
第 7 章 Win32 编程环境

使用汇编语言来开发 Win32 程序的软件包有 Microsoft 的 MASM 系列、Borland 的 TASM 系列等。本章首先介绍了 MASM32 软件包的组成及安装方法；然后通过 MASM32 软件包自带的例子，介绍集成环境 QEDITOR 的使用；最后介绍生成 Win32 可执行文件的步骤和方法。

7.1 MASM32 软件包简介

1. MASM 系列编译器介绍

MASM 是 Microsoft 公司推出的汇编编译器。它有众多版本，高版本和低版本的语法和功能相差很多，向下兼容性也不好。低版本的 MASM 固然无法编译高版本的源程序，但是，高版本的 MASM 也可能无法正常编译低版本的源程序。在使用 MASM 系列编译器时，一定要注意所使用的版本。

在前 6 章中，介绍 DOS 环境下的汇编编程，使用的是 MASM 6.00。从本章开始，介绍如何编制 Win32 编程，其使用的是 MASM 6.14 编译器。下面介绍一些重要的 MASM 版本及其关键演变历程。

1989 年 Microsoft 推出了 MASM 5.10B，它是传统的 DOS 汇编编译器中最完善的版本，运行稳定、速度快。该版本支持 80386 处理器指令，允许将段定义伪指令简化为“.code”与“.data”之类的形式，同时增加了对“@@”标号的支持。这样，程序员可以不再为标号的起名花掉很多时间。

1992 年 Microsoft 发布了 MASM 6.00。编译器使用扩展内存，编译更大的文件，这时，可执行文件名相应从 MASM.EXE 改为 ML.EXE。从该版本开始，可以在命令行上使用“*.ASM”，以便同时编译多个源文件。最大的改进之一是开始支持“.IF/ENDIF”等高级语法，使得复杂的条件分支语句的编写和用高级语言编写一样简单，可以在几千行代码中不定义一个标号；另外，增加了 invoke 伪指令，简化了带参数的子程序调用。这两个改进使汇编代码的风格越来越像 C，可读性和可维护性提高了很多。

MASM 6.11 于 1993 年 11 月发布，从该版本开始，支持 Win32 编程，支持 Windows NT、Pentium 指令。

MASM 6.14 是一个很完善的版本，它在 .XMM 中增加了对 Pentium III 的 SIMD 指令集的支持。

不同版本 MASM 产生的目标文件(.OBJ)格式不同。在 DOS 和 Win16 时期，Microsoft 使用的目标文件格式为 OMF(Intel Object Module Format)格式，到了 Win32 时期(从 MASM 6.11 版本开始)，改用了 COFF(Common Object File Format)格式，原因之一是 COFF 格式更像最终的 PE 文件，在链接时可以做更少的处理。用 Microsoft 的产品编写 Win32 程序，不管是使用 VC 还是 MASM，都必须使用 COFF 格式，因为 Microsoft 的 32 位的 Link 只支持将 COFF 格式的 OBJ 文件链接成 PE 文件，另外所有的导入库等支持文件的格式也全部是 COFF 格式的。

2. MASM32 软件包

MASM32 软件包是不同工具软件的大集合，它包含如下工具软件：

(1) 汇编编译器，来源于 MASM 软件包中的 ML.EXE；

- (2) 32 位链接器，来源于 Microsoft Visual Studio 的 LINK.EXE;
- (3) 资源编译器，同样来源于 Microsoft Visual Studio 的 RC.EXE。同时也包含了 Microsoft Visual Studio 中的其他一些工具，如 LIB.EXE 和 DUMPPE.EXE 等，所有的工具都适合于 Win32 编程的版本。
- (4) 导入库，包括详尽的头文件。导入库文件取自 Visual C++ 的导入库。
- (5) 头文件，包括详尽的头文件。规模庞大的头文件是发布者整理出来的。
- (6) W32 程序例子，涉及 Win32 汇编的很多方面，收集自世界各地 Win32 汇编爱好者发布的源程序。
- (7) 集成开发环境，为了使工具包更实用，发布者还为它编写了一个简单的集成开发环境（Integrated Develop Environment, IDE）QEDITOR.EXE。该 IDE 环境使用只是一个简单的文本编辑器加上用户可以自行设置菜单的 Shell。在开发软件时完全可以不用该集成开发环境。

MASM32 软件包完全是为了用汇编语言编写专业大型程序而开发的。它的发布者 Steve Hutchesson 做了所有汇编程序员都想做却又在庞大的工程量前止步的工作，如收集合适的工具软件，收集导入库，整理出完整的头文件，收集例子文件，写帮助文档，等等。

本书使用的是 MASM32V8 版本，在光盘上有一个 MASM32V8.EXE。和早期的 MASM32 版本相比，使用的编译器、链接器等可执行文件并没有什么改变，不同的是在头文件中增补了一些数据结构定义，增加了不少例子程序。最新版本的 MASM32 软件包可以在发布者的主页 <http://www.movsd.com> 中下载。MASM32 是一个免费的软件包，但其中的不同部分如编译器和例子程序等可能属于不同的公司和个人，使用时需要注意其版权声明。

7.2 MASM32 软件包安装

双击 MAMS32V8.EXE，出现如图 7.1 所示的 MASM32 的安装界面。

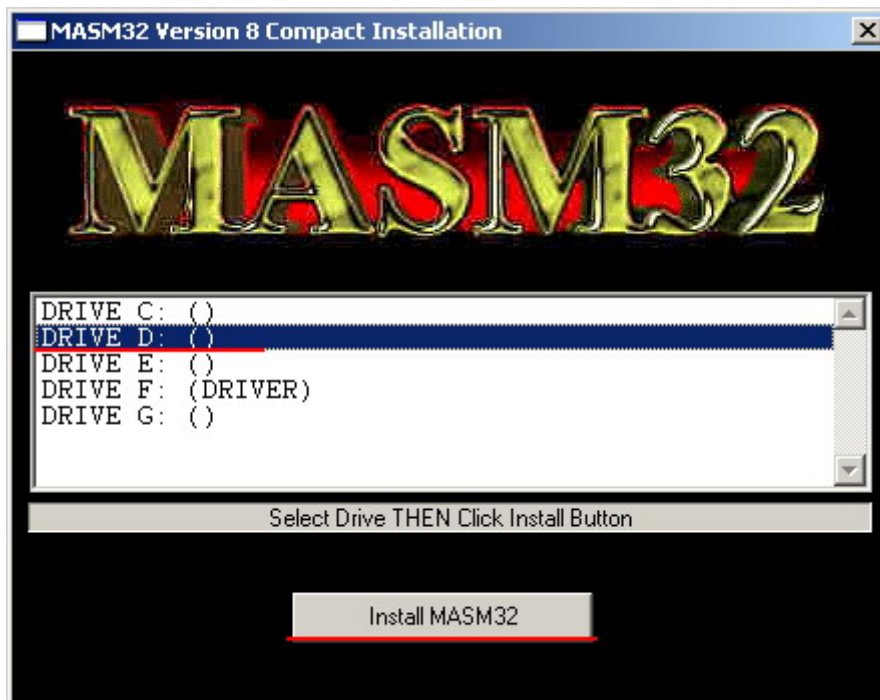


图 7.1 MASM32 的安装界面

首先，选择将 MASM32 安装在哪一个驱动器上。例如，我们要将其安装在 D:盘上，单击

“DRIVE D:”所在的行，该行上将出现蓝色亮条。这时，安装的目标位置将是 D:\masm32 目录。之后，单击“Install MASM32”，出现如图 7.2 所示的界面。

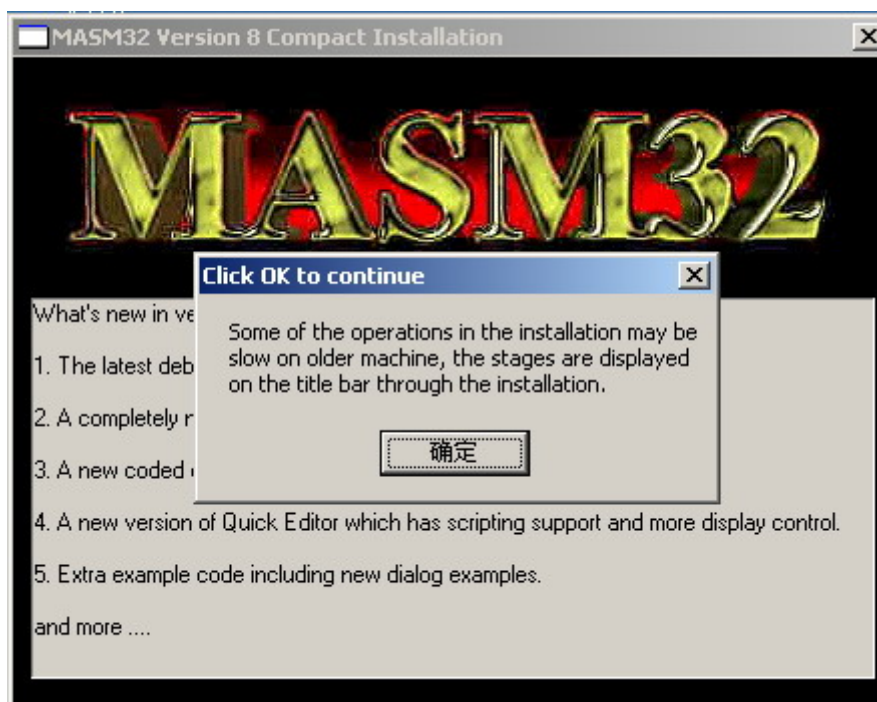


图 7.2 MASM32 的安装提示

弹出的对话框提示安装过程中的一些信息将显示在标题栏上。单击“确定”，开始安装。之后，我们可以看到标题栏上不断变化的提示信息。该过程要持续一会，请耐心等待，直到出现如图 7.3 所示的界面。

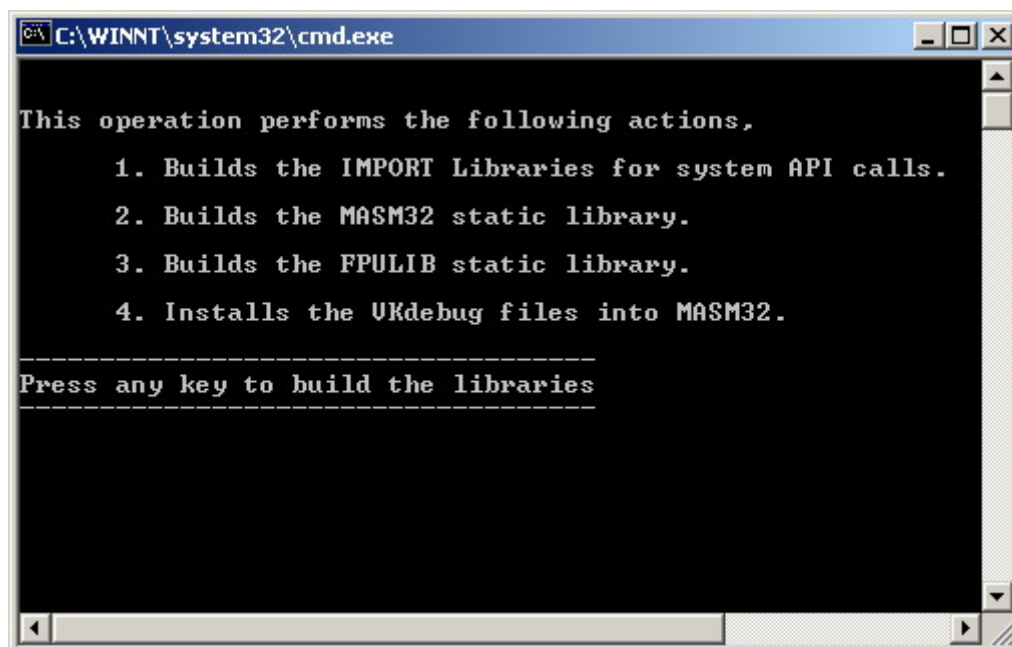


图 7.3 建立各种库

按任意键，开始建立各种库文件。安装完成后，有一个提示信息框，如图 7.4 所示。单击其中的“确定”即可。

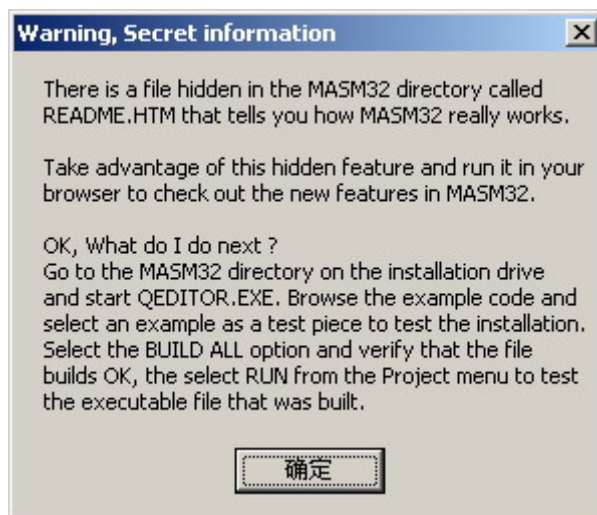


图 7.4 安装完成后的提示

图 7.4 提示信息中，一是指明在安装后的目录下（如 D:\masm32）有 README.HTM，该文件中有很多帮助信息；二是使用 QEDITOR.EXE 验证系统是否成功安装。7.3 节中将简单介绍 QEDITOR 的使用方法。

在安装完成后，我们可以浏览一下 MASM32 下各目录的内容，熟悉以后经常要用到的一些重要文件的位置。

\masm32\bin 可执行文件目录，里面包括 ML.EXE、LINK.EXE、LIB.EXE 等。在 bin 目录下，还有多个批处理文件 (*.BAT)，这些文件可以使用 UltraEdit 等工具打开。另外还有图像编辑工具 IMAGEDIT.EXE、资源编译器 RC.EXE、从资源文件到目标文件的转换工具 CVTRES.EXE 等。

\masm32\include 所有头文件的目录，WINDOWS.INC 为数据结构和预定义值的定义文件，RESOURCE.H 为资源文件的头文件，其他 .INC 文件为对应 DLL 文件中的 API 函数声明文件；

\masm32\lib 所有的导入库文件，每个.LIB 文件是对应 DLL 文件的导入库；

\masm32\help 帮助文件目录；

\masm32\m32lib 一些常用 C 子程序的汇编实现源程序，如熟悉的 STDIN 和 STDOUT 等，有一定的参考价值；

其他目录主要为例子和可用可不用的工具，例子中广泛收集自网上不同作者的作品，很有参考价值。

在众多目录中，最为关键的是 bin、include 和 lib，有了这三个目录中的内容，就可以进行 Win32 汇编编程了，其他目录中的文件仅起辅助作用。

7.3 使用 QEDITOR

7.3.1 首次使用 QEDITOR

本节我们使用 MASM32 软件包自带的例子，了解 Windows 程序编辑、链接、运行的过程。

双击 D:\masm32 目录下的 QEDITOR.EXE，出现如图 7.5 所示的界面。

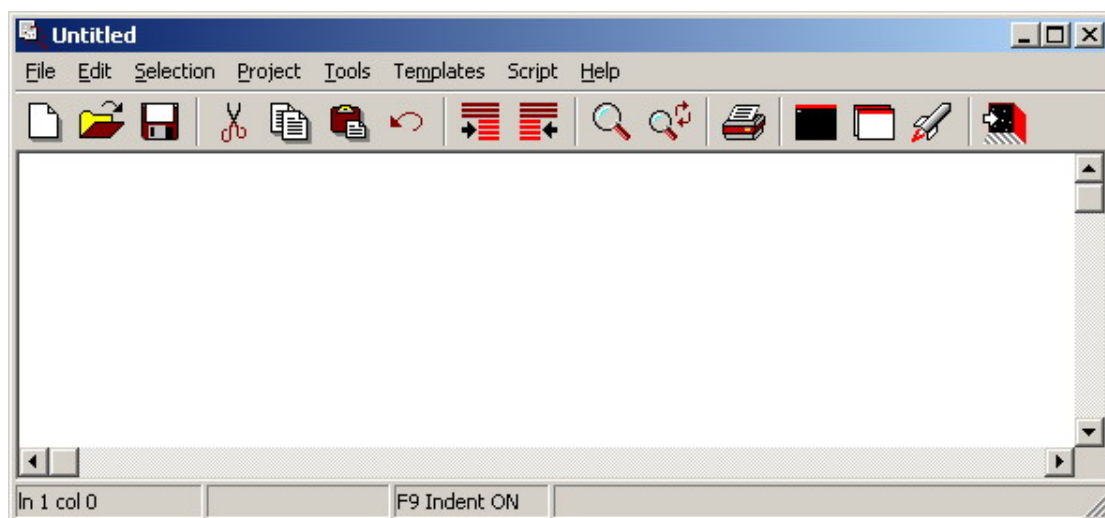


图 7.5 QEDITOR 的运行界面

单击“File / Open”，出现“Open File”对话框，如图 7.6 所示。

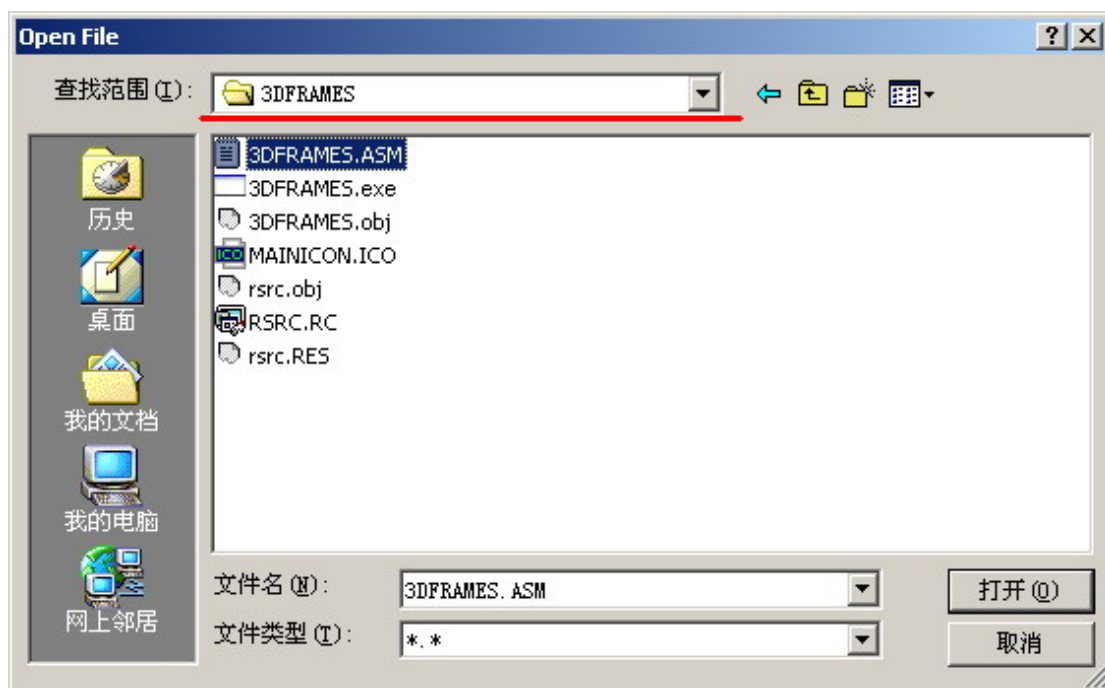


图 7.6 打开文件对话框

选择 EXAMPLE1 \ 3DFRAMES \ 3DFRAMES.ASM，也可出现图 7.6 所示的界面。单击“打开”，将出现 3DFRAMES.ASM 程序，在标题栏上也有该文件的路径。

执行“Project / Build All”，如图 7.7 所示，将对资源脚本文件、汇编源程序进行编译，之后执行链接操作，出现如图 7.8 所示的界面。

在图 7.8 所示的提示信息中，可以看到“Creating rsrc.RES”，即由资源脚本文件 RSRC.RC 生成资源文件 rsrc.RES；可以看到编译器的版本信息“Assembler Version 6.14”，还可以看到“Assembling D:\masm32\EXAMPLE1\3DFRAMES\3DFRAMES.asm”，即对源程序进行了汇编。之后，有增量式链接器的版本“Incremental Linker Version 5.12”。最后，生成了 3D FRAMES.EXE 文件。

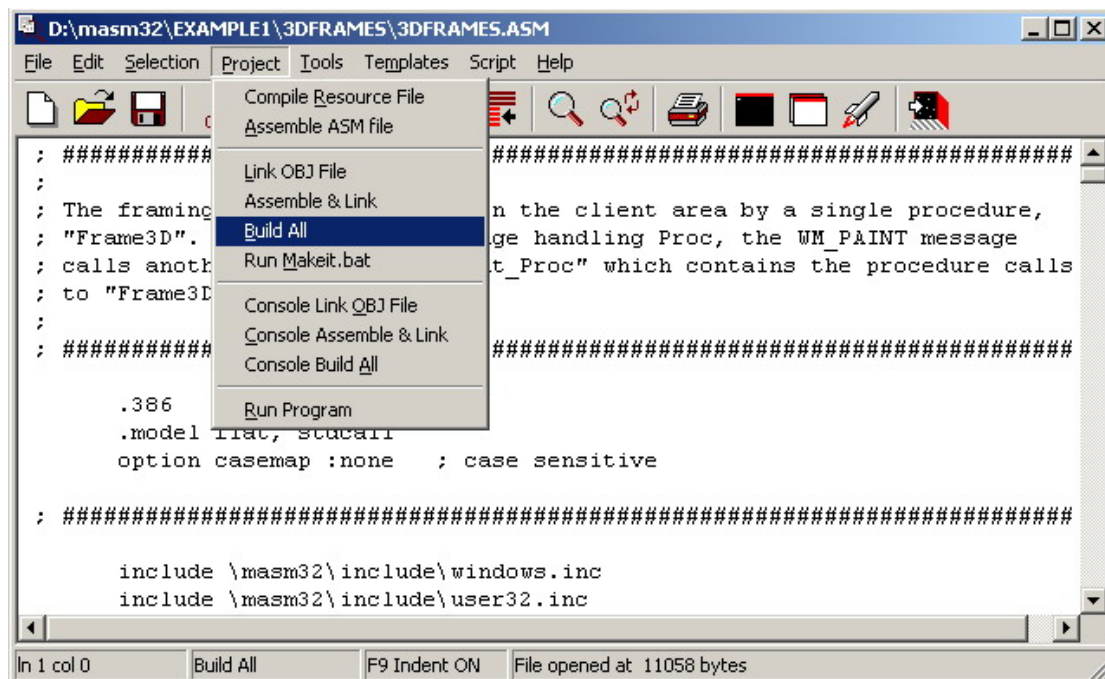


图 7.7 编译、链接程序

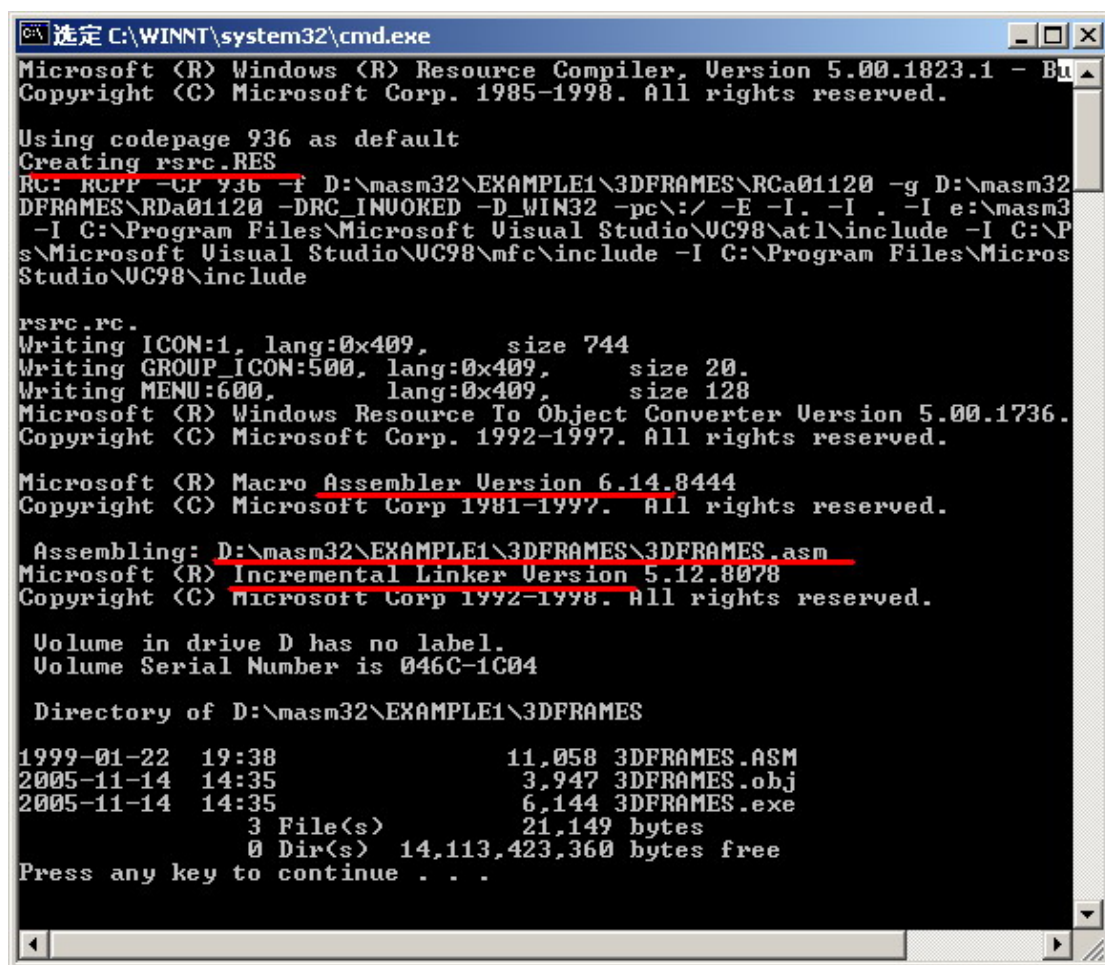


图 7.8 编译、链接后的提示信息

在 7.8 所示的界面中，按任意键返回到 QEDITOR 界面。

单击“Project / Run Program”，出现程序 3DFRAMES.EXE 的运行界面，如图 7.9 所示。

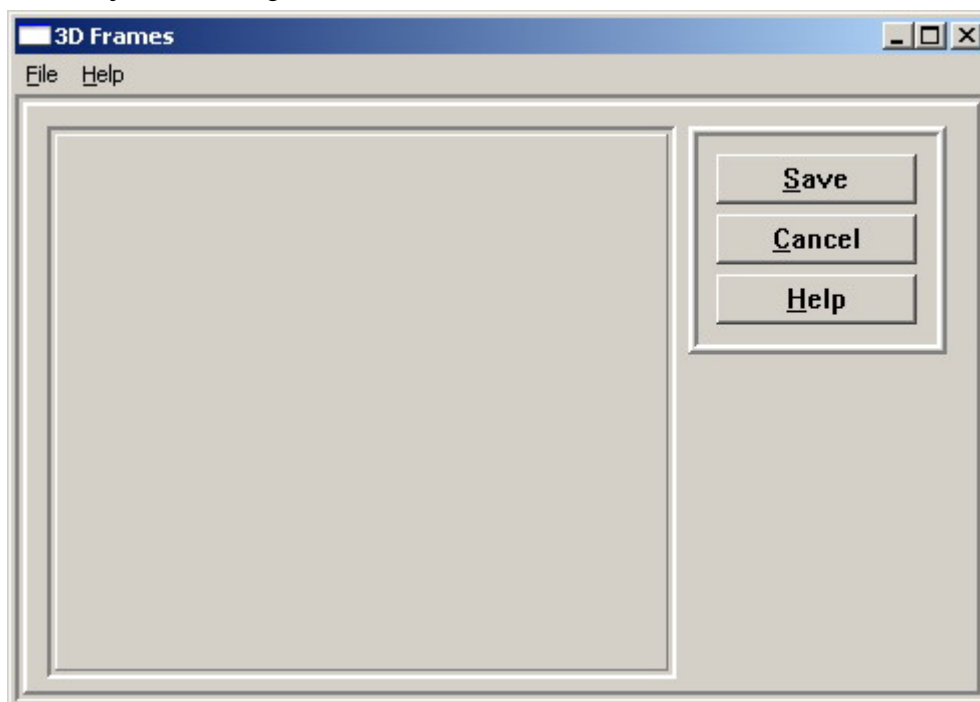


图 7.9 3DFRAMES.EXE 的运行界面

该程序具有菜单项 File、Help，还有三个按钮 Save、Cancel、Help。风格与常见的 Windows 程序一样。单击 Help 菜单下的 About，会出现一个消息框。单击 File 菜单下的 Exit，会出现一个确认退出的消息框。该程序仅仅是为了显示一个有立体效果的界面而编制的，各按钮并没有对应什么功能。

7.3.2 QEDITOR 揭密

打开 D:\masm32 目录下的 MENUS.INI，可看到如下信息。

```
[&Project]
Compile &Resource File,\MASM32\BIN\Bres.bat {b}
&Assemble ASM file,\MASM32\BIN\Assmbl.bat {b}
-
&Link OBJ File,\MASM32\BIN\Lnk.bat {b}
Assemble && Link,\MASM32\BIN\Build.bat {b}
&Build All,\MASM32\BIN\Bldall.bat {b}
Run &Makeit.bat,makeit.bat
-
Console Link &OBJ File,\MASM32\BIN\Lnkc.bat {b}
&Console Assemble && Link,\MASM32\BIN\Buildc.bat {b}
Console Build &All,\MASM32\BIN\Bldallc.bat {b}
-
&Run Program,{b}.exe

[&Tools]
```

.....

[&Project]即菜单 Project, &P 表示字母 P 下有下列线。在[&Project]下的行是菜单项及单击菜单项时执行的命令及相应参数, 前两者之间以逗号分隔。

Project 包括的菜单项如下:

Compile Resource File

Assemble ASM file

Link OBJ File

单击某一个菜单项, 执行的是菜单项右边的命令。

例如, 单击 “Build All”, 执行的是\MASM32\BIN 目录下的一个批处理文件 BLDALL.BAT, 它有一个参数{b}, 即为当前文件名。

通过修改 MENUS.INI, 仿照已有的菜单, 可以编写自己的菜单。

下面分析批处理文件 BLDALL.BAT 的内容。使用一个文本编辑器打开 BLDALL.BAT (可以就使用 QEDITOR, 也可以使用 UltraEdit 等)。其内容如下:

```
@echo off
if not exist rsrc.rc goto over1
\masm32\bin\rc /v rsrc.rc
\masm32\bin\cvtres /machine:ix86 rsrc.res
:over1
if exist "%1.obj" del "%1.obj"
if exist "%1.exe" del "%1.exe"
\masm32\bin\ml /c /coff "%1.asm"
if errorlevel 1 goto errasm
if not exist rsrc.obj goto nores
\masm32\bin\Link /SUBSYSTEM:WINDOWS "%1.obj" rsrc.obj
if errorlevel 1 goto errlink
dir "%1.*"
goto TheEnd
:nores
\masm32\bin\Link /SUBSYSTEM:WINDOWS "%1.obj"
if errorlevel 1 goto errlink
dir "%1.*"
goto TheEnd
:errlink
echo _
echo Link error
goto TheEnd
:errasm
echo _
echo Assembly Error
goto TheEnd
:TheEnd
Pause
```

BLDALL.BAT 是一个 DOS 的批处理文件。首先 “echo off”, 即可在屏幕上不显示执行

的命令信息。

如果文件 **RSRC.RC** 不存在，则转移到标号 **over1** 处；否则，使用 **RC.EXE** 对资源脚本文件 **RSRC.RC** 进行编译，生成资源文件“**RSRC.RES**”文件，之后使用 **CVTRES.EXE** 将 **RSRC.RES** 转换成 **RSRC.OBJ**。在资源脚本编译后，也执行到 **over1** 处。从这一小段程序可以看到，它对资源文件进行编译，并且将资源脚本的名称固定成了 **RSRC.RC**。在 **MASM32** 软件包所带的例子中，绝大部分使用的是 **RSRC.RC**。实际上，资源脚本完全可以使用其它名称，只是该处理程序将其写死罢了。例如，若将资源脚本文件命名为 **3DFRAMES_RC.RC**，则可以将批处理文件中的 **RSRC.RC** 替换为 **%1_RC.RC**。当然相应的 **RSRC.RES** 替换为 **%1_RC.RES**，**RSRC.OBJ** 替换成 **%1_RC.OBJ**。

在 **over1** 处，判断“**%1.obj**”和“**%1.exe**”是否存在，若存在，则删除它们。“**%1**”就是在 **BLDALL.BAT** 后面带的第一个参数，实际上就是在 **QEDITOR** 中打开的文件名(不含扩展名)。

紧接着，执行“**\masm32\bin\ml /c /coff "%1.asm"**”，即对文件进行编译。采用的编译器为 **masm32\bin** 目录下的 **ml**。有关 **ml** 的使用在 7.4 节介绍。如果编译有错误，将转到标号“**errasm**”处，显示出错提示。

在编译无错误的情况下，若有资源文件，则执行：

```
\masm32\bin\Link /SUBSYSTEM:WINDOWS "%1.obj" rsrc.obj
```

否则执行：

```
\masm32\bin\Link /SUBSYSTEM:WINDOWS "%1.obj"
```

该步执行了链接操作。有关 **Link** 的使用，也将在 7.4 节介绍。

如果链接有错误，转到标号“**errlink**”处。否则显示文件名等同参数 1 的文件，扩展名是任意的(**dir "%1.*"**)。

最后的一条命令是 **Pause**，即暂停执行 **DOS** 批处理文件的执行，等待用户按任意键。在按任意键后，继续执行 **DOS** 批处理程序。由于 **Pause** 之后无任何命令，批处理程序的执行也就结束了。此处写 **Pause** 的目的是让用户看清楚提示，否则一闪而过，也不知道编译、链接的状况。

与上述批处理程序类似，同样可以根据自己的需求编写的批处理文件，定制自己的菜单。

使用 **QEDITOR**，可以编辑汇编源程序，也还可以用来编辑资源脚本文件。资源脚本文件是一个文本文件，描述了程序中使用的菜单、对话框、位图、字符串等，具体的资源描述文法在第 8 章介绍。

在创建、编辑汇编源程序的时候，还可以使用系统提供的一些模板。例如，在 **Templates** 就有创建一个对话框程序的模板（**Dialog Application Template**）等，在 **Script** 菜单下有汇编程序框架模板(**ASM**)，也有提示消息框模板(**MessageBox**)，等等。感兴趣的读者可以定制菜单，进一步扩充系统的功能。

7.4 生成 Win32 程序的过程

生成一个可执行的 **Windows** 程序，大致要经过汇编源文件的编辑和编译、资源文件的编辑和编译、链接等过程。下面将对程序的编辑、编译、链接中使用到的软件进行更详细的说明。

7.4.1 汇编源文件和资源脚本的编辑

汇编源文件的编辑与 DOS 下写汇编源程序是一样的。使用文本编辑器（如 UltraEdit）就可以编写汇编源代码（*.ASM 文件）。和 DOS 程序不同的是，DOS 下的汇编源程序广泛地使用中断，而在 Win32 程序中，要大量使用 Win32 API，因此在 ASM 文件中要使用 include 语句包含相应的数据定义和函数声明头文件。Win32 汇编的头文件一般用“.INI”作扩展名，如 MASM32 软件包附带的 Windows.inc。Windows.inc 中定义了 Win32 API 中很多参数和数据结构，其他的 INC 文件则是对应同名 DLL 中的 Win32 API 函数声明。

Win32 程序的一个特点就是具有良好的用户界面，这些界面实际上是由大量资源所组成。Windows 程序中使用的资源包括：菜单、对话框、快捷键、字符串、版本信息和图标等。这些资源用一些文本描述，通常将它们放在一个脚本文件中，扩展名一般为“.RC”。不同类型的资源需要不同的描述信息，比较详细的语法将在第 8 章中介绍。资源脚本文件同样用到很多预定义值，所以软件包一般有资源头文件用来导入源文件。MASM32 软件包中的资源头文件是 Resource.h。

我们可以采用文本编辑器来生成资源脚本，但更为有效的方法是采用所见即所得的资源编辑器，第 8 章中将介绍 Visual C++ 提供的编辑资源功能。

7.4.2 汇编源文件的编译

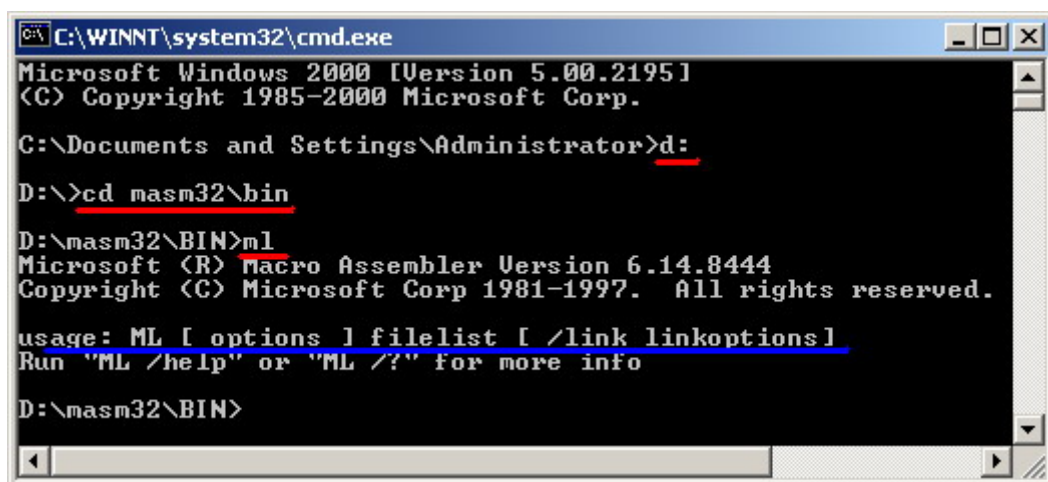
使用 ML.EXE 可以编译汇编源文件。

单击“开始/运行”，在弹出的窗口上输入 cmd，打开一个 cmd 窗口，依次执行命令：

```
C:\Documents and Settings\Administrator>d:↵  
D:\>cd masm32\bin↵  
D:\masm32\BIN>ml↵
```

可以看到 ML 的版本信息，以及 ML 的用法提示及得到更多帮助的方法，如图 7.10 所示。

图中，红线标示的地方为输入的信息，蓝线标示的是用法提示。



```
C:\WINNT\system32\cmd.exe  
Microsoft Windows 2000 [Version 5.00.2195]  
(C) Copyright 1985-2000 Microsoft Corp.  
  
C:\Documents and Settings\Administrator>d:  
D:\>cd masm32\bin  
D:\masm32\BIN>ml  
Microsoft (R) Macro Assembler Version 6.14.8444  
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.  
  
usage: ML [ options ] filelist [ /link linkoptions ]  
Run "ML /help" or "ML /?" for more info  
  
D:\masm32\BIN>
```

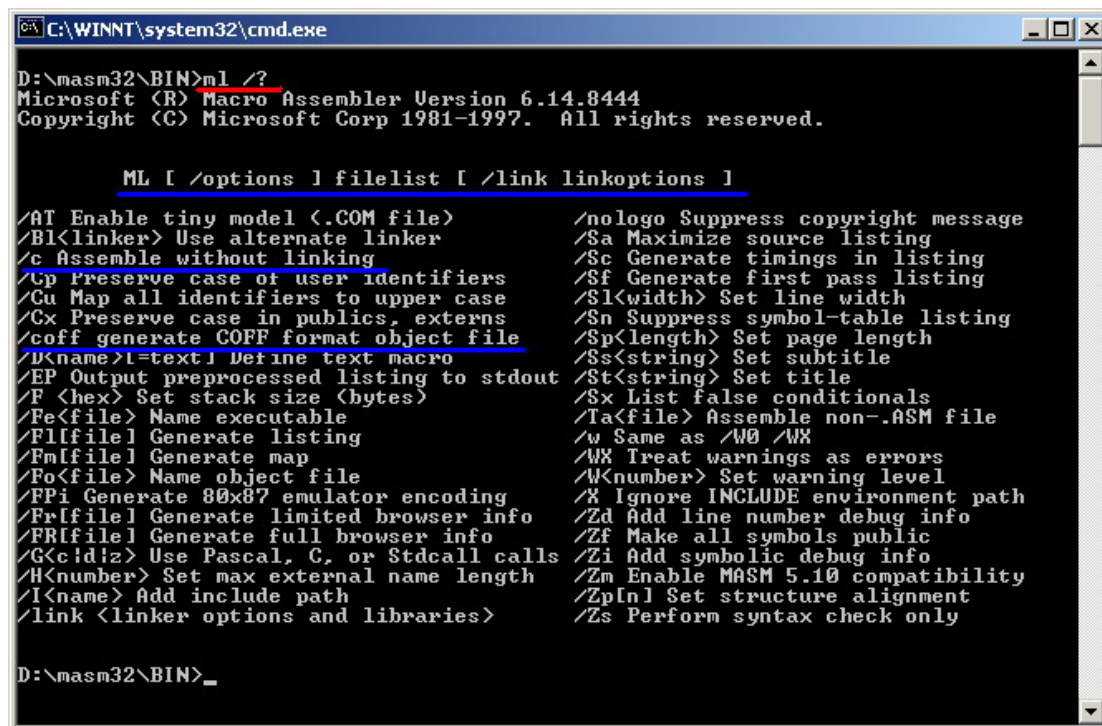
图 7.10 Ml 6.14 的用法

执行: D:\masm32\BIN>ml /?↵

可以看到 ML 的用法:

```
ML [/options] filelist [/link linkeoptions]
```

其中，options 为编译选项，filelist 为汇编源文件列表，linkeoptions 链接选项。
图 7.11 所示的是具体的选项及其描述。



```
C:\WINNT\system32\cmd.exe
D:\masm32\BIN>ml /?
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

    ML [ /options ] filelist [ /link linkoptions ]

/AT Enable tiny model (.COM file)
/Bl<linker> Use alternate linker
/c Assemble without linking
/Cp Preserve case of user identifiers
/Cu Map all identifiers to upper case
/Cx Preserve case in publics, externs
/coff generate COFF format object file
/D<name>=l=text1 Define text macro
/EP Output preprocessed listing to stdout
/F <hex> Set stack size (bytes)
/Fe<file> Name executable
/Fl[file] Generate listing
/Fm[file] Generate map
/Fo<file> Name object file
/FPi Generate 80x87 emulator encoding
/FR[file] Generate limited browser info
/FR[file] Generate full browser info
/G<c|d|z> Use Pascal, C, or Stdcall calls
/H<number> Set max external name length
/I<name> Add include path
/link <linker options and libraries>
/nologo Suppress copyright message
/Sa Maximize source listing
/Sc Generate timings in listing
/Sf Generate first pass listing
/Sl<width> Set line width
/Sn Suppress symbol-table listing
/Sp<length> Set page length
/Ss<string> Set subtitle
/St<string> Set title
/Sx List false conditionals
/Ta<file> Assemble non-.ASM file
/w Same as /W0 /WX
/WX Treat warnings as errors
/W<number> Set warning level
/X Ignore INCLUDE environment path
/Zd Add line number debug info
/Zf Make all symbols public
/Zi Add symbolic debug info
/Zm Enable MASM 5.10 compatibility
/Zp[1] Set structure alignment
/Zs Perform syntax check only

D:\masm32\BIN>
```

图 7.11 ml 的用法及参数

与 ML 6.0 相比, ML6.14 多了一个非常重要的也是生成 Win32 程序必需的选项 /coff, 即产生 COFF 格式的目标文件。/c 选项也是经常要用的, 使用了该参数就只生成目标文件, 而不链接, 否则要进行链接, 生成可执行文件。但是, 生成 Windows 程序时, 经常要使用资源文件, 一般不让 ML 直接用默认的方式链接。

除/coff、/c 外, 其它常用的选项说明如下:

- /Cp 源代码区分大小写
- /Fo filename 指定输出的 obj 文件名
- /Fe filename 指定链接后输出的.EXE 文件名
- /Fl [filename] 产生 .LST 列表文件
- /Gc 函数调用类型用 FORTRAN 或 PASCAL 形式
- /Gd 函数调用类型用 C 语言形式
- /Gz 函数调用类型用 StdCall 形式
- /I pathname 指定 INCLUDE 文件的路径
- /link 选项 指定链接时使用的选项
- /Sc 在列表文件中列出指令的时钟周期
- /Zi 增加符号调试信息

对于选项/Cp 和/Gz 也很重要, 但是可以在 ASM 源文件中用伪定义设置, 所以一般不在命令行中指定, 以免遗漏。

执行命令:

```
D:\masm32\BIN>cd ..\example\3dframes✓
D:\masm32\example1\3dframes>ml /c /coff 3dframes.asm✓
```

可以看到如图 7.12 所示的界面。出现的致命错误提示信息：无效的命令行参数/coff。仔细观察 7.12 上的信息，可以发现上述命令使用编译器的版本是 6.0。

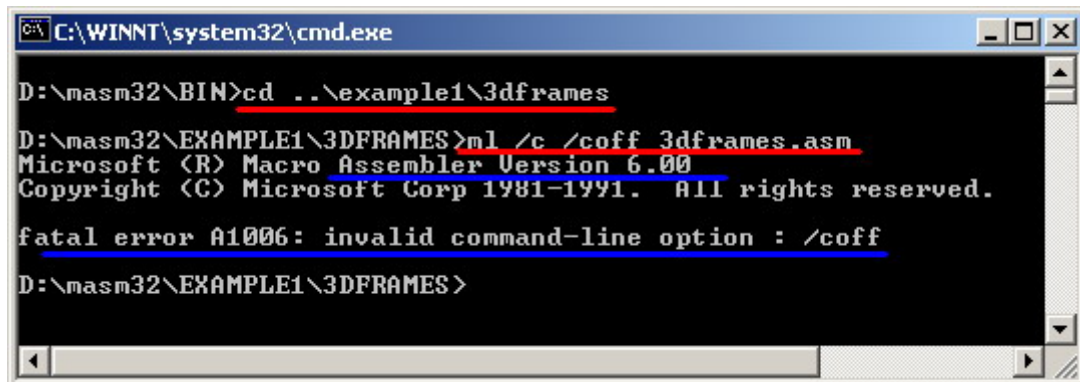


图 7.12 执行 ml 出现的错误

分析：我们在 1.5 节“建立更好的工作环境”时，曾在环境变量 Path 中添加了 masm 6。而 masm 6 中也含有 ML.EXE。在执行“ml /c /coff 3dframes.asm”时，首先在当前目录（即 d:\masm32\example1\3dframes）下寻找 ML.EXE。由于当前目录下找不到 ML.EXE，故再从环境变量 path 串的最左边开始，依次在各个目录中寻找，最后在 D:\masm6 目录下找到了 ML.EXE，查找工作停止，并运行该目录下的 ML.EXE。当然，如果在 path 指明的路径中也找不到 ML.EXE 时，则会给出错误提示。

执行命令：

D:\masm32\example1\3dframes>d:\masm32\bin\ml /c /coff 3dframes.asm

可以看到 3dframes.asm 已被编译的信息，如图 7.13 所示。

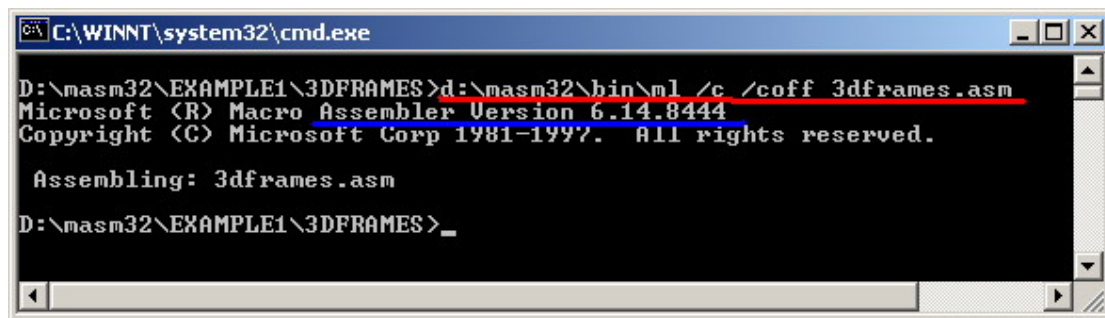


图 7.13 编译 3dframes.asm

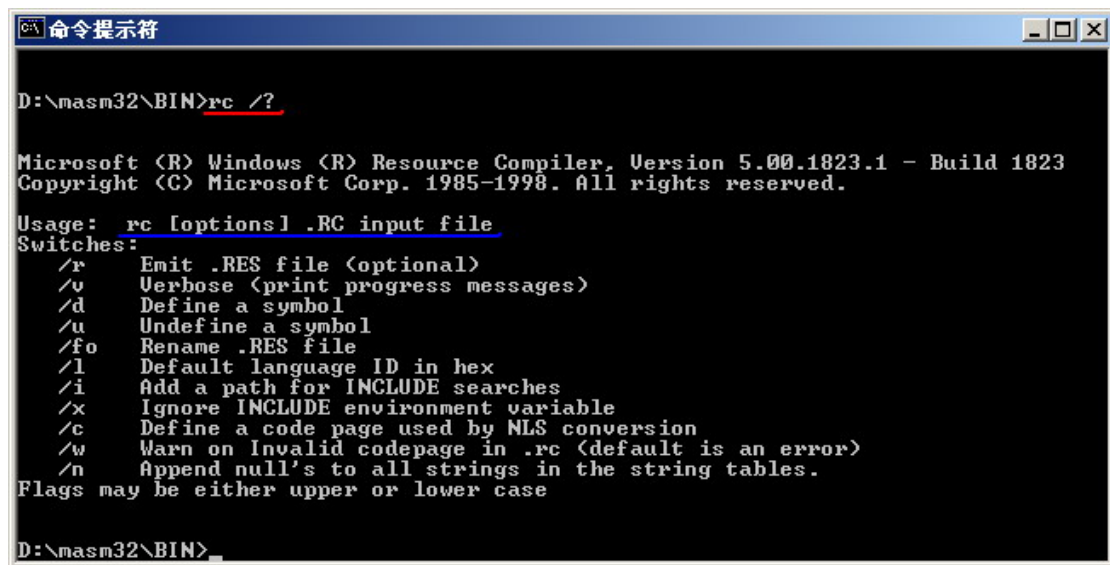
在上面的命令中，在 ML 前有一大串路径信息。如果每次都要这样输入的话，无疑是很痛苦的。读者可以将 D:\MASM32\bin 添加到环境变量 path 中。由于 D:\MASM6、D:\MASM32\bin 下面均含有 ML.EXE，要特别注意两者之间在 path 中的顺序，操作系统在寻找文件时，遵循谁写在 Path 串的前面就使用谁的规定。在 7.4.5 节，给出了另外一种建立更好工作环境的方法。

7.4.3 资源脚本的编译

采用 Microsoft 的资源编译器 RC.EXE，将资源脚本文件转换成资源文件(RES)。执行命令：“rc /?”，可以看到 RC.EXE 的用法及参数提示，如图 7.14 所示。

简单的资源脚本编译只需要执行如下命令：

D:\masm32\ example1\3dframes>d:\masm32\bin\rc rsrc.rc



```
命令提示符

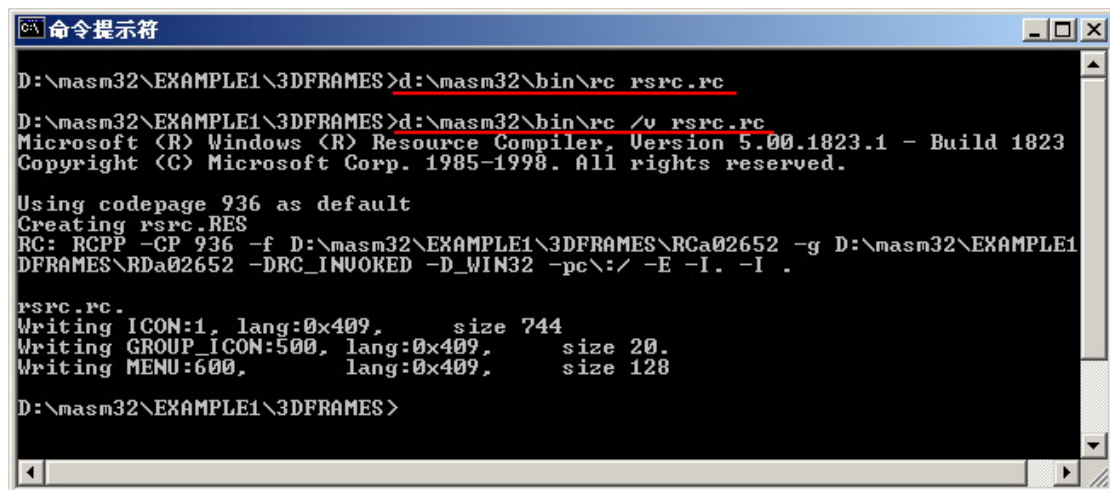
D:\masm32\BIN>rc /?

Microsoft (R) Windows (R) Resource Compiler, Version 5.00.1823.1 - Build 1823
Copyright (C) Microsoft Corp. 1985-1998. All rights reserved.

Usage: rc [options] .RC input file
Switches:
/r      Emit .RES file (optional)
/v      Verbose (print progress messages)
/d      Define a symbol
/u      Undefine a symbol
/fo     Rename .RES file
/l      Default language ID in hex
/i      Add a path for INCLUDE searches
/x      Ignore INCLUDE environment variable
/c      Define a code page used by NLS conversion
/w      Warn on Invalid codepage in .rc (default is an error)
/n      Append null's to all strings in the string tables.
Flags may be either upper or lower case

D:\masm32\BIN>
```

图 7.14 资源编译器的用法及参数



```
命令提示符

D:\masm32\EXAMPLE1\3DFRAMES>d:\masm32\bin\rc rsrc.rc
D:\masm32\EXAMPLE1\3DFRAMES>d:\masm32\bin\rc /v rsrc.rc
Microsoft (R) Windows (R) Resource Compiler, Version 5.00.1823.1 - Build 1823
Copyright (C) Microsoft Corp. 1985-1998. All rights reserved.

Using codepage 936 as default
Creating rsrc.RES
RC: RCPP -CP 936 -f D:\masm32\EXAMPLE1\3DFRAMES\RCa02652 -g D:\masm32\EXAMPLE1\
DFRAMES\RDa02652 -DRC_INVOKED -D_WIN32 -pc\:/ -E -I. -I .

rsrc.rc.
Writing ICON:1, lang:0x409, size 744
Writing GROUP_ICON:500, lang:0x409, size 20.
Writing MENU:600, lang:0x409, size 128

D:\masm32\EXAMPLE1\3DFRAMES>
```

图 7.15 资源脚本的编译

如果要显示信息，可以加选项/v, 执行的命令是：

D:\masm32\ example1\3dframes>d:\masm32\bin\rc /v rsrc.rc

显示的结果如图 7.15 所示。

7.4.4 执行文件的生成

采用增量式的链接器 LINK.EXE，将目标文件(*.OBJ)、资源文件(*.RES)、库函数链接在一起，形成可执行文件。

执行命令：

D:\masm32\EXAMPLE1\3DFRAMES>D:\masm32\bin\link /?

可以看到 Link 的用法及帮助信息。

执行命令：

D:\masm32\EXAMPLE1\3DFRAMES>D:\masm32\bin\link /? >linkhelp.txt

可以在 LINKHELP.TXT 中看到 LINK.EXE 的用法及参数。内容如下：

Microsoft (R) Incremental Linker Version 5.12.8078

Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

usage: LINK [options] [files] [@commandfile]

options:

/ALIGN:#
/BASE:{address|@filename,key}
/COMMENT:comment
/DEBUG
/DEBUGTYPE:{CV|COFF}
/DEF:filename
/DEFAULTLIB:library
/DLL
/DRIVER[:{UPONLY|WDM}]
/ENTRY:symbol
/EXETYPE:DYNAMIC
/EXPORT:symbol
/FIXED[:NO]
/FORCE[:{MULTIPLE|UNRESOLVED}]
/GPSize:#
/HEAP:reserve[,commit]
/IMPLIB:filename
/INCLUDE:symbol
/INCREMENTAL:{YES|NO}
/LARGEADDRESSAWARE[:NO]
/LIBPATH:dir
/MACHINE:{ALPHA|ARM|IX86|MIPS|MIPS16|MIPSR41XX|PPC|SH3|SH4}
/MAP[:filename]
/MAPINFO:{EXPORTS|FIXUPS|LINES}
/MERGE:from=to
/NODEFAULTLIB[:library]
/NOENTRY
/NOLOGO
/OPT:{ICF[,iterations]]|NOICF|NOREF|NOWIN98|REF|WIN98}
/ORDER:@filename
/OUT:filename
/PDB:{filename|NONE}
/PDBTYPE:{CON[SOLIDATE]]|SEPT[YPES]}
/PROFILE
/RELEASE
/SECTION:name,[E][R][W][S][D][K][L][P][X]
/STACK:reserve[,commit]
/STUB:filename
/SUBSYSTEM:{NATIVE|WINDOWS|CONSOLE|WINDOWSCE|POSIX}[,[.##]]
/SWAPRUN:{CD|NET}
/VERBOSE[:LIB]

```
/VERSION:#[.#]
/VXD
/WARN[:warninglevel]
/WINDOWSCE:{CONVERT|EMULATION}
/WS:AGGRESSIVE
```

执行命令：

```
D:\masm32\EXAMPLE1\3DFRAMES>d:\masm32\bin\link /subsystem:windows
/libpath"d:\masm32\lib" 3dframes.obj rsrc.res /
```

可以生成可执行文件 3dframes.EXE。在上述命令中，使用了两个常用的选项，一个是 subsystem:windows，指明程序运行的操作系统是 Windows；另一个是 /libpath，指明库文件所在的目录。

在使用 LINK 时，文件列表（files）用来列出所有需要链接到可执行文件中的模块，可以指定多个 obj 文件、res 资源文件以及导入库文件。

尽管 LINK 的参数很多，但最常用的是如下三种：

Link /subsystem:windows x.obj y.lib z.res（普通 PE 文件）

Link /subsystem:console x.obj y.lib z.res（控制台文件）

Link /subsystem:windows /dll /def:a.def x.obj y.lib z.res（DLL 文件）

使用 Link 的时候，/subsystem 选项必须被指定，一般指定为 windows，当编写控制台程序的时候要改为 console，写 dll 的时候要用 /def 指定列表定义文件，同时要指定 /dll 选项。其他的一些参数如 /stub，/section 和 /base 等只在编写特殊用途的程序时才使用。

Link 的常用选项如下：

/BASE: 地址，指定程序装入内存的基地址，一般 PE 文件默认的装入地址是 0x400000 处，dll 文件装入 0x10000000，用此选项可以修改这个默认值。

/COMMENT: 注释，在 PE 文件的文件头后面加上文本注释，想在可执行文件中加入版权字符串可以用这个办法，如果字符串中包括空格，那么要在头尾加双引号。

/DEBUG，在 PE 文件中加入调试信息。

/DEBUGTYPE: 类型，加入的调试信息的类型，可以是 CV 或 COFF。

/DRIVER: 类型，链接 Windows NT 的 WDM 驱动程序时用，类型可以是 WDM 或者 UPONLY。

/DLL，链接动态链接库文件时用。

/DEF: 文件名，编写链接库文件时使用的 def 文件名，用来指定要导出的函数列表。

/ENTRY: 标号，指定入口标号。

/IMPLIB: 文件名，当链接有导出函数的文件时（如 DLL）要建立的导入库名。

/INCREMENTAL:ON|OFF，是否增量链接，增量链接只重写可执行文件自上次链接后改动的部分，所以可以增加链接速度，但会增加文件长度。

/LIBPATH: 路径，指定库文件的目录。

/MACHINE: 平台名称，指定输出的可执行程序运行平台，可以是 ALPHA、ARM、IX86、MIPS、MIPS16、MIPSR41XX、PPC、SH3 和 SH4 等。

/MAP: 文件名，生成 MAP 文件。

/OUT: 文件名，指定输出文件名，默认的扩展名是 .exe，如果要生成其它文件名，如屏幕保护*.scr 等，则在这里指定。

/RELEASE，填写文件头中的校验字段。

/SECTION: 节区及其属性，改变节区的属性，如 exe 文件中代码节区的属性通常是不

可写的，用户也可以在这里将它设置为可写，属性可以是 E、R、W、S、D、K、L、P 和 X 等。

/STACK: 尺寸，设定堆栈尺寸。

/STUB: 文件名，这是一个有趣的参数，Win32 文件有个简单的 DOS 文件头，以便在 DOS 下执行时显示“必须在 Windows 下执行”一类的消息，这部分称为 DOS STUB。如果用户在这里指定了一个 DOS 可执行文件，例如 DOS 的 FDISK.EXE，那么程序在 Windows 下运行的会是用户编写的代码，但在 DOS 下运行的就是 FDISK.EXE，而不出现“”。

/SUBSYSTEM: 系统名，指定程序运行的操作系统，可以是 NATIVE，WINDOWS，CONSOLE，WINDOWSCE 和 POSIX 等。

/VXD，编写 Windows 95 VxD 驱动程序时指定。

7.4.5 建立更好的环境

在前面执行编译、链接等命令时，为了找到相应的处理程序，都在处理程序前面指定了路径，为了找到链接的库函数，使用了 /LIBPATH 参数。一种更简单的方法是建立程序运行环境。

建立一个批处理文件 VAR.BAT，内容如图 7.16 所示。

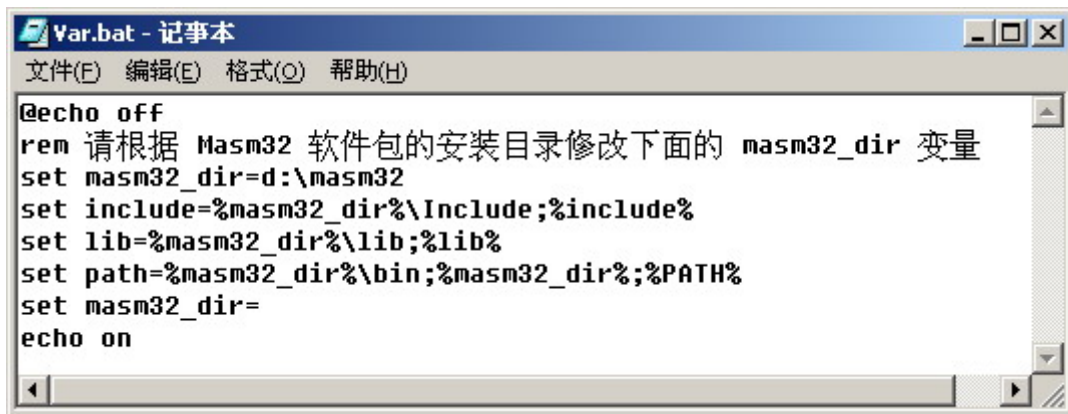


图 7.16 建立运行环境的批处理文件

在 VAR.BAT 中，将 masm32 及其子目录 bin 都放到了 path 中。同时也建立了 include、lib 变量。masm32_dir 只是一个中间变量。读者可以根据 masm32 的安装目录，仅修改 masm32_dir 右边的路径即可。在变量前后加百分号，如 %masm32_dir%，就是用该变量的值 (d:\masm32) 代替变量。

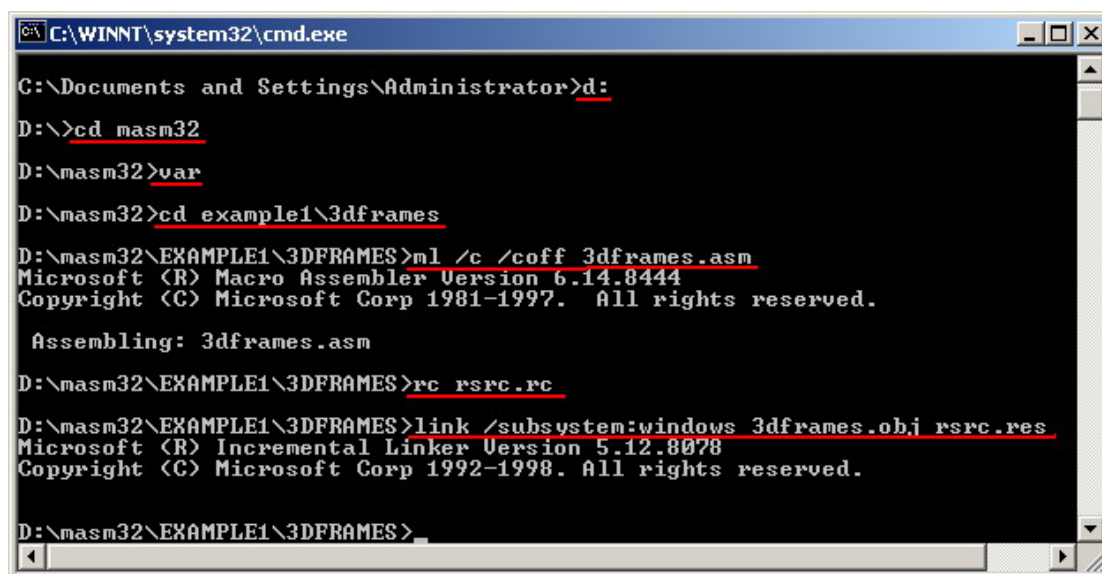
在 path 设置中，在已有的 path（即 %PATH%）前面添加了新的路径部分。

将 VAR.BAT 存放到 d:\masm32 目录下。每当我们打开一个 cmd 窗口时，都先执行该批处理文件建立环境。这与设置“系统 / 高级 / 环境变量”不同。设置系统环境变量只设置一次，即使关机后再开机，依然有效。

下面是使用 VAR.BAT 的一个操作实例：

```
C:\Documents and Settings\Administrator>d:↵  
D:\>cd masm32↵  
D:\masm32>var↵  
D:\masm32>cd example1\3dframes↵  
D:\masm32\example1\3dframes>ml /c /coff 3dframes.asm↵  
D:\masm32\example1\3dframes>rc rsrc.rc↵  
D:\masm32\example1\3dframes>link /subsystems:windows 3dframes.obj rsrc.res↵
```

图 7.17 给出了各操作执行后的显示信息。



```
C:\WINNT\system32\cmd.exe
C:\Documents and Settings\Administrator>d:
D:\>cd masm32
D:\masm32>var
D:\masm32>cd example1\3dframes
D:\masm32\EXAMPLE1\3DFRAMES>nl /c /coff 3dframes.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: 3dframes.asm
D:\masm32\EXAMPLE1\3DFRAMES>rc rsrc.rc
D:\masm32\EXAMPLE1\3DFRAMES>link /subsystem:windows 3dframes.obj rsrc.res
Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

D:\masm32\EXAMPLE1\3DFRAMES>
```

图 7.17 简化的操作命令

在上述操作中，无需指定编译器、链接器的路径，也未指定链接的函数库的路径。

7.5 nmake 工具

7.5.1 首次使用 make 工具

从 7.4 节可以看到，在命令窗口(cmd)下，即使设置了环境变量，逐行手工输入命令依然是一件麻烦事，编译器和链接器有多个参数选项，链接时还有多个文件名，因此很容易出现错误。我们也可以像 7.3 节介绍的批处理文件一样，编写一个批处理文件，但会遇到参数太多太乱、每个项目都不一样的情况。一种简单的解决办法就是为每个编程项目单独建立一个批处理，每次改动后，运行批处理把所有模块重新编译一次。不过当程序很庞大的时候，这将花费很长时间。有一种比较好的方案是使用 **nmake** 工具。

nmake 工具可以看成是一个智能的批处理工具，它本身并没有编译和链接的功能，只是用类似于批处理的方式，调用用户指定的语句来进行编译和链接。和批处理不同的是，**nmake** 工具会根据目标文件上一次编译的时间和所依赖的源文件的更新时间自动判断应当编译哪些源文件，对没有更新过的文件不会处理，这样就可以提高效率，而批处理会执行全部命令将全部源文件编译，包括那些不必重新编译的源文件。

使用 **nmake** 工具，需要 **NMAKE.EXE**，可以在 VC++ 的安装目录下找到 **NMAKE.EXE** 文件；还需要一个描述文件，在光盘上有多个描述文件的例子。我们在 7.5.2 节介绍描述文件的语法。

将 **NMAKE.EXE** 拷贝到 D:\masm32 目录下。将光盘 chapter7\3dframes 下的 makefile 拷贝到 d:\masm32\example1\3dframes 下（在 7.5.2 节给出了该文件的内容）。

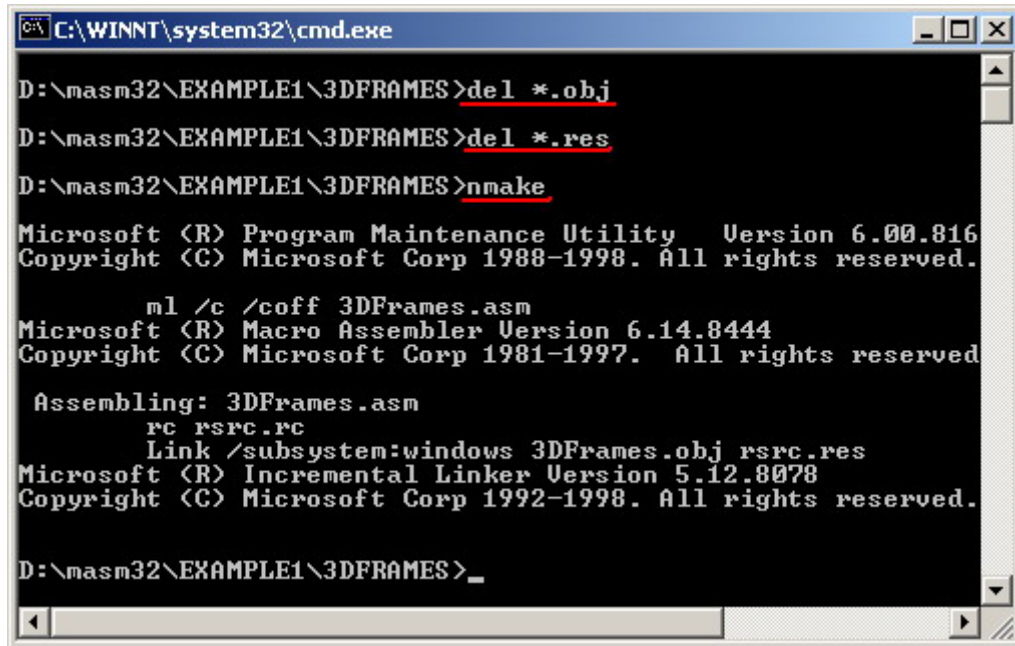
依次执行如下命令：

```
D:\masm32>var↵
D:\masm32>cd example1\3dframes↵
D:\masm32\example1\3dframes>nmake↵
```

可以看到编译等提示信息。如果“3DFRAMES.EXE”已存在，而我们又未修改过任何源文

件与资源文件，则会提示“3dframes.exe” is up-to-date。删除相应的目标文件或者资源文件(RES)，再次执行 nmake，就可以看到相应的提示信息。

图 7.18 给出了 nmake 的操作及提示。



```
C:\WINNT\system32\cmd.exe

D:\masm32\EXAMPLE1\3DFRAMES>del *.obj
D:\masm32\EXAMPLE1\3DFRAMES>del *.res
D:\masm32\EXAMPLE1\3DFRAMES>nmake

Microsoft (R) Program Maintenance Utility   Version 6.00.816
Copyright (C) Microsoft Corp 1988-1998. All rights reserved.

        ml /c /coff 3DFrames.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved

    Assembling: 3DFrames.asm
           rc rsrc.rc
           Link /subsystem:windows 3DFrames.obj rsrc.res
Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

D:\masm32\EXAMPLE1\3DFRAMES>_
```

图 7.18 nmake 的运行

注意，在 Makefile 中并未指明在何处找 ML.EXE, LINK.EXE 等，因此需要在执行 nmake 前设置环境，简单地运行 7.4.5 节的 VAR.BAT 即可。

7.5.2 描述文件的语法

描述文件是一个文本文件，可以用文本编辑器进行编辑。描述文件中包含的内容有：

- (1) 编译汇编源文件的方法
- (2) 编译资源文件的方法
- (3) 链接生成可执行文件的方法
- (4) 文件之间的依赖关系描述

为了简化描述文件，在文件中可以使用一些宏定义。下面是一个 chapter7\3dframes 中 makefile 的具体内容。

```
# 定义变量（也称为宏）
NAME = 3DFrames
OBJS = $(NAME).obj
RES = rsrc.res    # 一般是$(NAME).res，此处固定成了 rsrc.res
LINK_FLAG = /subsystem:windows
ML_FLAG = /c /coff
# 定义依赖关系和执行命令
$(NAME).exe: $(OBJS) $(RES)
           Link $(LINK_FLAG) $(OBJS) $(RES)
# 定义汇编编译和资源编译的默认规则
.asm.obj:
      ml $(ML_FLAG) $<
```

```
.rc.res:
    rc $<
# 清除临时文件
clean:
    del *.obj
    del *.res
```

描述文件一般包含注释、宏定义、显示规则、隐含规则等。

(1) 注释

在描述文件中，以#号开头一直到行尾的字符为注释。`nmake` 工具完全忽略#号及全部注释字符。这些内容仅帮助我们阅读。

当一行的内容过长的时候，可以用换行符来继续，描述文件的换行符是“\”，在“\”后面不能再加上其他字符，包括注释和空格，也即“\”应是一行的最后字符，否则 `nmake` 就不会把它当成换行符解释，就会出现错误。

(2) 宏定义

描述文件中可以使用宏（也称为变量）来指代源文件及其相关编译信息。

宏定义的语法如下：

变量名=变量内容

例如，在上面的描述文件中，定义了宏 `NAME`、`OBJS`、`RES`、`LINK_FLAG`、`ML_FLAG`。

(3) 宏引用

宏引用也非常简单，引用宏时只需在变量前加\$符号。但是要注意的是，如果变量名的长度超过一个字符，在引用时必须加圆括号“()”。

在上面的描述文件中，有如下引用：`$(NAME)`、`$(OBJS)`、`$(RES)`、`$(LINK_FLAG)`、`$(ML_FLAG)`。

假设我们定义了一个宏 `A`，则使用 `$A`、`$(A)` 均可。

宏定义的使用可以使描述文件的编写更简单、更易于修改。例如我们要更改 `ML` 的选项部分，只需要修改 `ML_FLAG` 的定义，不必重新阅读整个描述文件；另外，当不止一个地方用到同一个文件的时候，把文件名定义为宏，可减少错误，增加可读性。

(4) 显式规则

描述文件中包含有一些规则，这些规则定义了文件之间的依赖关系和产生命令，一个规则的格式如下：

目标文件：依赖文件
命令

例如：`$(NAME).exe: $(OBJS) $(RES)`

`Link $(LINK_FLAG) $(OBJS) $(RES)`

将其中的宏定义扩展，实际上它等同于

`3DFrames.exe: 3DFrames.obj rsrc.res`

`Link /subsystem:windows 3DFrames.obj rsrc.res`

其中目标文件是 `3DFrames.exe`，它依赖于 `3DFrames.obj` 和 `rsrc.res`，产生目标文件的命令就是下面的 `Link`。

注意：命令不能在一行的顶头，前面至少要一个空格。

(5) 隐含规则

隐含规则用于指出某一类文件的建立方法，即规定了带特定扩展名的文件转换成另一种扩展名的文件的方法。隐含规则格式如下：

.源扩展名.目标扩展名：

命令

例如: .asm.obj:

```
ml $(ML_FLAG) $<
```

源扩展名为“.ASM”，目标扩展名为“.OBJ”，“\$<”，表示当前目录下的所有源文件，即所有的“.ASM”为扩展名的文件。

隐含规则不能有依赖文件，无法指定确定的输入文件名，它采用了几个特殊的内定义宏来指带某一类的文件。

隐含规则中的内定义宏有：\$<、\$?、\$@、\$*，含义如下：

\$< 源文件名（只能用在隐含规则中）

\$? 所有源文件名

\$@ 全路径的目标文件。

\$* 除去扩展名的全路径的目标文件。

在一个描述文件中，有多条显式规则和隐含规则。**nmake** 默认将描述文件中的第一条规则的目标文件认为是最终文件（例如，执行“**nmake**”）。如果不想使用此默认规则，则可以给定目标。例如，使用命令“**nmake 3DFrames.obj**”，就只会生成 3DFrames.obj，而不会生成 rsrc.res 和 3DFrames.exe。“**nmake rsrc.res**”就只会生成 rsrc.res。执行“**nmake clean**”，就会删除当前目录下的 OBJ 文件和 RES 文件。

当用户要求 **nmake** 去建立一个目标（默认方式或者指定目标方式）时，**nmake** 就会去寻找该目标的依赖规则。在找到相应的规则后，其下的命令并不会立即执行。**nmake** 会先去检查该目标的依赖文件是否已构建好，只有在各依赖文件都构建好的情况下，才会执行这一条规则。依赖文件的构建过程和前面的最终目标的构建过程是一样的，因此，可将链接结构想象成一棵树，树中的节点为目标，树根是最终的目标。一个目标（文件）和其依赖目标（文件）构成父子关系。

注意 **nmake** 并不一定会执行规则中的命令，它检查了目标文件的依赖文件是否比目标文件新，如果新，才执行命令，否则就是 **up-to-date**，

目标可以没有依赖文件，也可以不是一个真正存在的文件。例如，前面例子中的 **clean** 是一个目标，但这里并不是要生成一个 **clean** 文件。这种目标也称为伪目标，不存在的目标总是被当成一个过时的目标，其下的规则一定会执行。

注意 如果只执行 **nmake**，则不会删除目标文件和 RES 文件，因为在构造相应的链接结构树时，**clean** 不会出现在树中，整个描述文件中没有调用过 **clean** 规则。

如果我们要在执行 **nmake** 后，调用 **clean**，则可以将如下规则作为第一条规则：

```
all: $(EXE) clean
```

该规则中目标 **all** 是一个伪目标，该规则也无实际的命令，仅指明依赖关系而已。

7.5.3 nmake 的高级用法

在命令行键入 **nmake /?** 可以显示帮助信息，内容如下：

Usage: NMAKE @commandfile

NMAKE [options] [/f makefile] [/x stderrfile] [macrodefs] [targets]

Options:

/A Build all evaluated targets	不检测文件时间，强制更新所有文件
/B Build if time stamps are equal	文件时间相等时也要更新文件
/C Suppress output messages	不输出信息
/D Display build information	显示文件新旧信息

/E Override env-var macros
/HELP Display brief usage message
/I Ignore exit codes from commands
/K Build unrelated targets on error
/N Display commands but do not execute 要执行的命令，但并不真正执行
/NOLOGO Suppress copyright message
/P Display NMAKE information 显示详细的信息
/Q Check time stamps but do not build
/R Ignore predefined rules/macros
/S Suppress executed-commands display
/T Change time stamps but do not build
/U Dump inline files
/Y Disable batch-mode
/? Display brief usage message

nmake 的语法为：nmake [选项] [/f 描述文件名] [/x 输出信息文件名] [宏定义] [目标]
说明如下：

/f: 指定描述文件名称，如果该项不出现，则使用默认的描述文件，即 MAKEFILE；

/x: 如果想把屏幕输出的信息存到一个文件中，可以用/x 参数指定。

宏定义：可以用新的定义覆盖描述文件中的宏定义。

例如，使用命令 nmake ML_FLAG="/c /coff /Zi"，就会以/c /coff /Zi 定义代替 makefile 中原来的 ML_FLAG 定义 "/c /coff"。

注意 一是宏名称要区分大小写，ML_FLAG 和 ml_flag 是不一样的；二是定义值中有空格的时候要用双引号引起来，没有空格时可以不用双引号，如 ML_FLAG=/c。执行 nmake 时采用宏定义，可以临时性的更改编译、链接等参数，而不必修改描述文件。

目标：指定建立描述文件中描述的某个文件。

例如，执行 nmake rsrc.res，就只对 rsrc.rc 进行编译，生成 rsrc.res。

有兴趣的读者可以在 MSDN(Microsoft Developer Network)中找到 nmake 的帮助信息。在第 8 章的例子中，我们都将使用描述文件 makefile，并用 nmake 工具来进行处理。

习题七：

7.1 采用 QEDITOR 编译、链接 MASM32 软件包中的 EXAMPLE1\COMCTLS 下的程序，并运行该程序，指出程序的功能。

7.2 采用 QEDITOR 编译、链接 MASM32 软件包中的 EXAMPLE1\MINIFILE 下的程序，并运行该程序，指出程序的功能。

7.3 对 EXAMPLE1\COMCTLS 下的程序，编写一个描述文件，利用 nmake 生成可执行程序。

7.4 对 EXAMPLE1\MINIFILE 下的程序，编写一个描述文件，利用 nmake 生成可执行程序。