第二十一章　动态连结程式库

动态连结程式库（也称为 DLL）是 Microsoft Windows 最重要的组成要素之一。大多数与 Windows 相关的磁碟档案如果不是程式模组，就是动态连结程式。迄今为止，我们都是在开发 Windows 应用程式；现在是尝试编写动态连结程式库的时候了。许多您已经学会的编写应用程式的规则同样适用於编写这些动态连结程式库模组，但也有一些重要的不同。

动态连结程式库的基本知识

正如前面所看到的，Windows 应用程式是一个可执行档案，它通常建立一个或几个视窗，并使用讯息回圈接收使用者输入。通常，动态连结程式库并不能直接执行，也不接收讯息。它们是一些独立的档案，其中包含能被程式或其他 DLL 呼叫来完成一定作业的函式。只有在其他模组呼叫动态连结程式库中的函式时，它才发挥作用。

所谓「动态连结」，是指 Windows 把一个模组中的函式呼叫连结到动态连结程式库模组中的实际函式上的程序。在程式开发中，您将各种目的模组(.OBJ)、执行时期程式库(.LIB)档案，以及经常是已编译的资源(.RES)档案连结在一起，以便建立 Windows 的.EXE 档案，这时的连结是「静态连结」。动态连结与此不同，它发生在执行时期。

KERNEL32.DLL、USER32.DLL 和 GDI32.DLL、各种驱动程式档案如 KEYBOARD.DRV、SYSTEM.DRV 和 MOUSE.DRV 和视讯及印表机驱动程式都是动态连结程式库。这些动态连结程式库能被所有 Windows 应用程式使用。

有些动态连结程式库（如字体档案等）被称为「纯资源」。它们只包含资料（通常是资源的形式）而不包含程式码。由此可见，动态连结程式库的目的之一就是提供能被许多不同的应用程式所使用的函式和资源。在一般的作业系统中，只有作业系统本身才包含其他应用程式能够呼叫来完成某一作业的常式。在 Windows 中，一个模组呼叫另一个模组函式的程序被推广了。结果使得编写一个动态连结程式库，也就是在扩充 Windows。当然，也可认为动态连结程式库（包括构成 Windows 的那些动态连结程式库常式）是对使用者程式的扩充。

尽管一个动态连结程式库模组可能有其他副档名（如.EXE 或.FON），但标准副档名是.DLL。只有带.DLL 副档名的动态连结程式库才能被 Windows 自动载入。如果档案有其他副档名，则程式必须另外使用 LoadLibrary 或者 LoadLibraryEx 函式载入该模组。

您通常会发现，动态连结程式库在大型应用程式中最有意义。例如，假设要为 Windows 编写一个由几个不同的程式组成的大型财务套装软体，就会发现这些应用程式会使用许多共同的常式。可以把这些公共常式放入一个一般性的目的码程式库（带.LIB 副档名）中，并在使用 LINK 静态连结时把它们加入各程式模组中。但这种方法是很浪费的，因为套装软体中的每个程式都包含与公共常式相同的程式码。而且，如果修改了程式库中的某个常式，就要重新连结使用此常式的所有程式。然而，如果把这些公共常式放到称为 ACCOUNT.DLL 的动态连结程式库中，就可解决这两个问题。只有动态连结程式库模组才包含所有程式都要用到的常式。这样能为储存档案节省磁碟空间，并且在同时执行多个应用程式时节省记忆体，而且，可以修改动态连结程式库模组而不用重新连结各个程式。

动态连结程式库实际上是可以独立存在的。例如，假设您编写了一系列 3D 绘图常式，并把它们放入名为 GDI3.DLL 的 DLL 中。如果其他软体发展者对此程式库很感兴趣，您就可以授权他们将其加入他们的图形程式中。使用多个这样的图形程式的使用者只需要一个 GDI3.DLL 档案。

程式库：一词多义

动态连结程式库有著令人困惑的印象，部分原因是由於「程式库」这个词被放在几种不同的用语之後。除了动态连结程式库之外，我们也用它来称呼「目的码程式库」或「引用程式库」。

目的码程式库是带.LIB 副档名的档案。在使用连结程式进行静态连结时，它的程式码就会加到程式的.EXE 档案中。例如，在 Microsoft Visual C++中，连同程式连结的一般 C 执行目的码程式库被称为 LIBC.LIB。

引用程式库是目的码程式库档案的一种特殊形式。像目的码程式库一样，引用程式库有.LIB 副档名，并且被连结器用来确定程式码中的函式呼叫来源。但引用程式库不含程式码，而是为连结程式提供资讯，以便在.EXE 档案中建立动态连结时要用到的重定位表。包含在 Microsoft 编译器中的 KERNEL32.LIB、USER32.LIB 和 GDI32.LIB 档案是 Windows 函式的引用程式库。如果一个程式呼叫 Rectangle 函式，Rectangle 将告诉 LINK，该函式在 GDI32.DLL 动态连结程式库中。该资讯被记录在.EXE 档案中，使得程式执行时，Windows 能够和 GDI32.DLL 动态连结程式库进行动态连结。

目的码程式库和引用程式库只用在程式开发期间使用，而动态连结程式库在执行期间使用。当一个使用动态连结程式库的程式执行时，该动态连结程式库必须在磁片上。当 Windows 要执行一个使用了动态连结程式库的程式而需要

载入该程式库时，动态连结程式库档案必须储存在含有该.EXE 程式的目录下、目前的目录下、Windows 系统目录下、Windows 目录下，或者是在通过 MS-DOS 环境中的 PATH 可以存取到的目录下（Windows 会按顺序搜索这些目录）。

一个简单的 DLL

虽然动态连结程式库的整体概念是它们可以被多个应用程式所使用，但您通常最初设计的动态连结程式库只与一个应用程式相联系，可能是一个「测试」程式在使用 DLL。

下面就是我们要做的。我们建立一个名为 EDRLIB.DLL 的 DLL。档案名中的「EDR」代表「简便的绘图常式（easy drawing routines）」。这里的 EDRLIB 只含有一个函式（名称为 EdrCenterText），但是您还可以将应用程式中其他简单的绘图函式添加进去。应用程式 EDRTEST.EXE 将通过呼叫 EDRLIB.DLL 中的函式来利用它。

要做到这一点，需要与我们以前所做的略有不同的方法，也包括 Visual C++ 中我们没有看过的特性。在 Visual C++ 中「工作空间（workspaces）」和「专案（projects）」不同。专案通常与建立的应用程式（.EXE）或者动态连结程式库（.DLL）相联系。一个工作空间可以包含一个或多个专案。迄今为止，我们所有的工作空间都只包含一个专案。我们现在就建立一个包含两个专案的工作空间EDRTEST——一个用於建立EDRTEST.EXE,而另一个用於建立EDRLIB.DLL，即 EDRTEST 使用的动态连结程式库。

现在就开始。在 Visual C++中，从「File」功能表选择「New」，然後选择「Workspaces」页面标签。（我们以前从来没有选择过。）在「Location」栏选择工作空间要储存的目录，然後在「Workspace Name」栏输入「EDRTEST」，按 Enter 键。

这样就建立了一个空的工作空间。Developer Studio 还建立了一个名为 EDRTEST 的子目录，以及工作空间档案 EDRTEST.DSW（就像两个其他档案）。

现在让我们在此工作空间里建立一个专案。从「File」功能表选择「New」，然後选择「Projects」页面标签。尽管过去您选择「Win32 Application」，但现在「Win32 Dynamic-Link Library」。另外，单击单选按钮「Add To Current Workspace」，这使得此专案是「EDRTEST」 工作空间的一部分。在「Project Name 栏输入 EDRLIB，但先不要按「OK」按钮。当您在 Project Name 栏输入 EDRLIB 时，Visual C++将改变「Location」栏，以显示 EDRLIB 作为 EDRTEST 的一个子目录。这不是我们要的，所以接著在「Location」栏删除 EDRLIB 子目录以便专案建立在 EDRTEST 目录。现在按「OK」。萤幕将显示一个对话方块，询问您建

立什么型态的 DLL。选择「An Empty DLL Project」，然後按「Finish」。Visual C++将建立一个专案档案 EDRLIB.DSP 和一个构造档案 EDRLIB.MAK（如果「Tools Options」对话方块的 B「uild 页面标签中选择了「Export Makefile」选项」。

现在您已经在此专案中添加了一对档案。从「File」功能表选择「New」，然後选择「Files」页面标签。选择「C/C++ Header File」，然後输入档案名 EDRLIB.H。输入程式 21-1 所示的档案（或者从本书光碟中复制）。再次从「File」功能表中选择「New」，然後选择「Files」页面标签。这次选择「C++ Source File」，然後输入档案名 EDRLIB.C。继续输入程式 21-1 所示的程式。

程式 21-1　EDRLIB 动态连结程式库

```
EDRLIB.H
/*--------------------------------------------------------------------------
      EDRLIB.H header file
--------------------------------------------------------------------------
*/

#ifdef      __cplusplus
#define     EXPORT extern "C" __declspec (dllexport)
#else
#define     EXPORT __declspec (dllexport)
#endif

EXPORT      BOOL CALLBACK EdrCenterTextA (HDC, PRECT, PCSTR) ;
EXPORT      BOOL CALLBACK EdrCenterTextW (HDC, PRECT, PCWSTR) ;

#ifdef      UNICODE
#define     EdrCenterText EdrCenterTextW
#else
#define     EdrCenterText EdrCenterTextA
#endif
EDRLIB.C
/*--------------------------------------------------------------------------
-
      EDRLIB.C -- Easy Drawing Routine Library module
                                                (c) Charles Petzold, 1998
--------------------------------------------------------------------------
-*/
#include  windows.h>
#include "edrlib.h"

int WINAPI DllMain (   HINSTANCE hInstance, DWORD fdwReason, PVOID pvReserved)
{
      return TRUE ;
}

EXPORT BOOL CALLBACK EdrCenterTextA (   HDC hdc, PRECT prc, PCSTR pString)
```

```
{
      int  iLength ;
      SIZE size ;

      iLength = lstrlenA (pString) ;
      GetTextExtentPoint32A (hdc, pString, iLength, &size) ;
      return TextOutA (hdc,( prc->right - prc->left - size.cx) / 2,
                        (    prc->bottom - prc->top - size.cy) / 2,
                                    pString, iLength) ;
}

EXPORT BOOL CALLBACK EdrCenterTextW (HDC hdc, PRECT prc, PCWSTR pString)
{
      int  iLength ;
      SIZE size ;

      iLength = lstrlenW (pString) ;
      GetTextExtentPoint32W (hdc, pString, iLength, &size) ;
      return TextOutW (hdc,  (     prc->right - prc->left - size.cx) / 2,
                             (     prc->bottom - prc->top - size.cy) / 2,
                                         pString, iLength) ;
}
```

这里您可以按 Release 设定，或者也可以按 Debug 设定来建立 EDRLIB.DLL。之後，RELEASE 和 DEBUG 目录将包含 EDRLIB.LIB（即动态连结程式库的引用程式库）和 EDRLIB.DLL（动态连结程式库本身）。

纵观全书，我们建立的所有程式都可以根据 UNICODE 识别字来编译成使用 Unicode 或非 Unicode 字串的程式码。当您建立一个 DLL 时，它应该包括处理字元和字串的 Unicode 和非 Unicode 版的所有函式。因此，EDRLIB.C 就包含函式 EdrCenterTextA（ANSI 版）和 EdrCenterTextW（宽字元版）。EdrCenterTextA 定义为带有参数 PCSTR（指向 const 字串的指标），而 EdrCenterTextW 则定义为带有参数 PCWSTR（指向 const 宽字串的指标）。EdrCenterTextA 函式将呼叫 lstrlenA、GetTextExtentPoint32A 和 TextOutA。EdrCenterTextW 将呼叫 lstrlenW、GetTextExtentPoint32W 和 TextOutW。如果定义了 UNICODE 识别字，则 EDRLIB.H 将 EdrCenterText 定义为 EdrCenterTextW，否则定义为 EdrCenterTextA。这样的做法很像 Windows 表头档案。

EDRLIB.H 也包含函式 DllMain，取代了 DLL 中的 WinMain。此函式用於执行初始化和未初始化（deinitialization），我将在下一节讨论。我们现在所需要的就是从 DllMain 传回 TRUE。

在这两个档案中，最後一点神秘之处就是定义了 EXPORT 识别字。DLL 中应用程式使用的函式必须是「输出（exported）」的。这跟税务或者商业制度无

关，只是确保函式名添加到 EDRLIB.LIB 的一个关键字（以便连结程式在连结使用此函式的应用程式时，能够解析出函式名称），而且该函式在 EDRLIB.DLL 中也是看得到的。EXPORT 识别字包括储存方式限定词__declspec（dllexport）以及在表头档案按 C++模式编译时附加的「C」。这将防止编译器使用 C++的名称轧压规则（name mangling）来处理函式名称，使 C 和 C++程式都能使用这个 DLL。

程式库入口／出口点

当动态连结程式库首次启动和结束时，我们呼叫了 DllMain 函式。DllMain 的第一个参数是程式库的执行实体代号。如果您的程式库使用需要执行实体代号（诸如 DialogBox）的资源，那么您应该将 hInstance 储存为一个整体变数。DllMain 的最後一个参数由系统保留。

fdwReason 参数可以是四个值之一，说明为什么 Windows 要呼叫 DllMain 函式。在下面的讨论中，请记住一个程式可以被载入多次，并在 Windows 下一起执行。每当一个程式载入时，它都被认为是一个独立的程序（process）。

fdwReason 的一个值 DLL_PROCESS_ATTACH 表示动态连结程式库被映射到一个程序的位址空间。程式库可以根据这个线索进行初始化，为以後来自该程序的请求提供服务。例如，这类初始化可能包括记忆体配置。在一个程序的生命周期内，只有一次对 DllMain 的呼叫以 DLL_PROCESS_ATTACH 为参数。使用同一 DLL 的其他任何程序都将导致另一个使用 DLL_PROCESS_ATTACH 参数的 DllMain 呼叫，但这是对新程序的呼叫。

如果初始化成功，DllMain 应该传回一个非 0 值。传回 0 将导致 Windows 不执行该程式。

当 fdwReason 的值为 DLL_PROCESS_DETACH 时，意味著程序不再需要 DLL 了，从而提供给程式库自己清除自己的机会。在 32 位元的 Windows 下，这种处理并不是严格必须的，但这是一种良好的程式写作习惯。

类似地，当以 DLL_THREAD_ATTACH 为 fdwReason 参数呼叫 DllMain 时，意味著某个程序建立了一个新的执行绪。当执行绪中止时，Windows 以 DLL_THREAD_DETACH 为 fdwReason 参数呼叫 DllMain。请注意，如果动态连结程式库是在执行绪被建立之後和一个程序连结的，那么可能会得到一个没有事先对应一个 DLL_THREAD_ATTACH 呼叫的 DLL_THREAD_DETACH 呼叫。

当使用一个 DLL_THREAD_DETACH 参数呼叫 DllMain 时，执行绪仍然存在。动态连结程式库甚至可以在这个程序期间发送执行绪讯息。但是它不应该使用 PostMessage，因为执行绪可能在此讯息被处理到之前就已经退出执行了。

测试程式

现在让我们在 EDRTEST 工作空间里建立第二个专案，程式名称为 EDRTEST，而且使用 EDRLIB.DLL。在 Visual C++中载入 EDRTEST 工作空间时，请从「File」功能表选择「New」，然後在「New」对话方块中选择「Projects」页面标签。这次选择「Win32 Application」，并确保选中了「Add To Current Workspace」按钮。输入专案名称 EDRTEST。再在「Locations」栏删除第二个 EDRTEST 子目录。按下「OK」，然後在下一个对话方块选择「An Empty Project」，按「Finish」。

从「File」功能表再次选择「New」。选择「Files」页面标签然後选择「C++ Source File」。确保「Add To Project」清单方块显示「EDRTEST」而不是「EDRLIB」。输入档案名称 EDRTEST.C，然後输入程式 21-2 所示的程式。此程式用 EdrCenterText 函式将显示区域中的字串居中对齐。

程式 21-2 EDRTEST

```
EDRTEST.C
/*-------------------------------------------------------------------------
-
      EDRTEST.C -- Program using EDRLIB dynamic-link library
                                                (c) Charles Petzold, 1998
-------------------------------------------------------------------------
*/

#include <windows.h>
#include "edrlib.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                    PSTR    szCmdLine,    int
iCmdShow)
{
      static TCHAR     szAppName[] = TEXT ("StrProg") ;
      HWND                              hwnd ;
      MSG                         msg ;
      WNDCLASS                    wndclass ;

      wndclass.style              = CS_HREDRAW | CS_VREDRAW ;
      wndclass.lpfnWndProc        = WndProc ;
      wndclass.cbClsExtra         = 0 ;
      wndclass.cbWndExtra         = 0 ;
      wndclass.hInstance          = hInstance ;
      wndclass.hIcon              = LoadIcon (NULL, IDI_APPLICATION) ;
      wndclass.hCursor            = LoadCursor (NULL, IDC_ARROW) ;
      wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
      wndclass.lpszMenuName  = NULL ;
```

```
     wndclass.lpszClassName = szAppName ;

     if (!RegisterClass (&wndclass))
     {
          MessageBox (     NULL, TEXT ("This program requires Windows NT!"),
                                                  szAppName, MB_ICONERROR) ;
               return 0 ;
    }

     hwnd = CreateWindow (szAppName, TEXT ("DLL Demonstration Program"),
                  WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT, CW_USEDEFAULT,
                  CW_USEDEFAULT, CW_USEDEFAULT,
                  NULL, NULL, hInstance, NULL) ;

     ShowWindow (hwnd, iCmdShow) ;
     UpdateWindow (hwnd) ;

     while (GetMessage (&msg, NULL, 0, 0))
     {
               TranslateMessage (&msg) ;
               DispatchMessage (&msg) ;
     }
     return msg.wParam ;
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
     HDC              hdc ;
     PAINTSTRUCT           ps ;
     RECT             rect ;

     switch (message)
     {
     case  WM_PAINT:
                  hdc = BeginPaint (hwnd, &ps) ;

                  GetClientRect (hwnd, &rect) ;

                  EdrCenterText (  hdc, &rect,
               TEXT ("This string was displayed by a DLL")) ;

                  EndPaint (hwnd, &ps) ;
                  return 0 ;

     case  WM_DESTROY:
                  PostQuitMessage (0) ;
```

```
                    return 0 ;
      }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

注意，为了定义 EdrCenterText 函式，EDRTEST.C 包括 EDRLIB.H 表头档案，此函式将在 WM_PAINT 讯息处理期间呼叫。

在编译此程式之前，您可能希望做以下几件事。首先，在「Project」功能表选择「Select Active Project」。这时您将看到「EDRLIB」和「EDRTEST」，选择「EDRTEST」。在重新编译此工作空间时，您真正要重新编译的是程式。另外，在「Project」功能表中，选择「Dependencies」，在「Select Project To Modify」清单方块中选择「EDRTEST」。在「Dependent On The Following Project(s)」列表选中「EDRLIB」。此操作的意思是：EDRTEST 需要 EDRLIB 动态连结程式库。以後每次重新编译 EDRTEST 时，如果必要的话，都将在编译和连结 EDRTEST 之前重新重新编译 EDRLIB。

从「Project」功能表选择「Settings」，单击「General」标签。当您在左边的窗格中选择「EDRLIB」或者「EDRTEST」专案时，如果设定为「Win32 Release」，则显示在右边窗格中的「Intermediate Files」和「Output Files」将位於 RELEASE 目录；如果设定为「Win32 Debug」，则位於 DEBUG 目录。如果不是，请按此修改。这样可确保 EDRLIB.DLL 与 EDRTEST.EXE 在同一个目录中，而且程式在使用 DLL 时也不会产生问题。

在「Project Setting」对话方块中依然选中「EDRTEST」，单击「C/C++」页面标签。按本书的惯例，在「Preprocessor Definitions」中，将「UNICODE」添加到 Debug 设定。

现在您就可以在「Debug」或「Release」设定中重新编译 EDRTEST.EXE 了。必要时，Visual C++将首先编译和连结 EDRLIB。RELEASE 和 DEBUG 目录都包含 EDRLIB.LIB（引用程式库）和 EDRLIB.DLL。当 Developer Studio 连结 EDRTEST 时，将自动包含引用程式库。

了解 EDRTEST.EXE 档案中不包含 EdrCenterText 程式码很重要。事实上，要证明执行了 EDRLIB.DLL 档案和 EdrCenterText 函式很简单:执行 EDRTEST.EXE 需要 EDRLIB.DLL。

执行 EDRTEST.EXE 时，Windows 按外部程式库模组执行固定的函式。其中许多函式都在一般 Windows 动态连结程式库中。但 Windows 也看到程式从 EDRLIB 呼叫了函式，因此 Windows 将 EDRLIB.DLL 档案载入到记忆体，然後呼叫 EDRLIB 的初始化常式。EDRTEST 呼叫 EdrCenterText 函式是动态连结到 EDRLIB 中函式的。

在 EDRTEST.C 原始码档案中包含 EDRLIB.H 与包含 WINDOWS.H 类似。连结 EDRLIB.LIB 与连结 Windows 引用程式库（例如 USER32.LIB）类似。当您的程式执行时，它连结 EDLIB.DLL 的方式与连结 USER32.DLL 的方式相同。恭喜您！您已经扩展了 Windows 的功能！

在继续之前，我还要对动态连结程式库多说明一些：

首先，虽然我们将 DLL 作为 Windows 的延伸，但它也是您的应用程式的延伸。DLL 所完成的每件工作对於应用程式来说都是应用程式所交代要完成的。例如，应用程式拥有 DLL 配置的全部记忆体、DLL 建立的全部视窗以及 DLL 打开的所有档案。多个应用程式可以同时使用同一个 DLL，但在 Windows 下，这些应用程式不会相互影响。

多个程序能够共用一个动态连结程式库中相同的程式码。但是，DLL 为每个程序所储存的资料都不同。每个程序都为 DLL 所使用的全部资料配置了自己的位址空间。我们将在下以节看到，共用记忆体需要额外的工作。

在 DLL 中共用记忆体

令人兴奋的是，Windows 能够将同时使用同一个动态连结程式库的应用程式分开。不过，有时却不太令人满意。您可能希望写一个 DLL，其中包含能够被不同应用程式或者同一个程式的不同常式所共用的记忆体。这包括使用共用记忆体。共用记忆体实际上是一种记忆体映射档案。

让我们测试一下，这项工作是如何在程式 STRPROG（「字串程式（string program）」）和动态连结程式库 STRLIB（「字串程式库（string library）」）中完成的。STRLIB 有三个输出函式被 STRPROG 呼叫，我们只对此感兴趣，STRLIB 中的一个函式使用了在 STRPROG 定义的 callback 函式。

STRLIB 是一个动态连结程式库模组，它储存并排序了最多 256 个字串。在 STRLIB 中，这些字串均为大写，并由共用记忆体维护。利用 STRLIB 的三个函式，STRPROG 能够添加字串、删除字串以及从 STRLIB 获得目前的所有字串。STRPROG 测试程式有两个功能表项（「Enter」和「Delete」），这两个功能表项将启动不同的对话方块来添加或删除字串。STRPROG 在其显示区域列出目前储存在 STRLIB 中的所有字串。

下面这个函式在 STRLIB 定义，它将一个字串添加到 STRLIB 的共用记忆体。

```
EXPORT BOOL CALLBACK AddString (pStringIn)
```

参数 pStringIn 是字串的指标。字串在 AddString 函式中变成大写。如果在 STRLIB 的列表中有一个相同的字串，那么此函式将添加一个字串的复本。如果成功，AddString 传回「TRUE」（非 0），否则传回「FALSE」（0）。如果字

串的长度为 0，或者不能配置储存字串的记忆体，或者已经储存了 256 个字串，则传回值将都是 FALSE。

STRLIB 函式从 STRLIB 的共用记忆体中删除一个字串。

```
EXPORT BOOL CALLBACK DeleteString (pStringIn)
```

另外，参数 pStringIn 是一个字串指标。如果有多个相同内容字串，则删除第一个。如果成功，那么 DeleteString 传回「TRUE」（非 0），否则传回「FALSE」（0）。传回「FALSE」表示字串长度为 0，或者找不到相同内容的字串。

STRLIB 函式使用了呼叫程式中的一个 callback 函式，以便列出目前储存在 STRLIB 共用记忆体中的字串：

```
EXPORT int CALLBACK GetStrings (pfnGetStrCallBack, pParam)
```

在呼叫程式中，callback 函式必须像下面这样定义：

```
EXPORT BOOL CALLBACK GetStrCallBack (PSTR pString, PVOID pParam)
```

GetStrings 的参数 pfnGetStrCallBack 指向 callback 函式。直到 callback 函式传回「FALSE」（0），GetStrings 将为每个字串都呼叫一次 GetStrCallBack。GetStrings 传回传递给 callback 函式的字串数。pParam 参数是一个远程指标，指向程式写作者定义的资料。

当然，此程式可以编译成 Unicode 程式，或者在 STRLIB 的支援下，编译成 Unicode 和非 Unicode 应用程式。与 EDRLIB 一样，所有的函式都有「A」和「W」两种版本。在内部，STRLIB 以 Unicode 储存所有的字串。如果非 Unicode 程式使用了 STRLIB（也就是说，程式将呼叫 AddStringA、DeleteStringA 和 GetStringsA），字串将在 Unicode 和非 Unicode 之间转换。

与 STRPROG 和 STRLIB 专案相关的工作空间名为 STRPROG。此档案按 EDRTEST 工作空间的方式组合。程式 21-3 显示了建立 STRLIB.DLL 动态连结程式库所必须的两个档案。

程式 21-3　STRLI

```
STRLIB.H
/*-------------------------------------------------------------------------
--
      STRLIB.H header file
-------------------------------------------------------------------------
-*/

#ifdef __cplusplus
#define EXPORT extern "C" __declspec (dllexport)
#else
#define EXPORT __declspec (dllexport)
#endif

      // The maximum number of strings STRLIB will store and their lengths
```

```
#define     MAX_STRINGS 256
#define     MAX_LENGTH  63

      // The callback function type definition uses generic strings

typedef BOOL (CALLBACK * GETSTRCB) (PCTSTR, PVOID) ;

      // Each function has ANSI and Unicode versions

EXPORT      BOOL CALLBACK AddStringA (PCSTR) ;
EXPORT      BOOL CALLBACK AddStringW (PCWSTR) ;

EXPORT      BOOL CALLBACK DeleteStringA (PCSTR) ;
EXPORT      BOOL CALLBACK DeleteStringW (PCWSTR) ;

EXPORT      int CALLBACK GetStringsA (GETSTRCB, PVOID) ;
EXPORT      int CALLBACK GetStringsW (GETSTRCB, PVOID) ;

      // Use the correct version depending on the UNICODE identifier

#ifdef      UNICODE
#define     AddString         AddStringW
#define     DeleteString            DeleteStringW
#define     GetStrings              GetStringsW
#else
#define     AddString         AddStringA
#define     DeleteString            DeleteStringA
#define     GetStrings              GetStringsA
#endif
STRLIB.C
/*------------------------------------------------------------------------
-
      STRLIB.C -  Library module for STRPROG program
                                        (c) Charles Petzold, 1998
------------------------------------------------------------------------
*/

#include <windows.h>
#include <wchar.h>                              //  for  wide-character  string
functions
#include "strlib.h"

      // shared memory section (requires /SECTION:shared,RWS in link options)
#pragma           data_seg ("shared")
int               iTotal = 0 ;
WCHAR                   szStrings [MAX_STRINGS][MAX_LENGTH + 1] = { '\0' } ;
#pragma           data_seg ()
```

```
#pragma          comment(linker,"/SECTION:shared,RWS")

int WINAPI DllMain (HINSTANCE hInstance, DWORD fdwReason, PVOID pvReserved)
{
      return TRUE ;
}

EXPORT BOOL CALLBACK AddStringA (PCSTR pStringIn)
{
      BOOL        bReturn ;
      int         iLength ;
      PWSTR             pWideStr ;

      // Convert string to Unicode and call AddStringW
      iLength = MultiByteToWideChar (CP_ACP, 0, pStringIn, -1, NULL, 0) ;
      pWideStr = malloc (iLength) ;
      MultiByteToWideChar (CP_ACP, 0, pStringIn, -1, pWideStr, iLength) ;
      bReturn = AddStringW (pWideStr) ;
      free (pWideStr) ;

      return bReturn ;
}

EXPORT BOOL CALLBACK AddStringW (PCWSTR pStringIn)
{
      PWSTR             pString ;
      int         i, iLength ;

      if (iTotal == MAX_STRINGS - 1)
                        return FALSE ;
      if ((iLength = wcslen (pStringIn)) == 0)
                        return FALSE ;
                  // Allocate memory for storing string, copy it, convert to
uppercase
      pString = malloc (sizeof (WCHAR) * (1 + iLength)) ;
      wcscpy (pString, pStringIn) ;
      _wcsupr (pString) ;

                  //  Alphabetize the strings
      for (i = iTotal ; i > 0 ; i-)
      {
                  if (wcscmp (pString, szStrings[i - 1]) >= 0)
                                    break ;
                  wcscpy (szStrings[i], szStrings[i - 1]) ;
      }
      wcscpy (szStrings[i], pString) ;
      iTotal++ ;
```

```
      free (pString) ;
      return TRUE ;
}

EXPORT BOOL CALLBACK DeleteStringA (PCSTR pStringIn)
{
      BOOL        bReturn ;
      int         iLength ;
      PWSTR             pWideStr ;

                        // Convert string to Unicode and call DeleteStringW

      iLength = MultiByteToWideChar (CP_ACP, 0, pStringIn, -1, NULL, 0) ;
      pWideStr = malloc (iLength) ;
      MultiByteToWideChar (CP_ACP, 0, pStringIn, -1, pWideStr, iLength) ;
      bReturn = DeleteStringW (pWideStr) ;
      free (pWideStr) ;

      return bReturn ;
}

EXPORT BOOL CALLBACK DeleteStringW (PCWSTR pStringIn)
{
      int i, j ;
      if (0 == wcslen (pStringIn))
                        return FALSE ;
      for (i = 0 ; i < iTotal ; i++)
      {
                if (_wcsicmp (szStrings[i], pStringIn) == 0)
                                  break ;
      }
                // If given string not in list, return without taking action
      if (i == iTotal)
                        return FALSE ;
                // Else adjust list downward
      for (j = i ; j < iTotal ; j++)
                        wcscpy (szStrings[j], szStrings[j + 1]) ;
      szStrings[iTotal-][0] = '\0' ;
      return TRUE ;
}

EXPORT int CALLBACK GetStringsA (GETSTRCB pfnGetStrCallBack, PVOID pParam)
{
      BOOL        bReturn ;
      int         i, iLength ;
      PSTR        pAnsiStr ;

      for (i = 0 ; i < iTotal ; i++)
```

```
      {
                                // Convert string from Unicode
iLength = WideCharToMultiByte (    CP_ACP, 0, szStrings[i], -1, NULL, 0, NULL,
NULL) ;
                  pAnsiStr = malloc (iLength) ;
         WideCharToMultiByte (     CP_ACP,  0,  szStrings[i],  -1,  pAnsiStr,
iLength, NULL, NULL) ;

                                          // Call callback function

                  bReturn = pfnGetStrCallBack (pAnsiStr, pParam) ;

                  if (bReturn == FALSE)
                                          return i + 1 ;

                  free (pAnsiStr) ;
      }
      return iTotal ;
}

EXPORT int CALLBACK GetStringsW (GETSTRCB pfnGetStrCallBack, PVOID pParam)
{
      BOOL        bReturn ;
      int         i ;

      for (i = 0 ; i < iTotal ; i++)
      {
                  bReturn = pfnGetStrCallBack (szStrings[i], pParam) ;
                  if (bReturn == FALSE)
                                    return i + 1 ;
      }
      return iTotal ;
}
```

除了 DllMain 函式以外，STRLIB 中只有六个函式供其他函式输出用。所有这些函式都按 EXPORT 定义。这会使 LINK 在 STRLIB.LIB 引用程式库中列出它们。

STRPROG 程式

STRPROG 程式如程式 21-4 所示，其内容相当浅显易懂。两个功能表选项(Enter 和 Delete)启动一个对话方块，让您输入一个字串，然後 STRPROG 呼叫 AddString 或者 DeleteString。当程式需要更新它的显示区域时，呼叫 GetStrings 并使用函式 GetStrCallBack 来列出所列举的字串。

程式 21-4　STRPROG

```
STRPROG.C
/*------------------------------------------------------------------------
```

```
--
      STRPROG.C - Program using STRLIB dynamic-link library
                                                (c) Charles Petzold, 1998
------------------------------------------------------------------------
-*/

#include <windows.h>
#include "strlib.h"
#include "resource.h"

typedef struct
{
      HDC   hdc ;
      int   xText ;
      int   yText ;
      int   xStart ;
      int   yStart ;
      int   xIncr ;
      int   yIncr ;
      int   xMax ;
      int   yMax ;
}
CBPARAM ;
LRESULT     CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR            szAppName [] = TEXT ("StrProg") ;
TCHAR szString [MAX_LENGTH + 1] ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                    PSTR    szCmdLine,    int
iCmdShow)
{
      HWND                  hwnd ;
      MSG                   msg ;
      WNDCLASS         wndclass ;

      wndclass.style                    = CS_HREDRAW | CS_VREDRAW ;
      wndclass.lpfnWndProc              = WndProc ;
      wndclass.cbClsExtra               = 0 ;
      wndclass.cbWndExtra               = 0 ;
      wndclass.hInstance                = hInstance ;
      wndclass.hIcon                    = LoadIcon (NULL, IDI_APPLICATION) ;
      wndclass.hCursor                  = LoadCursor (NULL, IDC_ARROW) ;
      wndclass.hbrBackground      = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
      wndclass.lpszMenuName       = szAppName ;
      wndclass.lpszClassName      = szAppName ;

      if (!RegisterClass (&wndclass))
      {
            MessageBox (     NULL, TEXT ("This program requires Windows NT!"),
```

```
                                                    szAppName, MB_ICONERROR) ;
                return 0 ;
    }

     hwnd = CreateWindow (  szAppName, TEXT ("DLL Demonstration Program"),
                            WS_OVERLAPPEDWINDOW,
                            CW_USEDEFAULT, CW_USEDEFAULT,
                            CW_USEDEFAULT, CW_USEDEFAULT,
                            NULL, NULL, hInstance, NULL) ;

     ShowWindow (hwnd, iCmdShow) ;
     UpdateWindow (hwnd) ;

     while (GetMessage (&msg, NULL, 0, 0))
     {
                TranslateMessage (&msg) ;
                DispatchMessage (&msg) ;
     }
     return msg.wParam ;
}

BOOL CALLBACK DlgProc (HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
     switch (message)
     {
     case  WM_INITDIALOG:
           SendDlgItemMessage (hDlg, IDC_STRING, EM_LIMITTEXT, MAX_LENGTH, 0) ;
           return TRUE ;

     case  WM_COMMAND:
                       switch (wParam)
                       {
                       case  IDOK:
                       GetDlgItemText (hDlg, IDC_STRING, szString, MAX_LENGTH) ;
                                       EndDialog (hDlg, TRUE) ;
                                       return TRUE ;

                       case  IDCANCEL:
                                       EndDialog (hDlg, FALSE) ;
                                       return TRUE ;
                       }
     }
     return FALSE ;
}

BOOL CALLBACK GetStrCallBack (PTSTR pString, CBPARAM * pcbp)
{
     TextOut (  pcbp->hdc, pcbp->xText, pcbp->yText,
```

```
                                    pString, lstrlen (pString)) ;

     if ((pcbp->yText += pcbp->yIncr) > pcbp->yMax)
     {
                    pcbp->yText = pcbp->yStart ;
                    if ((pcbp->xText += pcbp->xIncr) > pcbp->xMax)
                                    return FALSE ;
     }
     return TRUE ;
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
     static HINSTANCE            hInst ;
     static int                        cxChar, cyChar, cxClient, cyClient ;
     static UINT                 iDataChangeMsg ;
     CBPARAM                                 cbparam ;
     HDC                                          hdc ;
     PAINTSTRUCT                       ps ;
     TEXTMETRIC                        tm ;

     switch (message)
     {
     case  WM_CREATE:
                hInst            = ((LPCREATESTRUCT) lParam)->hInstance ;
                hdc   = GetDC (hwnd) ;
                GetTextMetrics (hdc, &tm) ;
                cxChar      = (int) tm.tmAveCharWidth ;
                cyChar      = (int) (tm.tmHeight + tm.tmExternalLeading) ;
                     ReleaseDC (hwnd, hdc) ;

                     // Register message for notifying instances of data changes

     iDataChangeMsg = RegisterWindowMessage (TEXT ("StrProgDataChange")) ;
                return 0 ;
     case  WM_COMMAND:
                     switch (wParam)
                     {
                     case  IDM_ENTER:
                     if (DialogBox (hInst, TEXT ("EnterDlg"), hwnd, &DlgProc))
                                    {
               if (AddString (szString))
                PostMessage (HWND_BROADCAST, iDataChangeMsg, 0, 0) ;
               else
                   MessageBeep (0) ;
                     }
                break ;
```

```
                    case  IDM_DELETE:
                    if (DialogBox (hInst, TEXT ("DeleteDlg"), hwnd, &DlgProc))
                                        {
                 if (DeleteString (szString))
                 PostMessage (HWND_BROADCAST, iDataChangeMsg, 0, 0) ;
                else
                MessageBeep (0) ;
                                        }
                                  break ;
                    }
                    return 0 ;

     case  WM_SIZE:
                    cxClient = (int) LOWORD (lParam) ;
                    cyClient = (int) HIWORD (lParam) ;
                    return 0 ;

     case  WM_PAINT:
           hdc = BeginPaint (hwnd, &ps) ;

           cbparam.hdc            = hdc ;
           cbparam.xText    = cbparam.xStart = cxChar ;
           cbparam.yText    = cbparam.yStart = cyChar ;
           cbparam.xIncr    = cxChar * MAX_LENGTH ;
           cbparam.yIncr    = cyChar ;
           cbparam.xMax     = cbparam.xIncr * (1 + cxClient / cbparam.xIncr) ;
                    cbparam.yMax     = cyChar * (cyClient / cyChar - 1) ;

                 GetStrings ((GETSTRCB) GetStrCallBack, (PVOID) &cbparam) ;

                    EndPaint (hwnd, &ps) ;
                    return 0 ;

     case  WM_DESTROY:
                    PostQuitMessage (0) ;
                    return 0 ;

     default:
                 if (message == iDataChangeMsg)
                                  InvalidateRect (hwnd, NULL, TRUE) ;
                 break ;
     }
     return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

STRPROG.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
```

```
#include "afxres.h"

/////////////////////////////////////////////////////////////////////////////
/
// Dialog
ENTERDLG DIALOG DISCARDABLE  20, 20, 186, 47
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Enter"
FONT 8, "MS Sans Serif"
BEGIN
      LTEXT                                    "&Enter:",IDC_STATIC,7,7,26,9
      EDITTEXT
      IDC_STRING,31,7,148,12,ES_AUTOHSCROLL
      DEFPUSHBUTTON          "OK",IDOK,32,26,50,14
      PUSHBUTTON                   "Cancel",IDCANCEL,104,26,50,14
END
DELETEDLG DIALOG DISCARDABLE  20, 20, 186, 47
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Delete"
FONT 8, "MS Sans Serif"
BEGIN
      LTEXT                                    "&Delete:",IDC_STATIC,7,7,26,9
      EDITTEXT
      IDC_STRING,31,7,148,12,ES_AUTOHSCROLL
      DEFPUSHBUTTON          "OK",IDOK,32,26,50,14
      PUSHBUTTON                   "Cancel",IDCANCEL,104,26,50,14
END

/////////////////////////////////////////////////////////////////////////////
/
// Menu
STRPROG MENU DISCARDABLE
BEGIN
      MENUITEM "&Enter!",                                          IDM_ENTER
      MENUITEM "&Delete!",
      IDM_DELETE
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by StrProg.rc

#define          IDC_STRING                                          1000
#define          IDM_ENTER                                           40001
#define          IDM_DELETE                                          40002
#define          IDC_STATIC                                          -1
```

STRPROG.C 包含 STRLIB.H 表头档案，其中定义了 STRPROG 将使用的 STRLIB 中的三个函式。

当您执行 STRPROG 的多个执行实体的时候，本程式的奥妙之处就会显露出来。STRLIB 将在共用记忆体中储存字串及其指标，并允许 STRPROG 中的所有执行实体共用此资料。让我们看一下它是如何执行的吧。

在 STRPROG 执行实体之间共用资料

Windows 在一个 Win32 程序的位址空间周围筑了一道墙。通常，一个程序的位址空间中的资料是私有的，对别的程序而言是不可见的。但是执行 STRPROG 的多个执行实体表示了 STRLIB 在程式的所有执行实体之间共用资料是毫无问题的。当您在一个 STRPROG 视窗中增加或者删除一个字串时，这种改变将立即反映在其他的视窗中。

在全部常式之间，STRLIB 共用两个变数：一个字元阵列和一个整数（记录已储存的有效字串的个数）。STRLIB 将这两个变数储存在共用的一个特殊记忆体区段中：

```
#pragma     data_seg ("shared")
int         iTotal = 0 ;
WCHAR            szStrings [MAX_STRINGS][MAX_LENGTH + 1] = { '\0' } ;
#pragma     data_seg ()
```

第一个#pragma 叙述建立资料段，这里命名为 shared。您可以将这段命名为任何一个您喜欢的名字。在这里的#pragma 叙述之後的所有初始化了的变数都放在 shared 资料段中。第二个#pragma 叙述标示段的结束。对变数进行专门的初始化是很重要的，否则编译器将把它们放在普通的未初始化资料段中而不是放在 shared 中。

连结器必须知道有一个「shared」共享资料段。在「Project Settings」对话方块选择「Link」页面标签。选中「STRLIB」时在「Project Options」栏位（在 Release 和 Debug 设定中均可），包含下面的连结叙述：

```
/SECTION:shared,RWS
```

字母 RWS 表示段具有读、写和共用属性。或者，您也可以直接用 DLL 原始码指定连结选项，就像我们在 STRLIB.C 那样：

```
#pragma comment(linker,"/SECTION:shared,RWS")
```

共用的记忆体段允许 iTotal 变数和 szStrings 字串阵列在 STRLIB 的所有常式之间共用。因为 MAX_STRINGS 等於 256，而 MAX_LENGTH 等於 63，所以，共用记忆体段的长度为 32,772 位元组——iTotal 变数需要 4 位元组，256 个指标中的每一个都需要 128 位元组。

使用共用记忆体段可能是在多个应用程式间共用资料的最简单的方法。如果需要动态配置共用记忆体空间，您应该查看记忆体映射档案物件的用法，文件在 /Platform SDK/Windows Base Services/Interprocess

Communication/File Mapping。

各式各样的 DLL 讨论

如前所述，动态连结程式库模组不接收讯息，但是，动态连结程式库模组可呼叫 GetMessage 和 PeekMessage。实际上，从讯息伫列中得到的讯息是发给呼叫程式库函式的程式的。一般来说，程式库是替呼叫它的程式工作的，这是一项对程式库所呼叫的大多数 Windows 函式都适用的规则。

动态连结程式库可以从程式库档案或者从呼叫程式库的程式档案中载入资源（如图示、字串和点阵图）。载入资源的函式需要执行实体代号。如果程式库使用它自己的执行实体代号（初始化期间传给程式库的），则程式库能从它自己的档案中获得资源。为了从呼叫程式的.EXE 档案中得到资源，程式程式库函式需要呼叫该函式的程式的执行实体代号。

在程式库中登录视窗类别和建立视窗需要一点技巧。视窗类别结构和 CreateWindow 呼叫都需要执行实体代号。尽管在建立视窗类别和视窗时可使用动态连结程式库模组的执行实体代号，但在程式库建立视窗时，视窗讯息仍会发送到呼叫程式库中程式的讯息伫列。如果使用者必须在程式库中建立视窗类别和视窗，最好的方法可能是使用呼叫程式的执行实体代号。

因为模态对话方块的讯息是在程式的讯息回圈之外接收到的，因此使用者可以在程式库中呼叫 DialogBox 来建立模态对话方块。执行实体代号可以是程式库代号，并且 DialogBox 的 hwndParent 参数可以为 NULL。

不用输入引用资讯的动态连结

除了在第一次把使用者程式载入记忆体时，由 Windows 执行动态连结外，程式执行时也可以把程式同动态连结程式库模组连结到一起。例如，您通常会这样呼叫 Rectangle 函式：

```
Rectangle (hdc, xLeft, yTop, xRight, yBottom) ;
```

因为程式和 GDI32.LIB 引用程式库连结，该程式库提供了 Rectangle 的位址，因此这种方法有效。

您也可以用更迂回的方法呼叫 Rectangle。首先用 typedef 为 Rectangle 定义一个函式型态：

```
typedef BOOL (WINAPI * PFNRECT) (HDC, int, int, int, int) ;
```

然後定义两个变数：

```
HANDLE      hLibrary ;
PFNRECT     pfnRectangle ;
```

现在将 hLibrary 设定为程式库代号，将 lpfnRectangle 设定为 Rectangle

函式的位址：

```
hLibrary = LoadLibrary (TEXT ("GDI32.DLL"))
pfnRectangle = (PFNPRECT) GetProcAddress (hLibrary, TEXT ("Rectangle"))
```

如果找不到程式库档案或者发生其他一些错误，LoadLibrary 函式传回 NULL。现在您可以呼叫函式然後释放程式库：

```
pfnRectangle (hdc, xLeft, yTop, xRight, yBottom) ;
FreeLibrary (hLibrary) ;
```

尽管这项执行时期动态连结的技术并没有为 Rectangle 函式增加多大好处，但它肯定是有用的，如果直到执行时还不知道程式动态连结程式库模组的名称，这时就需要使用它。

上面的程式码使用了 LoadLibrary 和 FreeLibrary 函式。Windows 为所有的动态连结程式库模组提供「引用计数」,LoadLibrary 使引用计数递增。当 Windows 载入任何使用了程式库的程式时，引用计数也会递增。FreeLibrary 使引用计数递减，在使用了程式库的程式执行实体结束时也是如此。当引用计数为零时，Windows 将从记忆体中把程式库删除掉，因为不再需要它了。

纯资源程式库

可由 Windows 程式或其他程式库使用的动态连结程式库中的任何函式都必须被输出。然而，DLL 也可以不包含任何输出函式。那么，DLL 到底包含什么呢？答案是资源。

假设使用者正在使用需要几幅点阵图的 Windows 应用程式进行工作。通常要在程式的资源描述档中列出资源，并用 LoadBitmap 函式把它们载入记忆体。但使用者可能希望建立若干套点阵图，每一套均适用於 Windows 所使用的不同显示卡。将不同套的点阵图存放到不同档案中可能是明智的，因为只需要在硬碟上保留一套点阵图。这些档案就是纯资源档案。

程式 21-5 说明如何建立包含 9 幅点阵图的名为 BITLIB.DLL 的纯资源程式库档案。BITLIB.RC 档案列出了所有独立的点阵图档案并为每个档案赋予一个序号。为了建立 BITLIB.DLL，需要 9 幅名为 BITMAP1.BMP、BITMAP2.BMP 等等的点阵图。您可以使用附带的光碟上提供的点阵图或者在 Visual C++中建立这些点阵图。它们与 ID 从 1 到 9 相对应。

程式 21-5 BITLIB

```
BITLIB.C
/*-------------------------------------------------------------
      BITLIB.C -- Code entry point for BITLIB dynamic-link library
                                          (c) Charles Petzold, 1998
-------------------------------------------------------------*/
```

```
#include <windows.h>
int WINAPI DllMain (HINSTANCE hInstance, DWORD fdwReason, PVOID pvReserved)
{
      return TRUE ;
}
```

BITLIB.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
/
// Bitmap
1               BITMAP  DISCARDABLE     "bitmap1.bmp"
2               BITMAP  DISCARDABLE     "bitmap2.bmp"
3               BITMAP  DISCARDABLE     "bitmap3.bmp"
4               BITMAP  DISCARDABLE     "bitmap4.bmp"
5               BITMAP  DISCARDABLE     "bitmap5.bmp"
6               BITMAP  DISCARDABLE     "bitmap6.bmp"
7               BITMAP  DISCARDABLE     "bitmap7.bmp"
8               BITMAP  DISCARDABLE     "bitmap8.bmp"
9               BITMAP  DISCARDABLE     "bitmap9.bmp"
```

在名为 SHOWBIT 的工作空间中建立 BITLIB 专案。在名为 SHOWBIT 的另一个专案中，建立程式 21-6 所示的 SHOWBIT 程式，这与前面的一样。不过，不要使 BITLIB 依赖於 SHOWBIT；否则，连结程序中将需要 BITLIB.LIB 档案，并且因为 BITLIB 没有任何输出函式，它也不会建立 BITLIB.LIB。事实上，要分别重新编译 BITLIB 和 SHOWBIT，可以交替设定其中一个为「Active Project」然後再重新编译。

SHOWBIT.C 从 BITLIB 读取点阵图资源，然後在其显示区域显示。按键盘上的任意键可以循环显示。

程式 21-6　SHOWBIT

```
SHOWBIT.C
/*------------------------------------------------------------------------
      SHOWBIT.C -- Shows bitmaps in BITLIB dynamic-link library
                                             (c) Charles Petzold, 1998
------------------------------------------------------------------------*
/

#include <windows.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

TCHAR szAppName [] = TEXT ("ShowBit") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                  PSTR    szCmdLine,    int
```

```
iCmdShow)
{
      HWND                    hwnd ;
      MSG                     msg ;
      WNDCLASS          wndclass ;

      wndclass.style                      = CS_HREDRAW | CS_VREDRAW ;
      wndclass.lpfnWndProc                = WndProc ;
      wndclass.cbClsExtra                 = 0 ;
      wndclass.cbWndExtra                 = 0 ;
      wndclass.hInstance                      = hInstance ;
      wndclass.hIcon                      = LoadIcon (NULL, IDI_APPLICATION) ;
      wndclass.hCursor                    = LoadCursor (NULL, IDC_ARROW) ;
      wndclass.hbrBackground       = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
      wndclass.lpszMenuName  = NULL ;
      wndclass.lpszClassName       = szAppName ;

      if (!RegisterClass (&wndclass))
    {
           MessageBox (     NULL, TEXT ("This program requires Windows NT!"),
                                                    szAppName, MB_ICONERROR) ;
                return 0 ;
    }

    hwnd = CreateWindow (szAppName,
          TEXT ("Show Bitmaps from BITLIB (Press Key)"),
              WS_OVERLAPPEDWINDOW,
              CW_USEDEFAULT, CW_USEDEFAULT,
              CW_USEDEFAULT, CW_USEDEFAULT,
              NULL, NULL, hInstance, NULL) ;

      if (!hwnd)
                        return 0 ;
      ShowWindow (hwnd, iCmdShow) ;
      UpdateWindow (hwnd) ;

      while (GetMessage (&msg, NULL, 0, 0))
      {
         TranslateMessage (&msg) ;
         DispatchMessage (&msg) ;
      }
      return msg.wParam ;
}

void DrawBitmap (HDC hdc, int xStart, int yStart, HBITMAP hBitmap)
{
      BITMAP            bm ;
      HDC               hMemDC ;
```

```
     POINT                pt ;

     hMemDC = CreateCompatibleDC (hdc) ;
     SelectObject (hMemDC, hBitmap) ;
     GetObject (hBitmap, sizeof (BITMAP), &bm) ;
     pt.x = bm.bmWidth ;
     pt.y = bm.bmHeight ;

     BitBlt (hdc, xStart, yStart, pt.x, pt.y, hMemDC, 0, 0, SRCCOPY) ;
     DeleteDC (hMemDC) ;
}

LRESULT CALLBACK WndProc (  HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
     static HINSTANCE       hLibrary ;
     static int                         iCurrent = 1 ;
     HBITMAP                                  hBitmap ;
     HDC                                           hdc ;
     PAINTSTRUCT                        ps ;

     switch (message)
     {
     case  WM_CREATE:
               if ((hLibrary = LoadLibrary (TEXT ("BITLIB.DLL"))) == NULL)
                    {
        MessageBox (  hwnd, TEXT ("Can't load BITLIB.DLL."),
                  szAppName, 0) ;
        return -1 ;
                    }
               return 0 ;

     case  WM_CHAR:
                    if (hLibrary)
                    {
                              iCurrent ++ ;
                              InvalidateRect (hwnd, NULL, TRUE) ;
                    }
                    return 0 ;
     case  WM_PAINT:
                    hdc = BeginPaint (hwnd, &ps) ;

                    if (hLibrary)
                    {
               hBitmap = LoadBitmap (hLibrary, MAKEINTRESOURCE (iCurrent)) ;

                              if (!hBitmap)
                              {
```

```
                                        iCurrent = 1 ;
                                        hBitmap = LoadBitmap ( hLibrary,
                                  MAKEINTRESOURCE (iCurrent)) ;
                                   }
                                   if (hBitmap)
                                        {
                                              DrawBitmap (hdc, 0, 0, hBitmap) ;
                                        DeleteObject (hBitmap) ;
            }
                        }
                        EndPaint (hwnd, &ps) ;
                        return 0 ;

      case  WM_DESTROY:
                        if (hLibrary)
                                        FreeLibrary (hLibrary) ;

                        PostQuitMessage (0) ;
                        return 0 ;
      }
      return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

在处理 WM_CREATE 讯息处理期间，SHOWBIT 获得了 BITLIB.DLL 的代号：

```
if ((hLibrary = LoadLibrary (TEXT ("BITLIB.DLL"))) == NULL)
```

如果 BITLIB.DLL 与 SHOWBIT.EXE 不在同一个目录，Windows 将按本章前面讨论的方法搜索。如果 LoadLibrary 传回 NULL，SHOWBIT 显示一个讯息方块来报告错误，并从 WM_CREATE 讯息传回-1。这将导致 WinMain 中的 CreateWindow 呼叫传回 NULL，而且程式终止程式。

SHOWBIT 透过程式库代号和点阵图号码来呼叫 LoadBitmap，从而得到一个点阵图代号：

```
hBitmap = LoadBitmap (hLibrary, MAKEINTRESOURCE (iCurrent)) ;
```

如果号码 iCurrent 对应的点阵图无效或者没有足够的记忆体载入点阵图，则传回一个错误。

在处理 WM_DESTROY 讯息时，SHOWBIT 释放程式库：

```
FreeLibrary (hLibrary) ;
```

当 SHOWBIT 的最後一个执行实体终止时，BITLIB.DLL 的引用计数变为 0，并且释放所占用的记忆体。这就是实作「图片剪辑」程式的一种简单方法，所谓的「图片剪辑」程式就是能够将预先建立的点阵图（或者 metafile、增强型 metafile）载入到剪贴簿，以供其他程式使用的程式。