第一篇

Windows 程序设计基础

本篇将从最简单的 Windows 应用程序着手，介绍如何利用 Visual C++ 软件(配合 C 语言)设计 Windows 应用程序。本篇还将介绍有关 Windows 程序设计的下列基本概念：

(1) 文本数据的输出

(2) 鼠标输入

(3) 键盘输入

(4) 窗口滚动条控制

(5) Windows 系统内的定时器

第一章　一个简单的实例

设计 Windows 应用程序比较困难，原因是，一个很简单的输出结果却需要很长的程序代码。本章将以一个最基本的 Windows 应用程序为例，深入讲解每一条语句的用法。

由于 Windows 应用程序较为复杂，本章将以 DOS 环境和 Windows 环境为例，一步一步地讲解编译、链接和运行程序的方法。本章将讲解下列内容：

(1) 软件和硬件需求

(2) Windows 程序的结构

(3) Windows 应用程序的编译与链接

(4) 一个简单的 Windows 程序实例

(5) 图标的设置

(6) 光标外形的设置

(7) 客户区的颜色

自 1990 年 5 月 22 日 Microsoft 公司在美国举办了一场壮观的 Microsoft Windows 3.0 发表会之后，几乎所有的 PC 软件厂商都以开发适用于 Windows 环境的软件为目标。轻敲按钮，弹指之间即可协助用户完成工作，的确为用户带来了很大的方便。

Windows 环境尽管好用，但是如何设计可在 Windows 环境下运行的 Windows 应用程序却一直是程序员的恶梦，本书的主要目的就是破除此恶梦，使用户可以花较少的时间学会 Windows 应用程序的设计方法。

1.1　软硬件需求

设计 Windows 程序的首要条件是，用户应备有 Microsoft Windows 软件。在当前市场上，用户可以使用 Borland 公司推出的 Borland C++（含 Application Framework，简称 AF）或者 Microsoft 公司最新开发的 Visual C++ 软件来设计 Windows 程序。本书的重点是以 Visual C++ 为工具来说明 Windows 应用程序的设计方法。

在 Visual C++ 软件中，用户基本可以采用下面两种方式来设计 Windows 应用程序：

(1) 利用 C 语言并配合应用程序接口函数（Application Programming Interface，简称 API）。

(2) 利用 C++ 语言并配合 Microsoft Foundation Class（简称 MFC）函数库及 API。

由于 C 语言仍是当前最广泛使用的程序语言，因此本书决定以 C 语言来讲解设计基本 Windows 应用程序的方法。

过去常听人说，要设计 Windows 应用程序，需要有软件开发工具（Software Development Toolkit，简称 SDK），不过在使用 Visual C++ 设计 Windows 应用程序时，就不必考虑这个工具了，因为 SDK 已经完全集成在 Visual C++ 软件中。

1.2 不同的工作环境

如果读者是C语言高手且精通函数库和DOS及BIOS调用，则在接触Windows应用程序设计之前，可能会认为利用C语言设计Windows应用程序只是多了一些有关窗口的函数调用罢了。如果是这样，将会大失所望。

在DOS环境下，程序员可以独占系统资源（即CPU、内存、屏幕及键盘等），进行数据的输入/输出并执行一般的运算。利用系统资源和执行一般运算时，大多数情况下只需要调用几个函数。但是在Windows环境下，以上情况都变得无效。下面将介绍Windows应用程序和DOS应用程序开发环境之间的差异。

1.2.1 多任务(Multitasking)特性

Windows是一个多任务的操作系统，也就是能在同一时间内执行多个应用程序。在这种情况下设计的应用程序无法独占所有系统资源（系统资源指CPU、内存、屏幕及键盘等）。DOS只是一个单用户的操作环境，在设计程序时，可以充分使用系统有效资源(available resource)。

例如，对DOS应用程序而言，一定可以在屏幕上输出数据，而不可能有其他程序共享屏幕资源。但是在Windows环境下，屏幕上可能同时出现多个应用程序的输出窗口。因此，对于任何一个Windows应用程序，在输出数据时必须考虑与其他正在运行的应用程序共享屏幕资源。Windows操作系统必须小心管理所有系统资源，以供所有应用程序分享。所有Windows应用程序必须根据Windows操作系统特有的界面(Interface)来执行操作，以确保Windows操作系统有效地管理系统资源，保证应用程序的使用。

1.2.2 操作界面

在DOS环境下，程序员可以独占屏幕。只要执行几个函数调用，就可以将资源输出到屏幕上，或者接收所输入的数据。

在Windows环境下，若想执行输入或输出操作，首先必须在屏幕上设计一个窗口，然后通过此窗口进行输入与输出。当然，一个应用程序可以存在多个应用程序窗口，这样就可以形成多文档界面(Multiple Document Interface)。

1.2.3 数据输入

在DOS环境下，程序员可以利用getchar(),scanf(),int86等函数来接收用户通过键盘或鼠标输入的数据。

在Windows环境下，Windows操作系统将处理用户通过键盘或鼠标输入的所有数据，然后再将接收到的数据送到适当的应用程序消息队列(application message queue，每个应用程序在运行时均产生此队列)，应用程序将自动在所属的应用程序消息队列中读取下一组数据。

1.2.4 数据输出

在DOS环境下，操作模式缺省假设为文本模式(Text Mode)。如果用户想输入图形，必须先切换到绘图模式(Graphics Mode)。如果不想输出图形，则必须切换回文本模式。

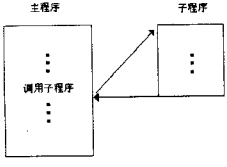
在Windows环境下，绘图模式是基本工作模式，用户可以很顺利地通过此模式进行文本或图形数据的输出。

1.3 Windows程序的结构

设计Windows应用程序比较困难。尽管同样是使用C语言来设计程序，但程序结构却有所不同。传统C语言的程序结构如下所示：

```
void main()
{
    .
    .
}
```

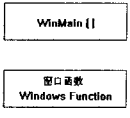
如果存在从主程序中调用子程序的情况，则调用子程序的基本流程如下：



在上图中，主程序调用子程序，子程序运行完毕后将立即返回主程序的原调用位置，然后继续往下运行。

尽管上述程序的逻辑概念仍然适用于Windows应用程序，不过Windows应用程序还包含了一些不同的逻辑概念。

Windows应用程序的基本结构如下：



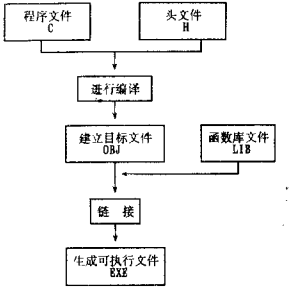
WinMain 和 Windows Function 是构成 Windows 应用程序的两大部分。WinMain 是主程序，也就是程序运行入口(entry point)，而 Windows Function 则是 Windows 应用程序的工作核心。程序员通常利用 WinMain 来执行设计和建立窗口的操作，而利用 Windows Function 来执行应用程序的操作。

WinMain 和 Windows Function 之间的区别在于，尽管从程序员的角度来看，Windows Function 属于 WinMain 的子程序，但是在 WinMain 中找不到任何一个调用 Windows Function 的语句。这时候用户一定会问，在哪一种情况下才执行用户所设计的 Windows Function 呢？

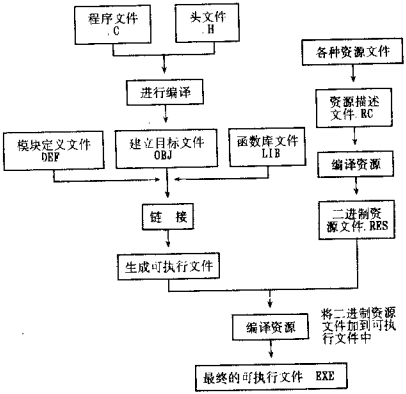
执行 Windows Function 的时机(后面章节将以实例详细说明)很多，重要的是当时机出现时，Windows 操作系统将主动调用 Windows Function。

1.4 Windows 应用程序的编译与链接

对于初学 Windows 应用程序设计的用户，另一个难点是，Windows 应用程序的编译(compile)及链接(link)过程比较复杂。一般 C 语言程序的编译及链接过程如下：



Windows 应用程序的编译及链接过程如下所示：



对于C语言程序员，通常只要设计C语言程序文件即可，至于编译及链接过程，通常可利用C语言本身的集成环境方便地完成。Windows应用程序则比较复杂，其步骤如下：

(1) 建立C语言程序文件。

(2) 建立模块定义文件(.def)，以定义程序的名称、属性(可搬移否)、堆栈(stack)和堆(heap)的大小。

(3) 利用资源编辑器(App Studio)编辑程序所需的资源文件，例如光标(.cur)、图标(.ico)、位图(.bmp)、对话框(.dia)及菜单等。

(4) 建立资源定义文件(.rc)，定义Windows应用程序所需的资源。

(5) 编译C语言程序文件，同时链接(配合模块定义文件和函数库文件)并建立可执行文件。

(6) 编译资源文件，产生资源可利用文件(.res)。

(7) 将资源可利用文件加到C程序语言的可执行文件内，形成Windows应用程序的可执行文件。

以上所述对于初学者来说可能稍感复杂，但是用户不必担心，本书将以实例一步一步讲解，用户只要遵循这些步骤，就一定可以学会建立自己的Windows应用程序。

1.5 资源类型

常见的C语言数据有下列几种类型：

数据类型	含　义
int	16位整数
unsigned int	16位无符号整数
long int	32位长整数
unsigned long int	32位无符号长整数
char	字符
float	32位浮点数
double	61位双精度浮点数

对于以C语言为基础的Windows应用程序，除了仍然沿用上述数据类型外，又扩充了下列几种常见的数据类型：

数据类型	含　义
WORD	16位无符号整数
DWORD	32位无符号整数
LONG	32位整数
HANDLE	16位无符号整数，此数据类型常用于表示Windows操作系统所建立的某个对象的句柄，使用此句柄相当于引用该对象
HWND	16位无符号整数，此数据类型常用于表示Windows环境内所打开的窗口句柄，使用此句柄相当于引用该窗口
BOOL	16位布尔值(Boolean value)
BYTE	8位无符号整数
FARPROC	32位指向函数的指针
LPSTR	32位指向字符(字符串)数据的指针
UINT	16位无符号整数
LPARAM	32位整数，常用于Windows函数的参数
WPARAM	16位整数，常用于Windows函数的参数

注意，如果未特别声明为无符号整数，则以上数据类型都代表符号整数。

以上只是一些常用的数据类型，后面将根据需要陆续配合程序实例介绍其他的数据类型。

1.6 一个简单的 Windows 程序实例

本程序将在屏幕上建立宽 300(x 轴)、高 200(y 轴)的窗口,基本单位是像素(pixel)。当光标在窗口内显示为"I"型时,窗口的标题为 ch1_1。

下面是 ch1_1.c 程序代码的全部内容:

```
// ------------------------------------------
//   程序名:ch1_1.c
//   第一个 windows 应用程序
// ------------------------------------------
#include <windows.h>
long FAR PASCAL WindowFun(HWND,UINT,WPARAM,LPARAM);

int PASCAL WinMain(HANDLE CurInstance,HANDLE PreInstance,
                   LPSTR CmdParaStr,int ShowStyle)
{
   static char ProgName[] = "ch1_1";   // 程序名
   HWND hwnd;                          // 窗口句柄
   MSG  msg;                           // 消息结构
   WNDCLASS wndclass;                  // 窗口类型

// 注册
// 通常只有第一次需要注册
   if ( ! PreInstance )
   {
      wndclass.lpszClassName = ProgName;
      wndclass.hInstance = CurInstance;
      wndclass.lpfnWndProc = WindowFun;
      wndclass.hIcon = LoadIcon(NULL,IDI_APPLICATION);
      wndclass.hCursor = LoadCursor(NULL,IDC_IBEAM);
      wndclass.lpszMenuName = NULL;
      wndclass.hbrBackground = GetStockObject(WHITE_BRUSH);
      wndclass.style = CS_HREDRAW | CS_VREDRAW;
      wndclass.cbClsExtra = 0;
      wndclass.cbWndExtra = 0;

      RegisterClass(&wndclass);                    // 注册
   }

// 建立窗口
   hwnd = CreateWindow(ProgName,                   // 首先注册名称
                       ProgName,                   // 窗口标题
                       WS_OVERLAPPEDWINDOW,// 窗口类型
                       100,                        // 窗口 x 位置
                       100,                        // 窗口 y 位置
                       300,                        // 窗口 x 长度
```

```
                    200,                    // 窗口 y 长度
                    NULL,                   // 父窗口的句柄
                    NULL,                   // 窗口菜单句柄
                    CurInstance,            // 当前窗口程序句柄
                    NULL);                  // 所用参数
  // 显示窗口
    ShowWindow(hwnd,ShowStyle);

  // 消息循环
    while (GetMessage(&msg,NULL,0,0))
    {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }
    return msg.wParam;
}
// ---------------------------------
// 窗口处理函数
// ---------------------------------
long FAR PASCAL WindowFunc(HWND hwnd,
                           UINT message,
                           WPARAM wParam,
                           LPARAM lParam)
{
  switch(message)
  {
    case WM_DESTROY:            // 释放所建窗口
      PostQuitMessage(0);
      return 0;
    default:                    // 执行其他情况
      return DefWindowProc(hwnd,message,wParam,lParam);
  }
}
```

下面是 ch1_1.def 的内容：

```
;-----------------------------
; ch1_1.def
;-----------------------------
NAME            ch1_1
DESCRIPTION     '第一个 Windows 应用程序'
EXETYPE         WINDOWS
STUB            'WINSTUB.EXE'
CODE            PRELOAD MOVEABLE DISCARDABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        1024
STACKSIZE       4096
```

下面是 ch1_1.mak 的内容：

```
#------------------
#  ch1_1.mak
#------------------

ch1_1.exe :ch1_1.obj ch1_1.def
    link /nod ch1_1,ch1_1,NUL,\
    slibcew libw commdlg,ch1_1
    rc -t ch1_1.exe

ch1_1.obj :ch1_1.c
    cl -c -G2sw -Ow -W3 -Zp -Tp ch1_1.c
```

1.6.1 程序的运行

运行 Windows 应用程序有两种方法，一是直接在 DOS 提示符下进行编译与链接(参考 1.6.1 节)，另一种是充分利用 Visual C++ 集成环境进行编译、链接与运行(参考 1.6.2 节)。

1. 在 DOS 环境下建立与运行程序

要在 DOS 环境下编译、链接与运行程序，首先应对启动盘中的 AUTOEXEC.BAT 文件作如下修改：

(1) 将下面一行加到 PATH 设置内：

C:\MSVC\BIN

(2) 请加入下列两行数据：

```
SET INCLUDE = C:\MSVC\INCLUDE;C:\MSVC\MFC\INCLUDE
SET LIB=C:\MSVC\LIB;C:\MSVC\MFC\LIB
```

以上假设将 Visual C++ 安装在 C:\MSVC 目录内。如果将 Visual C++ 安装在 D 盘的 MSVC 目录中，则只要将驱动器编号 C 改为 D 即可。当然，在上述设置更改完之后，必须重新开机，以上设置才有效。

在含有 ch1_1.c，ch1_1.def 和 ch1_1.mak 的目录下执行下列语句：

```
nmake ch1_1.mak
```

如果一切顺利，用户将看到下列编译及链接过程：

```
Microsoft (R) Program Maintenance Utility Version 1.30
Copyright (c) Microsoft Corp 1988-93.  All rights reserved.

        cl -c -G2sw -Ow -W3 -Zp -Tp ch1_1.c
Microsoft (R) C/C++ Optimizing Compiler Version 8.00
Copyright (c) Microsoft Corp 1984-93.  All rights reserved.
```

```
ch1_1.c
        link/nod ch1-1, ch1_1, NUL, slibcew libw commdlg, ch1_1

Microsoft (R) Segmented Executable Linker Version 5.50
Copyright (C) Microsoft Corp 1984-93. All rights reserved.

        rc -t ch1_1.exe
Microsoft (R) Windows Resource Compiler Version 3.31
Copyright (C) Microsoft Corp 1985-1992. All rights reserved.
```

见到上述运行结果后，表示已在 DOS 环境下成功地建立了某个 Windows 应用程序的可执行文件，该可执行文件的名称是 ch1_1.exe。只要键入下列语句即可运行这个可执行文件：

```
C:\WINDOWS\WIN  CH1-1
(启动 Windows)      (运行程序)
```

(1) 模块定义文件

ch1_1.def 文件又称模块定义文件，主要用于定义程序的模块名称以及一些相关属性。

1) 第 1 至第 3 行是注解。凡是分号右边的数据均是注解。

2) 第 4 行的 NAME 定义应用程序的名称，此行必须放在最前面。

3) 第 5 行的 DESCRIPTION 是程序注释。

4) 第 6 行的 EXETYPE 告诉链接程序(LINK)，这个应用程序在 Windows 环境下运行。

5) 第 7 行的 STUB 用于设置当此应用程序在 DOS 环境下运行失败时，将运行单引号内的程序 WINSTUB.EXE。运行此程序通常会出现下列信息：

This program requires Microsoft Windows

6) 第 8 行的 CODE 定义程序段的属性。只要一运行程序，PRELOAD 将立即装入程序段。MOVEABLE 的作用是，如果需要的话，Windows 操作系统将有权在内存中移动程序段。如果 Windows 操作系统想获取更多的内存空间，DISCARDABLE 可以将此程序从内存移走。如果以后需要此程序，Windows 操作系统会重新装入此程序。

7) 第 9 行的 DATA 定义数据段的属性。PRELOAD，MOVEABLE 和 CODE 内的定义相同，而 MULTIPLE 则表示，如果在同一时刻 Windows 操作系统包含此 Windows 应用程序的两份代码，则每一份 Windows 应用程序代码都将拥有自己的数据段。

8) 第 10 行的 HEAPSIZE 定义程序堆(heap)空间，在此设置为 1024 字节。堆空间主要供 C 语言程序动态利用 malloc()函数来分配内存。

9) 第 11 行的 STACKSIZE 定义程序堆栈(stack)空间，在此定义为 4096 字节。如果在程序运行时发生堆栈空间不够的情况，可以适当增加设置值。

(2) Make 文件(又称计划文件)

Make 文件的扩展名是. mak，可辅助 Windows 应用程序中各相关文件的编译和链接过程：

1）第 1 行至第 3 行是注解。凡是＃右边的字符均是注解。

2）第 5 行表示 ch1_1. exe 文件由 ch1_1. obj 和 ch1_1. def 链接而成。至于通过链接来产生 ch1_1. exe 的方式，则由第 6 行至第 7 行设置。同时第 5 行也表示，只有在 ch1_1. exe 文件的生成时间比 ch1_1. obj 或 ch1_1. def 早时，才进行重新链接，以便产生新的可执行文件。

3）第 6 行的内容及含义如下：

link /nod ch1_1, ch1_1, NULL, \

- /nod 告诉链接程序直接使用语句所定义的函数库（在第 7 行设置这些函数库），不必再搜索缺省的函数库。
- 第一个 ch1_1 表示这个 ch1_1 是 ch1_1. obj 文件。
- 第二个 ch1_1 表示这个 ch1_1 是 ch1_1. exe 文件。
- NUL 字段设置 map 类型的文件名，在此设置为 NULL，表示不产生 map 类型的文件。
- \符号表示下一行的参数属于本行语句。

4）第 7 行的内容及含义如下：

slibcew libw commdlg, ch1_1

- slibcew 的全名是 slibcew. lib，它是 small 模式下的 Windows C 函数库。
- libw 的全名是 libw. lib，使今后的程序可动态地将它们调用的函数命令链接到相关的函数库内。
- commdlg 的全名是 commdlg. lib，主要供建立对话框时使用。
- ch1_1 的全名是 ch1_1. def。

5）第 8 行充分利用资源编译程序（rc 是资源编译程序）产生 Windows 应用程序的可执行文件。－t 参数设置此应用程序只能在保护模式（protect mode）下运行。

6）第 10 行表示 ch1_1. obj 文件是通过编译 ch1_1. c 程序得到的。编译方式在 11 行设置。同时第 10 行也设置，只有在 ch1_1. obj 文件的生成时间比 ch1_1. c 早时，才重新编译，以产生新的目标文件。

7）第 11 行设置编译（cl 是编译程序）ch1_1. c 文件的方法，各参数的含义如下：

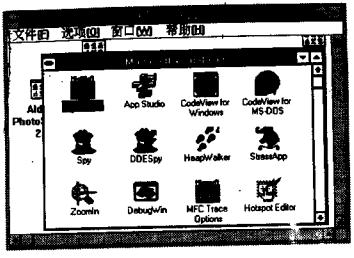
- －c：表示只编译即可。
- －G2sw：由下列三个参数组成：
  －G2：告诉编译程序产生 286 码。
  －Gs：不作堆栈溢出检查。用户最好设置足够的堆栈空间。
  －Gw：可在程序的 far 类型函数内换上特殊代码，以后此特殊代码将便于 Windows 操作系统在内存中移动此程序和数据段。
- －Ow：使用此参数可避免因执行某些编译优化操作而可能遇到的问题。
- －W3：可显示 level 3 级警告信息。通常某个程序经编译后，即使出现警告信息也不影响它的运行。建议用户编写不含警告信息的程序。
- －Zp：促使在字节边界上的结构字段“堆紧”，特别是当某些程序要与 Windows

操作系统通信时，Windows 操作系统均假设所有结构都是被堆栈的。

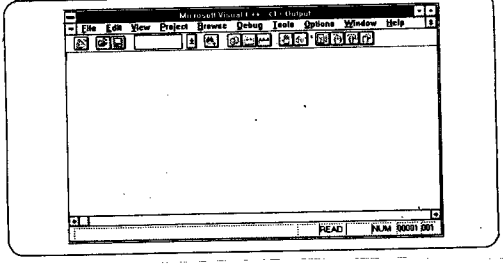
- -Tp：促使 C 语言程序在 C++环境下编译。如果以后将一些 C++特色的程序代码加到本程序内，则可避免一些错误。

2. 在集成环境下建立与运行程序

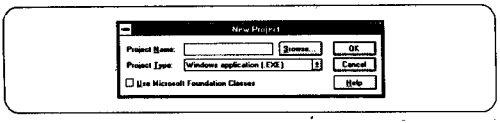
Visual C++集成环境又称 Workbench，在此环境下运行 Windows 应用程序非常方便。建立 C 语言程序文件后，可以利用 Visual C++所提供的 PROJECT 功能，让 Workbench 集成软件辅助建立 MAKE 类型的文件(.mak)和模块定义文件(.def)。例如，假设 c:\msvc\book4\ch1\ch1_1 目录中含有 ch1_1.c 的 Windows 应用程序，用户首先应进入 Windows 环境，如下所示(假定将 Visual C++安装在中文 Windows 下)：



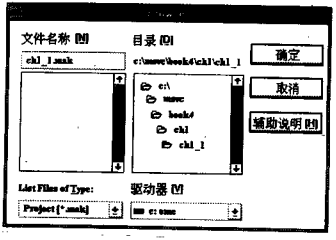
用鼠标双击 Visual C++图标后将可看到下列 Visual C++的 Workbench 集成环境：



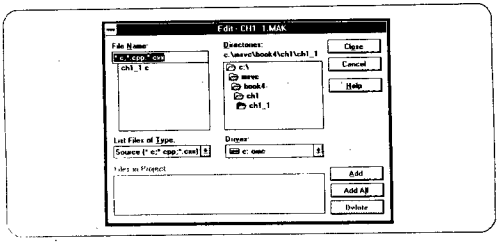
为了建立 MAKE 类型的文件(计划文件),应在 Project 菜单中选择 New,用户将在屏幕中看到下列对话框:



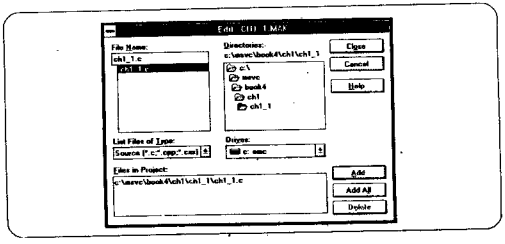
在上述对话框中,Use Microsoft Foundation Classes 字段用于设置是否使用 Microsoft Foundation Classes。对 C 语言程序而言,可以不使用它,所以它的核选框应为空白。Project Name 字段供用户输入将要建立的 MAKE 文件名(此文件名最好包含路径)。为了方便,可以先按下 Browse 按钮,设置目录后再输入文件名。例如,按下 Browse 按钮后,假设在目录字段设置 c:\msvc\book4\ch1\ch1_1,且在文件名称字段中输入 ch1_1.mak,如下所示:



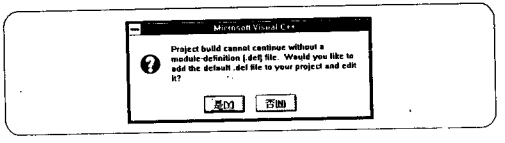
按下确定按钮(英文 Windows 中称 OK 按钮)后,将返回 New Project 对话框。请按 OK 按钮,此时出现下列对话框,要求用户编辑 ch1_1.mak 文件的内容,如下所示:



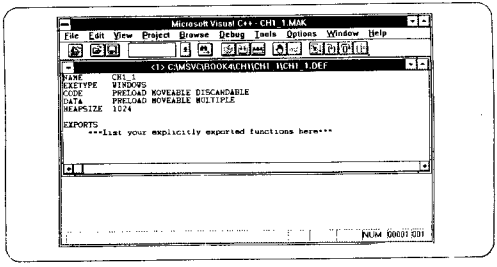
可以在 File Name 字段选择将要放在 chg1_1.mak 内的文件，选好后按 Add 按钮便可以将该文件放在下方的 File in Project 框内，如下所示：



按下 close 按钮之后，可以返回 Visual C++集成环境。如果想建立可执行文件，则可选择 Project 菜单中的 Build CH1_1.EXE 语句，此时可能会看到下列对话框：



总之，如果没有模块定义文件(.def)，则将无法建立可执行文件。是否要将系统缺省的模块定义文件加到当前的计划文件中呢？按下“是”按钮(英文 Windows 中称 Yes 按钮)，用户将可看到下列被加入计划文件内的模块定义文件：



此时如果执行 Project 菜单中的 Build All CH1_1.EXE 语句，便可直接运行 Windows 的可执行文件 CH1_1.EXE。也可以执行 Project 菜单中的 Build CH1_1.EXE，用户将看到下列对话框：

当计划文件的内容有变动时，如果用户想建立更改后的计划文件内容，按下“是”(英文 Windows 称 Yes)按钮，则可以看到“0 error(s)，1 warning(s)”，相当于零错误警告，这样就算成功地建立了 Windows 应用程序的可执行文件 CH1_1.EXE。

成功地建立了 CH1_1.EXE 可执行文件后，选择 Project 菜单的 Execute CH1_1.EXE 即可运行此应用程序。以后本书中的所有程序都以此方法建立可执行文件。

3. 运行结果

运行 CH1_1.EXE 之后，用户可在屏幕上看到下列运行结果：