_LAYERED when creating the window with the CreateWindowEx function or by setting WS_EX_LAYERED via SetWindowLong after the window has been created.

crKey

[in] COLORREF structure that specifies the transparency color key to be used when composing the layered window. All pixels painted by the window in this color will be transparent. To generate a **COLORREF**, use the RGB macro.

bAlpha

[in] Alpha value used to describe the opacity of the layered window. Similar to the **SourceConstantAlpha** member of the BLENDFUNCTION structure. When *bAlpha* is 0, the window is completely transparent. When *bAlpha* is 255, the window is opaque.

dwFlags

[in] Specifies an action to take. This parameter can be one or more of the following values.

LWA_COLORKEY

Use *crKey* as the transparency color.

LWA_ALPHA

Use *bAlpha* to determine the opacity of the layered window.

Return Value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

用 MasmPlus 学习汇编语言 8

一.单选按钮

Radio button 中文名称 “单选按钮”。从名称看就能知道是一种特殊的按钮。与之相对，

还有一种可以多选的，叫做 Check Box。今天我们的就编写一个选择更换桌面墙纸的程序。

更换墙纸，我们使用 SystemParametersInfo API，这个函数我们在前面多次使用过。与前面

不同，这次我们使用的是 SPI_SETDESKWALLPAPER 这个参数。有一点需要注意的是，2003/XP/2000

操作系统下这个参数后面给出的桌面墙纸不能是 jpg 格式。说明如下：

SPI_SETDESKWALLPAPER	Sets the desktop wallpaper. The value of the *pvParam* parameter determines the new wallpaper. To specify a wallpaper bitmap, set *pvParam* to point to a NULL-terminated string containing the full path to the bitmap file. Setting *pvParam* to "" removes the wallpaper. Setting *pvParam* to SETWALLPAPER_DEFAULT or NULL reverts to the default wallpaper. The *pvParam* parameter can specify a .jpg file. **Windows Server 2003 and Windows XP/2000:** The *pvParam* parameter cannot specify a .jpg file.

程序如下：

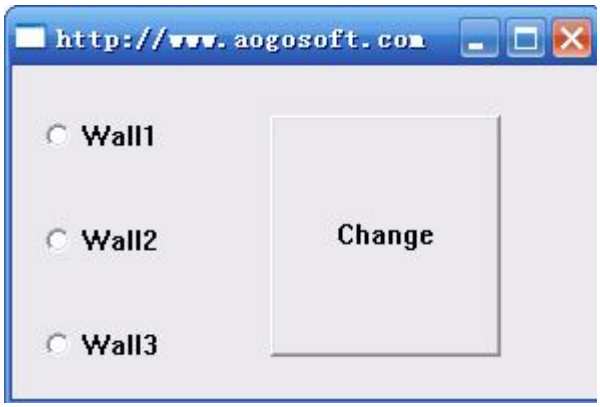
```
;MASMPlus 代码模板 - 普通的 Windows 程序代码

.386
.Model Flat, StdCall
Option Casemap :None

Include windows.inc
Include user32.inc
Include kernel32.inc
Include gdi32.inc

includelib gdi32.lib
IncludeLib user32.lib
IncludeLib kernel32.lib
include macro.asm


    RadioID01         equ      2001
    RadioID02         equ      2002
    RadioID03         equ      2003
    ButtonID              equ      2004

    WinMain PROTO :DWORD,:DWORD,:DWORD,:DWORD
    WndProc PROTO :DWORD,:DWORD,:DWORD,:DWORD

.DATA
    szClassName db "MASMPlus_Class",0
    num             dd 0abH
    szBuffer        db      100 dup (0)

    szWallPaper01 db    "Wall1",0
    szWallPaper02 db     "Wall2",0
    szWallPaper03 db    "Wall3",0

    WallPaper    dd    0

.DATA?
    hInstance    dd ?
    hRadio01        dd      ?
    hRadio02        dd      ?
    hRadio03        dd      ?
    hButton        dd ?
.CODE
```

```
START:

    invoke GetModuleHandle,NULL
    mov hInstance,eax
    invoke WinMain,hInstance,NULL,NULL,SW_SHOWDEFAULT
    invoke ExitProcess,0

WinMain proc hInst:DWORD,hPrevInst:DWORD,CmdLine:DWORD,CmdShow:DWORD
    LOCAL wc :WNDCLASSEX
    LOCAL msg :MSG
    local hWnd :HWND

    mov wc.cbSize,sizeof WNDCLASSEX
    mov wc.style,CS_HREDRAW or CS_VREDRAW or CS_BYTEALIGNWINDOW
    mov wc.lpfnWndProc,offset WndProc
    mov wc.cbClsExtra,NULL
    mov wc.cbWndExtra,NULL
    push hInst
    pop wc.hInstance
    mov wc.hbrBackground,COLOR_BTNFACE+1
    mov wc.lpszMenuName,NULL
    mov wc.lpszClassName,offset szClassName
    invoke LoadIcon,hInst,100
    mov wc.hIcon,eax
    invoke LoadCursor,NULL,IDC_ARROW
    mov wc.hCursor,eax
    mov wc.hIconSm,0
    invoke RegisterClassEx, ADDR wc
    invoke CreateWindowEx,NULL,ADDR
szClassName,CTXT("http://www.aogosoft.com"),WS_OVERLAPPEDWINDOW,200,2
00,300,200,NULL,NULL,hInst,NULL
    mov hWnd,eax

;---------- [Create Radio] ----------
INVOKE CreateWindowEx, NULL, CTXT("button"), addr szWallPaper01,\
WS_CHILD or WS_VISIBLE or WS_TABSTOP or BS_AUTORADIOBUTTON,\
16, 16, 100, 39, hWnd, RadioID01, hInst, NULL
mov hRadio01, eax

;---------- [Create Radio] ----------
INVOKE CreateWindowEx, NULL, CTXT("button"), addr szWallPaper02,\
WS_CHILD or WS_VISIBLE or WS_TABSTOP or BS_AUTORADIOBUTTON,\
16, 64, 100, 47, hWnd, RadioID02, hInst, NULL
```

```
mov hRadio02, eax

;---------- [Create Radio] ----------
INVOKE CreateWindowEx, NULL, CTXT("button"), addr szWallPaper03,\
WS_CHILD or WS_VISIBLE or WS_TABSTOP or BS_AUTORADIOBUTTON,\
16, 112, 100, 55, hWnd, RadioID03, hInst, NULL
mov hRadio03, eax

;---------- [Create the Control Window] ----------
INVOKE CreateWindowEx, NULL, CTXT("button"), CTXT("Change"),\
WS_CHILD or WS_VISIBLE or WS_TABSTOP,\
129, 25, 115, 120, hWnd, ButtonID, hInst, NULL
mov hButton, eax

    invoke ShowWindow,hWnd,SW_SHOWNORMAL
    invoke UpdateWindow,hWnd

    StartLoop:
        invoke GetMessage,ADDR msg,NULL,0,0
            cmp eax, 0
            je ExitLoop
                invoke TranslateMessage, ADDR msg
                invoke DispatchMessage, ADDR msg
            jmp StartLoop
    ExitLoop:

mov eax,msg.wParam
ret
WinMain endp

WndProc proc hWin:DWORD,uMsg:DWORD,wParam :DWORD,lParam :DWORD
    .if uMsg==WM_CREATE

.elseif uMsg == WM_COMMAND
mov eax,wParam

        .IF ax==RadioID01 ;我们在这里响应按键的动作
        shr eax,16
        .IF ax==BN_CLICKED
            mov     WallPaper,1
        .ENDIF
        .ENDIF
```

```
        .IF ax==RadioID02
        shr eax,16
        .IF ax==BN_CLICKED
            mov     WallPaper,2
        .ENDIF
        .ENDIF

        .IF ax==RadioID03
        shr eax,16
        .IF ax==BN_CLICKED
            mov     WallPaper,3
        .ENDIF
        .ENDIF

        .IF ax==ButtonID
        shr eax,16
        .IF ax==BN_CLICKED
            .if         WallPaper==1
                invoke      SystemParametersInfo,SPI_SETDESKWALLPAPER,
NULL,CTXT("F:\汇编通讯\MasmPlus\1.bmp"),1
            .elseif      WallPaper==2
                invoke      SystemParametersInfo,SPI_SETDESKWALLPAPER,
NULL,CTXT("F:\汇编通讯\MasmPlus\2.bmp"),1
            .elseif     WallPaper==3
                invoke      SystemParametersInfo,SPI_SETDESKWALLPAPER,
NULL,CTXT("F:\汇编通讯\MasmPlus\3.bmp"),1
            .endif
        .ENDIF
        .ENDIF

    .elseif uMsg == WM_DESTROY
        invoke PostQuitMessage,NULL
    .else
        invoke DefWindowProc,hWin,uMsg,wParam,lParam
    .endif
    ret
WndProc endp

END START
```

运行结果：



选择你想要的，然后点 Change 按钮，即可更换。

二. 光渗视幻觉

最近在读《数学趣闻集锦（上）》，上面提到了“光渗视幻觉”。

“视幻觉是由于人们的注意力和眼睛的构造两者造成的. 当我们观察一个具有明暗对象的区域时，

不固定的东西在我们眼睛中是不完全清楚的，光线进入位于我们眼 后的视网膜时扩散了. 结果明亮的

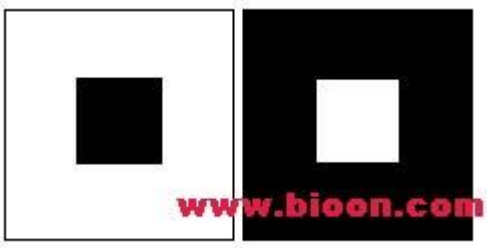
光线或亮的区域便溢出，并渗入到视网膜上影像的暗区. 这样一来，亮的区域就显得比同等大小的暗的

区域要大一些，就像下图所 示的那样. 这也解释了为什么穿暗色的衣服，特别是黑的，比起你穿亮丽的

衣服或同等式样的白色衣服，会使你显得更为瘦长. 这种幻觉称为光渗，它是由 19 世纪德国的物理学家和

生理学家赫尔姆霍兹(Herman L. F. von Helmholtz，1821—1894)发现的. ”【参考 1】

下面这个图片中心的方块，哪个看起来大一点呢？



下面我们就设计一个程序来验证这个问题，结合我们前面所学的绘图和获取键盘消息的知识，

```
;MASMPlus 代码模板 - 普通的 Windows 程序代码

.386
.Model Flat, StdCall
Option Casemap :None

Include windows.inc
Include user32.inc
Include kernel32.inc
Include gdi32.inc

includelib gdi32.lib
IncludeLib user32.lib
IncludeLib kernel32.lib
include macro.asm

    WinMain PROTO :DWORD,:DWORD,:DWORD,:DWORD
    WndProc PROTO :DWORD,:DWORD,:DWORD,:DWORD


.DATA
    szAppName db "Herman's",0
grRight        db    0h
grLeft         db    0h
rectwhole     RECT <0>

.DATA?
    hInstance     dd ?
    cxClient        dd ?
    cyClient        dd      ?
rectright     RECT    <>
rectleft        RECT    <>

.CODE
START: ;从这里开始执行

    invoke GetModuleHandle,NULL
    mov         hInstance,eax
    invoke WinMain,hInstance,NULL,NULL,SW_SHOWDEFAULT
    invoke ExitProcess,0
```

```
WinMain proc hInst:DWORD,hPrevInst:DWORD,CmdLine:DWORD,iCmdShow:DWORD
    LOCAL wndclass :WNDCLASSEX
    LOCAL msg      :MSG
    localhWnd      :HWND
    mov wndclass.cbSize,sizeof WNDCLASSEX
    mov wndclass.style,CS_HREDRAW or CS_VREDRAW
    mov wndclass.lpfnWndProc,offset WndProc

    mov wndclass.cbClsExtra,0
    mov wndclass.cbWndExtra,0

    push hInst
    pop wndclass.hInstance

    invoke LoadIcon,NULL,IDI_APPLICATION
    mov wndclass.hIcon,eax

    invoke LoadCursor,NULL,IDC_ARROW
    mov wndclass.hCursor,eax

    invoke GetStockObject,WHITE_BRUSH
    mov wndclass.hbrBackground,EAX

    mov wndclass.lpszMenuName,NULL
    mov wndclass.lpszClassName,offset szAppName

    mov wndclass.hIconSm,0

    invoke RegisterClassEx, ADDR wndclass
    .if (EAX==0)
        invoke MessageBox,NULL,CTXT("This program requires Windows NT!"),addr
szAppName,MB_ICONERROR
      ret
    .endif

    invoke CreateWindowEx,
              NULL,
              ADDR szAppName,                ;window class name
              CTXT("halo effect"), ;window caption
              WS_OVERLAPPEDWINDOW,           ;window style
              CW_USEDEFAULT,                 ;initial x position
              CW_USEDEFAULT,                 ;initial y position
```

```
                CW_USEDEFAULT,                  ;initial x size
                CW_USEDEFAULT,                  ;initial y size
                NULL,                           ;parent window handle
                NULL,                           ;window menu handle
                hInstance,                      ;program instance handle
                NULL                            ;creation parameters
    mov hWnd,eax

    invoke ShowWindow,hWnd,iCmdShow
    invoke UpdateWindow,hWnd

    StartLoop:
      invoke GetMessage,ADDR msg,NULL,0,0
        cmp eax, 0
        je ExitLoop
          invoke TranslateMessage, ADDR msg
          invoke DispatchMessage, ADDR msg
        jmp StartLoop
    ExitLoop:

    mov eax,msg.wParam
    ret
WinMain endp

WndProc proc hwnd:DWORD,message:DWORD,wParam :DWORD,lParam :DWORD

    LOCAL hdc               :HDC
    LOCAL tmpx,tmpy         :DWORD
    LOCAL ps                :PAINTSTRUCT
    LOCAL szBuffer[256]     :BYTE

    .if message == WM_SIZE
      mov       eax,lParam          ;cxClient = LOWORD (lParam)
      and       eax,0FFFFh
      mov       cxClient,eax
      mov     rectwhole.right,eax

      mov       eax,lParam
      shr       eax,16
      mov       cyClient,eax        ;cyClient = HIWORD (lParam)
      mov       rectwhole.bottom,eax
        ret
```

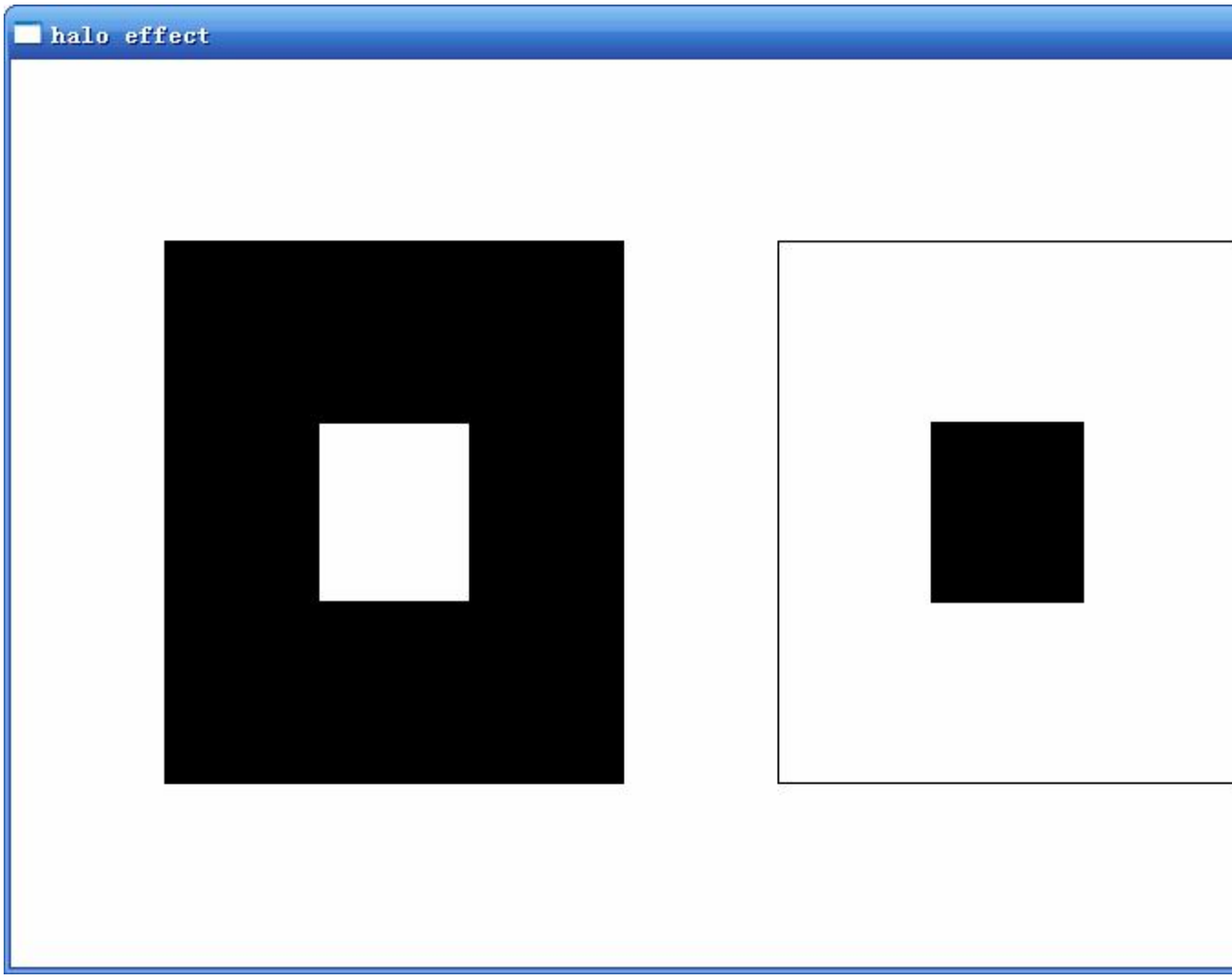
```
    .elseif message == WM_KEYDOWN
        .if     wParam == VK_UP
            add         grRight,5
        .elseif wParam == VK_DOWN
            sub         grRight,5
        .elseif wParam == VK_LEFT
            add         grLeft,5
        .elseif wParam == VK_RIGHT
            sub         grLeft,5
        .endif
        invoke      InvalidateRect,hwnd,addr rectwhole,FALSE
.elseif message == WM_PAINT
    invoke    BeginPaint,hwnd,addr ps
    mov         hdc,eax

    xor         edx,edx
    mov         eax,cxClient
    mov         ecx,9
    div         ecx
    mov         tmpx,eax          ;tmpx = cxClient / 9

    xor         edx,edx
    mov         eax,cyClient
    mov         ecx,5
    div         ecx
    mov         tmpy,eax          ;tmpy = cyClient / 5

    mov         eax,tmpy
    mov         ecx,4
    mul         ecx
    push        eax             ;eax = 4 * cyClient / 5
    mov         rectleft.bottom,eax

    mov         eax,tmpx
    mov         ecx,4
    mul         ecx
    push        eax             ;eax = 4 * cxClient / 9
    mov         rectleft.right,eax

    mov         eax,tmpy
    push        eax
    mov         rectleft.top,eax
```

```
    mov         eax,tmpx
    push        eax
    mov         rectleft.left,eax

    push        hdc
    call        Rectangle    ;绘制左边大长方形

    xor         eax,eax
    mov         al,grLeft
    shl         eax,8
    mov         al,grLeft
    shl         eax,8
    mov         al,grLeft
    mov         ebx,eax
 invoke     CreateSolidBrush,eax
    mov         ebx,eax
 invoke     FillRect,hdc,addr rectleft,ebx

    mov         eax,tmpy
    mov         ecx,3
    mul         ecx
    push        eax             ;eax = 3 * cyClient / 9

    mov         eax,tmpx
    mov         ecx,3
    mul         ecx
    push        eax             ;eax = 3 * cxClient / 5

    mov         eax,tmpy
    add         eax,eax
    push        eax

    mov         eax,tmpx
    add         eax,eax
    push        eax

    push        hdc
    call        Rectangle    ;绘制左边小长方形


    push        tmpy

    mov         eax,tmpx
```

```
mov         ecx,5
mul         ecx
push        eax             ;eax = 5 * cxClient / 9

mov         eax,tmpy
mov         ecx,4
mul         ecx
push        eax

mov         eax,tmpx
mov         ecx,8
mul         ecx
push        eax

push        hdc
call        Rectangle    ;绘制右边大长方形

mov         eax,tmpy
mov         ecx,3
mul         ecx
push        eax
mov         rectright.bottom,eax

mov         eax,tmpx
mov         ecx,7
mul         ecx
push        eax             ;eax = 8 * cxClient / 9
mov         rectright.right,eax

mov         eax,tmpy
mov         ecx,2
mul         ecx
push        eax
mov         rectright.top,eax

mov         eax,tmpx
mov         ecx,6
mul         ecx
push        eax
mov         rectright.left,eax

push        hdc
call        Rectangle    ;绘制右边小长方形
```

```
        xor         eax,eax
        mov         al,grRight
        shl         eax,8
        mov         al,grRight
        shl         eax,8
        mov         al,grRight
    invoke    CreateSolidBrush,eax
        mov         ebx,eax
    invoke    FillRect,hdc,addr rectright,ebx

        invoke    EndPaint,hwnd,addr ps
        ret
    .elseif message == WM_DESTROY

        invoke PostQuitMessage,NULL
    ret
    .endif

    invoke DefWindowProc,hwnd, message, wParam, lParam
    ret

WndProc endp
END START
```

运行结果：



你可以使用光标上下左右调整明暗。试试看什么时候感觉差别最大。

江湖传闻：据说美国码头搬运军火的装运工曾经因为箱子太沉太重而罢工。后来心理学家给军方一个建议，

就是使用绿色的箱子装军火，而原来的箱子都是黑色的。改变颜色之后，装运工普遍觉得箱子“轻了”。

个人观点：在目前所有的科学研究中，人类在心理学方面成就不大，思维是非线性不容易量化的，并且

出于道德伦理上面的限制，科学家也无法做过多的试验，而实验是科学的基础。很多时候，心理学的实验凭借

受试者的“感觉”，也是非常不精确的。

参考：

1. http://www.bioon.com/popular/math/305225.shtml

2. http://zhidao.baidu.com/question/2311680.html

色彩的膨胀与收缩感

据说法兰西国旗一开始是由面积完全相等的红、白、蓝三色制成的，但是旗帜升到空中后在感觉是三色的面积并不相等，

于是召集了有关色彩专家进行专门研究，最后把三色的比例调整到红 35％、白 33％、蓝 37％的比例时才感觉到面积相等。

这究竟是什么原因呢?

因为当各种不同波长的光同时通过水晶体时，聚集点并不完全在视网膜的一个平面上，因此在视网膜上的影像的清晰度

就有一定差别。长波长的暖色影像似焦距不准 确，因此在视网膜上所形成的影像模糊不清，似乎具有一种扩散性；短波长

的冷色影像就比较清晰，似乎具有某种收缩性。所以，我们平时在凝视红色的时候，时间 长了会产生眩晕现象，景物形象

模糊不清似有扩张运动的感觉。如果我们改看青色，就没有这种现象了。如果我们将红色与蓝色对照着看，由于色彩同时

对比的作 用，其面积错视现象就会更加明显。

色彩的膨胀、收缩感不仅与波长有关，而且还与明度有关。由于“球面像差”物理原理，光亮的物体在视网膜上所成影像

的轮廓外似乎有一圈光圈围绕着，使物体在 视网膜上的影像轮廓扩大了，看起来就觉得比实物大一些，如通电发亮的电灯

钨丝比通电前的钨丝似乎要粗得多，生理物理学上称这种现象为“光渗”现象。歌德在 《论颜色的科学》一文中指出：

“两个圆点同样面积大小，在白色背景上的黑圆点比黑色背景上的白圆点要小 1 / 5。”同一个人。日出和日落时，地平线上

仿佛出现一个凹陷似的，这也是光渗作 用而引起的视觉现象。

宽度相同的印花黑白条纹布，感觉上白条子总比黑条子宽；同样大小的黑白方格子布，白方格子要比黑方格略大一些。超市中，

小商品、小包装若要使它显眼一些， 宜采用鲜艳的浅色；如果要它显得高贵精致，宜采用沉着的深色或黑色。为了扩大建筑

或交通工具的室内空间感，色彩设计宜采用乳白、浅米、象牙等淡雅明快的色 调，像卫生间等特别狭小的空间还可以利用

镜子作墙面，利用镜子的反射来增加面积的宽畅度和明亮度。

Masm 错误指南

A1000 无法打开文件: filename

产生这个错误可能是如下原因

- 编译器无法打开或者源程序或者引用文件或者输出文件
- 产生原因可能是如下几种
- 文件不存在
- 其它进程正在访问
- 文件名无效
- 给定的输出的文件是一个已经存在的只读文件
- 当前硬盘空间不足
- 但前目录是根目录，已经达到文件数量的上限
- 无法写入设备
- 设备尚未准备好

A1001 关闭文件时遇到 I/O 错误

可能原因是你移除了移动存储介质（比如 U 盘）

A1002 写入文件时遇到 I/O 错误 可能原因同 A1000 类似

A1003 读取文件时遇到 I/O 错误 原因同 A1000 类似

A1005 编译器限制，宏参数名表满
可能原因是你的宏有太多的参数，局部标号或者全局标号，导致宏名称表空间不足
解决方法是简短定义的宏名称，或者删除不用的宏

A1006 无效的命令行参数：option
可能的原因主要是命令行语法错误

A1007 嵌套过深 编译器只能处理 20 层的嵌套
产生原因：
- 诸如.IF, .REPEAT, 或者 .WHILE 这样的高级指令嵌套过深
- 定义的结构体嵌套太深
- 条件编译嵌套太深
- 过程定义中发生了嵌套
- PUSHCONTEXT 指令（上限是 10）
- 段定义
- 引用文件中嵌套过深
- 宏

●

A1008 宏嵌套不匹配 文件结尾也没有找到所定义宏的终结指令 ENDM，或者 ENDM 定义在宏块的外面；另外的常见的原因是是忘记.REPEAT or .WHILE. 前面的“.”导致编译器理解上出现错误。

A1009 行太长 Masm 每行的上限是 512 字节，这个“行”包括用连字符“\”连接起来的逻辑行都不能超过 512 字节

A1010 块定义不匹配 发生这个问题的原因可能是：

- 高级编译指令比如 .IF, .REPEAT, or .WHILE
- 条件编译指令比如 IF, REPEAT, or WHILE
- 结构体和联合结构的定义
- 段定义
- POPCONTEXT 指令
- 汇编语言条件指令，比如找到了 ELSE, ELSEIF, 或者 ENDIF 但是没有对应的 IF

A1011 指令必须在控制块的内部 编译器发现错误的高级指令。可能原因，

- 出现.ELSE 却没有 .IF， 出现 .ENDIF 却没有.IF
- 出现.ENDW 没有 .WHILE
- 出现.UNTIL[[CXZ]] 没有 .REPEAT
- 出现.CONTINUE 没有 .WHILE 或者 .REPEAT
- 出现.BREAK 没有 .WHILE or .REPEAT
- .ELSE 之后又出现 .ELSE

A1012 当前发现的错误数量超过 100 个，停止编译

A1013 命令行出现无效数字 产生的原因是编译使用的命令行中，应该指定数字的地方出现了非数字

A1014 参数过多 产生的原因是命令行参数过多导致了内存不足

一个可能的原因是展开带有通配符的文件名时，发生这样的问题。为了解决，最好的办法是单独列出每个程序。

A1015 声明过于复杂 发生的原因是编译器在分析特定的声明时，堆栈耗尽通过下面的方法可以解决:

- 将复杂的声明拆解为短的声明
- 重新规划声明，减少复杂的嵌套
- 如果声明是宏的一部分，请常识将宏拆解为很多小的宏

A1017　源程序名称丢失　发生的原因是ML 工具无法找到欲编译的文件或者是需要传递给链接器的文件

A1901　汇编器内部错误 非常少见的错误……如果你遇到了最好的办法是暂时移除一部分代码，不断尝试，尝试找到导致问题的语句……

A2000　指令不允许内存操作

A2001　指令不允许立即数　指令无法接受常数或者内存偏移量作为立即数

A2002　每个 IF 块不能有超过一个 ELSE

比如 IF, IFE, IFB, IFNB, IFDEF, IFNDEF, IFDIF, 或者 IFIDN 指令都可以定义一个 IF 块，在这些 IF 块中只能有一个 ELSE。不过可以有很多个 ELSEIF 定义。

另外，出现这个问题时，建议检查前文是不是丢失了 ENDIF 指令

A2003　声明中出现多余字符 最好检查出现错误的位置是否出现了非注释状态的汉字，或者全角的空格字符等等。

A2004　标识符冲突　定义的变量标识符，结构体或者标号在同一个模块中有过定义，并且是不同的类型

A2005　标识符重定义

A2006　未定义的标识符

产生的原因如下：

- 标识符未定义
- 域未在指定结构体中定义
- 标识符定义在另一个文件中，但是该文件没有包含（include）在你使用的文件中
- 外部引用的标识符没有使用 EXTERN 或者 EXTERNDEF 声明
- 符号名拼写错误
- 在作用域之外引用了一个本地标号

A2007　记录体定义出错

记录体定义和之前的定义有冲突

可能是以下原因造成的:

- 二者域数量不同
- 某个域中“位”（BITS）数不同
- 标号不同
- 初始方式不同

A2008　语法错误

当前的记号导致一个语法错误

可能的原因:

- 指令前面漏掉了或者多加了一个 “.”
- 保留字 (比如说 C 或者 SIZE) 被当作标识符使用
- 当前选择的处理器模式下不允许使用这个指令
- 运行期比较操作 (比如 ==) 被当作关系操作（比如 EQ），误用在条件编译中
- 指令操作数缺少参数
- 使用了已经废弃的指令

A2009　表达式语法错误当前行的表达式有语法错误。需要注意的是：这个错误可能是预处理程序导致的副作用。

A2010　表达式中类型无效　THIS 或者 PTR 给定的操作数无效的表达式

A2011　当前段不适用 WORD 大小

过程定义或者代码标号中定义的地址大小同当前的段大小不匹配。

错误的原因可能是下面的情形之一:

- 在一个 32 位段中定了 NEAR16 或者 FAR16 的过程
- 在一个 16 位段中定了 NEAR32 或者 FAR32 的过程
- 在一个 32 位的段中，定义了 FAR16 或者 NEAR16 的标号
- 在一个 16 位的段中，定义了 FAR32 或者 NEAR32 的标号

A2012　过程或者宏中重复使用的标号必须添加 LOCAL 前缀

A2013　该指令之前必须有.model 定义

使用简化段定义或者.STARTUP 和.EXIT 之前必须先用.MODEL 定义

A2014　无法定义为 public 或者 external 属性

只有标号，过程和数字才可以通过 PUBLIC, EXTERN, or EXTERNDEF.定义为 public 或者 external 属性。本地代码标号无法定义为 pulic。

A2015　段属性无法修改

段再次定义了同之前定义不同的属性。当使用 SEGMENT 指令重新打开一个之前定义过的段时，新开的段继承自动继承之前定义的属性。

A2016　表达式异常

编译器在当前位置预期一个表达式，但是只发现下面的

- 没有操作数的一元运算符
- 二元操作符，但是缺少两个操作数
- 空的圆括号()或者空的方括号

A2017　需要运算符

当前位置期望一个表达式的运算符。发生这样错误的一个可能原因是在表达式列表中的表达式之前缺少逗号

A2018　外引用标识符无效　尝试使用关系操作符比较外部标识符。外部标识符在编译期是未知的，因而无法完成比较（链接的时候才能知道外部标识符）

A2019　操作对象必须是 RECORD 类型或者是 RECORD 的域成员

WIDTH 或者 MASK 操作符后面的操作数无效。WIDTH 或者 MASK 后面应该是一个域名或者一个 RECORD 类型.

A2020　标识符非记录类型当前位置期望是一个 record 类型

A2021　记录体常量不能跨行

记录体常量必须定义在一个完整的行中。

A2022　指令操作数大小必须相同

A2023　指令操数作必须有大小 操作数出现了未知大小

A2024　指令的操作数大小不对

A2025　操作数必须在同一个段中　发生错误的原因是使用了关系或者差值操作符号的可重定位运算操作必须在同一个段中

A2026　constant expected 期望常量　常量表达式是一种可以在编译期解析的数字表达式（比如说，mov eax,1000/4 在编译期就可以直接计算出实际操作是 mov eax,250）

A2027　操作数必须是一个内存表达式 出现问题的原因是 PTR 表达式右边操作数并非内存表达式。当 PTR 操作符左边是一个结构体或者联合类型的时候，右边必须是一个内存表达式

A2028　表达式必须是一个代码地址

发生这个问题的原因可能是：SHORT 后面没有跟着一个代码地址; NEAR PTR 或者 FAR PTR 指定的并非代码地址

A2029　不允许多基地址寄存器联用

尝试在内存表达式中同时使用 2 个基地址寄存器。比如，下面的表达式会导致这个错误

[bx+bp]

[bx][bp]

另外的例子，假设定义如下：

id1 proc arg1:byte

下面 2 行都会导致这个错误

mov al, [bx].arg1

lea ax, arg1[bx]

A2030　multiple index registers not allowed 不允许多索引寄存器联用

出问题的代码是在一个内存表达式中联用了 2 个索引寄存器

比如，下面的表达式会导致这个错误：

[si+di]

[di][si]

A2031　必须是索引寄存器或者基地址寄存器

出问题的代码是在一个内存表达式中使用了非基地址寄存器，也不是索引寄存器

比如，下面的表达式会导致这个错误：

[ax]

[bl]

A2032　寄存器使用错误

寄存器使用方法错误

可能的原因是：

- OFFSET 和寄存器联用
- 在无效的上下文中使用了 386 寄存器
- 寄存器通过 PTR 指向了一个无效的类型
- 寄存器作为段重载操作符“:”的右操作数
- 寄存器作为减法操作的右操作数
- 试图使用乘法操作符“*”操作寄存器
- 寄存器附近的括号“[]”丢失

A2033　INVOKE 参数出错

请检查 INVOKE 中是否包含了特殊的 386 寄存器，或者包含了字节大小的寄存器，这都会导致这个问题。

A2034　必须在段中

下面的命令出现在段外都会引起这个错误

- 指令
- 标号定义

- THIS 操作
- $ 操作
- 过程定义
- ALIGN 指令
- ORG 指令

A2035　DUP 结构过于复杂

数据结构中使用 DUP 操作符声明，展开过程中导致内部数据过大

A2036　给定结构体的初始值太多

初始化结构体给定的初始值比结构体声明的域多

A2037　结构体定义中不允许声明

结构体定义中出现无效的声明

结构体 结构体中不能有指令，标号，过程，流程控制指令，.STARTUP, or .EXIT。

A2038　宏操作中丢失操作数

汇编器发现宏参数列结尾有！或者%操作符

A2039　行过长

源程序一行不能超过 512 个字节

即使使用行连接符号（\）连接起来的逻辑行上限也是 512 字节

A2040　上下文中不允许出现段寄存器

因为前文段寄存器被指定给一条指令，所以当前无法当作段寄存器使用

A2041　字符串或者文本过长

字符串或者文本，或者宏返回值，超过 255 个字节

A2042　声明过于复杂

声明过于复杂，编译器无法进行语法分析。

解决方式减少记号数量或者减少前向引用标识符

A2043　标识符太长

标识符长度超过 247 字节

A2044 文件中出现无效字符

源程序中，注释和字符串之外出现的字符无法被识别为操作符或者其他有意义的字符

A2045　上下文中缺少括号

尖括号无法配对(缺少 < 或者 >)或者大括号无法配对(缺少 { 或者 })

可能原因如下：

- 括号对没有完成
- An angle bracket was intended to be literal, but it was not preceded by an exclamation point (!) to indicate a literal character.

A2046　字符串定义缺少单引号或者双引号

产生问题的原因是字符串定义中引号不配对(缺少' 或者 ")

产生这个错误的原因可能如下：

- 字符串前后的引号不配对
- 字符串前后的引号一个是单引号一个是双引号
- 引号是文本的一部分，但是前后使用同样的引号来表示字符串

A2047　空（null）字符串

引号引用的字符串中间没有字符

对于一个有效的字符串，必须包括 1-255 个字节

A2048 数字中出现了非数字符号

数字中出现了非当前进制的字符，比如说十进制数中出现了 B 或者 D 这个字母。

A2049　浮点常数出现语法错误

产生这个错误的原因通常是定义的常数浮点数中出现了无效的字符

A2050　不允许浮点数或者 BCD 数字

在数据初始化之外使用的浮点数（real）或者二进制编码（BCD）常量

可能的原因如下：

- 在表达式中使用了浮点数或者 BCD 数字
- 用 DWORD QWORD 和 TBYTE 之外的类型来初始化浮点数
- 使用非 TBYTE 来初始化 BCD 数字

A2051　需要文本

需要文本常量或者文本宏

可能的原因如下：

- 文本常量，在<>中定义
- 文本宏名称
- 宏功能调用 A macro function call
- % 后跟随了常量表达式

A2052　强制报错

条件错误指令.ERR 和 .ERR1 都可以引起这个错误 这个功能很有用，特别是你写复杂的大规模的程序的时候，各种编译条件很多，你可以将.err 插入到你要验证的代码位置来确定在编译过程中它是否会编译到。

A2053 强制错误：值为 0

条件错误指令.ERRE 会产生这个错误

A2054　强制错误：值不为 0

条件错误指令.ERRNZ 会产生这个错误

A2055　强制错误：标识符未定义

条件错误指令.ERRNDEF 会产生这个错误

A2056　强制错误：标识符已定义

条件错误指令.ERRDEF 会产生这个错误

A2057　强制错误：字符串为空白

条件错误指令.ERRB 会产生这个错误

A2058　强制错误：字符串不为空白

条件错误指令.ERRNB 会产生这个错误

A2059　强制错误：字符串相等

条件错误指令.ERRIDN 或者 .ERRIDNI 会产生这个错误

A2060　强制错误：字符串不相等

条件错误指令.ERRDIF or .ERRDIFI 会产生这个错误

A2061 不允许使用[[ELSE]]IF2/.ERR2 这样的一遍编译指令

Microsoft Macro Assembler (MASM) 是一遍汇编器，MASM 不接受 IF2, ELSEIF2,或者.ERR2 这样的指令。

另外，如果 IF1 指令后面跟随着 ELSE 也会产生这样的报错

A2062 UNTILCXZ 的表达式过于复杂

.UNTILCXZ 后面跟随的表达式过于复杂。.UNTILCXZ 指令只能接受一个表达式，其中只能包含 == 或者 ！=。其他的表达式比如||都会被认作过于复杂。

A2063　只能按照 2 的指数次对齐

由 ALIGN 指令指定的表达式无效。它后面的表达式只能是 2 的幂次，范围从 2 到 256，并且必须小于或者等于当前段，结构体或者联合类型的对齐值。

A2064　结构体只能按照 1，2，4 对齐

T 产生问题的原因是结构体定义中的对齐有错误

A2065　无法识别的 token

汇编器无法识别给定的 Token

A2066　当前 CPU 模式和段大小不匹配

试图创建一个 USE16, USE32, 或者 FLAT 属性的段，但是这个属性同当前的 CPU 模式不匹配，或者是在 32 位段中尝试切换为 16 位 CPU 模式。使用 USE32 和 FLAT 模式之前必须使用如下指令： .386, .386C, .386P, .486, or .486P.

A2067 LOCK 指令后面必须跟随者内存操作

LOCK 前缀遇到无效指令。使用它作为前缀时必须是内存操作

A2068 指令前缀不被容许

产生这个问题的原因是 REP, REPE, REPNE, 或者 LOCK 处理了一个无效的指令

A2069 该指令不需要操作数

产生错误的原因是指定给不需要操作数的指令一个或者多个操作数

A2070 无效的指令操作数

指令给定的操作数无效

A2071 预定义大小过大，无法初始化

初始化值尺寸过大，无法在数据区初始化

A2072 指定段或者 group 中无法访问给定的标识符

产生问题的原因是无法从指定的段或者 group 中访问给定的标识符。

A2073 操作数的 Frame 不同

一个表达式中的两个操作数在不同的 Frame 中。

A2074 无法通过段寄存器访问标号

产生问题的原因是：试图从过段寄存器访问标号，但是这个段寄存器没有 assume 到对应的段或者 group 中。

A2075 跳转过长[超过：n 字节]

产生问题的原因是：跳转指令的目标和这条指令之间的距离过长。

可以使用下面的方法解决：

- 使用 LJMP 选项

- 如果指令前面有 SHORT，可以试试去掉之
- 重新安排代码，使用跳转不会超过范围

A2076　跳转地址必须是一个指定的标号

直接跳转指令的目标必须是代码标号

A2077　该指令不允许使用 Near 间接寻址

条件跳转或者是 loop 循环不能使用内存操作数作为目标。它必须指定一个相对地址或者标号。

A2078　指令不允许 FAR 间接寻址

条件跳转或者是 loop 循环不能使用内存操作数作为目标。它必须指定一个相对地址或者标号

A2079　指令不允许 FAR 直接寻址

条件跳转或者 loop 不能以不同的段或者 group 为目标。

A2080　跳转距离超过当前 CPU 模式

跳转指令指定的距离和当前的 cpu 模式不匹配。

例如，48 位跳转需要.386 或者更高的 CPU 模式

A2081　一元运算符缺少操作数

操作需要操作数，但是没有。

A2082 无法混用 16 位和 32 位寄存器

地址表达式同时包含 16 位和 32 位寄存器

比如，下面的表达式会引起这个错误：

[bx+edi]

A2083　无效的变址

寄存器的变址只能是 1, 2, 4 或者 8.

A2084　常量太大

常量太大超过上下文环境默认的

A2085　当前模式下无法使用改指令或者寄存器

试图使用当前 cpu 模式无法使用的指令，寄存器或者关键字

比如，如果要使用 32 位寄存器，就需要指定.386。欲使用控制寄存器 CR0，就必须使用保护模式指令.386P 或者更高。同样的。在需要 NEAR32, FAR32, FLAT 关键字时，也会出现这个错误 。

A2086　需要保留键字

选项后面给定的列表中的单词不会被认为是保留字

A2087　指令需要指定 80386/486 模式

指令和当前的处理器模式不兼容。可能需要指定如下指令.386, .386C,

.386P, .486, 或者 .486P.

A2088　文件结尾需要 END 指令

编译器到达源程序结尾，没有发现.END 指令。

A2089　记录类型中：identifier 的位过多

产生问题可能是下面的原因：

- 对于给定的记录的域，位数过多
- 对于给定的记录，位数过多

记录或者记录的域的大小限制是 16 位（16 位模式）或者 32 位（32 位模式）

A2090 需要正数

下面一些需要正数的情况：

- 指定给 SUBSTR 或者 @SubStr 的起始位置
- 指定给 COMM 的数据对象的数量
- 指定给 COMM 的大小

A2091　索引值超过字符串最大长度

当使用 INSTR, SUBSTR, @InStr, or @SubStr 时，索引值超过字符串最大长度

A2092　计数器必须为正数或者为 0

SUBSTR 指令, @SubStr 宏, SHL 操作，SHR 操作, 和 DUP 指令的操作数不能为负

A2093　计数值过大

指定给 SUBSTR 或者 @SubStr 的长度超过字符串长度

A2094　操作数必须可以重定位

操作数没有和标号相关（意思操作数中使用了绝对地址之类的）

可能的原因是：

- END 指令跟随的操作数不是标号
- SEG 指令的操作数非标号
- 减法操作右侧同标号相关，但是左侧不是
- 关系操作符号操作的对象必须同时是整数常量或者同时为内存对象

A2095　需要常数或者可重定位的标号

指定的操作数必须是一个常数表达式或者内存偏移。

A2096　需要段，group 或者段寄存器

需要段或者 group，但是没能找到。

产生问题的原因可能如下：

- 由段操作符号（:）指定的左侧操作数并非段寄存器（CS, DS, SS, ES, FS, or GS）或者 group 名，段名或者段表达式。
- ASSUME 指令没有给定一个段寄存器，group 或者特定 FLAT group。

A2097　需要段: identifier

GROUP 指定的标识符是一个未定义的段。

A2098　OFSSET 指定了一个无效的操作数

OFFSET 后面必须跟着内存表达式或者立即数表达式

A2099　invalid use of external absolute

An attempt was made to subtract a constant defined in another module from an expression.

You can avoid this error by placing constants in include files rather than making them external.

A2100　不允许的段或者 group

产生问题的原因是试图使用无效的段或者 group。段或者 group 无法相加。

A2101　无法添加 2 个重定位表

产生问题的原因是试图将两个同一个标号相关的表达式相加。

A2102　无法将内存表达式和代码标号相加

产生问题的原因是试图将一个代码标号和内存表达式相加。

A2103　段超过 64K 限制

16 位段的大小超过 64K。

A2104　无效的数据类型声明:type

数据声明中的类型无效。

A2105　HIGH 和 LOW 操作需要立即数作为参数

产生问题的原因是：HIGH 或者 LOW 操作数不是立即数表达式。

A2107　无法使用隐式远跳转，或者 call 一个近标号

无法使用隐式远跳转，或者 call 另外一个段中的近标号

A2108　使用了 assume 到 ERROR 的寄存器

产生问题的原因是试图使用一个已经 ASSUME 到 ERROR 的寄存器

A2109　反斜杠后面只可以有空格或者注释

产生问题的原因是在反斜杠（\）后面发现非分号（;）或者空白字符（空格或者 TAB）的字符。

A2110　需要注释终结符

注释指令缺少终结符

A2111　定义中的参数相互冲突

函数原型声明和 PROC 中定义的参数不一致。

A2112　PROC 实际需要参数同原型冲突

函数原型(通过 PROTO, EXTERNDEF, or EXTERN 指令定义)，但是调用约定和 PROC 不一致。

A2113　无效的根

根必须是 2 到 16 之内的一个值

A2114　INVOKE 参数数量不匹配: argument number

通过 INVOKE 传递的参数的数量和函数原型中定义的不同。

A2115　无效的协处理器寄存器

协处理器索引为负数或者大于 7（St(0)……St(7)）

A2116　AT 段中出现了不允许的指令或者初始化

定义为 AT 属性的段中出现了指令或者数据初始化指令。AT 段中的数据必须通过？来初始化（初始值不确定）

A2117　/AT 选项需要指定 Tiny 内存模式

编译过程中，通过命令行指定了/AT 参数，但是程序中未使用 TINY 模式。

只有在指定了开始地址的模块或者使用了.STARTUP 指令后才会出现这个错误提示。

A2118　TINY 模式下不能有段地址参考

在 TINY 模式的程序中试图使用段

TINY 模式下全部的代码和数据只能使用 NEAR 方式寻址。

A2119　必须指令语言类型

A procedure definition or prototype was not given a language type.

A language type must be declared in each procedure definition or prototype if a default language type is not specified. A default language type is set using either the .MODEL directive, OPTION LANG, or the ML command-line options /Gc or /Gd.

A2120　PROLOGUE 必须是宏函数

The identifier specified with the OPTION PROLOGUE directive was not recognized as a defined macro function.

The user-defined prologue must be a macro function that returns the number of bytes needed for local variables and any extra space needed for the macro function.

A2121　EPILOGUE 必须是宏过程

The identifier specified with the OPTION EPILOGUE directive was not recognized as a defined macro procedure.

The user-defined epilogue macro cannot return a value.

A2122　alternate identifier not allowed with EXTERNDEF

An attempt was made to specify an alternate identifier with an EXTERNDEF directive. You can specify an optional alternate identifier with the EXTERN directive but not with EXTERNDEF.

A2123　文本宏嵌套过深

文本宏嵌套过深。对于文本宏来说最大的深度为 40

A2125　丢失宏参数

需要@InStr, @SubStr 或者用户定义的宏

A2126　EXITM 使用错误

EXITM 指令在同一个宏中使用了 2 次，一次指定了返回值，一次没有指定。宏过程返回值，宏函数不返回

A2127　宏参数过多

宏中的字符过多。这个错误也可能是由于使用 PROC 定义了一个宏导致的

A2129　VARARG（不定数量参数）只能是最后一个参数

VARARG 变量只能作为宏或者过程的最后一个参数。在一个宏中不能定义多个：VARARG 变量

A2130　VARARG 不能够定义为 LOCAL

试图将一个 VARARG 变量声明为 LOCAL 属性

A2131　VARARG（不定数量变量）参数要求 C 调用方式

过程定义或者过程原型中如果使用了 VARARG 参数，必须使用 C, SYSCALL, 或者 STDCALL 的调用方式

A2132　ORG 后接的参数必须定义为常量或者本地偏移量

ORG 后面必须是一个立即数表达式，这个表达式不能包含外部引用的标号（external label）或者段之外的标号

A2133　INVOKE 调用中发生了覆盖寄存器值

使用了寄存器当作参数传递给过程，但是使用 INVOKE 的过程中，寄存器的值被覆盖。AX, AL, AH, EAX, DX, DL, DH, 或者 EDX 寄存器可以用来传递数据。改正方法是换另一个寄存器。

A2134　结构体太大，不能在 INVOKE 中传递: argument number

产生问题的原因是：试图在 INVOKE 中传递一个大约 255 字节的结构体。如果结构体超过 255 字节，请使传递结构体的指针。

A2136　INVOKE 无法传递太多参数

通过 INVOKE 传递的参数超过原型中给定参数的数量

A2137　INVOKE 参数不够

通过 INVOKE 传递的参数少于原型中给定参数的数量

A2138　无效的数据初始化值

数据定义的初始化值列表无效。

A2140　RET 操作数过大

RET, RETN, 或者 RETF 的操作数超过 2 个字节

A2141　指令的操作过多

字符串控制指令给定了太多的操作数

A2142　对于每个.IF 块不能指定多个.ELSE

汇编器发现.IF 对应了超过 1 个的.ELSE.可以考虑使用.ELSEIF 替换中间的.ELSE。

A2143　需要数据标号

用 LENGTHOF, SIZEOF, LENGTH, or SIZE 操作一个非数据标号，或者用 SIZEOF 或者 SIZE 操作一个类型

A2144　过程无法嵌套

MASM 不支持在一个过程体中定义另外一个过程体（其他语言，比如 PASCAL 支持这样的定义）例如下面的情形是错误的：

```
uuu  proc h:DWORD

ppp  proc p:DWORD

     ret

ppp  endp

     ret

uuu  endp
```

A2145　EXPORT 必须是 FAR 属性: procedure

指定的过程给定了 EXPORT 可见和 NEAR 的距离。全部的 EXPORT 的过程必须声明为 FAR。默认的可见范围可以用过 OPTION PROC:EXPORT 参数设置，还指定为 SMALL 或者 COMPACT 内存模式

A2146　procedure 声明 2 种可见属性

指定的过程同时声明了两种冲突的可见属性（PUBLIC, PRIVATE, 或者 EXPORT）。PROC 和 PROTO 声明的可见属性必须相同。

A2147 宏中标号未定义: macrolabel

给定的宏标号未找到。

A2148 表达式中出现无效的标识符类型: identifier

表达式中出现的无效的标识符

例如：表达式中不容许出现宏过程名

A2149 字节大小的寄存器不可以是第一个操作数

指令不接受字节大小的寄存器作为操作数

A2150 字大小的寄存器不可以是第一个操作数

指令不接受字大小的寄存器作为操作数

A2151 特定的寄存器不可是第一个操作数

指令不接受字特定的寄存器作为操作数

A2152 协处理器寄存器不可以是第一个操作数

协处理器（堆栈）寄存器，不能作为这个指令的第一个操作数。

A2153 无法改变表达式计算大小

产生问题的原因是试图试图修改大小已经通过 EXPR16, EXPR32, SEGMENT:USE32, or SEGMENT:FLAT 选项或者。386 或者更高的 CPU 选择指令确定的表达式大小。

A2154 流程控制指令语法错误

流程控制指令（比如.IF 或者 .WHILE）中的条件出现语法错误

A2155 无法在表达式中混合 16 位寄存器和 32 位地址

错误的原因是在表达式中混合 16 位和 32 位的偏移。

请使用 32 位段中定义的符号和 32 位寄存器。

例如，Id 是定义在 32 位段中的标号，下面的调用方式会引起这个错误：idl [bx]

A2156　常数值超出范围

PAGE 指令后面指定了无效值

PAGE 指令的第一个参数可以是 0 或者从 10 到 255

PAGE 指令后面的第二个参数可以是 0 或者从 60 到 255

A2157　丢失右括号

在宏调用中，遗漏了右括号“)”

请确认括号配对使用

A2158　类型大小对于寄存器是错误的

试图将一个通用寄存器关联到一个和它大小不同的寄存器上。

例如，下面的声明会导致这个错误：

```
ASSUME bx:far ptr byte      ; far pointer is 4 or 6 bytes
ASSUME al:word              ; al is a byte reg, cannot hold word
```

A2159　结构体无法实例化

试图创建见一个结构体的实例，但是这个结构体中未定义域或者数据，也可能是在结构体的定义中使用了 ORG。

A2160　良性的结构重定义：label incorrect

结构体重定义中的标号不存在于原定义或者重定义结构中

A2161　良性结构重定义： too few labels

在一个结构体的重定义中，成员数量不足。

A2162　OLDSTRUCT/NOOLDSTRUCT 状态无法改变

当使用了 OLDSTRUCT/NOOLDSTRUCT 选项后，结构体已经定义，同一个模块中结构体的范围就不可以改变或者再次定义

A2163 非良性的结构重定义：错误的初始化值

STRUCT 或者 UNION 重定义与初始值不同。当结构体或者联合体多次定义时，必须使用相同的定义。定义不同时就会产生这个错误。

A2164　非良性结构体重定义：初始化值太少

STRUCT 或者 UNION 重定义中的初始化值过少。当多次结构体和共用体时，每次定义必须相同

A2165　非良性结构体重定义：lablel 的偏移错误: 标号的偏移不正确

在一个重定义的 STRUCT 或者 UNION 中，label 的偏移同原始定义不同。

当结构体和 union 定义一次以上的时候，定义必须是唯一的。产生错误的原因可能是丢掉了成员或者成员同之前的定义大小不同。

A2166　需要结构体内部项目

点操作符（.）右边不是结构体的域

产生这个问题的原因可能是编译的代码是为之前版本的编译器设计的。可以通过 OPTION OLDSTRUCTS 选项打开兼容模式，或者在命令行下使用 OPTION M510 或者 /Zm 打开这个模式

A2167　应该出现表达式的地方出现了不正确的字符

应该出现表达式的地方出现了不正确的字符

可能的原因如下：

- 字符串被当作了初始化值
- 记录类型常量中的标签被省略

A2169　表达式中出现除 0 错误

表达式中的除数为 0。

请检查表达式的语法，注意除数是否正常初始化

A2170　该指令只能在宏之内出现

GOTO 或者 EXITM 指令出现在宏外面。

A2171　无法展开宏

语法错误导致编译器无法展开宏功能

A2172　RECORD 中位太少

试图定义一个 0 位的记录域。

A2173　宏无法重定义自身

出现问题的原因是在一个宏功能内部定义一个同样名称的宏。同样当一个宏链试图定义之前的链时也会出现。

A2175　类型不合格

一个类型中的标识符并不是一个类型结构体或者记录体联合或者任何原型

A2176　用浮点数来初始化整数变量

产生问题的原因是试图使用 DWORD QWORD 或者 TBYTE 来初始化浮点数

A2177　嵌套结构体初始化不正确

嵌套的结构体无法解析

A2178　FLAT 使用方式无效

There was an ambiguous reference to FLAT as a group.

This error is generated when there is a reference to FLAT instead of a FLAT subgroup.

For example,

```
mov     ax, FLAT                ; Generates A2178
mov     ax, SEG FLAT:_data      ; Correct
```

A2179　结构体初始化方式错误

结构体初始化出现错误

可能的原因如下：

- 初始值不是一个有效的表达式
- 初始值是一个无效的 DUP 结构

A2180　列表初始化错误

结构体中，试图用一个值初始化个体列表，或者列表中的值大小错误

A2181　初始化体必须是字符串或者单个项目

试图使用非单独项目或者字符串来初始化一个结构体

产生错误的原因可能是初始化值缺少了（{}）

A2182　初始化值应该是一个单独的项目

初始化结构体时未使用单独的项目

产生错误的原因可能是初始化值缺少了（{}）

A2183　初始值必须是单字节大小

产生问题的原因是：试图将一个非单个字节大小的值装入结构体中字节大小的域中

A2184　列表初始化错误

该位置不应该出现一个开放性的大括号 （{）

A2185　文本初始化错误

初始化中丢失或者没有配对尖括号或者大括号(< >) ({ })

A2186　初始化时出现多余字符

文字结构体初始化时未能找到终结符。

可能是如下原因:

- 初始化中丢失或者没有配对尖括号或者大括号(< >) ({ })
- 初始化值最后出现额外的字符
- 结构体初始化过程中出现语法错误

A2187　必须使用浮点初始化

通过 REAL4,REAL8 或者 REAL10 声明的变量必须使用浮点数进行初始化或者使用问号（？）初始化。

通常出现这个问题的原因是你使用了整数形式（比如 18）而不是使用浮点数形式（18.0）

A2188 cannot use .EXIT for OS_OS2 with .8086

The INVOKE generated by the .EXIT statement under OS_OS2 requires the .186 (or higher) directive, since it must be able to use the PUSH instruction to push immediates directly.

A2189 段联合对齐无效

由 ALIGN 或者 EVEN 指令给定的对齐方式比当前 SEGMENT 指令给定的对齐方式要大。

A2190 INVOKE 需要过程的原型

使用 INVOKE 调用过程之前必须先使用 PROTO 声明原型

A2191 结构体不能包含本身

不可以在结构体中自引用

A2192 标识符语言属性冲突

对于同一个符号的两个声明存在语言冲突（比如 C 和 Pascal 的冲突）。

A2193 非良性的 COMM 重定义

使用 COMM 指令重新定义了一个变量，更改了语言属性、距离、大小或者实力数量。使用 COMM 多次定义一个变量必须是相同。

A2194 COMM 变量超过 64K

16 位段中使用 COMM 声明的变量不能超过 64K

A2195 参数或者本地变量不能是 void 类型

编译器尝试创一个没有类型的参数或者本地变量。可能的原因是遗漏了冒号或者将一个自定义的类型设定为 void

A2196 TINY 模式下不可以使用 OS_OS@2 标志

.MODEL 中指定了 TINY 内存模式，此外还指定了 OS_OS2 操作系统。Tiny 内存模式和 OS/2 操作系统存在矛盾。

A2197 表达式大小必须符合 32 位

试图在 32 位段(USE32 或者 FLAT)中使用 16 位的表达式。32 位段中（USE32 或者 FLAT），不能使用 16 位的表达式。

A2198 .EXIT 无法在 32 位段下工作

无法在 32 位段中使用.STARTUP 指令，它只在 16 位模式下有效。

A2199 .STARTUP 无法在 32 位段下工作

无法在 32 位段中使用.STARTUP 指令，它只在 16 位模式下有效。

A2200 联合类型中不允许使用 ORG 命令

你可以在结构体定义中使用 ORG 指令，但是在 UNION 中无法这样。

A2201 应用范围不能改变

在同一个模块中同时声明了 OPTION SCOPED 和 OPTION NOSCOPED。在同一个模块中不能切换声明范围。产生这样问题的一个原因可能是在包含的文件中声明了 OPTION SCOPED 或者 OPTION NOSCOPED

A2202 段寄存器使用方式非法

当使用/Fpi 命令行参数或者 OPTION EMULATOR 生成浮点模拟指令时，不能使用段覆盖 FS 和 GS 寄存器。

A2203 无法将代码标号的范围声明为 PUBLIC

代码试将形如“label:”的代码标号声明为 PUBLIC。你可以使用“label::”这样的形式，或者 LABEL 指令，或者 OPTION NOSCOPED 来消除这个错误

A2204 MSFLOAT 是过时的指令，将被忽略

目前已不支持微软二进制格式。请将你的数值转化为 80x87 支持的 IEEE 标准格式。

A2205 ECS 是过时的指令，将被忽略

目前不支持 ECS 指令。当前编译器支持全部数字协处理指令。

A2206 表达式中操作符丢失

表达式无法计算，因为运算符丢失。这个错误也可能是优化编译导致的副作用。

例如，下面的表达式会导致这个错误:

value1 = (1 + 2) 3

A2207　表达式中右括号丢失

表达式无法计算，因为右括弧丢失。这个错误也可能是优化编译导致的副作用。

例如，下面的表达式会导致这个错误:

value1 = ((1 + 2) * 3

A2208　表达式中左括弧丢失

表达式无法计算，因为左括弧丢失。这个错误也可能是优化编译导致的副作用。

例如，下面的表达式会导致这个错误:

value1 = ((1 + 2) * 3))

A2209　引用了前向宏重定义

当前无法使用宏，因为它还没有被定义。解决方法是在调用宏之前定义之。

A2901　无法调用 ML.EXE

MASM 无法调用 ML.EXE

可能原因是如下：

- ML.EXE 不在 PATH 指定的路径中
- ML.EXE 文件没有设置只读的文件属性
- 内存不足

ML 警告信息

A4000　无法修改 READONLY 属性的段

尝试修改的段中具有只读属性

A4002　由于 STUCT/UNION 域定义中出现重复因此该定义不合格

结构体或者联合的域必须是独一无二的。可以尝试重命名结构体的域，来解决冲突。

A4003　END 之后指定的起始地址由于.STARTUP 的存在而被忽略

同时使用了.STARTUP 和程序装载地址（END 命令指定这个地址），将会默认忽略 END 指定的这个地址。

A4004　ASSUME 的对象不能是 CS

尝试 assume 一个值给 cs 会导致这个问题。CS 只能设定为当前段或者组。

A4005　无法识别的默认 prologue

一个无法识别的参数被传递给默认的 prologue

A4006　宏调用中参数过多

宏指定参数多于定义

A4007　选项未翻译，需要其他指令: option

给定的 MASM 没有对应的 ML 命令行参数

参数　　指令

/A　　.ALPHA

/P　　OPTION READONLY

/S　　.SEQ

A4008　无效的命令行值，使用默认值

给定选项中的值无效。忽略该选项，并且使用默认值

A4009　内存不足，无法使用/EP 参数

内存不足，无法在一遍编译中产生列表

A4010 缺少 “>”

调用宏并且给定了一个文本信息参数，但是缺少 “>”

A4011　发现多个.MODEL 指令

在当前的模块中发现多个.MODEL 指令。默认第一个有效。

A4012　段的行号信息没有定义’code’类

There were instructions in a segment that did not have a class name that ends with

“CODE”。The assembler did not generate CodeView information for these instruction

CodeView cannot process modules with code in segments with class names that do not

end with “CODE”

A4013 AT 段不支持指令和数据初始化

在 AT 属性的段中的发现了指令或者初始化值的代码。在运行期代码和数据不会被调用

A4910 无法打开文件

当前路径下找不到给定的文件名

A5000 定义了@@，但是没有调用

跳转目标定义为@@:标号，但是没有跳转指令对应之。

产生这样问题的一个可能原因是，在你原来准备跳转的@@：标号中，不小心又插入了一个@@:标号。

A5001 需要表达式，假定为 0

使用 IF, ELSEIF, IFE, IFNE, ELSEIFE, or ELSEIFNE 没有指定关系表达式。编译器会默认表达式结果同 0 相比较。

A5002 externdef previously assumed to be external

The OPATTR or .TYPE operator was applied to a symbol after the symbol was used in

an EXTERNDEF statement but before it was declared. These operators were used on a

line where the assembler assumed that the symbol was external.

A5003 length of symbol previously assumed to be different 标识符长度同之前假设不同

The LENGTHOF, LENGTH, SIZEOF, or SIZE operator was applied to a symbol after

the symbol was used in an EXTERNDEF statement but before it was declared. These

operators were used on a line where the assembler assumed that the symbol had a

different length and size.

A5004　标识符同之前的假设不在一个 group 中

标识符在一个段的外面通过 EXTERNDEF 声明，然后在段内部再次声明

A5005 类型不同

通过 INVOKE 传递的参数和前面函数原型中声明的不同。编译器会进行适当的类型转换。

A6001　过程中未定义返回指令

使用 PROC 声明了一个过程，但是过程内部没有使用 RET 或者 IRET 指令返回。

A6003　条件跳转延长

条件跳转指令被编码为一个相反的条件跳转指令和一个近的无条件跳转指令。你可以通过重新调整代码来避免长条转形式。

A6004　过程参数或者位置未使用

当你使用传递一个过程参数或者使用 LOCAL 指令创建一个变量后，在这个过程中并未调用。不需要的参数或者本地参数会浪费堆栈。

A6005　表达式条件可能依赖于编译遍数

（简单解释“遍：对源程序（或其中间形式）从头到尾扫描一次，并进行有关加工处理，生成新的中间形式或最终目标程序，称为一遍。每遍的工作都由从外存上获前一遍的工作结果开始（或源程序），到完成它所包含的有关阶段程序的工作，并把结果记录于外存上为止。”形象理解，比如 MOV EAX,Foo 而 Foo 是一个宏，编译器处理的时候要将 Foo 展开，你可以想象一下，编译器刚开始的时候专门从头到尾，“来一遍”然后再交给下一步。有时候，表达式的条件和编译器这样的遍数有关系。这个比喻不是很恰当，更深入请阅读编译原理相关资料。）在命令行下使用/Zm 参数，或者指定 OPTION M510 时，表达式的值在不同遍数下结果不同。出现这个错误表明你的代码同遍数相关，必须重写之。

- 什么是 GROUP?

group　group 是一组段的集合，这些段的段基地址相同。

- 什么是 frame

段，Group 或者段寄存器这样的能够指定一个地址的段地址的成为 frame